CyBOK

The Cyber Security Body of Knowledge

Version 1.1.0
31st July 2021
https://www.cybok.org/

EDITORS

Awais Rashid | University of Bristol
Howard Chivers | University of York
Emil Lupu | Imperial College London
Andrew Martin | University of Oxford
Steve Schneider | University of Surrey

PROJECT MANAGERS

Helen Jones | University of Bristol
Yvonne Rigby | University of Bristol

PRODUCTION

Chao Chen | University of Bristol
Joseph Hallett | University of Bristol

COPYRIGHT

The CyBOK project would like to understand how the CyBOK is being used and its uptake. The project would like organisations using, or intending to use, CyBOK for the purposes of education, training, course development, professional development etc. to contact it at **contact@cybok.org** to let the project know how they are using CyBOK.

CHANGELOG

Introduction

Version date	Version number	Changes made
July 2021	1.1.0	Updated copyright statement; amended “issue” to “version”; added references to new KAs; updated diagram; removed Formal Methods cross-cutting topic.
October 2019	1.0	

Risk Management and Governance

Version date	Version number	Changes made
July 2021	1.1.1	Amended “issue” to “version”
March 2021	1.1.0	Updated copyright statement; added new section on Vulnerability Management; corrected typos
October 2019	1.0	

Law & Regulation

Version date	Version number	Changes made
July 2021	1.0.2	Amended “issue” to “version”; corrected typos
March 2021	1.0.1	Updated copyright statement; in Section 3.6.6 amended *“contacting parties”* to *“contracting parties”*
October 2019	1.0	

Human Factors

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Privacy & Online Rights

Version date	Version number	Changes made
July 2021	1.0.2	Amended “issue” to “version”; corrected typos
March 2021	1.0.1	Updated copyright statement; amended spelling error in Section 5.1.2: *“patters”* to *“patterns”*; added missing word in Section 5.1.2: *“On the hand”* to *“On the one hand”*
October 2019	1.0	

Malware & Attack Technologies

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Adversarial Behaviours

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Security Operations & Incident Management

Version date	Version number	Changes made
July 2021	1.0.2	Amended “issue” to “version”; corrected typos
March 2021	1.0.1	Updated copyright statement; updated SOAR acronym to *Security Orchestration, Automation and Response*; amended Section 8.1.2 from *“analysts man consoles”* to *“analysts monitor consoles”*
October 2019	1.0	

Forensics

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Cryptography

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Operating Systems and Virtualisation

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Distributed Systems Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Formal Methods for Security

Version date	Version number	Changes made
July 2021	1.0	

Authentication, Authorisation & Accountability

Version date	Version number	Changes made
July 2021	1.0.2	Fixed typographical issues in abstract; amended “issue” to “version”
March 2021	1.0.1	Updated copyright statement; removed AAA abbreviation from KA title; added footnote regarding use of the acronym AAA
October 2019	1.0	

Software Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Web & Mobile Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Secure Software Lifecycle

Version date	Version number	Changes made
July 2021	1.0.2	Amended “issue” to “version”; corrected typos
March 2021	1.0.1	Updated copyright statement; amended Section 17.4 from *“Four assessment approaches are described in this section”* to *“Three assessment approaches are described in this section”*
October 2019	1.0	

Applied Cryptography

Version date	Version number	Changes made
July 2021	1.0	

Network Security

Version date	Version number	Changes made
July 2021	2.0	Major revision with restructuring to place the network protocol discussion in a wider context and mention of a broader range of network architectures.
October 2019	1.0	

Hardware Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Cyber-Physical Systems Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”
October 2019	1.0	

Physical Layer and Telecommunications Security

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

Preface

We are pleased to share CyBOK Version 1.1.0 with you. This version has added two new Knowledge Areas to CyBOK: *Applied Cryptography* and *Formal Methods for Security*. It also includes a major revision to *Network Security*, a minor revision to *Risk Management & Governance* and a host of small typographical fixes. We have also added an extensive index of terms as a lookup mechanism.

CyBOK 1.1.0 is a continuation of the journey that began on the 1st of February 2017, when we started our *Scoping Phase (Phase I)*. This involved a range of community consultations, both within the UK and internationally, through a number of different activities designed to gain input from as wide an audience as possible. The activities included:

- 11 community workshops with 106 attendees across the UK;
- 44 responses through an online survey;
- 13 position statements;
- 10 in-depth interviews with key experts internationally across the socio-technical spectrum of cyber security; and
- 28 responses to a paper-based exercise as part of a panel at the Advances in Security Education Workshop at the USENIX Security Symposium 2017 in Vancouver, Canada.

There was a balance of inputs from academia and practitioners across most of these consultations.

We complemented the consultations with analysis of a number of documents that typically list key topics relevant to cyber security. Example documents included:

- Categorisations, such as the ACM Computing Classification System (CCS) taxonomy;

- Certifications, such as Certified Information Systems Security Professional (CISSP) and the Institute of Information Security Professionals (IISP) Skills Framework;
- Calls for papers such as IEEE Symposium on Security & Privacy and USENIX Symposium on Usable Privacy and Security;
- Existing curricula, such as the ACM computer science curriculum and the work of the Joint Task Force on Cybersecurity Education;
- Standards, such as BS ISO-IEC 27032 2021 and NIST IR 7298; and
- Tables of contents of various textbooks.

We used a variety of text-mining techniques, such as natural language processing and automatic text clustering to group relevant topics and identify relationships between topics. A two-day workshop of the editors and researchers involved in the scoping synthesised the various analyses to identify the 19 Knowledge Areas (KAs) that formed the scope of CyBOK Version 1.0. These were published for community feedback. Although none of the 19 KAs needed to be removed or new ones added on the basis of the feedback, the topics to be covered under each KA were refined. The KAs were also categorised into five top-level categories. Version 2.0 of the Scope document was published on the CyBOK website in October 2017 and formed the basis of the KAs in CyBOK Version 1.0. The details of the scoping work are discussed in the following article:

> Awais Rashid, George Danezis, Howard Chivers, Emil Lupu, Andrew Martin, Makayla Lewis, Claudia Peersman (2018). **Scoping the Cyber Security Body of Knowledge**. *IEEE Security & Privacy 16(3): 96-102*.

In *Phase II* (which started on the 1st of November 2017), the authoring of the 19 KAs began. For each KA, we drew up a list of internationally recognised experts on the topic as candidate authors and panels of reviewers. These lists were scrutinised by the CyBOK project's Professional Advisory Board and Academic Advisory Board. Following their input, and any updates, we invited a leading international expert to author each KA and a set of key experts as members of a peer-review panel to provide review and feedback on the KA, under the management of one of the CyBOK editors.

Each author prepared a *strawman* proposal for initial review and feedback by the expert peer-review panel. This was followed by a full draft (*woodenman*), which was reviewed by the panel, generating feedback for the authors—and often multiple iterations of discussion and updates. Once all feedback from the review panel had been addressed, the author prepared a draft for public review (*tinman*)[1]. The public review for each KA remained open for 4 weeks. All comments received from the public review were considered and, where appropriate, updates were made to the KA. If a comment was not addressed, a clear rationale was recorded for the decision. Following these updates, Version 1.0 of the KA was released on the CyBOK website. These collectively formed the CyBOK Version 1.0.

In addition to the authoring of the KAs, work was undertaken by the project team to identify learning pathways through CyBOK. This involved analysis of a number of curricular frameworks, professional certifications and academic degree programmes to study their coverage

[1]Due to the time constraints for *Phase II*, the panel and public reviews for Web & Mobile Security and Authentication, Authorisation & Accountability were conducted in parallel.

and focus with regards to CyBOK. A first analysis of four curricular frameworks was provided in the following paper:

Joseph Hallett, Robert Larson, Awais Rashid (2018). **Mirror, Mirror, On the Wall: What are we Teaching Them All? Characterising the Focus of Cybersecurity Curricular Frameworks**. Proceedings of the Advances in Security Education Workshop, USENIX Security Symposium.

Further analyses are available on the CyBOK website along with the knowledge trees derived from each KA for ease of mapping any curricular framework, degree programme or professional certification onto the CyBOK. The website also contains a wealth of other material such as webinars and podcasts—resources that complement the text and reference material covered within the KAs. Work was also done to identify the potential pathways for usage of CyBOK in school curricula, which was reported in the following paper:

Denny Pencheva, Joseph Hallett, Awais Rashid (2020). **Bringing Cyber to School: Integrating Cybersecurity Into Secondary School Education**. *IEEE Security & Privacy 18(2): 68-74*.

Phase II of the CyBOK project concluded with CyBOK Version 1.0 on the 31st of October 2019.

The project entered into *Phase III*, under the guidance of a newly-formed Steering Committee of academics and practitioners. *Phase III* focused on four main activities:

Maintenance and Evolution. A change management process was defined by the CyBOK Executive Team (the editors) and reviewed and approved by the CyBOK Steering Committee. The flowchart of this process is published on the CyBOK website. A call for community input for (major or minor) updates to CyBOK was opened (indefinitely) on the website. Any proposed changes are formulated into a change request, with major changes leading to an updated scope for the KA (or a new scope, in case of new KAs) that is published on the CyBOK website for community comments before forming the basis of a new or revised KA. The major revisions or new KAs followed the same process of development and community consultation as was the case for KAs in *Phase II*.

CyBOK Mapping Framework. An extensive framework was developed to support mapping of professional and academic education and training programmes onto CyBOK. This framework has been published on the CyBOK website along with the mapping resources, e.g., CyBOK knowledge trees, an extensive mapping reference, written step-by-step guides as well as exemplar mappings. An instantiation of the framework for the National Cyber Security Centre (NCSC) degree certification programme – which was updated in 2020 to use CyBOK Version 1.0 as its basis – was also provided along with the supporting resources for programme directors to be able to traceably map their degree contents onto the new certification requirements.

One-to-One Mapping Support and Feedback. Direct support was also provided by one of the researchers to programme directors with mapping their degree programmes onto the new CyBOK-based certification. Feedback on the mapping framework was also gathered through interviews. The NCSC certification programme provides a successful large-scale case study of CyBOK's usage – on a national scale – both within a certification

framework and a variety of university-level programmes.

Engagement with the Cyber Security Council Formation Project. The CyBOK Executive Board also actively supported the efforts underpinning the development of a new professional body in the UK: The Cyber Security Council. In this regard, members of the Executive Board and the chair of the Steering Committee contributed to the Formation Project's Board, its Advisory Board and a coordination group in order to improve synergy and cross-fertilisation of ideas and plans across CyBOK and the Council Formation Project. The Mapping Framework was also applied to a number of professional certification programmes, with detailed mappings and datasets produced to inform the qualifications framework being developed as part of the Council Formation Project.

Several key principles have underpinned the development of CyBOK throughout its development to date:

International focus. Though the project is funded by the National Cyber Security Programme in the UK, it is a truly international effort—engaging expert authors and reviewers around the world to develop a collective foundation for the discipline of cyber security.

For the community, by the community. The editors have led the effort but the scope, the KAs, the reviews and updates are all driven by community inputs.

Transparency. All outputs from the project work are made available on the website, including the scope, the guide to authors, draft and release versions of KAs and the analyses of curricular frameworks, university programmes and professional certifications. The only exceptions to this are the individual comments from the scoping work, expert panels or public review which, due to ethics requirements, cannot be made public. The same holds for the universities that voluntarily contributed their programmes for analysis on the assurance that the programme or university would not be made public.

Free and openly accessible. CyBOK is a community resource, freely available under the Open Government License. A key, over-arching guiding principle for CyBOK Version 1.0 and any future versions of CyBOK is that it remains an open and freely available resource for the community, e.g., it will not be placed behind a pay wall or a login page.

Academic independence. The editorial team has had full academic independence and all editorial decisions have solely rested with the editors.

CyBOK has now entered *Phase IV* where these principles will continue to be the basis of its development and evolution. The call for community input remains open and proposals for updates are reviewed and considered by the Executive Board on a fortnightly basis. Work is also underway to support the update of the NCSC's certification scheme – and also the degree programmes applying for this certification – to transition to CyBOK 1.1.0 with updates to associated resources such as the knowledge trees and the mapping reference. The Executive Board is also undertaking a review of the content to identify areas where further community consultation – in the form of further scoping discussions – may be required to scope future changes to CyBOK. The CyBOK team will also be working in collaboration with the newly-formed Cyber Security Council to identify further usages of CyBOK in professional qualifications in order to enable a rigorous *knowledge-based* underpinning for the profession.

Throughout the various phases of CyBOK to date, it has become clear that it offers a range of opportunities in transforming cyber security education and training programmes. It provides a rigorous knowledgebase to study, strengthen and update the focus of various professional

certification programmes. Opportunities also exist in terms of providing a basis for job descriptions so that employers can clearly articulate and evaluate the knowledge they expect from potential cyber security recruits. Furthermore, given its comprehensive nature, CyBOK can be used to benchmark cyber security capacity within an organisation or even across a nation. This is an on-going journey where more use cases and applications of CyBOK will emerge. We look forward to working with colleagues around the world on future updates and usage.

ACKNOWLEDGEMENTS

CyBOK Version 1.1.0 would not have been possible without the continuing support of the UK's National Cyber Security Programme that provided the funding for the project. We are grateful to the numerous colleagues across the cyber security community who provided input throughout the various phases of CyBOK: advice and comments on scopes, authoring and reviewing of KAs, acting in an advisory capacity in the various boards and the steering committee, or, informally guidance, during presentations and discussions at various conferences and events. We are also thankful to our project managers, Yvonne Rigby and Helen Jones, for their diligence, commitment and endless energy in coordinating the work of a large number of experts across the world. We are also grateful to the researchers who worked on the project, most notably Lata Nautiyal, who developed the mapping framework and the extensive resources for the NCSC degree certification programmes as well as the professional certifications, Joseph Hallett for his research on curricular frameworks, learning pathways and the knowledge trees, Chao Chen and Joseph Hallett for the extensive work on the production of CyBOK including management of the backend systems that control configurations in line with our change management process, Ben Shreeve and Anthony Mazeli for undertaking some of the mappings and James Clements for the detailed work on the CyBOK index.

We would like to thank our colleagues in NCSC who formulated the idea of, and secured funding for, a cyber security body of knowledge project. We also thank Fred Piper and Steven Furnell who provided input and comments as external reviewers for the NCSC during Phase I and II. The researchers, authors, expert panel members, advisory boards and steering committee are listed below. Last but not least, our thanks to the wider cyber security community of researchers, educators and practitioners for providing critical and constructive input throughout to help shape the scope and the KAs for CyBOK Version 1.1.0.

Researchers

Lata Nautiyal | University of Bristol
Joseph Hallett | University of Bristol
Chao Chen | University of Bristol
James Clements | University of Bristol
Benjamin Shreeve | University of Bristol
Anthony Mazeli | University of Bristol
Robert Larson | University of Bristol
Claudia Peersman | University of Bristol
Makayla Lewis | University of Lancaster

Authors, Editors and Reviewers

Knowledge Area	Author	Editor	Reviewers
Risk Management & Governance	Pete Burnap	Awais Rashid	Chris Johnson Ragnar Lofstedt Jason Nurse Adam Shostack
Law & Regulation	Robert Carolina	Howard Chivers	Tom Holt Madeline Carr Roderic Broadhurst
Human Factors	M. Angela Sasse Awais Rashid	Awais Rashid	Pam Briggs Lorrie Faith Cranor Matthew Smith Rick Wash Mary Ellen Zurko
Privacy & Online Rights	Carmela Troncoso	George Danezis Awais Rashid	Emiliano De Cristofaro Ian Goldberg Theresa Stadler
Malware & Attack Technologies	Wenke Lee	Howard Chivers	Alex Berry Lorenzo Cavallaro Mihai Christodorescu Igor Muttik
Adversarial Behaviours	Gianluca Stringhini	Awais Rashid	Tom Holt Adam Joinson Damon McCoy Paul Taylor
Security Operations & Incident Management	Hervé Debar	Howard Chivers	Douglas Wiemer Magnus Almgren Marc Dacier Sushil Jajodia
Forensics	Vassil Roussev	Howard Chivers	Bruce Nikkel Marc Kirby Paul Birch Zeno Geradts
Cryptography	Nigel Smart	George Danezis	Dan Bogdanov Kenny Paterson Liqun Chen
Operating Systems & Virtualisation	Herbert Bos	Andrew Martin	Chris Dalton David Lie Gernot Heiser Mathias Payer
Distributed Systems Security	Neeraj Suri	Emil Lupu	Konstantin Beznosov Marko Vukolić

Chapter	Author	Editor	Reviewers
Formal Methods for Security	David Basin	Steve Schneider	Rod Chapman Stephen Chong Mark Ryan Andrei Sabelfeld
Authentication, Authorisation & Accountability	Dieter Gollmann	Emil Lupu	Gail-Joon Ahn Michael Huth Indrakshi Ray
Software Security	Frank Piessens	Awais Rashid	Eric Bodden Rod Chapman Michael Hicks Jacques Klein Andrei Sablefeld
Web & Mobile Security	Sascha Fahl	Emil Lupu	Alastair Beresford Sven Bugiel Hao Chen Paul Freemantle Marco Viera
Secure Software Lifecycle	Laurie Williams	Andrew Martin	Rod Chapman Fabio Massacci Gary McGraw Nancy Mead James Noble Riccardo Scandariato
Applied Cryptography	Kenneth G. Paterson	Steve Schneider	Dan Bogdanov Liqun Chen Keith Martin Nigel Smart
Network Security	Christian Rossow Sanjay Jha	Andrew Martin	Amir Herzberg Matthias Hollick Sanjay Jha Shishir Nagaraja Gene Tsudik
Hardware Security	Ingrid Verbauwhede	Andrew Martin George Danezis	Srinivas Devadas Paul England Elisabeth Oswald Mark Ryan
Cyber-Physical Systems Security	Alvaro Cardenas	Emil Lupu	Henrik Sandberg Marina Krotofil Mauro Conti Nils Ole Tippenhauer Rakesh Bobba
Physical Layer & Telecommunications Security	Srdjan Čapkun	George Danezis Awais Rashid	Robert Piechocki Kasper Rasmussen

Professional Advisory Board (Phase I and II)

Sir Edmund Burton | Chair
Amanda Finch | Chartered Institute of Information Security (CIISec)
Nick Coleman | The Institute of Engineering & Technology (IET)
Diana Burley | The George Washington University
David King | Legal & General
Claire Vishik | Intel Corporation
Bill Mitchell | BCS Academy of Computing
Andrew Rose | NATS (formerly)

Academic Advisory Board (Phase I and II)

Bart Preneel | KU Leuven, Belgium
Mira Mezini | Technische Universität Darmstadt, Germany
L. Jean Camp | Indiana University Bloomington, USA
Jill Slay | La Trobe University, Melbourne, Australia
Trent Jaeger | Pennsylvania State University, USA

Steering Committee (Phase III)

Sir Edmund Burton | Chair
Amanda Finch | Chartered Institute of Information Security (CIISec)
Steven Furnell | Representative UK Cyber Security Council Formation Project
Heather Goldstraw | Defense Academy
Trent Jaeger | Pennsylvania State University, USA
David King | Legal & General
Mira Mezini | Technische Universität Darmstadt, Germany
Claire Vishik | Intel Corporation
Laurie Williams | North Carolina State University, USA
Awais Rashid | University of Bristol, UK (ex-officio, Executive Board)
Howard Chivers | University of York, UK (ex-officio, Executive Board)
Emil Lupu | Imperial College London, UK (ex-officio, Executive Board)
Andrew Martin | University of Oxford, UK (ex-officio, Executive Board)
Steve Schneider | University of Surrey, UK (ex-officio, Executive Board)

Steering Committee (Phase IV)

Sir Edmund Burton | Chair
Steven Furnell | Chartered Institute of Information Security (CIISec)
Heather Goldstraw | Defense Academy
Trent Jaeger | Pennsylvania State University, USA
David King | Legal & General
Emil Lupu | Imperial College London, UK
Mira Mezini | Technische Universität Darmstadt, Germany
Claire Vishik | Intel Corporation
Laurie Williams | North Carolina State University, USA
Awais Rashid | University of Bristol, UK (ex-officio, Executive Board)
Howard Chivers | University of York, UK (ex-officio, Executive Board)
Yulia Cherdantseva | University of York, UK (ex-officio, Executive Board)
Andrew Martin | University of Oxford, UK (ex-officio, Executive Board)
Steve Schneider | University of Surrey, UK (ex-officio, Executive Board)

Website Team

James Brown
Adrian Tucker
Kamil Hucal
David Graves

Contents

Chapter 1
Introduction

Andrew Martin | University of Oxford
Awais Rashid | University of Bristol
Howard Chivers | University of York
George Danezis | University College London
Steve Schneider | University of Surrey
Emil Lupu | Imperial College London

Cyber security is becoming an important element in curricula at all education levels. However, the foundational knowledge on which the field of cyber security is being developed is fragmented, and as a result, it can be difficult for both students and educators to map coherent paths of progression through the subject. By comparison, mature scientific disciplines like mathematics, physics, chemistry, and biology have established foundational knowledge and clear learning pathways. Within software engineering, the IEEE Software Engineering Body of Knowledge [3] codifies key foundational knowledge on which a range of educational programmes may be built. There are a number of previous and current efforts on establishing skills frameworks, key topic areas, and curricular guidelines for cyber security. However, a consensus has not been reached on what the diverse community of researchers, educators, and practitioners sees as established foundational knowledge in cyber security.

The Cyber Security Body of Knowledge (CyBOK) aims to codify the foundational and generally recognised knowledge on cyber security. In the same fashion as SWEBOK, CyBOK is meant to be a guide to the body of knowledge; the knowledge that it codifies already exists in literature such as textbooks, academic research articles, technical reports, white papers, and standards. Our focus is, therefore, on mapping established knowledge and not fully replicating everything that has ever been written on the subject. Educational programmes ranging from secondary and undergraduate education to postgraduate and continuing professional development programmes can then be developed on the basis of CyBOK.

This introduction sets out to place the 21 Knowledge Areas (KAs) of the CyBOK into a coherent overall framework. Each KA assumes a baseline agreement on the overall vocabulary, goals, and approaches to cyber security, and here we provide that common material which underpins the whole body of knowledege. We begin with an overview of cyber security as a topic, and some basic definitions, before introducing the knowledge areas. The KAs and their groupings into categories are, of course, not orthogonal and there are a number of dependencies across the KAs which are cross-referenced and also separately captured visually on the CyBOK web site (**https://www.cybok.org**). We then discuss how the knowledge in the KAs can be deployed to understand the means and objectives of cyber security, mitigate against failures and incidents, and manage risks.

Although we have necessarily divided the CyBOK into a number of discrete Knowledge Areas (KAs), it is clear that there are many inter-relationships among them. Those with professional responsibility for one area must typically have at least a moderate grasp of the adjacent topics; someone responsible for architecting a secure system must understand many. There are a number of *unifying principles* and crosscutting themes that underpin the development of systems that satisfy particular security properties. We conclude the introduction by discussing some of these.

1.1 CYBER SECURITY DEFINITION

The CyBOK Knowledge Areas assume a common vocabulary and core understanding of a number of topics central to the field. Whilst this *Body of Knowledge* is *descriptive* of existing knowledge (rather than seeking to innovate, or constrain), it is evident that use of widely-shared terminology in an established concept map is crucial to the development of the discipline as a whole. Since our main aim is to provide a *guide* to the Body of Knowledge, we will provide references to other definitions, rather than introducing our own.

Cyber security has become an encompassing term, as our working definition illustrates:

> **Definition:** ***Cyber security*** refers to the protection of information systems (hardware, software and associated infrastructure), the data on them, and the services they provide, from unauthorised access, harm or misuse. This includes harm caused intentionally by the operator of the system, or accidentally, as a result of failing to follow security procedures.
>
> *UK National Cyber Security Strategy [4]*

This is a succinct definition but expresses the breadth of coverage within the topic. Many other definitions are in use, and a document from ENISA [5] surveys a number of these.

The consideration of human behaviours is a crucial element of such a definition—but arguably still missing is a mention of the impact on them from loss of information or reduced safety, or of how security and privacy breaches impact trust in connected systems and infrastructures. Moreover, security must be balanced with other risks and requirements—from a human factors perspective there is a need not to disrupt the primary task.

A large contributor to the notion of cyber security is *Information Security*, widely regarded as comprised of three main elements:

> **Definition:** ***Information security.*** Preservation of confidentiality, integrity and availability of information.
>
> In addition, other properties, such as authenticity, accountability, non-repudiation, and reliability can also be involved.
>
> *ISO 27000 definition [6]*

For definitions of the subsidiary terms, the reader is referred to the ISO 27000 definitions [6].

Through the developing digital age other 'securities' have had prominence, including *Computer Security* and *Network Security*; related notions include *Information Assurance*, and *Systems Security* — perhaps within the context of *Systems Engineering* or *Security Engineering*. These terms are easily confused, and it seems that often one term is used when another is meant.

Many of those terms were subject to the criticism that they place an over-reliance on technical controls, and focus almost exclusively on *information*. Stretching them to relate to cyber-physical systems may be taking them too far: indeed, our working definition above privileges the notion of *information* (whilst also mentioning *services*) — whereas in the case of network-connected actuators, the pressing challenge is to prevent unwanted *physical actions*.

Moreover, in some accounts of the topic, cyberspace is best understood as a 'place' in which business is conducted, human communications take place, art is made and enjoyed, relationships are formed and developed, and so on. In this place, cyber crime, cyber terrorism, and cyber war may occur, having both 'real' and 'virtual' impacts. Taken as a whole, the CyBOK delineates a large range of topics which appear to be within the broad scope of *cyber security*, even if a succinct reduction of those into a short definition remains elusive. The full scope of CyBOK may serve as an extended definition of the topic—as summarised next.

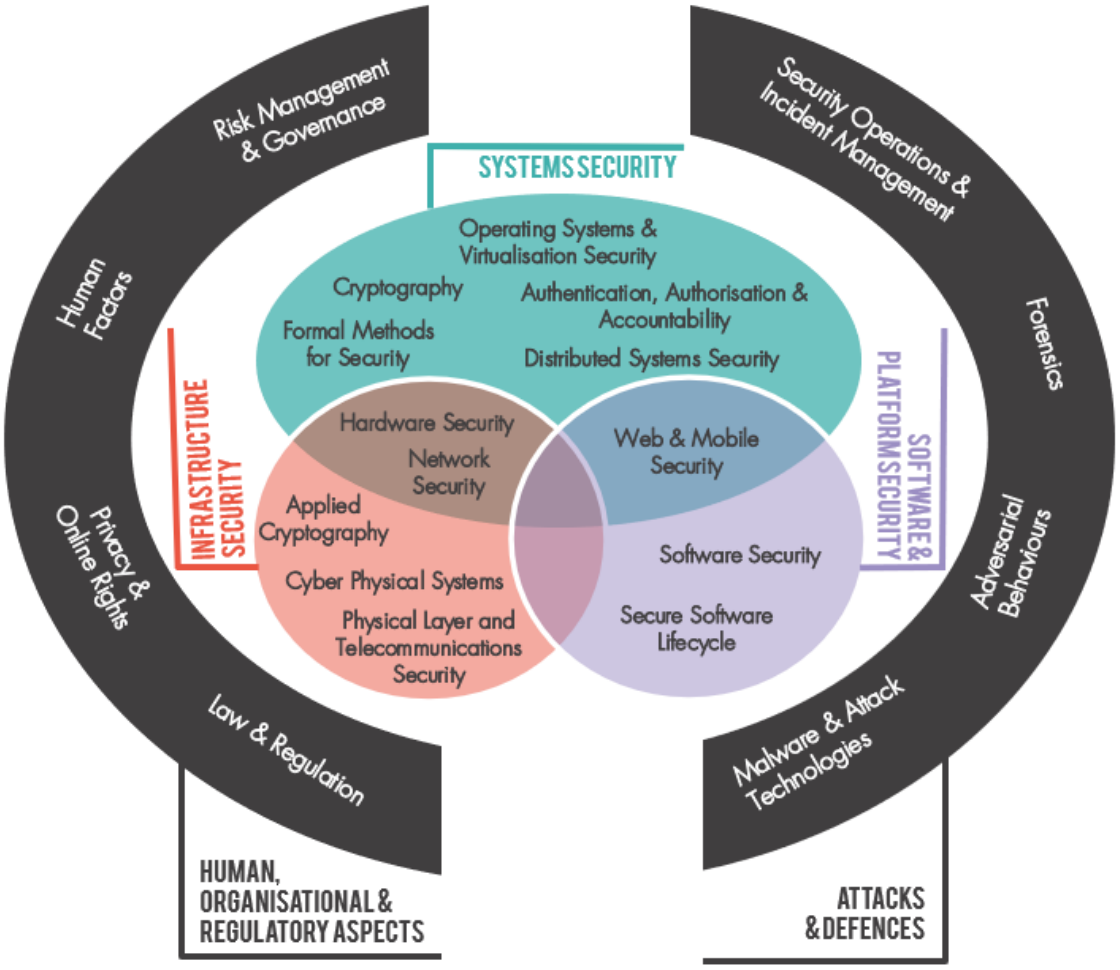


Figure 1.1: The 21 Knowledge Areas (KAs) in the CyBOK Scope

1.2 CYBOK KNOWLEDGE AREAS

The CyBOK is divided into 21 top-level Knowledge Areas (KAs), grouped into five broad categories, as shown in Figure 1.1. Clearly, other possible categorisations of these KAs may be equally valid, and ultimately some of the structure is relatively arbitrary. The CyBOK Preface describes the process by which these KAs were identified and chosen.

Our categories are not entirely orthogonal. These are intended to capture knowledge relating to cyber security *per se*: in order to make sense of some of that knowledge, auxiliary and background knowledge is needed — whether in the design of hardware and software, or in diverse other fields, such as law.

	Human, Organisational and Regulatory Aspects
Risk Management & Governance	Security management systems and organisational security controls, including standards, best practices, and approaches to risk assessment and mitigation.
Law & Regulation	International and national statutory and regulatory requirements, compliance obligations, and security ethics, including data protection and developing doctrines on cyber warfare.
Human Factors	Usable security, social & behavioural factors impacting security, security culture and awareness as well as the impact of security controls on user behaviours.
Privacy & Online Rights	Techniques for protecting personal information, including communications, applications, and inferences from databases and data processing. It also includes other systems supporting online rights touching on censorship and circumvention, covertness, electronic elections, and privacy in payment and identity systems.
	Attacks and Defences
Malware & Attack Technologies	Technical details of exploits and distributed malicious systems, together with associated discovery and analysis approaches.
Adversarial Behaviours	The motivations, behaviours, & methods used by attackers, including malware supply chains, attack vectors, and money transfers.
Security Operations & Incident Management	The configuration, operation and maintenance of secure systems including the detection of and response to security incidents and the collection and use of threat intelligence.
Forensics	The collection, analysis, & reporting of digital evidence in support of incidents or criminal events.
	Systems Security
Cryptography	Core primitives of cryptography as presently practised & emerging algorithms, techniques for analysis of these, and the protocols that use them.
Operating Systems & Virtualisation Security	Operating systems protection mechanisms, implementing secure abstraction of hardware, and sharing of resources, including isolation in multiuser systems, secure virtualisation, and security in database systems.
Distributed Systems Security	Security mechanisms relating to larger-scale coordinated distributed systems, including aspects of secure consensus, time, event systems, peer-to-peer systems, clouds, multitenant data centres, & distributed ledgers.
Formal Methods for Security	Formal specification, modelling and reasoning about the security of systems, software and protocols, covering the fundamental approaches, techniques and tool support.
Authentication, Authorisation & Accountability	All aspects of identity management and authentication technologies, and architectures and tools to support authorisation and accountability in both isolated and distributed systems.
	Software and Platform Security
Software Security	Known categories of programming errors resulting in security bugs, & techniques for avoiding these errors—both through coding practice and improved language design—and tools, techniques, and methods for detection of such errors in existing systems.
Web & Mobile Security	Issues related to web applications and services distributed across devices and frameworks, including the diverse programming paradigms and protection models.
Secure Software Lifecycle	The application of security software engineering techniques in the whole systems development lifecycle resulting in software that is secure by default.
	Infrastructure Security
Applied Cryptography	The application of cryptographic algorithms, schemes, and protocols, including issues around implementation, key management, and their use within protocols and systems.
Network Security	Security aspects of networking & telecommunication protocols, including the security of routing, network security elements, and specific cryptographic protocols used for network security.
Hardware Security	Security in the design, implementation, & deployment of general-purpose and specialist hardware, including trusted computing technologies and sources of randomness.
Cyber-Physical Systems Security	Security challenges in cyber-physical systems, such as the Internet of Things & industrial control systems, attacker models, safe-secure designs, and security of large-scale infrastructures.
Physical Layer & Telecommunications Security	Security concerns and limitations of the physical layer including aspects of radio frequency encodings and transmission techniques, unintended radiation, and interference.

Figure 1.2: Short descriptions of CyBOK Knowledge Areas

1.3 DEPLOYING CYBOK KNOWLEDGE TO ADDRESS SECURITY ISSUES

1.3.1 Means and objectives of cyber security

Implicit in the definitions above is that cyber security entails protection *against* an *adversary* or, possibly, against some other physical or random process. The latter implies some overlap between the notions of *safety* and *security*, although it is arguably possible to have either without the other. Within the security domain, if our modelling accounts for malice, it will necessarily encompass accidents and random processes. Therefore, core to any consideration of security is the modelling of these adversaries: their motives for attack, the threats they pose and the capabilities they may utilise.

In considering those threats, cyber security is often expressed in terms of instituting a number of *controls* affecting people, process, and technology. Some of these will focus on the *prevention* of bad outcomes, whereas others are better approached through *detection* and *reaction*. Selection of those controls is generally approached through a process of Risk Management (see below, and the Risk Management & Governance Knowledge Area (Chapter 2)) — although increasing emphasis is placed on Human Factors (see the Human Factors Knowledge Area (Chapter 4)), noting the need to leverage humans as a lynchpin for improving cyber security cultures, as well as supporting them to protect their privacy online (see the Privacy & Online Rights Knowledge Area (Chapter 5)).

Equally, security requires an analysis of *vulnerabilities* within the system under consideration: a (hypothetical) system without vulnerabilities would be impervious to all threats; a highly vulnerable system placed in totally benign circumstances (no threats) would have no security incidents, either.

The intended use of security controls gives rise to its own questions about whether they are deployed appropriately, and whether they are effective: these belong to the domain of *security assurance*, which has processes and controls of its own. These will involve residual risk analysis (see below, and the Risk Management & Governance Knowledge Area (Chapter 2)) which includes an attempt to measure and quantify the presence of vulnerabilities.

1.3.2 Failures and Incidents

When adversaries achieve their goal (wholly or partially) — when attacks succeed — the collection of security controls may be said to have failed. Alternatively, we may say that insufficient or ineffective controls were in place. Operationally speaking, one or more failures may give rise to a security incident. Typically such incidents may be described in terms of the harm to which they give rise: according to our definition of cyber security, these typically amount to harm from theft or damage of information, devices, services, or networks. The cyber-physical domain (see the Cyber-Physical Systems Security Knowledge Area (Chapter 21)) gives rise to many additional potential harms—harms to humans may come from either information, or from unintended physical action, or from both.

A significant sub-discipline of operational security considers detection of security failures, and reactions to them (remediation where possible). The Security Operations & Incident Management Knowledge Area (Chapter 8) addresses the context; the Malware & Attack Technologies Knowledge Area (Chapter 6) deals with analysis of attack vectors while the

Forensics Knowledge Area (Chapter 9) considers the technical details and processes for post-attack analysis in a robust and reliable manner.

A recurrent theme in security analysis is that it is not sufficient to define good security controls solely within a particular abstraction or frame of reference: it is necessary also to consider what may happen if an adversary chooses to ignore that abstraction or frame.

This arises, for example, in communication *side channels*, where an adversary may infer much from capturing radio frequency emissions from a cable, say, without needing to tap that cable physically. Similar eavesdropping effects have been observed against cryptography implemented on smartcards: simple analysis of the power consumption of the processor as it addresses each bit in turn can be sufficient to disclose the cryptographic key (see the Hardware Security Knowledge Area (Chapter 20) and the Software Security Knowledge Area (Chapter 15)). Appropriate use and handling of cryptographic keys is considered in Applied Cryptography Knowledge Area (Chapter 18), with underlying theory considered in Cryptography Knowledge Area (Chapter 10).

These problems occur at every level in the system design. In software, the *SQL injection attack* arises (see Software Security and Web & Mobile Security Knowledge Areas) because a string of characters intended to be interpreted as a database entry is forced to become a database command. Files holding secrets written by one application may give up those secrets when read by another, or by a general-purpose debugger or dump program.

Mathematical theories of refinement (and software development contracts) explore the relationship of an 'abstract' expression of an algorithm and a more 'concrete' version which is implemented: but security properties proven of the one may not be true of the other (for example, reducing uncertainty can increase information content and lead to the leak of information such as a cryptographic key), so great care must be taken in the construction of the theories. 'Black-box testing' relies on the same notion and, since it cannot possibly test every input, may easily miss the particular combination of circumstances which — by accident or design — destroys the security of the program. Carefully-constructed mathematical theories and models for security aim avoid these pitfalls and permit precise reasoning about security properties. Some of these are discussed in the Formal Methods for Security Knowledge Area (Chapter 13).

Operational security of a system may be predicated upon the operators following a particular procedure or avoiding particular dangerous circumstances: there is an assumption that if people are told in a professional context (not) to do something, then they will (not) do it. This is demonstrably false (see the Human Factors Knowledge Area (Chapter 4)).

These — and an endless array of other — security problems arise because it is necessary to think (and design systems) using abstractions. Not only can no individual comprehend every detail of the operation of a networked computing system (from the device physics upwards), even if they had the requisite knowledge they must work in abstractions in order to make progress and avoid being overwhelmed with detail. But, for the majority of security controls, the abstraction is no more than a thinking tool: and so the adversary is able to disregard it entirely.

Since abstractions are usually built in layers (and computing systems are usually explicitly designed in that way), this is sometimes known as the 'layer below' problem [7] because the adversary often attacks the layer below the one in which the abstraction defining the control sits (see, for example, the threats and attacks discussed in the Operating Systems & Virtualisation Knowledge Area (Chapter 11) and the Hardware Security Knowledge Area

(Chapter 20)).

1.3.3 Risk

There is no limit in principle to the amount of effort or money that might be expended on security controls. In order to balance these with the available resources and the harms and opportunities that might arise from emerging threats to security, a common over-arching approach to security analysis is a process of Risk Assessment — and selection of controls, a process of Risk Management. These are explored in depth in the Risk Management & Governance Knowledge Area (Chapter 2).

As with any process of risk management, a key calculation relates to expected impact, being calculated from some estimate of likelihood of events that may lead to impact, and an estimate of the impact arising from those events. The likelihood has two elements: the presence of *vulnerabilities* (known or unknown—the latter not always being capable of being mitigated), and the nature of the *threat*. The management response to the risk assessment may take many forms, including additional controls to reduce the impact or likelihood of a threat, accepting the risk, or transferring/sharing it with a third party (e.g., insurance), or in some cases deciding not to proceed because all of these outcomes are unacceptable.

Security management encompasses all the management and security actions necessary to maintain the security of a system during its lifetime. Important in this context, but outside of the scope of the CyBOK, are quality management practices. Such practices are long-established in industry, essentially requiring that all work follows documented processes, and that the processes provide metrics which are, in turn, reviewed and used to correct processes that are not fit for purpose ('nonconformities').

The analogy between quality management and security is not perfect because the threat environment is not static; however, the trend is for security management standards such as ISO/IEC 27001 to embody standard quality management processes which are then specialised for security. The primary specialisation is the periodic use of risk management (see the Risk Management & Governance Knowledge Area (Chapter 2)), which must also take account of the changing threat environment. It is necessary to supplement periodic risk management with continuous measures of the effectiveness of the security processes. For example, system patching and maintenance can be continuously reviewed via vulnerability scanning, logs relating to failed access attempts, user lock-outs or password resets can provide indicators of the usability of security features.

The functions within a security management system can be grouped into Physical, Personnel, Information Systems and Incident Management and are a mixture of standard IT system management functions and those that are specific to cyber security.

Physical security includes physical protection of the system, including access control, asset management and the handling and protection of data storage media. These aspects are outside the scope of the CyBOK.

Personnel security is concerned with a wide range of security usability and behaviour shaping, including education and training (see the Human Factors Knowledge Area (Chapter 4)). It also includes formal human-resource management elements such as the selection and vetting of staff, terms and conditions of acceptable usage for IT systems (see the Law & Regulation Knowledge Area (Chapter 3)) and disciplinary sanctions for security breaches.

Information system management includes access management (see the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)) and system logging (see the Security Operations & Incident Management Knowledge Area (Chapter 8)). The audit function is divided into security monitoring (see the Security Operations & Incident Management Knowledge Area (Chapter 8)) and other IT functions, such as volumetric review for system provisioning. Management of the information system also involves standard IT functions such as backup and recovery, and the management of supplier relationships.

Incident management functions (see the Security Operations & Incident Management Knowledge Area (Chapter 8)) are specific to cyber security and include security monitoring, incident detection and response.

1.4 PRINCIPLES

Sound thinking and good practice in security has been codified by a number of authors. The principles they describe touch many different KAs, and taken together help to develop a holistic approach to the design, development, and deployment of secure systems.

1.4.1 Saltzer and Schroeder Principles

The earliest collected design principles for engineering security controls were enumerated by Saltzer and Schroeder in 1975 [8]. These were proposed in the context of engineering secure multi-user operating systems supporting confidentiality properties for use in government and military organisations. This motivation does bias them in some ways, however they have also stood the test of time in being applicable to the design of security controls much more broadly.

The eight principles they enumerate are as follows:

- *Economy of mechanism.* The design of security controls should remain as simple as possible, to ensure high assurance. Simpler designs are easier to reason about formally or informally, to argue correctness. Further, simpler designs have simpler implementations that are easier to manually audit or verify for high assurance. This principle underlies the notion of Trusted Computing Base (TCB) — namely the collection of all software and hardware components on which a security mechanism or policy relies. It implies that the TCB of a system should remain small to ensure that it maintain the security properties expected.
- *Fail-safe defaults.* Security controls need to define and enable operations that can positively be identified as being in accordance with a security policy, and reject all others. In particular, Saltzer and Schroeder warn against mechanisms that determine access by attempting to identify and reject malicious behaviour. Malicious behaviour, as it is under the control of the adversary and will therefore adapt, is difficult to enumerate and identify exhaustively. As a result basing controls on exclusion of detected violation, rather than inclusion of known good behaviour, is error prone. It is notable that some modern security controls violate this principle including signature based anti-virus software and intrusion detection.
- *Complete mediation.* All operations on all objects in a system should be checked to ensure that they are in accordance with the security policy. Such checks would usually

involve ensuring that the subject that initiated the operation is authorised to perform it, presuming a robust mechanism for authentication. However, modern security controls may not base checks on the identity of such a subject but other considerations, such as holding a 'capability'.

- *Open design.* The security of the control must not rely on the secrecy of how it operates, but only on well specified secrets or passwords. This principle underpins cyber security as a field of open study: it allows scholars, engineers, auditors, and regulators to examine how security controls operate to ensure their correctness, or identify flaws, without undermining their security. The opposite approach, often called 'security by obscurity', is fragile as it restricts who may audit a security control, and is ineffective against insider threats or controls that can be reverse engineered.
- *Separation of privilege.* Security controls that rely on multiple subjects to authorise an operation, provide higher assurance than those relying on a single subject. This principle is embodied in traditional banking systems, and carries forward to cyber security controls. However, while it is usually the case that increasing the number of authorities involved in authorising an operation increases assurance around integrity properties, it usually also decreases assurance around availability properties. The principle also has limits, relating to over diluting responsibility leading to a 'tragedy of the security commons' in which no authority has incentives to invest in security assuming the others will.
- *Least privilege.* Subjects and the operations they perform in a system should be performed using the fewest possible privileges. For example, if an operation needs to only read some information, it should not also be granted the privileges to write or delete this information. Granting the minimum set of privileges ensures that, if the subject is corrupt or software incorrect, the damage they may do to the security properties of the system is diminished. Defining security architectures heavily relies on this principle, and consists of separating large systems into components, each with the least privileges possible — to ensure that partial compromises cannot affect, or have a minimal effect on, the overall security properties of a whole system.
- *Least common mechanism.* It is preferable to minimise sharing of resources and system mechanisms between different parties. This principle is heavily influenced by the context of engineering secure multi-user systems. In such systems common mechanisms (such as shared memory, disk, CPU, etc.) are vectors for potential leaks of confidential information from one user to the other, as well as potential interference from one user into the operations of another. Its extreme realisation sees systems that must not interfere with each other being 'air-gapped'. Yet, the principle has limits when it comes to using shared infrastructures (such as the Internet), or shared computing resources (such as multi-user operating systems, that naturally share CPUs and other resources).
- *Psychological acceptability.* The security control should be naturally usable so that users 'routinely and automatically' apply the protection. Saltzer and Schroeder, specifically state that 'to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimised'. This principle is the basis for the Human Factors Knowledge Area (Chapter 4).

Saltzer and Schroeder also provide two further principles, but warn that those are only imperfectly applicable to cyber security controls:

- *Work factor.* Good security controls require more resources to circumvent than those available to the adversary. In some cases, such as the cost of brute forcing a key, the

work factor may be computed and designers can be assured that adversaries cannot be sufficiently endowed to try them all. For other controls, however, this work factor is harder to compute accurately. For example, it is hard to estimate the cost of a corrupt insider, or the cost of finding a bug in software.

- *Compromise recording.* It is sometimes suggested that reliable records or logs, that allow detection of a compromise, may be used instead of controls that prevent a compromise. Most systems do log security events, and security operations heavily rely on such reliable logs to detect intrusions. The relative merits — and costs — of the two approaches are highly context-dependent.

Those principles in turn draw on much older precedents such as Kerckhoff's principles relating to cryptographic systems [9]. Kerchoff highlights that cryptographic systems must be practically secure, without requiring the secrecy of how they operate (open design). He also highlights that keys should be short and memorable, the equipment must be easy to use, and applicable to telecommunications — all of which relate to the psychological acceptability of the designs.

1.4.2 NIST Principles

More contemporary principles in systems design are enumerated by NIST[10, Appendix F]. They incorporate and extend the principles from Saltzer and Schroeder. They are categorised into three broad families relating to: 'Security Architecture and Design' (i.e., organisation, structure and interfaces); 'Security Capability and Intrinsic Behaviours' (i.e., what the protections are about); and 'Life Cycle Security' (i.e., those related to process and management). As such those principles specifically refer to security architecture, specific controls, as well as engineering process management.

A number of the NIST principles map directly to those by Saltzer and Schroeder, such as Least Common Mechanism, Efficiently Mediated Access, Minimised Sharing, Minimised Security Elements, Reduced Complexity, Least Privilege, Secure Defaults and Predicate Permission, and Acceptable Security.

Notably, new principles deal with the increased complexity of modern computing systems and emphasise clean modular design, i.e. with Clear Abstraction, Modularity and Layering, Partially Ordered Dependencies, Secure Evolvability. Other principles recognise that not all components in a secure system may operate at the same level of assurance, and call for those to benefit from a Hierarchical Trust structure, in which the security failure of some components does not endanger all properties in the system. The principle of Inverse Modification Threshold states that those components that are the most critical to security, should also be the most protected against unauthorised modification or tampering. Hierarchical protection states that least critical security components need not be protected from more critical ones.

The NIST framework also recognises that modern systems are interconnected, and provides principles of how to secure them. These should be networked using Trusted Communication Channels. They should enjoy Secure Distributed Composition, meaning that if two systems that enforce the same policy are composed, their composition should also at least enforce the same policy. Finally, the principle of Self-Reliant Trustworthiness states that a secure system should remain secure even if disconnected from other remote components.

The NIST principles expand on what types of security mechanisms are acceptable for real-world systems. In particular the principles of Economic Security, Performance Security, Human

Factored Security, and Acceptable Security state that security controls should not be overly expensive, overly degrade performance, or be unusable or otherwise unacceptable to users. This is a recognition that security controls support functional properties of systems and are not a goal in themselves.

Besides principles, NIST also outlines three key security architecture strategies. The Reference Monitor Concept is an abstract control that is sufficient to enforce the security properties of a system. Defence in Depth describes a security architecture composed on multiple overlapping controls. Isolation is a strategy by which different components are physically or logically separated to minimise interference or information leakage.

Both NIST, as well as Saltzer and Schroeder, highlight that principles provide guidance only, and need to be applied with skill to specific problems at hand to design secure architectures and controls. Deviation from a principle does not automatically lead to any problems, but such deviations need to be identified to ensure that any issues that may arise have been mitigated appropriately.

1.4.3 Latent Design Conditions

As more and more cyber-physical systems are connected to other systems and the Internet, the inherent complexity emerging from such large-scale connectivity and the safety critical nature of some of the cyber-physical systems means other principles also become highly relevant. One such principle is that of *Latent Design Conditions* from research in the safety-critical systems domain by James Reason [11]. In the context of cyber security, latent design conditions arise from past decisions about a system (or systems). They often remain hidden (or unconsidered) and only come to the fore when certain events or settings align — in the case of cyber-physical systems security vulnerabilities being exposed as they become connected to other systems or the Internet. Reason refers to this as the Swiss Cheese model where different holes in the slices align. These issues are discussed further in the Human Factors Knowledge Area (Chapter 4). The key point to note is that we can no longer just consider information loss as a potential consequence of cyber security breaches — but must also consider safety implications. Furthermore, security by design is not always a possibility and, as legacy systems become connected to other networked environments, one must consider the latent (insecure) design conditions that may be manifested and how to mitigate their impact.

1.4.4 The Precautionary Principle

As the participatory data economy leads to a range of innovative products and services, there are also growing concerns about privacy and potential misuse of data as has been highlighted by recent cases of interference in democratic processes. As such the *Precautionary Principle* — reflecting on the potential harmful effect of design choices before technological innovations are put into large-scale deployment — also becomes relevant. Designers must consider the security and privacy implications of their choices from conception, through to modelling, implementation, maintenance, evolution and also decommissioning of large-scale connected systems and infrastructures on which society increasingly relies. *Function creep* as the system evolves over its lifetime and its impact on the society-at-large must also be considered [12].

1.5 CROSSCUTTING THEMES

A number of topics and themes recur across various KAs — implicitly or explicitly — and provide a context or unification of ideas across those KAs which cuts across the structure chosen for the CyBOK. In a different decomposition of the CyBOK they might have been KAs in their own right. These are an important part of the body of knowledge, and so we document here the most substantial of them.

1.5.1 Security Economics

Economics of information security is a synthesis between computer and social science. It combines microeconomic theory, and to a lesser extent game theory, with information security to gain an in-depth understanding of the trade-offs and misaligned incentives in the design and deployment of technical computer security policies and mechanisms [13, 14]. For example, Van Eeten and Bauer studied the incentives of legitimate market players — such as Internet Service Providers (ISPs) and software vendors — when confronted with malware[1] and how the actions driven by such incentives lead to optimal or sub-optimal security for the wider interconnected system. Attacker economics is gaining importance as well (for example, [15, 16, 17]). Attacker economics exposes cost-benefit analyses of attackers to exploit vulnerabilities in the security of the victim target, to subsequently formulate protective countermeasures for law-abiding entities [18]. Lastly, there is the economics of deviant security [19]. This subdiscipline of attacker economics focuses on understanding how cyber criminals apply, i.e., practice, security to defend their systems and operations against disruption from law enforcement (e.g., resilience mechanisms built into botnets [20] or anti-forensics techniques [21]).

Security economics is, therefore, of high relevance across the various attacks and countermeasures discussed within the different KAs within CyBOK. It also plays a key role in understanding the cost of security to legitimate users of the system and to the cybercriminals — the strength of such a socio-technical approach is its acknowledgement that security is very much a human problem, and the cost versus benefits trade-offs are key to increasing our understanding of the decisions of defenders and attackers to respectively secure their systems or optimise attacks [13].

1.5.2 Security Architecture and Lifecycle

The word ‘architecture’ is used at all levels of detail within a system; here we are concerned with the high-level design of a system from a security perspective, in particular how the primary security controls are motivated and positioned within the system. This, in turn, is bound up with an understanding of the *systems* lifecycle, from conception to decommissioning. Within this, the secure *software* lifecycle is crucial (the subject of the Secure Software Lifecycle Knowledge Area).

The fundamental design decision is how a system is compartmentalised — how users, data, and services are organised to ensure that the highest risk potential interactions are protected by the simplest and most self-contained security mechanisms (see Section 1.4). For example, a network may be divided into front-office/back-office compartments by a network router or firewall that permits no inward connections from the front to the back. Such a mechanism is

[1] **http://www.oecd.org/internet/ieconomy/40722462.pdf**

simpler and more robust than one that uses access controls to separate the two functions in a shared network.

The first step is to review the proposed use of the system. The business processes to be supported should identify the interactions between the users, data or services in the system. Potential high risk interactions between users (see the Adversarial Behaviours Knowledge Area (Chapter 7) and data should then be identified with an outline risk assessment (see the Risk Management & Governance Knowledge Area (Chapter 2)) which will also need to take account of external requirements such as compliance (see the Law & Regulation Knowledge Area (Chapter 3)) and contractual obligations. If users with a legitimate need to access specific data items also pose a high risk to those items, or if any user has unconstrained authority to effect an undesired security outcome, the business process itself must be revised. Often such cases require a 'two person' rule, for example, counter-authorisation for payments.

The next step is to group users and data into broad categories using role-access requirements, together with formal data classification and user clearance. Such categories are potential system compartments, for example, Internet users and public data, or engineers and design data. Compartments should ensure that the highest risk user-data interactions cross compartment boundaries, and that common user-data interactions do not. Such compartments are usually enforced with network partitioning controls (see the Network Security Knowledge Area (Chapter 19)). Detailed design is then required within compartments, with the first steps being to focus on concrete user roles, data design and access controls (see the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)), with more detailed risk assessments being conducted as the design matures.

Systems benefit from a uniform approach to security infrastructure, for example, the management of keys and network protocols (see the Network Security Knowledge Area (Chapter 19)), resource management and coordination (see the Distributed Systems Security Knowledge Area (Chapter 12)), roles (see the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)), user access (see the Human Factors Knowledge Area (Chapter 4)), and intrusion detection (see the Security Operations & Incident Management Knowledge Area (Chapter 8)). CyBOK provides important foundation knowledge in these areas, but neither this nor risk assessment are sufficient to motivate the detailed implementation of infrastructure; they need to be complemented by current good practice. In some industries best practice is mandated (e.g., the Payment Card Industries). In other cases it may be available from open sources (e.g., OWASP[2]) or as a result of corporate benchmarking.

Orthogonal to these concerns are a number of topics which relate to the context of the system development and operation. It is increasingly clear that a code of conduct, as prescribed by many professional bodies, offers a valuable framework for system designers and those who explore weaknesses and vulnerabilities within such systems. Initiatives around responsible research and innovation are gaining ground. The discovery of vulnerabilities necessitates a disclosure policy — and the parameters of responsible disclosure have prompted much debate, together with the role of this in a security equities process.

These broad consideration of architecture and lifecycle have been captured within the notions of 'security by design', and 'secure by default'[3]. The former term is often applied to detailed practices in software engineering, such as input checking, to avoid buffer overflows and the like (see the Secure Software Lifecycle Knowledge Area (Chapter 17)). More generally,

[2] **https://www.owasp.org**

[3] A related notion is 'privacy by design' (see the Privacy & Online Rights Knowledge Area (Chapter 5)).

consideration of security throughout the lifecycle, including in the default configuration 'out of the box' (although not much software is delivered in boxes these days), demonstrably leads to less insecurity in deployed systems.

ACKNOWLEDGEMENTS

The authors thank Erik van de Sandt for permission to use text from his PhD thesis [19] in the section on Security Economics.

I Human, Organisational & Regulatory Aspects

Chapter 2
Risk Management and Governance

Pete Burnap | Cardiff University

2.1 INTRODUCTION

This Knowledge Area will explain the fundamental principles of cyber risk assessment and management and their role in risk governance, expanding on these to cover the knowledge required to gain a working understanding of the topic and its sub-areas. We begin by discussing the relationship between everyday risk and why this is important in today's interconnected digital world. We explain why, as humans, we need effective risk assessment and management principles to support the capture and communication of factors that may impact our values. We then move on to describe different perspectives on cyber risk assessment – from individual assets, to whole-system goals and objectives. We unpick some of the major risk assessment methods and highlight their main uses and limitations, as well as providing pointers to more detailed information.

Security metrics are an ongoing topic of debate in the risk assessment and management domain: which system features to measure for risk, how to measure risk, and why measure risk at all? These questions are framed in the context of existing literature on this topic. This links into risk governance, which explains why effective governance is important to uphold cyber security and some of the social and cultural factors that are essential to consider when developing governance frameworks. Almost all systems still include a human element of control, which must be considered from the outset. Finally, even with well defined and executed risk assessment and management plans, it is still possible that a risk will turn into reality. In such cases, incident response is required. We discuss the importance of incident response and its link to the risk governance process.

2.2 WHAT IS RISK?

[22, 23, 24]

Risk is at the heart of everyday life. From a child making a decision to jump out of a tree to an investment decision by the CEO of a multi-billion dollar company, we all make decisions that potentially impact us as individuals, and impact our broader social networks and surroundings. Defining risk is, therefore, a highly philosophical and contentious matter. Seminal works by Slovic [23] and Renn [22] on risk perception capture the broad-reaching issues surrounding this debate, and provide a working definition that abstracts the question to allow us to engage with the topic of risk on a socio-technical level. Renn's working definition of risk is *the possibility that human actions or events lead to consequences that have an impact on what humans value*. This fundamentally grounds risk in human value, which applies to both the child and CEO examples. It also applies to cyber security contexts in a world where people and technology are intrinsically linked. The failure of one to support the success of the other can lead to social, economic and technical disaster. The working definition of *impact on values* raises a further question of how to define the value and capture indicators that can be used to measure and manage the risk. Renn defines three basic abstract elements required for this: outcomes that have an impact on what humans value, possibility of occurrence (uncertainty), and a formula to combine both elements. These elements are at the core of most *risk assessment methods*. Such methods aim to provide a structured approach to capturing the entities of value and the likelihood of unwanted outcomes affecting the entities, while also bearing in mind that even something with very low probability may be realised and may have significant impact on a value. We, therefore, use Renn's working definition of risk for discussion in this KA in the context of cyber risk.

A key challenge with risk assessment and management is making assumptions explicit and finding the balance between subjective risk perceptions and objective evidence. *Risk assessment* is, therefore, a process of collating observations and perceptions of the world that can be justified by logical reasoning or comparisons with actual outcomes [24]. *Risk management*, on the other hand, is the process of developing and evaluating options to address the risks in a manner that is agreeable to people whose values may be impacted, bearing in mind agreement on how to address risk may involve a spectrum of (in)tolerance – from acceptance to rejection. *Risk Governance* is an overarching set of ongoing processes and principles that aims to ensure an awareness and education of the risks faced when certain actions occur, and to instil a sense of responsibility and accountability to all involved in managing it. It underpins collective decision-making and encompasses both risk assessment and management, including consideration of the legal, social, organisational and economic contexts in which risk is evaluated [24]. This Knowledge Area explores all these topics and provides insights into risk assessment, management and governance from a cyber security science perspective that is accessible to individuals, SMEs and large organisations alike.

2.3 WHY IS RISK ASSESSMENT AND MANAGEMENT IMPORTANT?

[23, 24, 25, 26]

Risk assessment involves three core components [24]: (i) identification and, if possible, estimation of hazard; (ii) assessment of exposure and/or vulnerability; and (iii) estimation of risk, combining the likelihood and severity. Identification relates to the establishment of events and subsequent outcomes, while estimation is related to the relative strength of the outcome. Exposure relates to the aspects of a system open to threat actors (e.g., people, devices, databases), while vulnerability relates to the attributes of these aspects that could be targeted (e.g., susceptibility to deception, hardware flaws, software exploits). Risk estimation can be quantitative (e.g., probabilistic) or qualitative (e.g., scenario-based) and captures the expected impact of outcomes. The fundamental concept of risk assessment is to use analytic and structured processes to capture information, perceptions and evidence relating what is at stake, the potential for desirable and undesirable events, and a measure of the likely outcomes and impact. Without any of this information we have no basis from which to understand our exposure to threats nor devise a plan to manage them. An often overlooked part of the risk assessment process is *concern assessment*. This stems from public risk perception literature but is also important for cyber security risk assessment as we will discuss later in the document. In addition to the more evidential, scientific aspects of risk, concern assessment includes wider stakeholder perceptions of: hazards, repercussions of risk effects, fear and dread, personal or institutional control over risk management and trust in the risk managers.

The risk management process involves reviewing the information collected as part of the risk (and concern) assessments. This information forms the basis of decisions leading to three outcomes for each perceived risk [24]:

- *Intolerable:* the aspect of the system at risk needs to be abandoned or replaced, or if not possible, vulnerabilities need to be reduced and exposure limited.
- *Tolerable:* risks have been reduced with reasonable and appropriate methods to a level as low as reasonably possible (ALARP) [27] or as low as reasonably allowable (ALARA).

A range of choices may include mitigating, sharing, or transferring risk [28], selection of which will depend on the risk managers' (and more general company) appetite for taking risks.

- *Acceptable:* risk reduction is not necessary and can proceed without intervention. Furthermore, risk can also be used to pursue opportunities (also known as 'upside risk'), thus the outcome may be to accept and embrace the risk rather than reduce it. Hillson discusses this perspective in further detail [25].

Deciding which to select will be dependent on a number of factors, for example (as suggested in ISO 31000:2018 [29]), tangible and intangible uncertainty, consequences of risk realisation (good or bad), appetite for risk, organisational capacity to handle risk etc.

Beyond this decision framework Renn defines four types of risk that require different risk management plans [24]. These include:

- *Routine risks:* these follow a fairly normal decision-making process for management. Statistics and relevant data are provided, desirable outcomes and limits of acceptability are defined, and risk reduction measures are implemented and enforced. Renn gives examples of car accidents and safety devices.
- *Complex risks:* where risks are less clear cut, there may be a need to include a broader set of evidence and consider a comparative approach such as cost-benefit analysis or cost-effectiveness. Scientific dissent such as drug treatment effects or climate change are examples of this.
- *Uncertain risks:* where a lack of predictability exists, factors such as reversibility, persistence and ubiquity become useful considerations. A precautionary approach should be taken with a continual and managed approach to system development whereby negative side effects can be contained and rolled-back. Resilience to uncertain outcomes is key here.
- *Ambiguous risks:* where broader stakeholders, such as operational staff or civil society, interpret risk differently (e.g., different viewpoints exist or lack of agreement on management controls), risk management needs to address the causes for the differing views. Renn uses the example of genetically modified foods where well-being concerns conflict with sustainability options. In this instance, risk management must enable participatory decision-making, with discursive measures aiming to reduce the ambiguity to a number of manageable options that can be further assessed and evaluated.

Management options, therefore, include a risk-based management approach (risk-benefit analysis or comparative options), a resilience-based approach (where it is accepted that risk will likely remain but needs to be contained, e.g. using ALARA/ALARP principles), or a discourse-based approach (including risk communication and conflict resolution to deal with ambiguities). Without effective consideration of the acceptability of risk and an appropriate risk reduction plan, it is likely that the response to adverse outcomes will be disorganised, ineffective, and likely lead to further spreading of undesirable outcomes.

Effective risk management through structured assessment methods is particularly important because, although our working definition of risk is grounded in consequences of interest to people, we (as a society) are not very good at assessing this risk. Slovic's article on risk perception highlights that perceptions related to *dread risk* (e.g., nuclear accidents) are ranked highest risk by lay people, but much lower by domain experts who understand the evidence relating to safety limitations and controls for such systems. Expert risk ranking tends to follow

expected or recorded undesirable outcomes such as deaths, while lay people are influenced more by their intuitive judgment (a nuclear accident could impact my whole family). There is, therefore, a mismatch between perceived vs. actual risk. As people we tend to exaggerate *dread-related* but rare risks (e.g., nuclear incidents and terrorist attacks) but downplay common ones (e.g., street crime and accidents in the home) – even though the latter kill far more people.

This is also why concern assessment is important in the risk management process alongside risk assessment. Schneier's book *Beyond Fear* [26] notes that we have a natural sense of safety in our own environment and a heightened sense of risk outside of this. For instance, we feel safe walking down a street next to our house but on edge when arriving in a new city. As a society, we rarely study statistics when making decisions; they are based on perceptions of exposure to threat, our perceived control over threats, and their possible impact. Risk assessment helps us capture quantitative and qualitative aspects of the world that enable us to put a realistic estimate of how certain we can be that adverse events will come to pass, and how they will impact on what we value most. This applies to us personally as individuals, and as groups of people with a common aim – saving the planet, running a business, or educating children. We need to capture our goals, understand what could lead to the failure to achieve them, and put processes in place to align realistic measures to reduce harms inflicted upon our objectives.

When done well, risk assessment and management enables decision makers, who are responsible, to ensure that the system operates to achieve the desired goals as defined by its stakeholders. It can also ensure the system is not manipulated (intentionally or otherwise) to produce undesired outcomes, as well as having processes in place that minimise the impact should undesirable outcomes occur. Risk assessment and management is also about presenting information in a transparent, understandable and easily interpreted way to different audiences, so that accountable stakeholders are aware of the risks, how they are being managed, who is responsible for managing them, and are in agreement on what is the acceptable limit of risk exposure. This is absolutely crucial to successfully managing risk because, if the risks are not presented clearly to decision makers (be they technical, social, economic or otherwise), the impact of not managing them will be overlooked, and the system will remain exposed. Likewise, if the purpose of risk management is not made clear to the people at the operational level, alongside their own responsibilities and accountability for risk impacts, they will not buy in to the risk management plan and the system will remain exposed. More broadly, if wider stakeholder concerns (e.g., civil society) are not heard or there is lack of confidence in the risk management plan, there could be widespread rejection of the planned system being proposed.

As important as it is to convey risks clearly to stakeholders, it is equally as important to stress that risks cannot always be removed. There is likely to be some residual risk to the things we value, so discussions must be held between decision makers and those who are involved with the operations of a system. Ultimately, decision makers, who will be held to account for failure to manage risk, will determine the level of risk tolerance – whether risk is accepted, avoided, mitigated, shared, or transferred. However, it is possible that wider stakeholders, such as those involved with system operations, may have differing views on how to manage risk, given they are likely to have different values they are trying to protect. For some, saving money will be key. For others, reputation is the main focus. For people working within the system it may be speed of process or ease of carrying out daily tasks. The purpose of risk assessment and management is to communicate these values and ensure decisions are taken to minimise the risks to an agreed set of values by managing them appropriately,

while maximising 'buy in' to the risk management process. In the broader health and safety risk context, this concept relates to the notion of ALARP (as low as reasonably practicable) [27] – being able to demonstrate that significant efforts and computation have been made to calculate the balance between risk acceptance and mitigation, in the favour of security and safety. Again it is important to highlight here that concern assessment is an important part of risk assessment to ensure the risk assessment policy (the agreed approach to risk assessment) is informed by those responsible for, and impacted by risk, and those who are required to act in a way that upholds the management plan day-to-day. Crucially, it must be recognised that the impact of single events can often extend beyond direct harms and spread far wider into supply chains. As Slovic puts it, the results of an event act like ripples from a stone dropped into a pond, first directly within the company or system in which it occurred, and then into sub-systems and interdependent companies and components [23].

One of the major drivers for risk assessment and management is to demonstrate compliance. This can be a result of the need to have audited compliance approval from international standards bodies in order to gain commercial contracts; to comply with legal or regulatory demands (e.g., in Europe the Network and Information Systems (NIS) directive [30] mandates that operators of essential services (such as critical national infrastructure) follow a set of 14 goal-oriented principles [31]); or to improve the marketability of a company through perceived improvements in public trust if certification is obtained. This can sometimes lead to 'tick-box' risk assessment whereby the outcome is less focused on managing the risk, and more about achieving compliance. This can result in a false sense of security and leave the organisation exposed to risks. This bring us back to Renn's working definition of risk. These examples focus on managing risk of failing compliance with various policy positions, and as a result, they may neglect the broader focus on impact on values held by wider organisational, societal or economic stakeholders. The context and scope of risk management must take this broader outcomes-view in order to be a useful and valuable exercise that improves preparedness and resilience to adverse outcomes.

Based on these factors, risk assessment and management is most certainly a process not a product. It is something that, when done well, has the potential to significantly improve the resilience of a system. When done badly (or not at all) it can lead to confusion, reputational damage, and serious impact on system functionality. It is a process that is sometimes perceived to be unimportant before one needs it, but critical for business continuity in a time of crisis. Throughout the process of risk assessment we must remain aware that risk perception varies significantly based on a variety of factors, and that despite objective evidence, it will not change. To use an example from [23], providing evidence that the annual risk from living next to a nuclear power plant is equivalent to the risk of riding an extra 3 miles in an automobile, does not necessarily reduce the perception of risk given the differences surrounding the general perception of the different scenarios. Intuitively, communication and a respect for qualitative and quantitative measures of risk assessment are core to its practice. Both measures exhibit ambiguity (e.g., [32]) and often we lack quality data on risk so evidence only goes so far. There will always be a need for subjective human judgment to determine relevance and management plans [33], which in itself comes with its own limitations such as lack of expert knowledge and cognitive bias [34].

2.4 WHAT IS CYBER RISK ASSESSMENT AND MANAGEMENT?

[35]

The introductory sections have made the case for risk assessment and management more generally, but the main focus of this document is to frame risk assessment and management in a cyber security context. Digital technology is becoming evermore pervasive and underpins almost every facet of our daily lives. With the growth of the Internet of Things, connected devices are expected to reach levels of more than 50 billion by 2022 [36]. Further, human decision-based tasks such as driving and decision-making are being replaced by automated technologies, and the digital infrastructures that we are increasingly reliant upon can be disrupted indiscriminately as a result of, for example, ransomware [37]. Cyber security risk assessment and management is, therefore, a fundamental special case that everyone living and working within the digital domain should understand and be a participant in it.

There are a number of global standards that aim to formalise and provide a common framework for cyber risk assessment and management, and, in this section, we will study some of them. We will begin with high level definitions of some of the foremost positions on risk. The United Kingdom was ranked first in the 2018 Global Cybersecurity Index (GCI) [38], a scientifically grounded review of the cyber security commitment and situation at a global country-by-country level. The review covers five pillars: (i) legal, (ii) technical, (iii) organisational, (iv) capacity building, and (v) cooperation – and then aggregates them into an overall score. As the lead nation in the GCI, the technical authority for cyber security, the UK National Cyber Security Centre (NCSC) has published guidance on risk management [35]. Importantly, the NCSC is clear that there is no one-size-fits-all for risk assessment and management. Indeed conducting risk assessment and management as a tick-box exercise produces a false sense of security, which potentially increases the Vulnerability of the people impacted by risk because they are not properly prepared. Cyber security is such a rapidly evolving domain that we must accept that we cannot be fully cyber secure. However, we can increase our preparedness. The Potomac Institute for Policy Studies provides a framework for studying cyber readiness along with a country-specific profile for a range of nations (inc. USA, India, South Africa, France, UK) and an associated cyber readiness index [39].

2.5 RISK GOVERNANCE

[40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]

2.5.1 What is risk governance and why is it essential?

Risk assessment and developing mitigation principles to manage risk is only likely to be effective where a coordinated and well communicated governance policy is put in place within the system being managed. Millstone et al. [40] proposed three governance models:

- *Technocratic:* where policy is directly informed by science and evidence from domain expertise.
- *Decisionistic:* where risk evaluation and policy are developed using inputs beyond science alone. For instance, incorporating social and economic drivers.
- *Transparent (inclusive):* where context for risk assessment is considered from the outset with input from science, politics, economics and civil society. This develops a model of 'pre-assessment' – that includes the views of wider stakeholders – that shapes risk assessment and subsequent management policy.

None are correct or incorrect. There is a fine balance between the knowledge and findings of scientific experts, and perceptions of the lay public. While the technocratic approach may seem logical to some risk owners who work on the basis of reasoning using evidence, it is absolutely crucial for effective risk governance to include the wider stakeholder view. Rohrmann and Renn's work on risk perception highlights some key reasons for this [41]. They identify four elements that influence the perception of risk:

- intuitive judgment associated with probabilities and damages;
- contextual factors surrounding the perceived characteristics of the risk (e.g., familiarity) and the risk situation (e.g., personal control);
- semantic associations linked to the risk source, people associated with the risk, and circumstances of the risk-taking situation;
- trust and credibility of the actors involved in the risk debate.

These factors are not particularly scientific, structured or evidence-based but, as noted by Fischoff et al. [42], such forms of defining probabilities are countered by the strength of belief people have about the likelihood of an undesirable event impacting their own values. Ultimately, from a governance perspective, the more inclusive and transparent the policy development, the more likely the support and buy-in from the wider stakeholder group – including lay people as well as operational staff – for the risk management policies and principles.

There are several elements that are key to successful risk governance. Like much of the risk assessment process, there is no one-size solution for all endeavours. However, a major principle is ensuring that the governance activity (see below) is tightly coupled with everyday activity and decision-making. Cyber risk is as important as health and safety, financial processes, and human resources. These activities are now well established in decision-making. For instance, when hiring staff, the HR process is at the forefront of the recruiter's activity. When travelling overseas, employees will always consult the financial constraints and processes for travel. Cyber security should be thought of in the same way – a clear set of processes that reduce

the risk of harm to individuals and the business. Everyone involved in the daily running of the system in question must understand that, for security to be effective, it must be part of everyday operational culture. The cyber risk governance approach is key to this cultural adoption.

2.5.2 The human factor and risk communication

Sasse and Flechais [43] identified human factors that can impact security governance, including people: having problems using security tools correctly; not understanding the importance of data, software, and systems for their organisation; not believing that the assets are at risk (i.e., that they would be attacked); or not understanding that their behaviour puts the system at risk. This highlights that *risk cannot be mitigated with technology alone*, and that *concern assessment* is important. If risk perception is such that there is a widely held view that people do not believe their assets will be attacked (as noted by [43]), despite statistics showing cyber security breaches are on the rise year-on-year, then there is likely to be a problem with the cyber security culture in the organisation. Educating people within an organisation is vital to ensuring cultural adoption of the principles defined in the risk management plan and associated security governance policy. People will generally follow the path of least resistance to get a job done, or seek the path of highest reward. As Sasse and Flechais note, people fail to follow the required security behaviour for one of two reasons: (1) they are unable to behave as required (one example being that it is not technically possible to do so; another being that the security procedures and policies available to them are large, difficult to digest, or unclear) , (2) they do not want to behave in the way required (an example of this may be that they find it easier to work around the proposed low-risk but time consuming policy; another being that they disagree with the proposed policy).

Weirich and Sasse studied compliance with password rules as an example of compliance with security policy [44] and found that a lack of compliance was associated with people not believing that they were personally at risk and or that they would be held accountable for failure to follow security rules. There is thus a need to ensure a sense of responsibility and process for accountability, should there be a breach of policy. This must, of course, be mindful of legal and ethical implications, as well as the cultural issues around breaching rules, which is a balancing act. Risk communication, therefore, plays an important role in governance [45] [22] including aspects, such as:

- *Education:* particularly around risk awareness and day-to-day handling of risks, including risk and concern assessment and management;
- *Training and inducement of behaviour change:* taking the awareness provided by education and changing internal practices and processes to adhere to security policy;
- *Creation of confidence:* both around organisational risk management and key individuals – develop trust over time, and maintain this through strong performance and handling of risks.
- *Involvement:* particularly in the risk decision-making process – giving stakeholders an opportunity to take part in risk and concern assessment and partake in conflict resolution.

Finally, leading by example is of paramount importance in the risk communication process. People are likely to be resentful if it appears that senior management are not abiding by

the same risk management rules and principles. Visible senior engagement in an important cultural aspect of risk communication.

2.5.3 Security culture and awareness

Dekker's principles on *Just Culture* [46] aim to balance accountability with learning in the context of security. He proposes the need to change the way in which we think about accountability so that it becomes compatible with learning and improving the security posture of an organisation. It is important that people feel able to report issues and concerns, particularly if they think they may be at fault. Accountability needs to be intrinsically linked to *helping the organisation*, without concern of being stigmatised and penalised. There is often an issue where those responsible for security governance have limited awareness and understanding of what it means to practise it in the operational world. In these cases there needs to be an awareness that there is possibly no clear right or wrong, and that poorly thought-out processes and practices are likely to have been behind the security breach, as opposed to malicious human behaviour. If this is the case, these need to be addressed and the person at fault needs to feel supported by their peers and free of anxiety. One suggestion Dekker makes is to have an independent team to handle security breach reports so people do not have to go through their line manager. If people are aware of the pathways and outcomes following security breaches it will reduce anxiety about what will happen and, therefore, lead to a more open security culture.

Given that security awareness and education is such an important factor in effective governance, Jaquith [47] links security awareness with security metrics through a range of questions that may be considered as useful pointers for improving security culture:

- Are employees acknowledging their security responsibilities as users of information systems? (Metric: % new employees completing security awareness training).
- Are employees receiving training at intervals consistent with company policies? (Metric: % existing employees completing refresher training per policy).
- Do security staff members possess sufficient skills and professional certifications? (Metric: % security staff with professional security certifications).
- Are security staff members acquiring new skills at rates consistent with management objectives? (Metrics: # security skill mastered, average per employee and per security team member, fulfillment rate of target external security training workshops and classroom seminars).
- Are security awareness and training efforts leading to measurable results? (Metrics: By business unit or office, correlation of password strength with the elapsed time since training classes; by business unit or office, correlation of tailgating rate with training latency).

Metrics may be a crude way to capture adherence to security policy, but when linked to questions that are related to the initial risk assessment, they can provide an objective and measurable way to continually monitor and report on the security of a system to the decision makers, as well as those responsible for its governance in an understandable and meaningful way. However, it is worth noting the complexity of metrics here with the use of the term 'acknowledging' in the first bullet point. It does not necessarily mean the person will acknowledge their responsibilities merely by completing awareness training. This reinforces the points

already made about the importance of human factors and security culture, and the following section on enacting security policy.

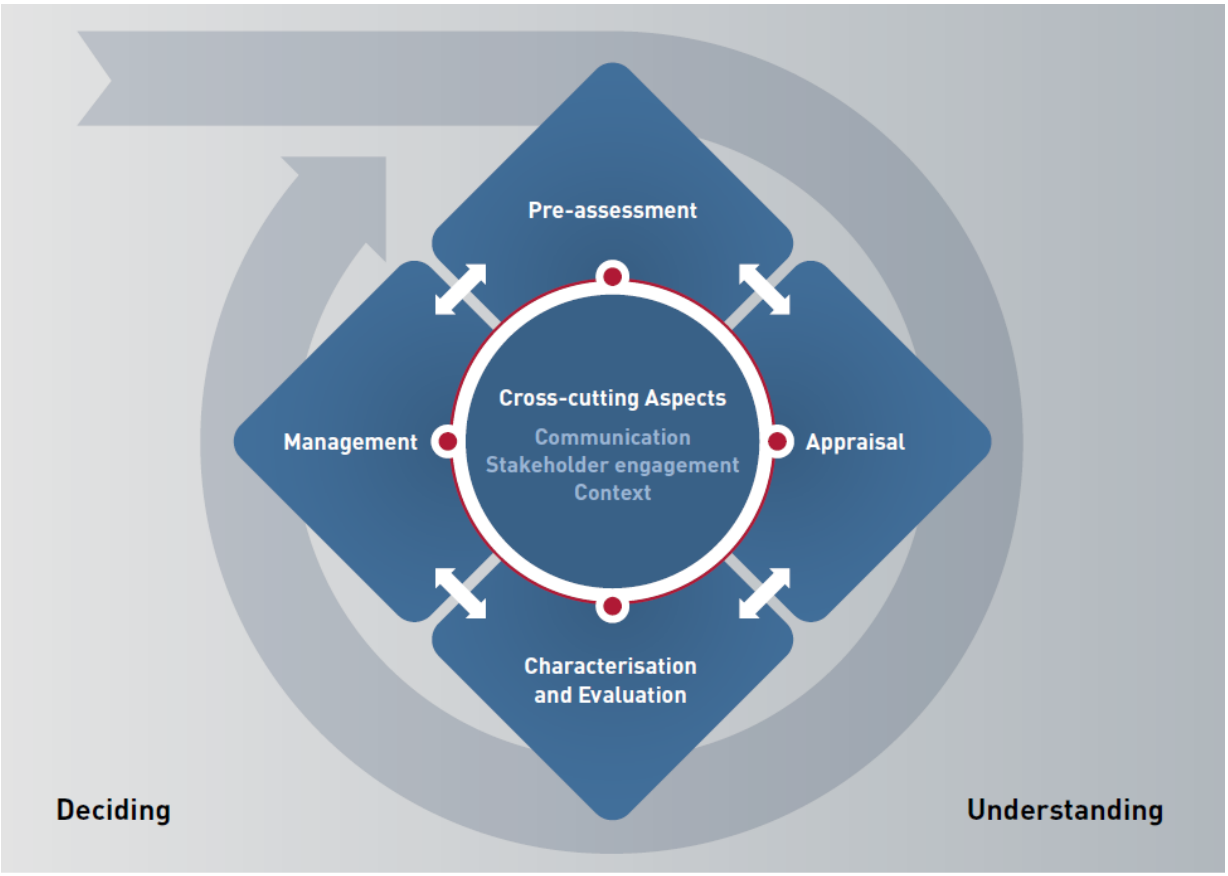


Figure 2.1: Risk Governance Framework from IRGC - taken from [49]

2.5.4 Enacting Security Policy

Overall, effective cyber risk governance will be underpinned by a clear and enactable security policy. This section focuses on the elements of risk assessment and management that are relevant to achieving this. From the initial phase of the risk assessment there should be a clear focus on the purpose and scope of the risk assessment exercise. During this phase, for more complex systems or whole system security, there should be a focus on identifying the objectives and goals of the system. These should be achievable with clear links from objectives to the processes that underpin them. Risks should be articulated as clear statements that capture the interdependencies between the vulnerabilities, threats, likelihoods and outcomes (e.g., causes and effects) that comprise the risk. Risk management decisions will be taken to mitigate threats identified for these processes, and these should be linked to the security policy, which will clearly articulate the required actions and activities taken (and by whom), often along with a clear timeline, to mitigate the risks. This should also include what is expected to happen as a consequence of this risk becoming a reality.

Presentation of risk assessment information in this context is important. Often heat maps and risk matrices are used to visualise the risks. However, research has identified limitations in the concept of combining multiple risk measurements (such as likelihood and impact) into a single matrix and heat map [51]. Attention should, therefore, be paid to the purpose of the

visualisation and the accuracy of the evidence it represents for the goal of developing security policy decisions.

Human factors (see the Human Factors Knowledge Area (Chapter 4)), and security culture are fundamental to the enactment of the security policy. As discussed, people fail to follow the required security behaviour because they are unable to behave as required, or they do not want to behave in the way required [43]. A set of rules dictating how security risk management should operate will almost certainly fail unless the necessary actions are seen as linked to broader organisational governance, and therefore security policy, in the same way HR and finance policy requires. People must be enabled to operate in a secure way and not be the subject of a blame culture when things fail. It is highly likely that there will be security breaches, but the majority of these will not be intentional. Therefore, the security policy must be reflective and reactive to issues, responding to the *Just Culture* agenda and creating a policy of accountability for learning, and using mistakes to refine the security policy and underpinning processes – not blame and penalise people.

Security education should be a formal part of all employees' continual professional development, with reinforced messaging around why cyber security is important to the organisation, and the employee's role and duties within this. Principles of risk communication are an important aspect of the human factor in security education. We have discussed the need for credible and trustworthy narratives and stakeholder engagement in the risk management process. There are additional principles to consider such as early and frequent communication, tailoring the message to the audience, pretesting the message and considering existing risk perceptions that should be part of the planning around security education. Extensive discussion of such risk communication principles that are particularly relevant for messaging regarding risk can be found in [50].

Part of the final risk assessment and management outcomes should be a list of accepted risks with associated owners who have oversight for the organisational goals and assets underpinning the processes at risk. These individuals should be tightly coupled with review activity and should be clearly identifiable as responsible and accountable for risk management.

Figure 2.1 summarises the core elements of the risk governance process as discussed so far. This model from the International Risk Governance Council (IRGC) [49], which is heavily inspired by Renn's work [24], highlights that risk communication sits at the heart of the governance process and draws on problem framing, risk and concern assessment, risk evaluation, and risk management. The governance process is iterative, always seeking awareness of new problems and evolving threats, and continually reflecting on best practice to manage new risks.

2.6 RISK ASSESSMENT AND MANAGEMENT PRINCIPLES

[30, 35, 47, 49, 52, 53, 54, 55, 56, 57, 58]

2.6.1 Component vs. Systems Perspectives

The UK NCSC guidance [35] breaks down risk management into *Component-driven risk management*, which focuses on technical components, and the threats and vulnerabilities they face (also known as bottom up); and *System-driven risk management*, which analyses systems as a whole (also known as top down). A major difference between the two is that component-driven approaches tend to focus on the specific risk to an individual component (e.g., hardware, software, data, staff), while system-driven approaches focus more on the goals of an entire system – requiring the definition of a higher level purpose and subsequent understanding of sub-systems and how various parts interact.

Rasmussen's work [52] enables us to consider a hierarchy of abstraction and show how systems-driven and component-driven risk assessment techniques are complementary. As illustrated in Figure 2.2, the goals and purposes of the system can be considered at the higher level. Notably, this includes a focus on dependencies between sub-goals and also what the system must not do (pre-defined failure states). These are important to design into the system and, if omitted, lead to having to retrofit cyber security into a system that has already been deployed. The lower levels then consider capabilities and functionality needed to achieve the overarching goals. At this level component-driven risk assessments of real-world artefacts (e.g., hardware, software, data, staff) consider how these may be impacted by adverse actions or events.

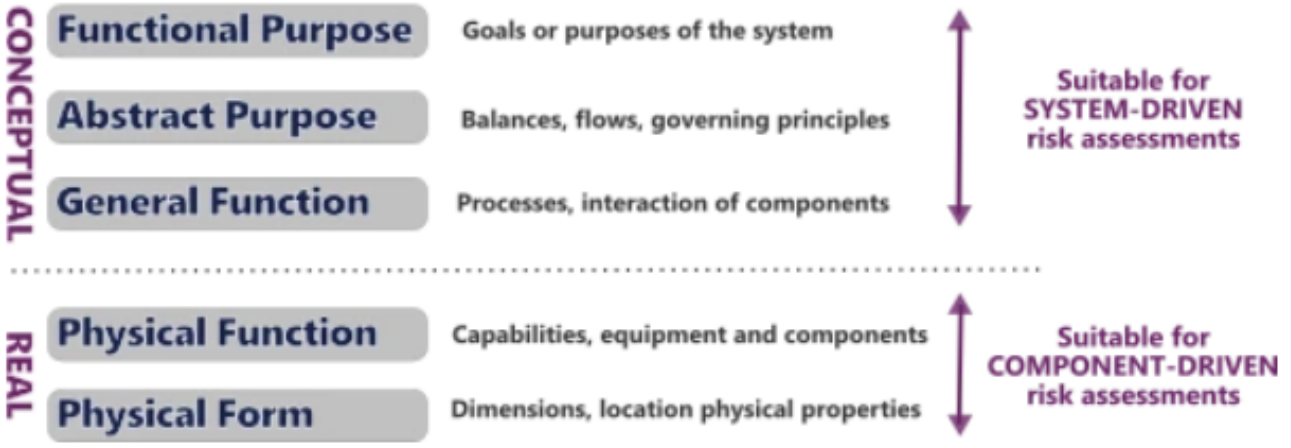


Figure 2.2: Jens Rasmussen's Hierarchy

System-driven approaches can help to better understand the complexity between sub-components and their components. These may include people, technology, and organisational processes for which the interactions and dependencies are non-trivial. Taking such an approach (which may perhaps prove more resource intensive than component based approaches, due to identification of inter-dependencies) is only necessary where complexity actually exists. If interactions and dependencies are clear and the system is less complex (e.g., a simple office-based IT infrastructure) then a component-driven approach may be more appropriate.

The NCSC guidance provides a summary table (reproduced here as Figure 2.3) that is helpful in

guiding the selection of component-driven or system-driven methods based on the kind of risk management problem being addressed. The major differentiator is that the component view is individual asset-based, where complexity is well-understood and expected functionality is clear. The system view supports an analysis of risk in situations of greater complexity, before physical function is agreed and implemented, and to support discussions by key stakeholders on what the system should and should not do. These discussions are crucial in finding the balance between component-level and system-level failure and how best to manage the risk. Component-risk is likely to be more important to operational employees who need the component to be functioning in order for their part of a bigger system to perform (e.g., staff, data, devices). Systems-level risk is likely to be more important to higher-level managers who have a vested interest in the strategic direction of the system. For them a component further down the value/supply chain may not be perceived to be important, while for the operational employee it's the number one risk. The challenge is to work with both perspectives to develop a representation of risk and an associated risk management policy enacted by all parties.

	Good For
Component-driven methods	• Analysing the risks faced by individual technical components. • Deconstructing less complex systems, with well-understood connections between component parts. • Working at levels of abstraction where a system's physical function has already been agreed amongst stakeholders.
System-driven methods	• Exploring security breaches which emerge out of the complex interaction of many parts of your system. • Establishing system security requirements before you have decided on the system's exact physical design. • Bringing together multiple stakeholders' views of what a system should and should not do (e.g., safety, security, legal views). • Analysing security breaches which cannot be tracked back to a single point of failure.

Figure 2.3: Guidelines for mapping risk management problem types to component or system driven methods

2.6.2 Elements of Risk

While it is useful to avoid creating a universal definition of risk, to support inclusivity of different views and perspectives, it is important to have agreed definitions of the concepts that underpin risk assessment and management. This ensures a common language throughout the process and avoids talking at cross purposes. There are four concepts that are core to a risk assessment in most models – vulnerability, threat, likelihood and impact.

A *Vulnerability* is something open to attack or misuse that could lead to an undesirable outcome. If the vulnerability were to be exploited it could lead to an impact on a process or system. Vulnerabilities can be diverse and include technology (e.g., a software interface

being vulnerable to invalid input), people (e.g., a business is vulnerable to a lack of human resources), legal (e.g., databases being vulnerable and linked to large legal fines if data is mishandled and exposed) etc. This is a non-exhaustive list, but highlights that vulnerabilities are socio-technical.

A *Threat* is an individual, event, or action that has the capability to exploit a vulnerability. Threats are also socio-technical and could include hackers, disgruntled or poorly trained employees, poorly designed software, a poorly articulated or understood operational process etc. To give a concrete example that differentiates vulnerabilities from threats – a software interface has a vulnerability in that malicious input could cause the software to behave in an undesirable manner (e.g., delete tables from a database on the system), while the threat is an action or event that exploits the vulnerability (e.g., the hacker who introduces the malicious input to the system).

Likelihood represents a measure capturing the degree of possibility that a threat will exploit a vulnerability, and therefore produce an undesirable outcome affecting the values at the core of the system. This can be a qualitative indicator (e.g., low, medium, high), or a quantitative value (e.g., a scale of 1-10 or a percentage).

Impact is the result of a threat exploiting a vulnerability, which has a negative effect on the success of the objectives for which we are assessing the risk. From a systems view this could be the failure to manufacture a new product on time, while from a component view it could be the failure of a specific manufacturing production component.

2.6.3 Risk assessment and management methods

The purpose of capturing these four elements of risk is for use in dialogue that aims to represent how best to determine the exposure of a system to cyber risk, and how to manage it. There are a range of methods, some of which have been established as international standards and guidelines, that provide a structured means to transform vulnerability, threat, likelihood and impact into a ranked list in order to be able to prioritise and treat them. While each method has its own particular approach to risk assessment and management, there are some features common to a number of the most widely used methods that are useful for framing risk assessment and management activities, which can be mapped back to Renn's seminal work on risk governance [24] as discussed in earlier sections. The International Risk Governance Council (IRGC) capture these in its risk governance framework (developed by an expert group chaired by Renn), summarised in Figure 2.1, which includes four core areas and crosscutting components. *Pre-assessment* includes the framing of risk, identification of relevant actors and stakeholders, and captures perspectives on risk. *Appraisal* includes the assessment of causes and consequences of risk (including risk concern), developing a knowledge base of risks and mitigation options (e.g., preventing, sharing etc). *Characterisation* involves a decision process, making a judgment about the significance and tolerance of the risks. *Appraisal* and *Characterisation* forms the basis of the implementation of Renn's three core components of risk assessment [24]. *Management* processes include deciding on the risk management plan and how to implement it, including risk tolerance (accepting, avoiding, mitigating, sharing, transferring). Cutting across all four is *communication, engagement and context setting* through open and inclusive dialogue.

The US Government NIST [53] guidelines capture the vulnerability, threats, likelihood and impact elements inside the *prepare (pre-assessment), conduct (appraisal and characterise), communicate (cross-cutting), maintain (management)* cycle (see Figure 2.4). A step-by-step

detailed guide can be found in the full document, but we summarise the actions here.

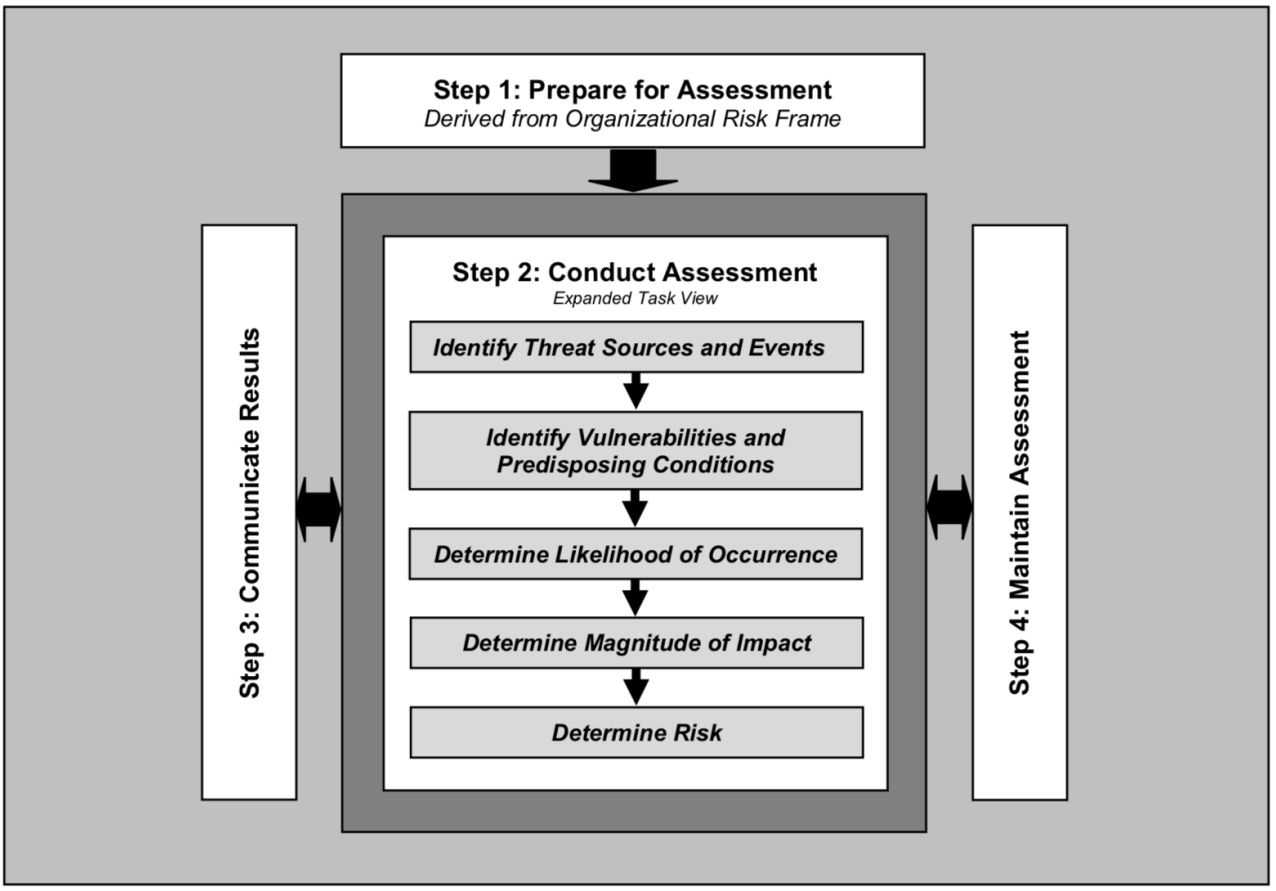


Figure 2.4: NIST SP-800-30 Risk Assessment Process

Prepare involves identifying the *purpose* (e.g., establishing a baseline of risk or identifying vulnerabilities, threats, likelihood and impact) and *scope* (e.g., what parts of a system/organisation are to be included in the risk assessment?; what decisions are the results informing?). It also involves defining *assumptions* and *constraints* on elements such as resources required and predisposing conditions that need to inform the risk assessment. The *assessment approach* and tolerances for risk are also defined at this stage along with identifying *sources of information* relating to threats, vulnerabilities and impact.

Conduct is the phase where threats, vulnerabilities, likelihood and impact are identified. There are a range of ways that this can be conducted, and this will vary depending on the nature of the system being risk assessed and the results of the *prepare* stage. NIST has a very specific set of tasks to be performed. These may not be relevant to all systems, but there are some useful tasks that generalise across multiple system perspectives, including identifying: threat sources and adversary capability, intent and targets; threat events and relevance to the system in question; vulnerabilities and predisposing conditions; likelihood that the threats identified will exploit the vulnerabilities; and the impacts and affected assets. Note that the ordering of actions in the NIST approach puts threat identification before vulnerabilities, which presupposes that all threats can be identified and mapped to vulnerabilities. It is worth highlighting that risk assessment must also be effective in situations where threats are less obvious or yet to be mainstream (e.g., IoT Botnets) and, therefore, some organisations that are particularly ingrained in digital adoption may also wish to consider conducting a vulnerability assessment independently or prior to the identification of likely threats to avoid making

assumptions on what or who the threats actors may be.

Communicate is one of the most important phases, and one often overlooked. Conducting the risk assessment gives one the data to be able to inform actions that will improve the security of the system. However, it is crucial this is communicated using an appropriate method. Executive boards will expect and need information to be presented in a different way to operational team members, and general organisational staff will need educating and guiding in an entirely different way. The results and evidence of the risk assessment must be communicated in a manner accessible to each stakeholder and in a way that is likely to engage them in risk management planning and execution.

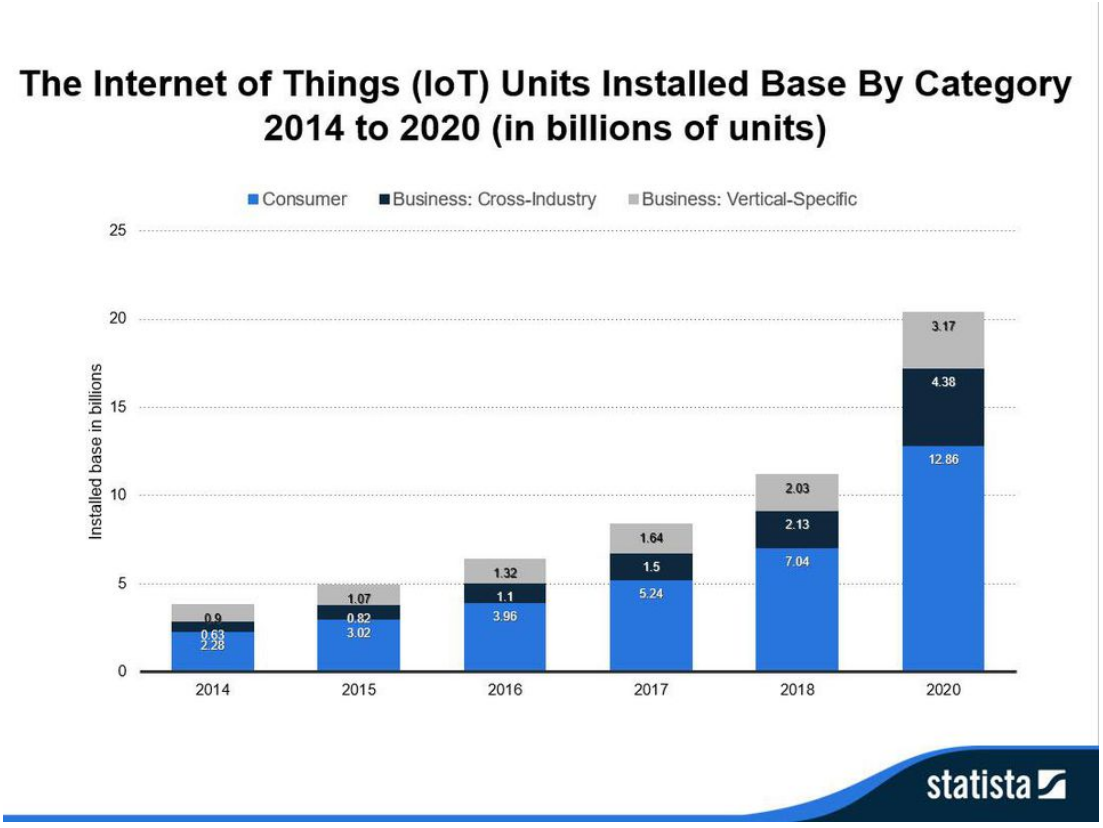


Figure 2.5: IoT Devices Use Figures: Source: [36]

Maintain is an ongoing phase that is essential to continually update the risk assessment in the light of changes to the system environment and configuration. Security postures change regularly in digital environments. For instance, Figure 2.5 shows the volume of IoT units installed from 2014 to 2020 with a rapid increase in adoption of 2.63 million across the business sector between 2014 and 2018. By 2020 this is projected to grow by a further 3.39 million. This kind of rapid integration of devices into corporate IT systems is likely to change the exposure to risk and, therefore, the scope would need to be refined, new risk assessments carried out, and action taken and communicated to all stakeholders to ensure that the new risk is managed. This scenario indicates that (i) risk assessment maintenance must be *proactive* and undertaken much more regularly than an annual basis, and (ii) conducting risk assessment for compliance purposes (possibly only once a year) will leave the organisation wide open to new technological threats unless the *maintain* phase is taken seriously. Risk factors should be identified for ongoing monitoring (e.g., changes in technology use within the system), frequency of risk factor monitoring should be agreed, and change-triggered reviews should revisit and refine the scope, purpose and assumptions of the risk assessment—remembering

to communicate the results each time new risks are identified.

The international standard ISO/IEC 27005 for risk management [54] contains analogous activities to the NIST guidance (see Figure 2.6). It includes an *Establish Context* phase, which is broadly aimed at achieving the outcomes of the *Prepare* phase of NIST and the IRGC *Pre-assessment* phase. The *Risk Assessment* phase is multi-layered, with *identification, estimation, evaluation* stages. This aligns with the IRGC's appraisal and *characterisation* phases. ISO 27005 also has *Risk Communication* and *Risk Monitoring and Review* phases, which relate broadly to the aims of the NIST *Communicate* and *Maintain* phases, and IRGC's crosscutting communication, context and engagement phases. ISO/IEC 27005 has additional elements that explicitly capture risk management decision processes but it is not prescriptive on how to implement them. The inclusion of the *treatment and acceptance* phases linked to communication and review capture some of the fundamental management aspects, offering the choice of treatment or acceptance as part of the assessment process. This aspect of the ISO/IEC 27005 approach is analogous to the risk response element of the NIST-SP800-39 guidance on risk management [28], where the risk response options include acceptance, avoidance, mitigation, or sharing/transfer. The take-away message from this comparison is that, while the risk assessment methods may differ at the risk assessment phase (depending on the type of system being analysed and the scope of the study), the preparation, communication, and continual monitoring phases are must-haves in both widely-used international guidelines, as are the important decisions around risk tolerance. ISO/IEC 27005 is less prescriptive than NIST so offers the option to include a range of assessment and management approaches within the overall process.

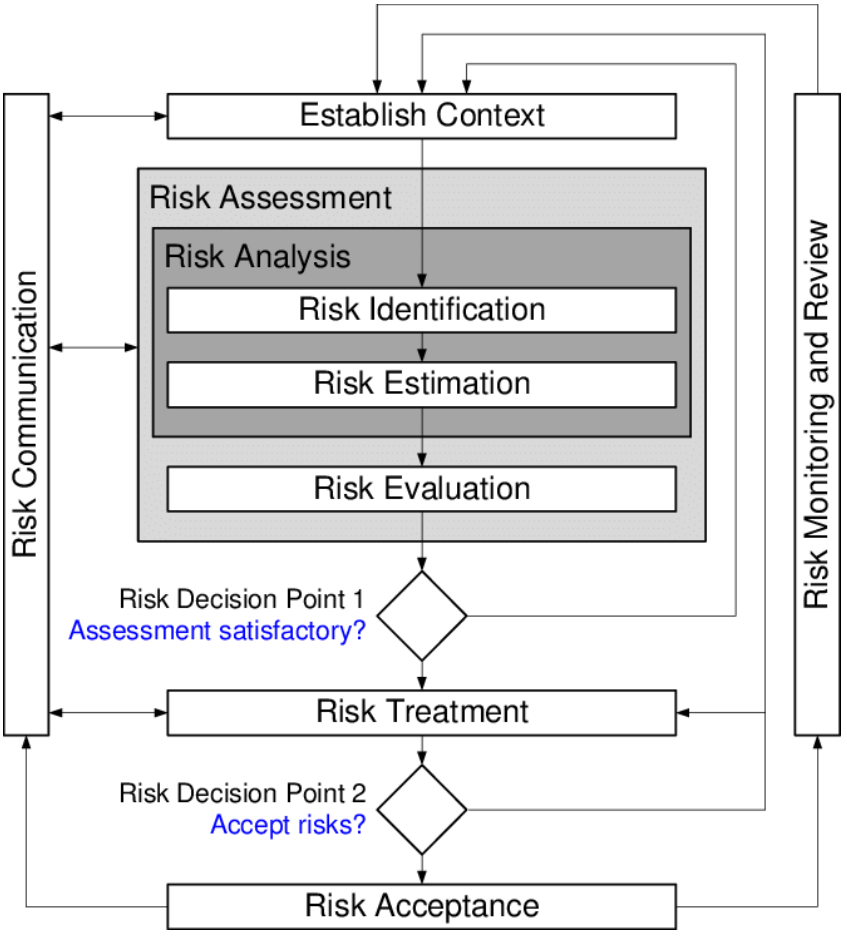


Figure 2.6: ISO/IEC 27005 Process - taken from [59]

A list of commonly used component-driven cyber risk management frameworks can be found at [55]. The list also includes a brief description, an overview of how they work, who should use it, and an indication of cost and prerequisites. While not wishing to reproduce the whole list here, we provide an overview for the purposes of comparison.

- ISO/IEC 27005:2018 is an international standard set of guidelines for information risk management. It does not prescribe a specific risk assessment technique but does have a component-driven focus and requires vulnerabilities, threats and impact to be specified.
- NIST SP800-30/39 are the US Government's preferred risk assessment/management methods and are mandated for US government agencies. They have a strong regulatory focus, which may not be relevant for countries other than the US, but they have a clear set of guiding steps to support the whole risk assessment and management process from establishing context to risk tolerance, and effective controls, including determining likelihood of impact. They are freely available and consistent with ISO standards (which are not free but are low cost).
- The Information Security Forum (ISF) produced the IRAM 2 risk management methodology that uses a number of phases to identify, evaluate and treat risks using the vulnerability, threats and impact measures. It is provided to (paid up) members of the ISF and requires information risk management expertise to use it effectively, which may come at additional cost.
- FAIR, initially developed by Jones [60] and subsequently collaboratively developed with the Open Group into OpenFAIR [61], proposes a taxonomy of risk factors and a framework for combining them. Threat surface can be considered very broad and there is a clear focus on loss event frequency, threat capability, control strength and loss magnitude. It also breaks financial loss factors into multiple levels and supports a scenario model to build comparable loss profiles.
- Octave Allegro is oriented towards operational risk and security practices rather than technology. Qualitative risk assessment is linked with organisational goals. Real-world scenarios are used to identify risks through threat and impact analysis. The risks are then prioritised and mitigation is planned. The approach is designed for workshop style risk assessment and could be performed in-house possibly resulting in a lower cost than a consultant-led risk assessment.
- STRIDE is a failure-oriented threat modelling approach focusing on six core areas: spoofing (faking identity), tampering (unauthorised modification), repudiation (denying actions), information disclosure (data leakage), denial of service (slowing down or disabling a system), and elevation of privilege (having unauthorised control of the system). The approach considers threat targets (including what an attacker may do), mitigation strategy, and mitigation technique. Threats can be considered for multiple interactions on the same threat target in the system and can include people, process and technology. Shostack presents STRIDE as part of a four-stage framework in his book [58] – model the system, find threats, address threats, validate. Threat modelling, of course, cannot guarantee that all failures can be predicted, but the iterative process supports continual assessment of evolving threats if time and resources allow.
- Attack Trees [62] formulate an overall goal based on the objectives of an attacker (the root node), and develop sub-nodes relating to actions that would lead to the successful compromise of components within a system. Like STRIDE, attack trees are required to be iterative, continually considering pruning the tree and checking for completeness.

Attack libraries such as Common Vulnerabilities and Exposuress (CVEs) and Open Web Application Security Project (OWASP) can be used to augment internal knowledge of evolving threats and attacks.

Using and extending the analysis developed in [63] and [55], we provide a comparison table below to enable selection based on the organisational and technical differences for each of these methods (see Table 2.1). While core principles of risk based around vulnerability, threat and impact exist across all methods, there are individual attributes (we refer to as strengths) of each method, as well as resource and reporting differences, that may make them a better fit to an organisation depending on what the risk stakeholders require as evidence of exposure.

Table 2.1: Risk assessment and management methods differences

	Methodology	Assessment Team and Cost	Information Gathering and Reporting
ISO/IEC 2005 :2018	Covers people, process and technology. Not prescriptive in assessment and management method (i.e. other methods in this list could be used to manage risk) but covers threats, vulnerabilities, and impacts. Intended to target higher level management and decision makers. Clear focus on people - internal and external *Strength*:Socio-technical	Aims to include a range of relevant backgrounds in the assessment (covering people, process and tech) and applicable across varying sizes of organisation. Typically externally led due to size and complexity in large organisations, which comes at a cost in addition to the cost of purchasing the documentation. Smaller organisations with less complexity can also follow the principles in-house.	Questionnaire, interviews, document review, process observation. Documentation covers all security controls
NIST SP800-30/39	Focused on technical risk management of IT systems with a prescriptive approach. Includes threats, vulnerabilities, likelihood and impact - along with control monitoring and compliance verification. People not considered as a core organisational asset. *Strength*:Technology-driven	Includes roles and should be usable by organisations of all sizes (albeit it is very US focused). Free to access.	Questionnaire, interviews, document reviews. Checklist reports for operational, management and technical security
ISF	Broad business impact assessment, practitioner led. Threat, vulnerability and impact based *Strength*:Business impact-driven	Only available to members at cost and requires a team with expertise in risk assessment	Information required on impact of losses. Reports on business impact, threat assessment, vulnerability assessment, security requirements evaluation and control selection
FAIR	Taxonomy-based - loss events, threat capability, control strength and loss magnitude. Scenario driven with very well defined measures on economic impact. People are part of the method, both internal business and external threat actors *Strength*: Economic impact-driven	Well-defined method could be used by a small internal team. OpenFAIR standard available via the Open Group	Information sources may vary depending who hold the necessary information. Reports on financial loss magnitudes

	Methodology	Assessment Team and Cost	Information Gathering and Reporting
Octave Allegro	Covers people, technology and physical security. Identifies core IT staff. Self-directed methods intended for internal use, including qualitative management and evaluation workshops linked to identification of organisational goals and related assets. Followed by threat identification and mitigation. Qualitative risks (e.g. reputation, productivity) have relative impact scores (low, medium, high multiplied by categorical risk score) to support prioritisation *Strength*: Qualitative goal-oriented focus	Collaborative assessment team from within and across business including management, staff and IT. Free to access. Documentation states it is targeted at organisations with 300+ employees	Workshops and questionnaires. Baseline reports profile of practices, threat profile, and vulnerabilities
STRIDE	Threat assessment method. Can include people, technology and physical security. Well documented and clear approach based on threats, mitigation (including tolerance levels for risk), and mitigation including who signs off on risk. *Strength*: Threat-driven	Small threat modelling team from within and across business including management and IT. Free to access	Threat workshops. Graphical threat models and tables capturing STRIDE analysis for systems elements and interactions.
Attack Trees	Similar threat assessment to STRIDE, but more attack-specific, focusing on key details of attack methods. *Strength*: Attack-driven	Small attack modelling team from within the business with a technical focus. Openly accessible method	Attack modelling workshops. Attack trees and quantitative measures of likelihood of attack with associated impact.

A list of commonly used system-driven cyber risk management methods can be found at [56]. Below we provide an overview and identify the attributes that can act as differentiators based on the core focus of each method. These all focus on system-level risk and, as such, may require significant human resource effort depending on the size of the organisation. The main objective of these methods is to capture interactions and interdependent aspects of the system and thus requires extensive engagement with process owners and seeking the 'right' people with knowledge of sub-systems.

- *Systems-Theoretic Accident Model and Process (STAMP)* is an ensemble of methods used for modelling causation of accidents and hazards, developed at MIT [64]. Initially focused on safety as a dynamic control problem including direct and indirect causality, it has also been applied to cyber security (e.g., STPA-Sec) and has a focus on socio-technical aspects of risk. The method uses a feedback loop with a controller and a controlled process linked via actuation and feedback. It is based on systems thinking and involves: identification of system purpose, unacceptable losses, hazards, and constraints; development of a hierarchical control structure; identification of unsafe control actions; and the analysis of causal scenarios that lead to these unsafe control actions. This can be supplemented by a timeline or sequence of events.
  Strength: Causality – helps identify risks emerging from subsystem interactions.

- *The Open Group Architectural Framework (TOGAF)* [65] is an enterprise architecture standard that supports component-driven and system-driven approaches to manage risk. The concept of an enterprise in this context encompasses all the business activities and capabilities, information, and technology that make up the entire infrastructure and governance activities of the enterprise. If this extends into partners, suppliers, and customers, as well as internal business units, then the model can also encompass this aspect. Risk assessment in TOGAF is based on a qualitative approach combining effect and frequency labels to produce an overall impact assessment. Risk assessment and mitigation worksheets are then maintained as governance artefacts [66].
  Strength: Linked to structured architectural representation of the enterprise.
- *Dependency Modelling*. The Open Group also developed the Open Dependency Modelling (O-DM) Framework for Goal-oriented risk modelling in a top-down method [67]. This method begins by asking 'What is the overall goal of the system or enterprise?' (e.g., continual manufacturing operations), then asks a further question 'What does this goal depend on to be successful?' (e.g., functioning machinery, operational staff, supply of materials). The method then iterates the questions until a tree of dependencies is created. Goals are abstract so not dependent on actual processes, and allow a connectionist view of an enterprise, its suppliers, and customers to be developed. Recent work has developed tools to support the capturing of dependencies in a workshop setting and apply quantitative probabilities to goals, underpinning Bayesian analysis and modelling cascading failure [68].
  Strength: Capturing interdependencies between abstract goals that sit above, and are linked to, actual business processes.
- *SABSA* [69] is another architecture-based approach. It includes four phases. The first phase identifies the risk associated with achieving objectives so mitigation plans can be identified. The output then feeds into the design phase that determines the security management processes and how they will be used. The third phase implements, deploys and tests the management processes by the operations teams. The final phase relates to management and measurement, which collects security information and reports to the governance stakeholders. The method is enacted by decomposing business processes at different architectural layers, from high-level capabilities (context and concept) down to logical and physical aspects, technology components and activities. Risk is addressed at every layer in a top-down approach to managing risk through activities in all layers, and filtering security requirements from top to bottom to ensure cyber risk is considered throughout. Cutting through all layers is a focus on assets (what), motivation (why), process (how), people (who), location (where) and time (when).
  Strength: Matrix-structured layered approach linked to business model (could sit within TOGAF).

2.6.4 Vulnerability management

A key outcome of the risk assessment and management exercise will be the identification of vulnerabilities. As we discussed in the *maintain* phase of risk management in the previous section, risk assessment needs to be proactive in the context of an ever-changing technology landscape, and include ongoing monitoring and management processes.

A particularly important aspect of this is vulnerability management, which is generally undertaken with a focus on software. New software vulnerabilities are discovered and reported all the time, with vendors often releasing *patches* - fixes for software vulnerabilities - once a month.

It is an often-discussed concern that around one in three cyber breaches occur due to vulnerabilities for which a patch was available, but had not been applied. Closing the loop between the issuance of patches, and their application within a digital infrastructure, is therefore ideally achieved in as little time as possible. Applying patches once a week, or even once a month (per Microsoft's 'Patch Tuesday') significantly reduces the likelihood of the vulnerability being exploited.

Understanding where the latest vulnerabilities lie in software is potentially a time-intensive task, and one that SMEs in particular may not prioritise. However, a number of automated tools exist for this. They scan the network, gathering information about services and software that are running (in the same way an attacker might), and can produce reports on which assets are vulnerable (e.g software version numbers). Of course, these should be treated with caution (they have been known to create false positives), and run only with specific access-controlled user accounts to avoid attackers using them for insider knowledge.

Individuals or groups responsible for risk governance should meet regularly to triage these reports, deciding priorities and timelines for fixing vulnerabilities (i.e. patching or re-configuring). If fixes are not implemented, then the reason for this should be justified, documented, and approved by the relevant owner of the risk to which this vulnerability pertains. Prioritisation might include factors such as the impact of the vulnerability being exploited – financially, or a cascading impact on other aspects of a system; the visibility of the vulnerability (i.e., is it Internet-facing and therefore an open target for attackers?); or the ease with which an automated vulnerability exploit could be deployed into the system (e.g., via an email attachment).

Cost and availability of skilled resource to be able to fix issues is also likely to be a consideration, and these need to be weighed up as a business decision alongside the prioritised fixes – ensuring all vulnerability management decisions are ratified and documented.

2.6.5 Risk assessment and management in cyber-physical systems and operational technology

We start with a note on security vs. safety. While traditional IT security (e.g., corporate desktop computers, devices and servers) may generally take a risk assessment perspective focused on minimising access (confidentiality), modification (integrity) and downtime (availability) within components and systems, the world of cyber-physical systems and Operational Technology (OT) typically has a greater focus on *safety*. These components and systems, also known as Industrial Control Systems (ICSs) underpin Critical National Infrastructure (CNI) such as energy provision, transportation, and water treatment. They also underpin complex manufacturing systems where processes are too heavy-duty, monotonous, or dangerous

for human involvement. As a result, OT risks will more often involve a safety or reliability context due to the nature of failure impacting worker and general public safety and livelihood by having a direct impact in the physical world. This is perhaps a prime case for the use of systems-driven methods over component-driven, as the former support the abstraction away from components to high-level objectives (e.g., avoiding death, complying with regulation). Taking this view can bridge the security and safety perspective and support discussion on how to best mitigate risk with shared system-level objectives in mind.

Efforts to continually monitor and control OT remotely have led to increasing convergence of OT with IT, linking the business (and its associated risks) to its safety critical systems. Technology such as Supervisory Control and Data Acquisition (SCADA) provides capability to continually monitor and control OT but must be suitably designed to prevent risks from IT impacting OT. In Europe the Network and Information Systems (NIS) directive [30] mandates that operators of essential services (such as CNI) follow a set of 14 goal-oriented principles [31], focused on outcomes broadly based around risk assessment, cyber defence, detection and minimising impact. Safety critical systems have a history of significant global impacts when failure occurs in the control systems (e.g., Chernobyl, Fukushima), and the addition of connectivity to this environment has the potential to further increase the threat surface, introducing the additional risk elements of global politics and highly-resourced attackers (e.g., Stuxnet, BlackEnergy). Recent additions to this debate include the uptake and adoption of IoT devices, including, for example, smart tools on manufacturing shop-floors. These are a more recent example of an interface to safety critical systems that could offer a window for attackers to breach systems security. IoT security is in its infancy and the approach to risk management is yet to be completely understood.

The cyber security of cyber-physical systems, including vulnerabilities, attacks and counter-measures is beyond the scope of this KA and is discussed in detail in the Cyber-Physical Systems Security Knowledge Area (Chapter 21). However, there is an important point to note around vulnerability management. Referring back to the previous Section on the topic, we noted that the continual identification of vulnerabilities is a crucial part of the risk governance process. We discussed the use of automated tools to support this potentially time intensive task. However, operational technology is particularly susceptible to adverse impact from automated scans. For instance, active scanning of legacy devices can disrupt their operations (e.g. impacting real-time properties or putting devices into stop mode) – rendering the device useless. Consideration for legacy systems when scanning for vulnerabilities is an important point to remember in the context of cyber-physical systems.

2.6.6 Security Metrics

Security metrics is a long-standing area of contention within the risk community as there is debate over the value of measuring security. It is often difficult to quantify – with confidence – how *secure* an organisation is, or could be. Qualitative representations such as *low, medium, high* or *red, amber, green* are typically used in the absence of trusted quantitative data, but there is often a concern that such values are subjective and mean different things to different stakeholders. Open questions include: what features of a system should be measured for risk?, how to measure risk?, and why measure risk at all? Some metrics may be related to risk levels, some to system performance, and others related to service provision or reliability. Jaquith provides some useful pointers on what constitutes *good* and *bad* metrics to help select appropriate measures [47].

Good metrics should be:

- Consistently measured, without subjective criteria.
- Cheap to gather, preferably in an automated way.
- Expressed as a cardinal number or percentage, not with qualitative labels like "high", "medium", and "low".
- Expressed using at least one unit of measure, such as "defects", "hours", or "dollars".
- Contextually specific and relevant enough to decision-makers that they can take action. If the response to a metric is a shrug of the shoulders and "so what?", it is not worth gathering. [47]

Bad metrics:

- Are inconsistently measured, usually because they rely on subjective judgments that vary from person to person.
- Cannot be gathered cheaply, as is typical of labour-intensive surveys and one-off spreadsheets.
- Do not express results with cardinal numbers and units of measure. Instead, they rely on qualitative high/medium/low ratings, traffic lights, and letter grades. [47]

More extensive discussions of options to select metrics, along with case studies can be found in Jaquith's book [47].

The work of Herrmann [57] provides a more pragmatic view based on regulatory compliance, resilience and return on investment. There are examples of metrics that could provide utility in domains such as healthcare, privacy and national security. The perspective on metrics is grounded in the understanding that we cannot be completely secure, so measuring *actual* security against *necessary* security is arguably a defensible approach, and the metrics described are tailored towards measuring the effectiveness of vulnerability management. Essentially, is it possible to quantify whether the risk management plan and associated controls are fit for purpose based on the threats identified, and do the metrics provide evidence that these controls are appropriate? Furthermore, are the controls put in place likely to add more value in the savings they produce than the cost of their implementation? This point is particularly pertinent in the current era of artificial intelligence technology being marketed widely at an international level to protect digital infrastructure. With a large price tag there is a question mark over an evidence-based understanding of the actual added-value of such security mechanisms and the cost-effectiveness of such solutions in the light of potential savings.

Jones and Ashenden [70] take an actor-oriented approach to security metrics, providing a range of scenarios where threats are ranked based on a mixed qualitative and quantitative method. For instance, nation state threats are based on metrics such as population, literacy and cultural factors; terrorist groups on technical expertise, level of education and history of activity; and pressure groups are ranked on spread of membership, number of activists, and funding. The framework provides a perspective on how to capture measures that ground threat metrics in information that can support discursive, intelligence-led and culturally-grounded risk assessment. However, the approach of "thinking like an attacker" or profiling the adversary has been reported to fail even at nation-state level (with a lot of investment and intelligence). In an article with President Obama on the complications and failures of risk management in the state of Libya, he notes that the US analytical teams underestimated the attacker profile

(particularly socio-cultural aspects), which led to failure in risk management [71]. Assuming knowledge of the adversary can be very risky, but metrics to profile possible threats and attacks (while explicitly accepting our limitations in knowledge) can be used as part of a threat modelling approach such as STRIDE [58] or Attack Trees [62]. Shostack (the author of [58]) discusses the limitations of attacker profiling in a blog post [72].

While quantitative metrics framed in this way appear preferable to qualitative metrics, it is not always a trivial process to collect consistently measured data, either manually or automated. This brings us back to the point around communication and agreeing common language in the risk assessment phase. While metrics may be limited in their accessibility and consistent collection, agreeing the upper and lower bounds, or specific meaning of qualitative labels also provides a degree of value to measuring the security of a system through well-defined links between threats and their relationship to vulnerabilities and impact.

2.7 BUSINESS CONTINUITY: INCIDENT RESPONSE AND RECOVERY PLANNING

[73, 74]

Ultimately, despite all best efforts of accountable individuals or boards within a company who have understood and managed the risk they face, it is likely that at some point cyber security defences will be breached. An essential part of the risk assessment, management and governance process includes consideration and planning of the process of managing incidents and rapidly responding to cyber attacks. The aim is to understand the impact on the system and minimise it, develop and implement a remediation plan, and use this understanding to improve defences to better protect against successful exploitation of vulnerabilities in future (feedback loop). This is still a nascent area of cyber security maturity. Organisations typically prefer to keep information about cyber security breaches anonymous to prevent reputational damage and cover up lapses in security. However, it is likely that other organisations, including competitors will succumb to the same fate in the future, and could benefit from prior knowledge of the incident that occurred. At a broad scale, this is something that needs to be addressed, especially given the offensive side of cyber security will communicate and collaborate to share intelligence about opportunities and vulnerabilities for exploiting systems. Certain industries such as financial and pharmaceutical sectors have arrangements for sharing such intelligence but it is yet to become commonplace for all types of organisations. Large public consortia such as Cyber Defence Alliance Limited (CDA), Cyber Information Sharing Partnership (CISP), and the Open Web Application Security Project (OWASP) are all aiming to support the community in sharing and providing access to intelligence on the latest threats to cyber security. For more detailed information on incident management see the Security Operations & Incident Management Knowledge Area (Chapter 8).

ISO/IEC 27035-1:2016 [74] is an international standard defining principles for incident management. It expands on the aforementioned ISO/IEC 27005 model and includes steps for incident response, including:

- *Plan and Prepare:* including the definition of an incident management policy and establishing a team to deal with incidents.
- *Detection and Reporting:* observing, monitoring detecting and reporting of security incidents.

- *Assessment and Decision:* determining the presence (or otherwise) and associated severity of the incident and taking decisive action on steps to handle it.
- *Response:* this may include forensic analysis, system patching, or containment and remediation of the incident.
- *Learning:* a key part of incident management is learning – making improvements to the system defences to reduce the likelihood of future breaches.

The NCSC also provides ten steps to help guide the incident management process [75] which, broadly speaking, relate the to the Plan, Detect, Assess, Respond and Learn phases of ISO/IEC 27035. In summary, the steps include:

- *Establish incident response capability:* including funding and resources, either in-house or externally to manage incidents. This should include reporting incidents and managing any regulatory expectations.
- *Training:* ensuring that necessary expertise is in place to manage incidents (e.g., forensic response and understanding of reporting expectations).
- *Roles:* assign duties to individuals to handle incidents and empower them to respond to incidents in line with a clear action plan – and make sure this person is well known to people who may be likely to identify an incident.
- *Recovery:* particularly for data and critical applications, make sure a backup is physically separated from the system – and test the ability to restore from backup.
- *Test:* play out scenarios to test out the recovery plans; these should be refined based on practical and timely restoration under different attack scenarios.
- *Report:* ensure that information is shared with the appropriate personnel internally to improve risk management and security controls, plus externally to ensure legal or regulatory requirements are met.
- *Gather evidence:* forensic response may be crucial following an incident – the preservation of evidence could be critical to legal proceedings or, at a minimum, understanding the events that led to the breach.
- *Develop:* take note of the actions taken as part of the incident response. What worked and what did not? Where could the process be improved? As well as defences, the response plan may also benefit from refinement. Security is an ever-evolving issue and requires continual reflection. Security policies, training, and communication may all help reduce the impact of future breaches.
- *Awareness:* continue to remind employees of their responsibilities and accountability regarding cyber security – remind them of how to report incidents and what to look out for. Vigilance is key whether it involves reporting suspicious behaviour or a known personal error that has led to a breach.
- *Report:* Cyber crime must be reported to relevant law enforcement agencies.

As a final word on business continuity we highlight the significance of supply chains. Incident management approaches along with systems-level risk assessment methods are designed to enable the capture of risks relating to interactions and interdependent aspects of the system, which, of course, can and should include supply chains, but will only do so if due attention is

given this aspect of risk. Cyber security of supply chains risk, while nascent as a topic with regards to risk assessment and governance [76][77], is an important issue.

2.8 CONCLUSION

We have explained the fundamental concepts of risk, using a working definition of *the possibility that human actions or events may lead to consequences that have an impact on what humans value*, and placed this in the context of cyber risk management and governance. Using academic foundations that have been widely adopted in international practice, we have explained the links between pre-assessment and context setting, risk and concern assessment, characterisation and evaluation, management, and governance. Risk governance is the overarching set of ongoing processes and principles that underpin collective decision-making and encompasses both risk assessment and management, including consideration of the legal, social, organisational and economic contexts in which risk is evaluated. We have defined some of the core terminology used as part of the structured processes that capture information, perceptions and evidence relating to what is at stake, the potential for desirable and undesirable events, and measures of likely outcomes and impact – whether they be qualitative or quantitative.

A major aspect of risk is human perception and tolerance of risk and we have framed these in the extant literature to argue their significance in risk governance aligned with varying types of risk – routine, complex, uncertain and ambiguous. We have particularly drawn on factors that influence the perception of risk and discussed how these link to the human factors of cyber security in the context of security culture. Training, behaviour change, creation of confidence and trust, and stakeholder involvement in the risk governance process have been highlighted as crucial success factors. This is based on well-established literature that people's intuition and bias will often outweigh evidence about risk likelihood if they believe the management of the risk is not trustworthy, does not apply to them, or is beyond their control. We need people to buy into risk governance rather than impose it upon them. Accordingly, we introduced the concept of balancing accountability with learning, proposing that failures in the risk governance process should lead to feedback and improvement where individuals that may have breached risk management policies should feel able to bring this to the attention of risk managers without fear of stigmatisation.

We differentiated between system-level risk management that analyses the risk of a system as a whole and considers inter-dependencies between sub-systems; and component-level risk management that focuses on risk to individual elements. A number of well-established risk management methods from the systems and component perspectives were analysed with core strengths of each highlighted and some insights into how the methods function, the resources (human and economic) required, and information gathering/reporting requirements. While the core principles of risk – based around vulnerability, threat and impact – exist across all methods, there are individual attributes (we referred to as strengths) of each method that may make them a better fit to an organisation depending on what the risk stakeholders require as evidence of exposure. We reflected briefly on the context of safety in risk assessment for operational technology, which also included the growth of IoT and the need to consider additional directives for critical national infrastructure risk.

Measuring security and the limitations of metrics were discussed in the context of possible options for security metrics, as well as differing views in the community on the benefits and limitations of metricised risk. Finally, we linked to incident response and recovery, which

should provide a feedback loop to risk management planning within the risk governance process. Even with the best laid plans, it is likely a breach of cyber security defences will occur at some point and, in addition to the cultural aspects of learning and improvements of staff, we highlighted a number of key steps from international standards that are required to be considered as part of the governance process.

Risk governance is a cyclical and iterative process, and not something that can be performed once. The crosscutting aspects of communication, stakeholder engagement and context bind the risk assessment and management processes and are core to the continual reflection and review of risk governance practices. Incidents, when they occur, must inform risk management policy to improve cyber security in future – and we must accept that we will likely never be completely secure. In line with this, human factors and security culture must respond to the ever changing need to manage cyber risk, enabling and instilling continual professional development through education and *Just Culture* where lessons can be learned and governance methods improved.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Section	Cites
2.2 What is risk?	[22, 23, 24]
2.3 Why is risk assessment and management important?	[23, 24, 25, 26]
2.4 What is cyber risk assessment and management?	[35]
2.5 Risk governance	
2.5.1 What is risk governance and why is it essential?	[40, 41, 42]
2.5.2 The human factor and risk communication	[43, 44, 45]
2.5.3 Security culture and awareness	[46, 47, 48]
2.5.4 Enacting Security Policy	[48, 49, 50]
2.6 Risk assessment and management principles	
2.6.1 Component vs. Systems Perspectives	[35, 52]
2.6.2 Elements of Risk	
2.6.3 Risk assessment and management methods	[49, 53, 54, 55, 56, 58]
2.6.5 Risk assessment and management in cyber-physical systems and operational technology	[30]
2.6.6 Security Metrics	[47, 57]
2.7 Business continuity: incident response and recovery planning	[73, 74]

Chapter 3
Law & Regulation

Robert Carolina | Royal Holloway, University of London

NOTICE AND DISCLAIMER: This knowledge area does not constitute the provision of legal advice or legal services and should not be relied upon as such. The work is presented as an educational aid for cyber security practitioners. Opinions expressed are solely those of the author. This work does not represent official policy or opinion of the NCSC, the government of the United Kingdom, any state, any persons involved in its production or review, or any of their staff, employers, funders, or other persons affiliated with any of them.

INTRODUCTION

The purpose of this knowledge area is to provide a snapshot of legal and regulatory topics that merit consideration when conducting various activities in the field of cyber security such as: security management, risk assessment, security testing, forensic investigation, research, product and service development, and cyber operations (defensive and offensive). The hope is to provide a framework that shows the cyber security practitioner the most common categories of legal and regulatory risk that apply to these activities, and to highlight (where possible) some sources of legal authority and scholarship.

The nature and breadth of the subject matter addressed renders this knowledge area, and the sources cited, a mere starting rather than ending point. Undoubtedly, some favoured, even significant, sources of authority and scholarship have been overlooked.

The reader is assumed to hold no formal qualification or training in the subject of law. The audience is further assumed to be multinational. To make the material practically accessible to such a diverse body of cyber security domain specialists, subjects are presented at a level that would be considered introductory for those who are already well educated in law or public policy.

The rules of mathematics and physical sciences are both immutable and identical around the world. Laws and regulations are not. The foundation of the world's legal and regulatory systems has for many centuries been based on the principle of territorial sovereignty. Various international efforts to harmonise differences in laws and regulations have met with variable degrees of success. In practice, this means that laws and regulations differ – sometimes significantly – from state to state. These differences are not erased simply because people act through the instrumentality of cyberspace [78].

This knowledge area, however, addresses a multinational audience of practitioners who will be called upon to conduct their activities under laws and regulations imposed by different states - both the home state in which they practice, and foreign states with which they make contact. While respecting the reality that legal details vary by state, this knowledge area will attempt to identify some widely shared norms among various systems of domestic law and regulation, and some aspects of public international law, that may (or should) influence the work of the security practitioner.

In the search for generalisable norms that retain utility for the practitioner, this knowledge area focuses primarily on substantive law. Substantive law focuses on the obligations, responsibilities, and behaviours, of persons. Examples include computer crime, contract, tort, data protection, etc.

Procedural rules are mostly excluded from coverage. Procedural rules tend to focus on

managing the dispute resolution process or specifying methods of communication with a state authority. Examples include civil procedure,[1] criminal procedure,[2] and rules of evidence.[3] Although these are significant to the administration of justice, they are often parochial in nature and bound up with quirks of local practice. Cyber security practitioners who need to become familiar with the details of these rules (e.g., forensic investigators, law enforcement officers, expert witnesses, and others who collect or present evidence to tribunals) invariably require specialist guidance or training from relevant local legal practitioners who understand the procedural rules of a given tribunal.[4]

As with many efforts at legal taxonomy, the difference between substance and procedure is imprecise at the boundary. The test for inclusion in this knowledge area is less to do with divining the boundary between substance and procedure, and springs instead from the desire to make normative statements that remain useful to practitioners in a multinational context.

Section 3.1 starts the knowledge area with an introduction to principles of law and legal research, contrasting the study of law and science and explaining the role of evidence and proof. Section 3.2 then explores various aspects of jurisdiction in an online environment.

Sections 3.3 and 3.4 discuss general principles of privacy law (including interception of communications) and the more detailed regulatory regime of data protection law. Section 3.5 presents an outline of computer crime laws, and more specifically crimes against information systems.

Sections 3.6 and 3.7 provide an introduction to principles of contract and tort law of interest to practitioners. Section 3.8 provides a general introduction to relevant topics in intellectual property, while Section 3.9 provides an overview of laws that reduce liability of content intermediaries.

Sections 3.10 and 3.11 address a few specialist topics, with an exploration of rights and responsibilities in trust services systems and a brief survey of other topics of interest such as export restrictions on cryptography products. Sections 3.12, 3.13, and 3.14, conclude the knowledge area with a survey of public international law, ethics, and a checklist for legal risk management.

The author of this knowledge area is trained in the common law[5] (nearly ubiquitous in anglophone territories) and experienced in international commercial legal practice conducted in London. Examples of legal norms are therefore drawn from common law (as interpreted by different states), various anglophone statutes and case decisions, European Union law, and public international law.[6] The author welcomes thoughtful correspondence confirming, further qualifying, or challenging the normative status of issues presented.

Finally, a note on terminology and presentation. ‘Alice’ and ‘Bob’ and similar terms are used in an effort to present ideas in a form likely to be familiar to security practitioners. There is one significant difference in how these terms are used. In most of the technical security literature ‘Alice’ and ‘Bob’ refer to technological devices. In this knowledge area, however, ‘Alice’ and ‘Bob’ refer to persons.[7] Unusually for CyBOK (but in common with legal research and scholarship) this knowledge area makes extensive use of notes. Notes are used for a variety of purposes, including providing specific examples, further explanation of issues, and additional argument in support of or against a given a proposition. In some circumstances notes have been used to suggest potential future legal developments, subjects worthy of further study, or to provide other comments.[8]

CONTENT

3.1 INTRODUCTORY PRINCIPLES OF LAW AND LEGAL RESEARCH

Cyber security practitioners and researchers come from an incredibly wide array of educational backgrounds. Experience teaching legal and regulatory subjects to cyber security post-graduate students, and providing legal advice to cyber security practitioners, suggests that much of this knowledge area's content will be novel to those whose education is based in science, technology, engineering, mathematics, many social sciences, and many of the humanities. These introductory observations are offered as an aid for those who are approaching the subject without significant experience.

3.1.1 The nature of law and legal analysis

Although the reader is assumed to have some degree of familiarity with the process of law making and law enforcement, a review of some of the most common sources of law should help to orient those who are unfamiliar with legal research and analysis.

Law should be analysed with rigorous logic. Unlike scientific disciplines such as physics or mathematics, however, the study of law is not conceptualised as an effort to discover immutable principles of our world. Law is bound together with social and political values, human desire, and human frailty [79].

Society influences the development and interpretation of law even as law influences the behaviour of members of society. Societies evolve and values change. Changes to law and to methods of interpreting law tend to follow.[9] This creates a number of challenges for legal scholarship,[10] as the topic under study continues to change.[11] Perhaps as a result the study of law is often presented in the form of historical dialectic: examining the evolution of law and its interpretation over time, often through case studies. This method provides all-important context, aids in the interpretation of law as it exists, and often suggests the direction of future developments.

The study of law endeavours to share at least one characteristic with the sciences: the ability to predict outcomes. While sciences like chemistry predict the outcome of events such as the introduction of solid sodium to liquid water, the study of law attempts to predict the outcome of disputes submitted to a suitably expert legal tribunal. Although the study of law can never predict outcomes of dispute with 100% certainty, in states with well-developed systems of law and well-qualified adjudicators, it is possible to achieve a degree of predictability of outcome that is sufficiently high to maintain confidence in the system as a whole.[12]

Legal studies often begin with a mechanistic review of the governance processes surrounding the adoption and enforcement of law. Laws are made (legislative authority), laws are interpreted (judicial authority), and laws are enforced (executive authority). Understanding different governance structures adopted by states to manage these three processes requires an examination of comparative constitutional law which is beyond the scope of this knowledge area.

Most legal research and analysis proceeds on the basis of argument from authority, drawn from an analysis of historical texts that embody expressions of law. There follow a few

observations about differing sources of legal authority and how these vary in different contexts. No standards body exists to harmonise the definition of legal terms of art as they are used by different states. Confusion over legal terminology is therefore commonplace in a multinational context.

Primary legislation. In both common law[13] and civil law[14] jurisdictions, primary legislation (typically a statute such as an Act of Parliament in the UK, or an Act of Congress in the US) is the most easily understood embodiment of 'the law'. In civil law jurisdictions primary legislation typically takes the form of adopting or amending a comprehensive legal code.[15] A statute (a law promulgated by a legislature) should be distinguished from a bill (a draft law which may or may not be adopted as a statute)[16] which normally has no force of law.[17]

Secondary legislation. Sometime a degree of law-making authority is delegated by a senior legislative body (such as the UK Parliament or the US Congress) to some other agency of the state (such as the Foreign Minister of the UK or the US Commerce Department). Delegation is often made for reasons of technical expertise, or the need for frequent periodic review of adopted rules. Laws promulgated by such subordinate agencies are generally termed secondary legislation. The term 'regulation' is sometimes used colloquially to refer to secondary legislation as distinct from primary legislation.

European Union legislation. A 'Directive' of the European Union (formerly European Economic Community) is a specific type of primary legislation addressed to the member states of the Union. Each member state is required to examine the terms of the Directive, and then to implement these terms within its own domestic law within a specified time frame. Directives are normally said to lack 'direct effect' in member state law, with some exceptions. By contrast, a European Union 'Regulation' constitutes immediately applicable binding law within all member states.[18]

Judicial decisions. In common law jurisdictions, the published decisions of domestic courts that interpret the law tend to constitute significant and binding interpretative authority depending upon the seniority and jurisdiction of the court. Decisions by the courts of foreign states may constitute persuasive authority, or indeed their interpretation of the law may be ignored entirely.[19] In civil law jurisdictions, the decisions of judges are generally accorded less interpretive authority than similar decisions in a common law jurisdiction.

Codes. In legal research, 'code' can refer to any systemised collection of primary legislation,[20] secondary legislation,[21] model laws,[22] or merely a set of rules published by public or private organisations.[23]

Restatements of the law. A restatement of the law is a carefully constructed work, normally undertaken by a committee of legal experts over a number of years, which seeks to explain, clarify, and codify existing law. Although restatements are not normally considered a source of mandatory authority, as carefully considered expressions of expert opinion they are often extremely influential.[24]

Treaties. Treaties are instruments of agreement among and between states. In some states, the legal terms of a treaty are automatically carried into operation of a contracting state's domestic law once the state has fully acceded to the treaty. In others, domestic law is not amended unless and until the domestic legislature acts to amend domestic law in accordance with the treaty requirements. (Public international law is discussed in Section 3.12.)

Scholarly articles. Within common law jurisdictions, scholarly articles written by legal academics can constitute a type of persuasive, albeit weak, authority. Judges typically adopt the

arguments of legal scholars only to the extent that the scholar's work persuades a jurist to adopt their view. In many civil law systems, by contrast, scholarly articles by leading legal academics may be accorded significant deference by tribunals who are called upon to interpret the law.

3.1.2 Applying law to cyberspace and information technologies

The birth of cyberspace caused a great deal of anxiety with regard to the application of laws and regulations to this new domain.

Two prevailing schools of thought emerged. The first school posited that cyberspace is so radically different from anything in human experience, that old laws were unsuitable and should be widely inapplicable to actions taken using this new domain. Law makers and judges were encouraged by this school to re-examine all doctrines afresh and to abandon large swathes of precedent when considering disputes. Radical proponents of this view went so far as to deny the authority of sovereign states to enforce laws and regulations in the context of Internet-related activities [80].

The second school held instead that the Internet is, like so many tools developed in human history, merely an instrumentality of human action. As such, laws could – and perhaps should – continue to be applied to persons who use cyberspace in most respects just as they applied before it existed [81, 82, 83]. Members of this second school described a 'cyberspace fallacy' – the false belief that cyberspace was a legal jurisdiction somehow separate and distinct from real space [84].[25]

For the time being, the second school has almost universally prevailed with state authorities [78, 85, 86, 87]. The practitioner is confronted with the reality that existing laws, some centuries old and some amended or born anew each year, are applied by states, their law makers, judges, police and defence forces to cyberspace-related activity whether or not cyberspace was expressly contemplated by those same laws.[26]

One must be cautious when attempting to map legal rules onto activities. While lawyers and legal scholars divide the law into neat categories, real-life and cyber operations do not always fit neatly within a single category. For example, a single data processing action that does not infringe copyright and is not defamatory may still constitute a violation of data protection rights. Any given action should be assessed by reference to whatever laws or regulations present risk. The problem of conflicting obligations that can arise as a result of multi-state regulation is introduced in Section 3.2.

Practitioners increasingly ask questions concerning the application of law to artificial intelligence. Laws are generally framed to influence and respond to the behaviours of persons, or to address the disposition or use of property. (This can be seen in the discussion of enforcement jurisdiction in Section 3.2.3.) Instances of artificial intelligence are not currently defined as persons under the law.[27] Therefore an AI, as such, cannot be guilty of a crime, enter into a contract, own property, or be liable for a tort. If an object controlled by an AI causes harm, the law would normally be expected to look beyond the AI to the persons who created or made use of it and the responsibility of such persons would be assessed using existing legal standards. This subject is explored briefly in Section 3.7.2, which touches upon circumstances where persons could become strictly liable for AI-related actions which cause death or personal injury.[28]

3.1.3 Distinguishing criminal and civil law

3.1.3.1 Criminal law

Criminal law is the body of law that prohibits behaviour generally abhorred by society. Criminal law is normally enforced by an agency of the state. Examples include prohibitions against bank fraud and computer hacking. Depending upon the society in question, the purposes of criminal law are usually described as some combination of:

- deterrence (seeking to deter bad behaviour, for both members of society generally and a criminal specifically);
- incapacitation (limiting the ability of the criminal to further harm society);
- retribution (causing a criminal to suffer some type of loss in response to crime);
- restitution (causing a criminal to compensate a victim or some related person);
- rehabilitation (seeking to change the long-term behaviour of a criminal).

Terms such as 'guilty' and 'innocent' are normally reserved as descriptions of verdicts (outcomes) in a criminal case. These terms should not be used when referring to outcomes of civil actions.

Punishments available in criminal law include custodial prison sentences, criminal fines normally remitted to the state, seizure and forfeiture of criminal proceeds, and financial or other restitution remitted to victims.

There is often no requirement for an accused to have understood that their actions were defined as criminal, although states normally must prove that the accused intended to take those actions. Some crimes are defined in a fashion that guilt only attaches if the state can prove that the accused was aware that they were doing something 'wrong'.[29] An accused, therefore, may not be able to escape criminal liability by suggesting, or even proving, that an act was undertaken with good intentions or otherwise 'in the public interest'.[30]

3.1.3.2 Civil (non-criminal) law

Civil law[31] is the area of law that regulates private relationships among and between persons. Examples include the laws of contract and negligence. A person injured as a result of breach of civil law can normally bring legal action against the responsible party.

Remedies available under civil law (depending on the circumstances) may include some combination of:

- an order for the liable party to pay compensation to the injured party;
- an order to terminate some legal relationship between the parties;
- an order for the liable party to discontinue harmful activity; or
- an order for the liable party to take some type of affirmative act (e.g., transferring ownership of property).

The principles of civil law are often crafted in an effort to redress negative externalities of behaviour in a modern economy. This makes civil law especially interesting in cyber security, as poor security in the development of ICT products and services is a sadly recurring negative externality that often falls short of criminal behaviour [88]. Policy makers hope that people

who become aware that certain types of risk-taking carry an associated liability for resulting harm will alter their behaviour for the better.

3.1.3.3 One act: two types of liability & two courts

A single act or series of connected acts can create liability simultaneously under both criminal and civil law. Consider the act of Alice making unauthorised access to Bob's computer. Her actions in turn cause Bob's LAN and related infrastructure to fail. Alice's single hacking spree results in two types of liability. The state can prosecute Alice for the relevant crime (i.e., unauthorised access, see Section 3.5) and Bob can bring a civil legal action (i.e., negligence, see Section 3.7.1) against Alice.

The two types of legal action would normally be contested in two separate tribunals, and subject to two different standards of proof (see Section 3.1.4).[32] The purpose of the criminal case is to protect the interests of society as a whole, while the purpose of the civil case is to compensate Bob.

3.1.4 The nature of evidence and proof

The concept of 'proof' in law is different from the term as it is used in the field of mathematics or logic. This can create confusion in discussions of cyber security topics and the law.

In law, to 'prove' something means simply to use permissible evidence in an effort to demonstrate the truth of contested events to a fact finder *to a prescribed degree of certainty*. Permissible evidence can take a variety of forms. Subject to the rules of different legal systems, evidence might include direct witness testimony, business records, correspondence, surveillance records, recordings of intercepted telephone conversations,[33] server logs, etc.[34]

As a gross generalisation, legal analysis in a dispute consists of two elements. A 'fact finder' (a judge, jury, regulator, etc.) must first consider competing versions of events and establish a factual narrative or 'finding'. This factual narrative is then subjected to analysis under relevant law.

A person who brings a legal action is said to carry the burden of proof with respect to the elements that define their right of action. This is also known as proving the claiming party's *prima facie* case. An accused then bears the burden to prove affirmative defences which might serve to reduce or eliminate their liability.[35]

The applicable standard of proof, which is to say the degree of certainty that must be achieved by the fact finder to reach a finding on a given contested issue, depends upon the issue under consideration. A non-exhaustive sample of different standards of proof used in various legal contexts is presented in Table 3.1.

3.1.5 A more holistic approach to legal risk analysis

Those who approach the study of law for the first time often fall victim to seeing only one aspect of the law: 'the rules'. More specifically, the elemental framework from a given law which defines the evidentiary burden to be met by a person seeking to prove the guilt or liability of a second person. This ignores other factors that must be taken into account when analysing legal risk.

Consider a circumstance in which Alice has some right of action against Bob. (Alice could be a state considering prosecution of Bob for a crime or Alice could be a person considering a civil law suit against Bob for breach of contract or tort.) Alice might pursue a legal action against Bob, or she might not. If Alice pursues legal action against Bob, she might win the action or she might lose. Bob must take different factors into consideration when analysing the relevant risk of Alice taking legal action.

It may aid understanding to consider a function:

$$R = f(P, D, Q, X)$$

in which:

- $R =$ the risk-weighted cost to Bob that Alice will commence and win this legal action;
- $P =$ Alice's relative ability (using admissible evidence) to prove her *prima facie* case against Bob (adjusted by Bob's ability to rebut such evidence);
- $D =$ Bob's relative ability (using admissible evidence) to prove any affirmative defence that might reduce or eliminate Bob's liability (adjusted by Alice's ability to rebut such evidence);
- $Q =$ the total cost to Bob (other than transaction costs) if Alice pursues and wins her legal action; and
- $X =$ a variety of additional factors, such as Alice's willingness and ability to commence legal action, Bob's willingness and ability to defend, Alice's ability to secure enforcement jurisdiction over Bob or his assets, plus transaction costs such as investigation costs, legal costs, and court costs.

The purpose of the function above is merely to highlight that legal risk analysis involves more than consideration of 'the rules'.[36] Thus, the discussions of substantive law in this knowledge area (e.g., data protection, criminal law, contract, tort) begin with some examination of the framework used to prove liability (P). Discussion also touches on some affirmative defences (D) as well as relevant penalties and remedies (Q). The knowledge area gives significant, separate, consideration to the problem of jurisdiction (which falls within X). In assessing each of these factors, one must also consider the probative value of available evidence as well as the relevant standard of proof to be met in each element (see Section 3.1.4).

Some areas of risk, such as risks related to transaction costs including mechanisms that may shift some transaction costs from winner to loser (which also fall within X), are highly individualised and process-oriented and beyond the scope of this knowledge area.

The issues introduced here significantly underpin the observations concerning legal risk management in Section 3.14.

Standard of proof	Degree of Certainty Required	Example context
Beyond a reasonable doubt.	Extremely high. Almost incontrovertible. No other reasonable explanation exists to make sense of the evidence.	States are most often required to meet this, or a similar standard, in proving the elements of a crime for a fact finder to hold an accused person guilty. This higher standard is heavily influenced by notions of human rights law because individual life and liberty are at stake.
Clear and convincing evidence.	Reasonably high certainty. Much more than simply 'probable'.	This standard of proof is used in US law, for example, when a court is asked to invalidate a previously granted patent. The burden of proof placed upon the person seeking to invalidate the patent is set high because this would deprive a rights-holder of property previously granted by the patent office. This phrase is also used to describe the standard to be met by prisoners who challenge the validity of their criminal conviction in US federal courts using a *habeas corpus* petition long after normal routes of appeal have been exhausted. In this circumstance, the higher standard is required as a means of preserving the integrity of the original criminal justice process (including the original appeals) while not foreclosing all possibility of post-conviction review.[37]
Preponderance of evidence. Balance of probabilities.	More probable than not. Greater than 50%. When weighed on the scales of justice, the evidence on one side is at least a feather-weight greater than the other.	The most common formulations of the standard of proof required to prevail in a civil case.
Probable cause.	The evidence suggests that the target of an investigation has committed a crime, although evidence is not yet conclusive.	The standard required in the US to persuade a judicial officer to issue a search warrant or arrest warrant. This standard serves to filter out trivial or unsubstantiated requests to intrude into privacy or detain a suspect.
Reasonable suspicion.		The standard typically required in the US to justify a police officer temporarily stopping and questioning a person. This lower standard is often justified on policy grounds of minimising threats to the safety of police officers. This phrase has also been suggested by the United Nations High Commissioner for Human Rights on the right to privacy in the digital age as a threshold for justifying state electronic surveillance [89].

Table 3.1: Example Standards of Proof

3.2 JURISDICTION

[78, 90, 91, 92, 93, 94]

Cyberspace enables persons located in different states to communicate with one another in a fashion that is unprecedented in history. Once-unusual international contacts and relationships have become commonplace. Those who face a potential threat of enforcement by a person in a foreign state must consider a few threshold questions before the relevant legal risk can be analysed: jurisdiction and conflict of law.

Jurisdiction describes scope of state authority and the mechanisms used by a state to assert power. Private international law, or conflict of law, examines how to determine which domestic state law(s) will be applied to resolve certain aspects of a given dispute. This section of the knowledge area discusses jurisdiction. Conflict of law is addressed separately in the context of individual substantive headings of law.

Many of the principles concerning jurisdiction and conflict of law are not new. What has changed are the larger numbers of people who benefit from considering these principles now that persons are facing cross-border legal responsibilities at increased rates.

3.2.1 Territorial jurisdiction

The term 'jurisdiction' is often used in a rather informal manner to refer to a state, or any political sub-division of a state, that has the authority to make or enforce laws or regulations.[38] In this sense, the term is nearly synonymous with the territory of that state or its political sub-division. The purpose of this section, however, is to focus more specifically on the territorial extent of a state's power – its territorial jurisdiction.[39]

When reviewing legal risks from multi-state activities conducted via cyberspace, it may be helpful to consider three different aspects of jurisdiction: prescriptive jurisdiction, juridical jurisdiction, and enforcement jurisdiction.

Prescriptive jurisdiction describes the scope of authority claimed by a state to regulate the activities of persons or take possession of property. Law makers normally adopt laws for the purpose of protecting the residents of their home state and may declare their desire to regulate the actions of foreign-resident persons to the extent that such actions are prejudicial to home state-resident persons.

Juridical jurisdiction describes the authority of a tribunal to decide a case or controversy. The rules of such jurisdiction vary widely from tribunal to tribunal. In civil cases, courts usually demand a minimum degree of contact between the residential territory of the court and the property or person against which legal action is taken. Such minimum contact might involve obvious examples such as the presence of a branch office. It might be extremely minimal, indeed, resting upon little more than correspondence soliciting business from a resident of the court's territory.[40] In the context of criminal prosecutions, courts normally demand the physical presence of an accused before proceedings commence. Some states allow courts to make exceptions to this rule and are prepared to conduct a criminal trial *in absentia* if the defendant cannot be found within the territorial jurisdiction of the court.

Enforcement jurisdiction describes the authority of a state to enforce law. This is sometimes described as police power, power to arrest and detain, authority to use force against persons, etc. In civil matters, this may describe other methods used to project force over persons or

property resident in a territory, such as seizing plant and equipment, evicting tenants from property, garnishing wages, seizing funds on deposit with a bank, etc. In practice, enforcement jurisdiction is limited by the ability of the state and its agents to project power over the objects of enforcement.[41]

3.2.2 Prescriptive jurisdiction

It has long been commonplace for states to exert a degree of prescriptive and juridical jurisdiction over non-resident persons who solicit business relationships with residents. A theory often espoused is that non-resident persons who remotely solicit or enter into business relationships with residents avail themselves of the benefits of the domestic market and, therefore, become amenable to the rules of that market. This principle long predates the Internet.

More controversial are cases where a non-resident person is not soliciting business from a state resident but may nonetheless be acting in a fashion which somehow harms state residents. Some of the best-known examples arise in competition law (a.k.a. anti-trust law). These cases follow a familiar pattern. A cartel of persons who produce commodities (e.g., bananas, aluminium, wood pulp, diamonds) outside of the state's territory, convene a meeting that also takes place outside the state's territory. In this meeting the cartel members conspire to fix the wholesale prices of a given commodity. This kind of offshore price-fixing conspiracy, which would be disallowed if it took place within the state's territory, eventually results in inflated prices inside the state as well. The only communication between the prohibited act (price fixing) and the state is the price inflation in the overseas (exporting) market, which in turn causes inflation of domestic (importing) market prices.

At the start of the twentieth century the notion of applying a state's domestic competition law to such overseas activity was considered wholly inappropriate [95]. The growth of international trade in the modern economy, however, caused courts to reconsider this position. US courts decided in 1945 that extending prescriptive jurisdiction to foreign price-fixing activity was justified due to the consequential harm to the domestic market and the sovereign interest in protecting the functioning of that market [96]. A substantially similar (if not identical) doctrine was announced in 1988 by the European Court of Justice when applying European competition law [97, 98]. Although these jurisdictional theories have been criticised, they are now exercised routinely.

States also claim prescriptive jurisdiction over some actions taken by their own nationals while present in a foreign state even if no express 'effect' is claimed within the territory of the home state. Examples include laws prohibiting bribery of foreign officials [99] and laws against child sex tourism [100, 101]. States may also claim prescriptive jurisdiction over violent acts committed against a state's own nationals outside of the state's territory by any person, especially in cases of terrorism.[42]

Instances where more than one state claims jurisdiction over a single act or occurrence are not uncommon. Claims of prescriptive jurisdiction tend to be founded on notions of protecting the interests of a state and its residents. Some of the rules of jurisdiction have been adopted with a view to reducing instances where persons might face irreconcilable conflict between the mandates of two states. Although such irreconcilable conflicts are less common than some might believe, they still arise from time to time. In cases where a person faces an irreconcilable conflict of mandates imposed by two states, the person is required to make hard choices. For businesses, these choices often involve changing business processes,

structure or governance to avoid or limit the potential for such conflicts.

3.2.2.1 Prescriptive jurisdiction over online content

Numerous court decisions around the world have confirmed the willingness of states to assert prescriptive jurisdiction over actions where criminal or tortious content originates from outside of the state's territory, is transferred via the internet, and displayed within the state's territory. Examples of laws that have been enforced on this basis include copyright, defamation, gaming/gambling services, and state-specific subject matter prohibitions such as the prohibition against displaying or offering for sale Nazi memorabilia within France [78, 90, 91, 102].

These exercises of jurisdiction do not necessarily rest on the more attenuated 'effects doctrine' used in competition law. Courts seem willing to interpret domestic law in a fashion which asserts prescriptive jurisdiction, and then to assert their own juridical jurisdiction on the basis that content is visible to persons within the state irrespective of the location of the server from which it originates. In this fashion, the offending act (e.g., copying, publishing, transmitting, displaying, offering for sale) is said to take place within the state asserting jurisdiction.

3.2.2.2 Prescriptive jurisdiction over computer crime

States adopting computer crime laws often legislate to include cross-border acts. As a result, it is common for a state with such laws on their books to exercise prescriptive jurisdiction over persons – no matter where they are located – who take actions directed to computer equipment located within the state. Similarly, persons who act while physically located within the state's territory are often caught within the scope of the criminal law when conducting offensive operations against computers resident in foreign states [92, 93, 94, 103, 104]. Public international law recognises such exercises of prescriptive jurisdiction as a function of territorial sovereignty ([87] at R.1-4, R.10).

When a hacker who is physically present in one state directs offensive activity to a computer in another state, that hacker may violate the criminal law of both states. If the relevant hacking activity does not constitute a crime in the first state for whatever reason,[43] it may still constitute a crime under the law of the second state where the target computer is located [103, 104].

3.2.2.3 Prescriptive jurisdiction and data protection (GDPR)

GDPR brought about a significant change in the territorial prescriptive jurisdiction of European data protection law [105].

GDPR, in common with its predecessor 1995 legislation, applies first to any 'processing of personal data in the context of the activities of an establishment of a controller or a processor in the Union, regardless of whether the processing takes place in the Union or not' (Art. 3(1)). The term 'establishment of a controller' as used in EU data protection law generally, is extraordinarily broad when compared with other commonly understood legal principles. Creating or maintaining an establishment in the territory of the EU merely means the ability to direct business affairs or activities. This definition is not restricted by the usual niceties of corporate or international tax law. A holding company in the US, for example, can be deemed to have a personal data processing establishment in the EU through the non-processing activities of its wholly owned subsidiary [106]. Thus, legal persons that have no 'permanent establishment' or 'taxable presence' in the EU for purposes of analysing direct tax liability may

nonetheless be deemed to be carrying out data processing in the context of an 'establishment' in the EU for the purposes of analysing GDPR liability.

GDPR now also asserts prescriptive jurisdiction over the personal data processing activities of any person, anywhere in the world, related to offering goods or services to data subjects in the EU (Art. 3(2)(a)). Prescriptive jurisdiction is believed to extend only to circumstances when the supplier volitionally offers such goods or services to data subjects in the EU.

Finally, GDPR applies to any person who monitors the behaviour of data subjects located in the EU, to the extent that this monitored behaviour 'takes place in' the EU (Art. 3(2)(b)). This heading of jurisdiction appears to have been motivated primarily by the emergence of services which monitor and analyse a variety of human behaviours including actions performed by persons using web browsers, or physical movement patterns exhibited by persons on the ground such as shopping behaviour.

Persons located outside the EU, who are nonetheless subject to the prescriptive jurisdiction of GDPR because they offer goods or services to, or monitor the behaviour of, persons resident in the EU, are often required to appoint a representative in the EU (Art 27; Recital 80).

Interpreting the scope of GDPR's territorial jurisdictional can be difficult, especially given the rapid emergence of new forms of online services. The European Data Protection Board is expected to finalise formal guidance in due course [107].

3.2.3 Enforcement jurisdiction

While it is relatively easy to imagine a state exercising broad prescriptive and juridical jurisdiction over activities and controversies, more difficult questions arise with respect to enforcement jurisdiction: how a state practically enforces its rules.

As a general proposition, one state has no right under public international law to exercise enforcement jurisdiction within the territory of another state ([87] at R.11).[44]

This section considers some of the more common enforcement mechanisms used by states in a cyber security context. Enforcing the law tends to turn on three different mechanisms of state power: power over persons (*in personum* jurisdiction), power over property (*in rem* jurisdiction), and requests or demands for international assistance.

3.2.3.1 Asset seizure and forfeiture generally

It is common to assert *in rem* jurisdiction over the property or other legal rights that are present within a state's territory and amenable to that state's police powers. The state might seize such property in an effort to compel attendance at court proceedings, or eventually sell the property to meet the financial obligations of an absent person. Examples of objects seized for this purpose include immovable property such as office buildings or factories, movable property such as plant and equipment, trucks, maritime vessels, or merchandise in transit, and intangibles such as intellectual property rights or rights to withdraw funds on deposit with a bank.

3.2.3.2 Seizure and forfeiture of servers, domain names, and registries

When a server located in a state is used to conduct activity that constitutes a crime in that state, seizing the server as an enforcement mechanism might be considered. Moving beyond the server, however, US law enforcement authorities have also used *in rem* jurisdiction for seizure and forfeiture of domain names where the domain TLD registry is maintained in the US. Actions for infringement of trademark rights have used similar *in rem* powers for domain name seizure and forfeiture. This is a potentially interesting enforcement tool in the US, especially as TLD registries administered and maintained from within the territory of the US include '.com', '.org' and '.net' [108, 109].

Similar *in rem* powers have been asserted by various states to regulate the administration of the ccTLD registry associated with their state, or to forcibly transfer the administration and operation of the ccTLD to a different in-state administrator [110].[45]

3.2.3.3 Territorial location of the right to demand repayment of bank deposits

Efforts to enforce laws that freeze or otherwise restrict depositor access to funds on deposit have raised difficult questions about the territorial scope of state enforcement authority. Asset freeze orders directed to enemy states or their citizens are not unusual, especially at times of international conflict.

A case highlighting limits of this power arose from the 1986 order issued by the United States mandating the freeze of assets held by the state of Libya. This order by the Reagan administration was unusual. In addition to mandating the freeze of money on deposit in the United States, it also ordered any US person who maintained effective control over any bank account anywhere in the world to freeze money on deposit in any of these global bank accounts.

The Libyan Arab Foreign Bank (a state-owned Libyan bank) took legal action against US banks in the courts of England demanding the repayment of deposits (denominated in US dollars) held in London branches. The resulting English court judgment makes for interesting reading, as the court discussed at length the extensive role of electronic funds transfer systems in international banking at that time. Having looked at the question, however, the dematerialised nature of funds transfers ultimately had almost no impact on the outcome of the case. The court held that money deposited with the London branch of a bank constitutes a legal right for the depositor to demand payment of that money in England [111, 112].[46]

In other words, a bank account may be conceptualised as being situated within the territory of the state in which the branch to which the deposit is made is located. This analysis continues to apply if the relationship is carried out entirely through online interactions, and indeed even if the depositor remains offshore and never attends the branch in person.

3.2.3.4 Foreign recognition and enforcement of civil judgments

A civil judgment issued by the court of one state may under certain circumstances be enforced by the courts of a friendly second state. This is normally achieved when the prevailing party transmits the judgment to the courts of the second state where the adverse party has assets, requesting enforcement of the judgment against those assets. Foreign recognition and enforcement of civil judgments is often granted under the principle of *comity*: a doctrine which can be expressed in this context as, 'We will enforce your civil judgments because, as a friendly state, we anticipate you will enforce ours.'[47]

A foreign court's willingness to enforce such civil judgments is not universal. Requests for civil enforcement are sometimes rejected for policy reasons. Nonetheless, this remains a relatively common mechanism in the context of judgments for money damages arising from many contract and tort disputes.

3.2.3.5 The arrest of natural persons in state territory

It is normally straightforward for police officers to arrest persons present within their state's territory. When a criminal suspect is outside the state's territory, officials are sometimes able to arrest that suspect when they subsequently appear in state – whether or not it was an intended destination. Law enforcement officers can normally arrest the accused upon their arrival in state territory.[48]

State authorities can normally exercise the power of arrest on any seagoing vessel within the state's territorial waters, as well as vessels registered under the flag of the arresting state when in international waters. Additional maritime enforcement scenarios are possible [113].

3.2.3.6 Extradition of natural persons

If an accused criminal is not present within the state, a traditional method of obtaining custody is to request extradition from another state [92, 94]. Extradition is normally governed by bilateral extradition treaties, and is normally only allowed when the alleged criminal act constitutes a crime in both states (the requirement of dual criminality).

If two states that are contracting parties to the Budapest convention (see Section 3.5.1) maintain a bilateral extradition treaty between them, the Convention obliges them to incorporate within their extradition procedures those computer crimes mandated by the Convention. The Convention can (optionally) also serve as an independent legal basis for extradition between two contracting states which do not maintain a bilateral extradition treaty [103] at article 24.

Extradition has a troubled history in cyber security. Extradition requests for accused cyber criminals might be denied by another state for a number of reasons: lack of an extradition treaty between the two states, lack of dual criminality, public policy concerns over the severity of punishment to be imposed by the requesting state, and concerns for the health or welfare of the accused, are all reasons that have been cited for refusal to grant the extradition of persons accused of cybercrime [90].

3.2.3.7 Technological content filtering

Technological intervention can be adopted as a practical expression of state power – either by a state directly ordering such intervention, or by other persons adopting a technical intervention to avoid or limit liability.

Content filtering is merely one type of technological intervention that can be used to enforce law or to reduce the risk of adverse enforcement activity. This approach fits generally within the concept explored by Lawrence Lessig and expressed with the phrase, 'code is law' [81].[49]

An enforcing state can direct an enforcement order to a person mandating that they filter content at the point of origination, whether the content is hosted on an in-state or out-of-state server [102]. Such an order carries with it the implicit or explicit threat that failure to implement the order could result in the use of other, more aggressive, enforcement mechanisms directed to in-state persons or property.

If an out-of-state person who originates or hosts offending online content from out-of-state infrastructure fails or refuses to filter it, the enforcing state might look to other technologically-based enforcement methods. A state might issue an order to in-state ISPs to block the in-state receipt of offending content [114]. Although such technical mechanisms are far from perfect (as is the case with any border enforcement technology), they may be sufficiently effective to accomplish the purpose of the enforcing state.

Filtering efforts are also initiated in the absence of specific state enforcement activity. Persons create and impose their own filters at point of origin to limit content transfers to states where filtered content might result in liability.[50] Filtering efforts can be conducted collaboratively between private and public sector actors.[51]

3.2.3.8 Orders to in-state persons directing production of data under their control whether held on domestic or foreign IT systems

States may also order state-resident persons to produce data under their control, irrespective of the territorial location of data storage.

Such orders are especially common under court procedural rules that govern disclosure (a.k.a. discovery) of potential evidence by the parties to a dispute. Those who find themselves party to a dispute that is subject to the jurisdiction of a foreign court must quickly become familiar with that court's rules of mandated disclosure. Courts normally do not feel constrained by the location of potential evidence – only that the parties to the dispute disclose it as required according to forum court rules.

More controversial are cases where a state, often in the context of a criminal investigation or intelligence gathering operation, demands the production of data under the control of a state-resident person who is not the target of (criminal) investigation or a party to (civil) legal action. Critics claim that such demands are inappropriate and the state should be limited to submitting requests for international legal assistance (see Section 3.2.3.9). Supporters argue that such demands represent a legitimate exercise of state enforcement jurisdiction against persons present within state territory.

An early example involved a previously secret program in which the United States demanded lawful access to banking transaction records held by SWIFT. The orders to produce data were addressed to US-resident SWIFT offices. Failure to comply with the US demands could have resulted in criminal prosecution of US-resident persons under US law. Complying with these

demands, however, very probably constituted a violation of the data protection law of Belgium (SWIFT's headquarters), among others. News of the programme leaked in 2007 and created a diplomatic dispute between the US and Belgium (among others). This diplomatic issue was eventually resolved through negotiation and agreement concerning the scope of future investigatory operations [115].

Another well-known example involved a request made by an unknown agency of the US government under the Stored Communications Act. The government asked the US court to issue an order to the Microsoft Corporation in the US demanding the production of the contents of an email account maintained by Microsoft on behalf of an unnamed customer who was not resident in the US. The US court issued the order to Microsoft in the US, although the email account itself was maintained on a server in a data centre in Dublin, Ireland. US-resident staff of Microsoft had the technological ability to access the contents of the Dublin server, and the act of producing the requested data would have been technologically trivial. Microsoft asked the court to quash (invalidate) this order, generally on the grounds that the relevant US law did not authorise an order of this type with respect to data stored offshore.

After multiple skirmishes in the District court, the US Court of Appeals (2nd Circuit) eventually quashed the order against Microsoft on the extremely narrow basis that the Stored Communications Act (adopted in 1986) did not expressly and unambiguously claim prescriptive jurisdiction over data stored on equipment located outside the territorial United States [116, 117, 118].[52] This decision was appealed to the US supreme court where it was fully briefed and argued. Following argument but before judgment, the US Congress in 2018 adopted the CLOUD Act. This legislation amended the Stored Communications Act to bring data stored on foreign servers expressly into the prescriptive jurisdiction of that Act, and the US government immediately requested a replacement warrant under the revised law. The supreme court then dismissed the pending appeal without issuing a substantive judgment, as the new law had resolved any dispute about the scope of prescriptive jurisdiction claimed by the US Congress [119].[53]

3.2.3.9 International legal assistance

States can make requests for assistance from persons outside of their territory to gather evidence in support of criminal investigation. Traditionally, such requests are made pursuant to a mutual legal assistance treaty and are transmitted by the authorities of a first state to the designated authority of a second state for consideration and possible action. Such requests can also be made in the absence of a treaty, although the second state retains discretion over how it chooses to respond in the absence of international legal obligation.

The Budapest convention (see Section 3.5.1) imposes a series of requirements upon contracting states to provide mutual legal assistance in the investigation of cybercrime [103]. The Convention also sets a series of requirements concerning preservation of electronic evidence, including metadata.

Formal state-to-state requests for mutual legal assistance have gained a reputation for being heavily bureaucratic and slow [92]. Although there are many examples of successful international cooperation in the investigation of cybercrime, it has been observed that 'the use of formal cooperation mechanisms occurs on a timescale of months, rather than days' [120].

There are some options available to gather cross-border evidence that do not involve seeking permission from the state in which evidence resides. The Budapest convention provides two additional methods. Authorities of a given Convention State A may gather evidence from

publicly available (open sources) of data stored in a given Convention State B without prior notice to or authorisation from State B [103] at article 32a.

Convention State A is also said to be allowed to use a computer in the territory of State A to access data from a closed source in Convention State B if State A 'obtains the lawful and voluntary consent of the person who has the lawful authority to disclose the data' [103] at article 32b. A formal Guidance Note to the Convention cites the example of a criminal suspect detained in State A who provides consent to State A authorities to access their email or documents stored on a server in State B [121].[54]

article 32b has been discussed at length by the Council of Europe's Cybercrime Convention Committee (T-CY). An *ad hoc* subgroup of this Committee set out an extensive discussion of issues arising and specific examples in which use of this authority might be considered [122]. The Committee itself went on to publish a Guidance Note which clarifies the authority granted by article 32b [121].

Practitioners should note that article 32 powers are permissive, not prohibitive. If State A is unable to demonstrate that a proposed evidence gathering activity complies with article 32b this only means that the activity is not expressly authorised by the Budapest convention. article 32 of the Convention would not prohibit the proposed activity, although some other features of public international law might. See Section 3.12.4.

Critics argue that article 32b constitutes an unwelcome intrusion into state sovereignty. This has been cited by some states as a reason for refusing to sign the Budapest convention ([123] at p.19, fn.39).

Another cross-border investigation method in the absence of consent by the second state is described in Section 3.2.3.8.

3.2.4 The problem of data sovereignty

The phrase 'data sovereignty' is sometimes used to struggle with the various jurisdictional demands outlined above. The extremely low technological cost of storing and then retrieving data outside the territory of a state, raises concerns about the number of states that might seek to compel some form of intervention with respect to such data.

Cloud services merely provide 'a *sense* of location independence' rather than actual location independence [124]. The location of a service provider's infrastructure and the location of persons who maintain effective control over that infrastructure are both important for understanding which states might be able to assert enforcement jurisdiction mandating some type of intervention with respect to such data [125].[55]

Users of cloud services have become increasingly aware that locating a data storage facility in any given state increases that state's opportunity to exercise enforcement jurisdiction over such facilities. Practitioners should also consider enforcement jurisdiction opportunities presented to a state when persons within its territory have technical or organisational ability to access or otherwise interfere with data held on infrastructure physically outside that state. (See the discussion in Section 3.2.3.8.) Enforcement risk can arise from the geo-location of data storage equipment, or the geo-location of persons able to access such data.[56]

Some states have responded to potential jurisdictional conflicts by mandating local storage and processing (localisation) for some types of data. Indeed, under its data protection laws the European Union has long imposed an EEA localisation requirement (in the form of a rule

prohibiting export) for personal data although in practice there are multiple mechanisms available to enable exports from the EEA (see Section 3.4.6). Other states outside the EEA have imposed localisation requirements for a variety of reasons [126, 127, 128, 129, 130].

Some states within the EEA have imposed single-state data localisation rules for certain types of sensitive data, prohibiting exports even to fellow member states of the EEA. Possibly in response to this single state localisation trend, the European Union adopted a Regulation in 2018 that prohibits member state legal restrictions on the free movement of *non-personal* data within the Union. (I.e., the Regulation does not prohibit member states from adopting data localisation requirements for personal data.) This Regulation also includes multiple exceptions for member states that wish to impose localisation requirements for reasons of important public policy [131].[57]

3.3 PRIVACY LAWS IN GENERAL AND ELECTRONIC INTERCEPTION

The concept of 'privacy' is both widely cited and challenging to articulate. This section addresses privacy in the sense described in the seminal nineteenth century article, 'The Right to Privacy' [132]. In this context, privacy has been described simply as the right for a person[58] to be free from intrusion by others into personal affairs or the 'right to be left alone'.

In the work of a cyber security practitioner, the issue of privacy most often arises in the context of electronic surveillance and related investigatory activity, which is the focus of this section. This area of law can be expected to continue to evolve quickly in response to new use cases enabled by cloud data processing services.

Data protection law is addressed in Section 3.4 and crimes against information systems are considered in Section 3.5. Most of these areas of law stem from or are related to privacy concepts.

3.3.1 International norms: foundations from international human rights law

Privacy is widely recognised internationally as a human right, although not an absolute right.[59] The right to privacy is conditional – subject to limitations and exceptions.

The 1948 Universal Declaration of Human Rights states at Art 12 that, 'No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence...' [133]. Freedom from interference with privacy extends only to 'arbitrary' interference, which clearly contemplates the legitimacy of 'non-arbitrary' interference. Similar expressions, with similar qualifications, can be found in article 8 of the European Convention on Human Rights and again in article 7 of the Charter of Fundamental Rights of the European Union [134].[60]

In the more narrow context of limiting government authority, the Fourth Amendment of the US Constitution adopted in 1791 states, 'The right of the people to be secure in their persons, houses, papers, and effects, against unreasonable searches and seizures, shall not be violated, and no warrants [authorizing search or seizure] shall issue, but upon probable cause...' [135]. Again, this right is limited to protect only against such actions that are 'unreasonable'.

The application of these principles to intangible data evolved significantly during the twentieth century. In 1928, for example, the US supreme court interpreted the Fourth Amendment narrowly as protecting persons only from physical intrusion into their property [136]. Four decades later, after electronic communication had become a ubiquitous feature of everyday life, the Court changed its position and re-interpreted the Fourth Amendment to protect persons from unwarranted intrusion into electronic communications. The 1967 Court observed that laws like the Fourth Amendment are intended to 'protect people not places' [137]. The privacy right expressed in the European Convention on Human Rights has long been understood to apply to electronic communications [138]. By the early twenty-first century it appears to have become a widely accepted international norm that privacy rights (however they are interpreted) apply to intangible expressions of information as well as physical space [139].

While the principles described above are widely accepted in the international community, the interpretation and implementation of these principles remains subject to significant divergence. Some laws extend a general right of privacy into almost every situation, while others focus solely on limiting the power of the state to intrude into private affairs.[61]

A given person's expectation of privacy may vary by reference to the nature of their relationship with the party who seeks to intrude. For example, there tend to be few restrictions imposed by any state's laws with respect to intrusion by a parent into the affairs of their minor children. By contrast, states vary significantly when considering when it is appropriate for employers to intrude into the affairs of their employees. In the latter context, the UN has published recommended approaches to the application of human rights in a business setting [140].

Expectations of privacy can also vary significantly between different societies. An intrusion viewed by one society as relatively innocuous and to be expected might be viewed by another society as a breach of human rights.

As persons rely on cloud services to manage increasingly intimate aspects of their lives, expectations of privacy over the variety of data processed using these systems will continue to evolve.[62] Policy makers, service providers, and civil society organisations, regularly seek to explain or to adjust expectations of privacy through education and advocacy.

An additional aspect of privacy relates to limits imposed upon the degree of permitted intrusion. In cases of state-warranted lawful interception, for example, warrants may be narrowly drawn to limit interception to named places, specified equipment, specified persons, or specified categories of persons.

Privacy laws often treat metadata differently from content data, usually based on the theory that persons have a lower expectation of privacy in metadata [141].[63] This distinction is increasingly criticised, and policy makers and courts are under pressure to reconsider the nature of metadata given:

- the private quality of some information disclosed by modern metadata such as URLs,[64]
- the incredible growth in the volume and types of metadata available in the age of ubiquitous personal mobile data communications;[65] and
- the growing volume of otherwise-private information that can be inferred from metadata using modern traffic analysis and visualisation techniques.[66]

3.3.2 Interception by a state

State intrusion into electronic communication for purposes of law enforcement or state security is often treated under specialist legal regimes that are highly heterogenous. There is broad agreement in public international law dating to the mid-nineteenth century that each state has the right to intercept or interrupt electronic communications in appropriate circumstances [142]. These principles continue to apply to cyberspace [87, 143].

As electronic communications (especially telephones) became commonplace and interception methods became more cost-effective in the 1960s and 1970s, a trend emerged to move state interception of communications activity away from informal or customary practice onto a more clearly regulated footing [138, 144]. Although legal governance processes and standards adopted to authorise state interception have evolved significantly, these legal processes and standards differ significantly from state to state. Some states require a prior examination of each request for state interception by an independent judicial officer; some delegate this decision-making authority broadly with limited oversight; and others adopt mechanisms that fall anywhere between these extremes.

Although there does not yet appear to be any obvious international harmonisation of legal standards and procedures concerning lawful interception, there are examples of recommended practice for states that wish to place their legal procedures onto a robust and predictable foundation [145].

By contrast, some technical standards for facilitating lawful access (such as the ETSI LI series) have developed successfully on a multilateral basis [146, 147]. These technical standards make it possible for product and service developers to design lawful access technologies to a common multinational standard, while leaving substantive decision-making about their use in the hands of domestic authorities.[67]

Practitioners who work in a police or state security environment must become familiar with the rules that apply to their interception activity. Some state organisations employ large teams of lawyers dedicated solely to assessing the legality of various intelligence-gathering and investigation activities.

Those who work for communication service providers must also become familiar with obligations imposed on them by applicable laws to assist in state interception activity. This can be especially challenging for multinational communication service providers, as they are normally subject to the prescriptive jurisdiction of each state where their service is supplied.[68] Service providers often localise responsibility for compliance with lawful interception by domestic authorities in each state where they supply services.

State regulations concerning lawful interception tend to impose a combination of obligations upon the providers of public communications services, such as:

- procuring and maintaining facilities designed to facilitate lawful interception within the service provider's domain (this obligation may be imposed under telecommunication regulation as a condition of telecommunications licensing, especially for those that operate in-state physical infrastructure such as PSTN operators);
- providing technical assistance in response to lawful interception requests; and
- maintaining the secrecy of the content of lawful interception requests, especially the identity of investigation targets.

Some states impose additional legal obligations to maintain secrecy over the existence, nature, or frequency, of lawful interception requests, the location or operation of interception facilities, etc. Communication service providers that wish to report publicly about the nature and frequency of state interception requests (a.k.a. transparency reports) must be careful to conduct this reporting in compliance with applicable law.[69]

As easy-to-use cryptographic technologies have become ubiquitous, and larger volumes of message traffic are transmitted as ciphertext, states conducting lawful access activity face increasing difficulty obtaining access to plaintext messages [144]. States have attempted to recover plaintext by using a variety of creative legal mechanisms including warrants for the physical search and seizure of end point devices and requests for technical assistance from device manufacturers or third-party analysts. These procedures are of variable effectiveness and remain subject to much debate [144]. Efforts to compel an end user to decrypt ciphertext or to disclose relevant passwords or keys also face a variety of legal challenges [148, 149].[70] Some states have adopted laws that specifically address compelled disclosure of plaintext or keys that enable decipherment.[71]

The emergence of virtual communication service providers (i.e., those that provide communication services via third-party infrastructure – or 'over the top' service providers) have created challenges for both states and service providers. These service providers remain subject to the jurisdiction of states in which their service is supplied, as states show a clear sovereign interest in services provided to persons within their territory.[72] States have, however, taken different approaches when choosing how and when to exercise jurisdiction over these providers. Enforcement actions by states against such persons have included orders to facilitate in-territory lawful interception at the risk of a variety of sanctions including: prohibiting the service provider from entering into business relationships with in-state residents, or ordering third-party state-resident service providers to block or filter such services at the PSTN or IP layer, thus making it inaccessible to (many or most) in-state residents. Changes in enforcement practices are likely as this subject continues to develop.

3.3.3 Interception by persons other than states

Laws concerning interception activity by non-state actors are also highly heterogenous.

Persons that provide public telecommunications services are often specifically restricted from intercepting communications that transit their own public service networks [117, 150]. This might be framed legally as a restriction imposed only on providers of these public services, or a more general restriction limiting the ability of any person to intercept communications on public networks.

In many cases, efforts to intercept communications while transiting a third-party network will also constitute a crime under computer anti-intrusion laws. This was a significant motivating factor in the adoption of these laws (see Section 3.5).

The interception of communications by a person during the course of transmission over its own non-public network, such as interception on a router, bridge or IMAP server operated by that person on their own LAN for purposes other than providing a public communications service, presents other challenges to analysis. This type of interception activity would not normally expect to fall foul of traditional computer crime legislation, as the relevant person is normally authorised to gain entry to the relevant computer (see Section 3.5). It might, however, be regulated generally within the same legal framework used to govern the interception of

communications, although interception by an owner/controller on their own system is often treated more liberally [150]. Finally, in-house interception activity may also be limited by the terms of general privacy statutes or data protection laws (see Section 3.4).

3.3.4 Enforcement of privacy laws – penalties for violation

Enforcing a legal right of privacy brings a number of challenges. From an evidentiary perspective, a person whose privacy rights have been violated might never learn that a violation has occurred. Some legal rules serve, among other things, to redress this knowledge imbalance. These include breach notification requirements which reveal inappropriate disclosures of personal data to the effected person (see Section 3.4.7), criminal procedure rules that require the disclosure of prosecutorial evidence to the accused which in turn reveals intrusive investigatory techniques,[73] and civil procedure rules which require similar disclosures in civil legal actions (e.g., employment disputes).

Remedies available to persons whose privacy rights have been violated might include the ability to bring a tort action against the violator claiming monetary compensation (see Section 3.7.4). These individual tort remedies are a regular feature of data protection laws as well as various US privacy laws. The US criminal courts also employ an exclusionary rule prohibiting the introduction of evidence gathered in violation of the US Constitutional privacy rights of the accused [151].[74]

Finally, some violations of privacy – especially unwarranted interception of communications during the course of transmission on a public network or unauthorised intrusions into data at rest – are defined as and may be prosecuted as crimes [152].

3.4 DATA PROTECTION

[90, 91, 153, 154]

Data protection law developed from a foundation of general privacy law. This generalisation can be a bit misleading, however, as data protection law has evolved to address a number of related issues that arise from modern data processing techniques that might not traditionally have been defined as 'privacy'.

Data protection is of significant interest to cyber security practitioners, as it includes numerous obligations related to data security. This section will focus primarily on issues that recur in a security-related context. Data protection law is not, however, a generalised system of regulations that address every aspect of cyber security. The focus remains on specific principles adopted to support individual rights in a data processing context.

Data protection law has developed primarily from European legislative initiatives. European Union law has been tremendously influential around the world through various mechanisms, including states seeking 'adequacy determinations' from the European Union, which enable exports of personal data, and private law contract requirements imposed upon non-EU resident data processors [155]. This international impact continues to grow as the EU now expressly claims prescriptive jurisdiction over personal data processing activity anywhere in the world that relates to data subjects present in the EU (see discussion in Section 3.2.2.3).

The foundational laws that define data protection obligations in the EU are Regulation 2016/679 – GDPR (EU-wide regulation applicable to most persons) and Directive 2016/680 (obligations

to be imposed by member states in domestic law in the context of investigation or prosecution of crime by the state) [105, 156].[75] This section primarily addresses obligations imposed by GDPR. Practitioners engaged by a state in conduct related to investigation or prosecution of crime must be aware of the modified obligations that apply to that activity described by Directive 2016/680 as transposed into member state law [157, 158].

3.4.1 Subject matter and regulatory focus

The overriding purpose of EU data protection law is to protect the interests of *data subjects* (GDPR at article 1; Recital 1, 2, 4, 75, 78, et al.).[76] Data protection law accomplishes this by regulating acts of *controllers* and *processors* when *processing* data that incorporates *personal data*. Any such processing activity activates the application of data protection law. Each of these terms is considered in this section.

3.4.1.1 Data subject, personal data (and PII)

In data protection law, the terms 'personal data' and 'data subject' are defined concurrently:

> *personal data* means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person (GDPR, Art 4(1))

Only natural persons, not legal persons, are data subjects. GDPR does not apply to personal data of deceased natural persons, although member states may individually adopt such protections if they wish (GDPR at Recital 27).

Because the definition of data subject extends to persons who are *identified or identifiable*, data can incorporate personal data even when the data include no obvious information identifying a data subject. It is sufficient that a data subject is capable of being identified, by anyone, through analysing the data or by applying additional information known to any person - even if this additional information is unknown and inaccessible to the person controlling or processing data. Pseudonymised data remains personal data (GDPR at Recital 26).

The Court of Justice of the European Union has held that a server log with IP address numbers incorporates personal data, as it remains possible for third parties (telecommunications service providers) to match static or dynamic IP numbers to individual customer premises and from there to a living person. This made some server log entries 'related to' a data subject [159]. The fact that the holder of the server logs did not have access to the IP number allocation or customer identification data was irrelevant.

As de-anonymisation and similar analysis techniques increase the capability to identify living persons from data that has no obvious personal identifiers, it becomes increasingly difficult to maintain data sets that are truly devoid of personal data [160, 161].[77]

The term 'personal data' is often confused in practice with 'personally identifiable information' (PII). This confusion arises because of the ubiquity of the term 'PII' in cyber security as well as significant variance in its definition.

Definitions and detailed discussions of PII are found in Section 4.4 of ISO/IEC 29100:2011, and Section 2.1 of NIST SP-800-122 [162, 163]. Although it is arguable whether the ISO and NIST definitions of PII are contiguous with the legal definition of personal data, both technical standards clearly conclude that data containing no obvious personal identifiers may nonetheless constitute PII.

Complicating matters further, the phrase 'personally identifiable information' is used in a variety of US federal statutes and regulations, either without statutory definition, or with definitions specifically addressed to individual use cases [164].[78] In this specific context, some US courts have interpreted this phrase narrowly to include only obvious personal identifiers. Thus some US courts have held that data such as MAC codes and IP numbers do not fall within the meaning of 'personally identifiable information' as that phrase is used in some US statutes [165, 166, 167].[79] As explained above, these same identifiers often constitute 'personal data' as that term is defined in European law.

Irrespective of how one defines PII, European data protection law contains a clear and broad definition of 'personal data'. It is this definition of personal data, not PII, that triggers the application of European data protection law.[80]

3.4.1.2 Processing

In data protection law, the term *processing* is defined as:

> any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means, such as collection, recording, organisation, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, restriction, erasure or destruction (GDPR, Art 4(2))

Processing therefore incorporates almost any action one can imagine taking with respect to personal data.

3.4.1.3 Controller and processor

In data protection law, the term *controller* is defined as:

> the natural or legal person, public authority, agency or other body which, alone or jointly with others, determines the purposes and means of the processing of personal data; where the purposes and means of such processing are determined by Union or Member State law, the controller or the specific criteria for its nomination may be provided for by Union or Member State law (GDPR, Art 4(7))

In data protection law, the term *processor* is defined as:

> a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller (GDPR, Art 4(8))

These definitions make clear the relationship between controller and processor. A controller decides; a processor executes. In the history of data protection law, many policy makers

originally believed that the most effective way to protect individual rights was to focus regulation on persons who operated and maintained computer equipment – processors. The focus was on the machine. As the PC revolution changed our social relationship with computers, however, policy makers began to appreciate that the focus should be turned to persons in a position to command and control how the machines were used – controllers.

As between these two persons, Directive 95/46 tended to place the heaviest regulatory burden on controllers. Processors were advised that their obligation consisted primarily of following directions provided by controllers. There are many valid reasons for placing primary compliance responsibility on data controllers, especially because they are most often able to communicate and manage relationships with the relevant data subjects.

This regulatory distinction started to break down as cloud services became ubiquitous – especially SaaS. A typical SaaS provider might spend an enormous amount of time and effort designing their system and user interfaces, and then present the operational characteristics of that system to controller-customers in a service level agreement on a 'take it or leave it' basis. As a technical matter, the SaaS provider might be keen to demonstrate that they are acting only in the capacity of a processor and that their customers are acting as controllers – shifting the burden of assessing compliance to individual controllers. In the revisions to data protection law embodied in GDPR, policy makers have responded by generally increasing the regulatory responsibility of processors. Compliance responsibility under GDPR is now more evenly shared by controllers and processors, although their responsibilities depend upon their respective area of competence.

3.4.2 Core regulatory principles

Data protection law is built on a foundation of regulatory principles governing processing of personal data outlined in GDPR article 5, being:

- lawfulness, fairness and transparency;
- purpose limitation;
- data minimisation;
- accuracy;
- storage limitation;
- integrity and confidentiality.

These core principles are well rehearsed and there are many published commentaries and guidelines available in forms accessible to practitioners to aid understanding [153, 154, 168, 169].

Practitioners should be especially alert to the presence of certain types of sensitive personal data in any system with which they are involved. Such data includes, 'personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation' (GDPR, Art 9). Sensitive personal data triggers a series of additional protections and generally increased levels of regulatory scrutiny, as improper use of such data often presents a disproportional risk to the interests of the data subject.

The topic of 'consent' in data protection law is worth a brief comment, as it remains a subject of some confusion. As a threshold matter, data subject consent is not always required when processing personal data. There may be multiple lawful grounds for processing personal data other than consent depending upon context. If data subject consent is required, however, data protection law sets a very high bar that this must be 'freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her' (GDPR, Art 4(11)). A series of conditions that apply to consent are set out in GDPR, Art 7 (and Art 8 relating to children's consent).[81]

3.4.3 Investigation and prevention of crime, and similar activities

Practitioners engaged by a state benefit from certain reductions in data protection obligations when processing personal data related to criminal investigation and prosecution. These reduced obligations are described in general in Directive 2016/680 and then transposed into member state law.

Practitioners who conduct activities with similar goals, but are not engaged by a state, remain subject to GDPR. In this context, however, GDPR makes it clear that purposes such as fraud prevention constitute a legitimate interest of data controllers (GDPR at Recital 47). GDPR also provides member states with the option to adopt in their domestic laws reduced data protection obligations for non-state actors when conducting activities designed to prevent, investigate, detect, or prosecute crime, etc. (GDPR, Art 23; [170] at s.15 & Sched 2).

3.4.4 Appropriate security measures

Data protection law imposes an obligation on controllers and processors to 'implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk' associated with processing personal data (GDPR, Art 32(1)). This security principle is a long-standing feature of data protection law.

The obligation clearly encompasses both technical measures as well as human management and oversight (i.e., 'organisational measures'). Compliance requires that both components are appropriate. Compliance requires a consideration of the state of the art and an assessment of costs of various measures in comparison with risks presented. Assessing this obligation to take appropriate security measures might therefore be aided by analogy with the law of negligence which presents various frameworks used to assess 'reasonable' care (see discussion in Section 3.7.1.2).

GDPR has expanded significantly the discussion of security measures to provide examples of measures that might assist in creating appropriate security. This includes many past practices that developed organically such as pseudonymisation and encryption of personal data, assuring ongoing confidentiality, integrity, availability and resilience of systems, and robust incident recovery plans. To be clear, GDPR does not expressly mandate encryption of all personal data. It simply highlights encryption as a technical measure that can be adopted to enhance security. As encryption methods or other security technologies become standardised and costs fall, however, it becomes increasingly difficult to justify why such technologies are not adopted.

Organisational methods used to protect the security of personal data may include contract obligations with supply chain partners and others. (See also the discussions in Sec-

tions 3.4.6.2 and 3.6.2)

Although security certification or compliance with security codes of practice might help to prove appropriateness of security measures, these certifications are not dispositive of compliance with the law (GDPR, Art 32(3)).

3.4.5 Assessment and design of processing systems

Sometimes, the most effective way to prevent violations of data protection law is to design a system that minimises the ability of persons to take inappropriate action. GDPR, therefore, has adopted an obligation to implement data protection strategies by design, and by default. As with the general security principle, this obligation extends to both technological and organisation measures and is assessed on a risk balancing basis. This obligation arises at the planning phase, before processing commences, as controllers are required to consider this issue 'at the time of determining the means of processing' (GDPR, Art 25).

If a new personal data processing activity presents significant risk of harm to data subjects, especially in the context of developing or migrating to systems that process large volumes of data, the controller is required to undertake a data protection impact assessment (GDPR, Art 35, Recital 91, et al.). If the assessment reveals significant risks, the controller is further required to consult with the relevant supervisory authority about the proposed processing activity (GDPR, Art 36).

3.4.6 International data transfer

European data protection law imposes a general prohibition on the transfer of personal data to any state outside the European Economic Area or to any international governmental organisation (GDPR, Art 44). Such transfers remain commonplace, however, when enabled by an appropriate export compliance mechanism.

3.4.6.1 Adequacy determinations and Privacy Shield

Transfers of personal data can be made to territories in accordance with an adequacy decision: a finding by the European Commission that the receiving territory (or IGO) has established adequate legal protections concerning personal data (GDPR, Art 45). The process of obtaining an adequacy decision is instigated at the request of the proposed receiving state and often requires years of technical evaluation and diplomatic negotiation [155].

Adequacy determinations fall into two categories: decisions that a receiving territory's laws are generally adequate to protect personal data, and decisions that a receiving territory's laws are adequate provided that special conditions are met. Decisions concerning Canada and the United States both fall into the second category. In the case of Canada, adequacy is only assured with respect to transfers to the commercial for-profit sector, as the relevant Canadian laws do not apply to processing by governments or charities.

The US adequacy determination has a difficult history. The US has nothing like the EU's generalised legal protections concerning processing personal data. To enable transfers of data, the US and the EU have negotiated specific agreements to support an adequacy finding. This agreement enables most US businesses, if they wish, to opt in to a regulatory system that provides adequacy. This regulatory system is then enforced by agencies of the US state against opted-in US businesses. The original system, Safe Harbour, was invalidated by the

European Court of Justice in October 2015 in the Schrems case [171]. It was quickly replaced by the EU-US Privacy Shield regime in 2016, which operates in a fashion similar to Safe Harbour with enhanced protections for data subjects.

3.4.6.2 Transfers subject to safeguards

Transfers are also allowed when appropriate safeguards are put into place (GDPR, Art 46). The most common safeguards normally encountered are binding corporate rules, and approved data protection clauses in contracts between exporters and importers.

Binding corporate rules are governance procedures normally adopted by multinational enterprises in an effort to demonstrate to data protection authorities that they will comply with data protection principles (GDPR, Art 47). To be effective for data transfer compliance, such rules must be approved by relevant public authorities. This can take years to negotiate. While such rules were originally developed as a tool to enable sharing of personal data among the members of a multinational data controller enterprise that operates both inside and outside the EEA, they have more recently been adopted by non-resident cloud service providers as a compliance tool to facilitate business from customers in the EEA. Practitioners may be called upon to assist in drafting or negotiating binding corporate rules, as they have a significant impact on IT services, security architectures and governance procedures.

Approved contract clauses are simply contract obligations between a data exporter and importer that serve to protect the interests of data subjects. They can be either standard clauses approved for use by the Commission, or special clauses submitted to the relevant authorities for prior approval (GDPR, Art 46(2)(c)-(d) & 46(3)(a)). Although the Commission-approved clauses are standardised, to be effective the parties to the relevant contract are required to incorporate a significant amount of operational detail about the nature of the personal data to be transferred, the purposes of the data processing to be undertaken, etc.

3.4.6.3 Transfers pursuant to international mutual legal assistance treaty

Transfers of personal data that are otherwise prohibited by GDPR can be made in circumstances such as requests for assistance by a foreign state police agency pursuant to the terms of a mutual legal assistance treaty (GDPR, Art 48). (See also Section 3.2.3.9.) Such transfers are addressed specifically in Directive 2016/680, GDPR, Art 35-40.

3.4.6.4 Derogations allowing transfers

In the absence of any other mechanism allowing a transfer, exports from the EEA are still allowed under certain limited circumstances such as:

- the data subject provides knowing informed express consent to the transfer;
- the transfer is necessary in order to perform a contract with the data subject, or a contract with a third party adopted in the interests of the data subject;
- the transfer serves an important public interest;
- the transfer is connected to the pursuit or defence of a legal claim; or
- the transfer is necessary to protect the life or welfare of the data subject, who is physically unable to consent.

These derogations (GDPR, Art 49) are meant to be interpreted narrowly, and the European Data Protection Board has issued guidance on the interpretation and application of these measures [172].

3.4.7 Personal data breach notification

Laws mandating the notification of personal data breaches to data subjects[82] began to emerge in both the EU and the US around the turn of the twenty-first century [173, 174]. In a pattern that is curiously the reverse of the development of data protection laws generally, EU notification requirements arose first in narrowly defined subject matter areas while US states (beginning with California) imposed a more general duty to notify effected persons of personal data breaches.[83]

GDPR marked the emergence in Europe of a general duty placed on processors and controllers of personal data to make certain notifications following a 'personal data breach', which is defined as 'a breach of security leading to the accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed' (GDPR, Art 4(12)). Thus, events as diverse as personal data exfiltration, the unauthorised modification of personal data and ransomware can all constitute personal data breaches.

A processor is first required to notify the circumstances of a breach to the relevant controller 'without undue delay'. The controller is then required to notify the relevant supervisory authority of the breach 'without undue delay and, where feasible, not later than 72 hours after having become aware of it' (GDPR, Art 33(1)-(2)). The content of the notice is set out in Art 33(3). There is a limited exception to the controller's duty to notify a supervisory authority if the breach is 'unlikely to result in a risk to the rights and freedoms of natural persons'. Whether or not notified to the supervisory authority, the controller is required to document all such breach events and these records are subject to periodic review by the supervisory authority.

If such a breach is 'likely to result in a high risk to the rights and freedoms of natural persons', then the controller is required to communicate the circumstances of the breach to the relevant data subjects without undue delay (GDPR, Art 34(1)-(2)). Communication to the data subjects can be avoided if the controller has implemented methods that limit the harm that might be caused by such a breach, such as encrypting data that was then exfiltrated as ciphertext. While such ciphertext remains personal data for legal purposes, the encrypted state of the data reduces the potential harm to data subject to some degree (depending upon the type of encryption, etc.) This ability to avoid communication to data subjects when harm is unlikely is a useful feature of GDPR. Many US state notification laws originally demanded notifying data subjects irrespective of the relevant risks presented by the breach.[84] Supervisory authorities retain the right to compel communication to data subjects about the breach if they disagree with the controller's risk assessment.

Various states around the world continue to adopt mandatory breach disclosure laws, each with their own unique characteristics [175].

3.4.8 Enforcement and penalties

Egregious violations of data protection law can be prosecuted as crimes under member state domestic law. Relevant actions can be prosecuted simultaneously as crimes against information systems (see Section 3.5) [176].

Data protection laws also enable data subjects to bring tort claims for violation of data protection rights. Such claims implicate the risk of vicarious liability for employee misdeeds, especially if a large group of data subjects are able to bring a claim as a group or class. (See the discussion of the *Morrisons* case at Section 3.7.5.1)

Public enforcement authorities are also given powers to serve enforcement notices, demanding changes in processing behaviour to achieve compliance with the law (GDPR, Art 58.) In particularly egregious cases, public authorities might serve a notice prohibiting large categories of processing activity. Breaching such an enforcement notice is an independent cause for more severe enforcement action.

Perhaps the greatest change to legal risk presented by EU data protection law in the past few decades has been the steady and accelerating increase in the size of penalties assessed by public authorities. (Adopting the terminology of Section 3.1.5, this has dramatically increased the Q term over time.)

Historically, civil or administrative fines imposed by public authorities for violation of data protection law were perceived in some member states as relatively minor. The disparity in approach among member states to data protection law was a motivating factor for the adoption of the original 1995 Directive, which tended to increase data protection rights in most member states. Following the 1995 Directive, increasingly larger fines started to emerge as state authorities began to increase enforcement pressure. By the time GDPR was adopted in 2016, administrative fines in the region of €500,000 were not uncommon for significant violations of the law.

One of the most-discussed features of GDPR concerns the authority granted to impose large administrative fines (GDPR, Art 83). Violations of some of the more procedural or operational requirements of GDPR, including the requirement to adopt appropriate security measures, can incur administrative fines of up to €10,000,000, or 2% of an undertaking's annual worldwide turnover, whichever is greater. Violations of more fundamental principles of GDPR, such as failure to respect the rights of data subjects, processing personal data without lawful authority, or exporting data in violation of the law, can incur administrative fines of up to €20,000,000, or 4% of an undertaking's annual worldwide turnover, whichever is greater. Authorities are instructed to calculate fines at a level to make them 'effective, proportionate and dissuasive' in individual circumstances. GDPR lists a number of both mitigating and aggravating factors for consideration when setting these fines that are worth closer study (GDPR, Art 83(2)).

The emergence in GDPR of the potential for 'eight figure' and 'nine figure' fines, together with the increased scope of territorial jurisdiction, instantly promoted data protection law into the category of a significant risk to be assessed and managed at senior leadership levels – a position that this law had rarely occupied prior to these changes. Some persons who provide online information services from outside the EU (who presumably fear that their business models are not compatible with GDPR compliance) responded by withdrawing from the European market by using geographic filtering mechanisms (see Section 3.2.3.7). Other offshore service providers have embraced the change and worked to comply with the rules (presumably as they value their ongoing contact with the European market).

In July 2019, the Information Commissioner's Office of the United Kingdom issued two notices of their intention to issue large fines under GDPR: a proposed fine of GB£183.39 million to British Airways[85] and a proposed fine of GB£99.2 million to Marriott International, Inc.[86] At time of writing, both companies have expressed their intention to contest the fines.

3.5 COMPUTER CRIME

[92, 93, 94]

The term 'cybercrime' is often used to identify three different categories of criminal activity: crimes in which cyberspace infrastructure is merely an instrumentality of some other traditional crime (e.g., financial fraud), distribution of criminal content (e.g., pornography and hate speech), and crimes directed against cyberspace infrastructure itself (e.g., unlawful intrusion into a computer system).

This section is addressed solely to the last category, computer crimes or crimes against information systems. These tend to be of concern as they are of interest to those who work for state enforcement authorities, as well as those who manage cyber security risk, research cyber security technologies, and develop cyber security products and services.

Although some practitioners are engaged by states in the investigation and prosecution of crimes where cyberspace is an instrumentality of crime, it is difficult to draw out generalisable statements about those crimes that remain useful in a multinational context. Crimes based on message content are especially problematic, as these rest upon widely diverging opinion from different societies about what constitutes 'illegitimate' content worthy of criminal prosecution.[87] (One area in which there appears to be growing international consensus for criminalising message content concerns child exploitation materials [93, 103, 177]. Even with this subject matter, where high level normative principles may be quickly agreed, attempting to translate these principles into widely agreed legal standards remains challenging [94].)

3.5.1 Crimes against information systems

In the 1980s and 1990s, many states confronted the problem that an emerging set of anti-social behaviours related to cyberspace infrastructure were not clearly identified as crimes.[88]

The UK Parliament responded by adopting the Computer Misuse Act 1990, which defined a series of computer-related criminal offences. This law has been subsequently amended from time to time [178].

In 1984, the US Congress adopted the Computer Fraud and Abuse Act, which has also been regularly amended [179, 180].[89] Many US states have additionally adopted their own statutes to prosecute computer crime.[90] The US landscape is especially complex, as a variety of federal and state law enforcement agencies have varying subject matter jurisdiction over computer crimes [93].

Similar laws have been adopted by many, but not all, states around the world. The Council of Europe Convention on Cybercrime (a.k.a. the Budapest convention) is a multilateral treaty which has had a significant impact on harmonising both computer crime laws and rules of state international assistance [103]. The Convention opened for signature in 2001, and as of July 2019 had been ratified by 44 member states of the Council of Europe and 19 non-European states including Canada, Japan and the US [181].

In 2013, the European Union adopted Directive 2013/40. This mandates that member states modify their criminal laws to address commonly recognised computer crimes which the Directive describes as crimes 'against information systems' [104].

This introductory section on crimes against information systems is influenced by the taxonomy adopted by the Budapest convention and further reflected in Directive 2013/40. Although these two international legal instruments are cited repeatedly, practitioners should keep in mind that they are instruments of public international law and relevant crimes are defined by, and prosecuted under, the domestic law of individual states.[91]

3.5.1.1 Improper access to a system

Improper system access laws criminalise the act of accessing a computer system (in whole or in part) without the right to do so, colloquially known as hacking.[92] (Budapest convention at Art. 2; Directive 2013/40 at Art 3.) The UK Computer Misuse Act 1990 at s.1, for example, defines as criminal an action by a person which causes a computer to perform an act with the intent to secure unauthorised access to any program or data [178]. Thus, the mere act of entering a password into a system without authorisation in an effort to access that system constitutes a crime under the UK statute whether or not access is successfully achieved.

Some debate persists concerning how to distinguish rightful from wrongful action in cases where an otherwise-authorised person exceeds the scope of permission granted to them. Critics argue that an overly-broad interpretation of statutory terms like 'unauthorised access' can produce criminal prosecution based only on breaching an acceptable use policy or website terms and conditions. This might serve to re-define as a crime what otherwise could be a civil breach of contract claim [182]. The issue remains open to argument in some circumstances [94, 183, 184, 185].

3.5.1.2 Improper interference with data

Improper system interference with data laws criminalise the act of inappropriately 'deleting, damaging, deteriorating, altering or suppressing' data. (Budapest convention at Art. 4; Directive 2013/40 at Art 5.) These laws can be used to prosecute actions such as release or installation of malware, including ransomware.

3.5.1.3 Improper interference with systems

Early computer crime laws tended to focus on the act of intrusion into a computer system, or improperly modifying the contents of those systems. With the emergence of DoS and DDoS attacks, some of these early criminal laws were found to be inadequate to address this new threatening behaviour.

These laws now more commonly include a prohibition against acts that cause a material degradation in the performance of an information system. (Budapest convention at Art. 5; Directive 2013/40 at Art 4; Computer Misuse Act 1990 at s.3, as amended in 2007-08.)

3.5.1.4 Improper interception of communication

Often as a corollary to various rights of privacy, many legal systems define the act of wrongfully intercepting electronic communications as a crime. (Budapest convention at Art. 3; Directive 2013/40 at Art 6.) The rules and penalties tend to be most restrictive in the context of intercepting communications during the course of their conveyance on public networks. This subject is discussed in Section 3.3.

3.5.1.5 Producing hacking tools with improper intentions

Many states also define as crimes the production or distribution of tools with the intention that they are used to facilitate other crimes against information systems. (Budapest convention at Art. 6; Directive 2013/40, Art 7; Computer Misuse Act 1990, s.3A.) These laws can create challenges for those who produce or distribute security testing tools, as discussed in Section 3.5.5.

3.5.2 *De minimis* exceptions to crimes against information systems

Some laws may limit the definition of computer crime to acts which are somehow significant. Directive 2013/40, for example, only mandates that member states criminalise acts against systems 'which are not minor' (Art 3-7). The concept of a 'minor' act against a system is discussed in Recital 11 to the Directive, which suggests that states might define this by reference to the relative insignificance of any risk created or damage caused by the given act [104].

This type of *de minimis* exception to the definition of computer crime is far from universal. EU member states remain free to criminalise such *de minimis* acts. At the time of writing, the UK legislation contains no such *de minimis* exception.[93]

The very idea of a *de minimis* exception to crimes against information systems raises a recurring debate over the nature of the harm that these types of laws seek to redress. It is not always clear how to assess the relative damage or risk caused by any given act against information systems. For some criminal acts such as remote intrusion into a chemical plant industrial control system the risk presented or harm caused is clear to see, as the attack is concentrated against a single and volatile target. In others, such as controlling the actions of a multinational botnet comprising tens of thousands of suborned machines, the risk created or harm caused may be widely diffused among the bots and more difficult to quantify.[94]

3.5.3 The enforcement of and penalties for crimes against information systems

States normally have absolute discretion to decide whether or not to investigate alleged crimes. Having investigated, states normally have absolute discretion regarding the decision to prosecute a criminal matter.[95] Some states have set out guidance to explain how this discretion is exercised [186].

Penalties for committing a crime against information systems vary widely. In criminal cases custodial sentences are often bounded in law by a maximum, and occasionally by a minimum, length of term. Within these policy-imposed limits judges are usually given a wide degree of discretion to decide an appropriate sentence.

Under the UK Computer Misuse Act, for example, a custodial sentence for the crime of improper system access is normally limited to a maximum of two years, while the crime of interfering with data or system integrity is normally limited to a maximum of five years. Prosecution and sentencing history both suggest that actual sentences issued under the UK legislation for these crimes are rarely, if ever, this severe. By contrast, in the US, both federal and state laws have consistently provided for longer maximum custodial sentences of 20 years or more for unlawful intrusion or unlawful interference with data.[96]

The question of appropriate punishment for crimes against information systems remains the subject of review and debate. The emergence of the Internet of Things arguably increases the risk that these crimes might pose to life and property.[97] EU Directive 2013/40, for example, requires that member states provide for the possibility of longer custodial sentences when attacks are directed against critical national infrastructure or when they actually cause significant damage (Art 9(b)-(c)). The UK amended its Computer Misuse Act in 2015 (s.3ZA) to increase the maximum available custodial sentence if criminals are proven to have created significant risk or caused serious damage. Such a person could now be subjected to a maximum custodial sentence of 14 years. In cases where the criminal act causes (or creates significant risk of) serious damage to human welfare or national security, the maximum custodial sentence under UK law increases to life imprisonment (s.3ZA(7)).

Arguments continue over appropriate punishments for crimes against information systems. This debate is complicated by difficulties in understanding or quantifying the degree of risk or the degree of harm caused by these criminal acts. (See the discussion in Section 3.5.2.)

3.5.4 Warranted state activity

When actions related to investigation of crime or in defence of state security are conducted with state authorisation such as a warrant, the person using the warranted technique is often expressly exempted from that state's criminal liability for intrusion into information systems to the extent that the intrusion conforms with expressly warranted activity.

An example can be found in the UK's Investigatory Powers Act 2016, which holds that certain activity conducted with lawful authority under the terms of that Act are 'lawful for all other purposes' [150] in ss.6(2)-(3), 81(1), 99(11), 176(9), 252(8). In other words, actions in compliance with a warrant issued pursuant to the 2016 legislation will not constitute a crime against information systems under the Computer Misuse Act 1990 etc.[98]

State-sponsored acts of remote investigation into cyberspace infrastructure located in foreign states are considered in Section 3.12.4.

3.5.5 Research and development activities conducted by non-state persons

Those who research cyber security issues and develop security products and services outside of the domain of state-sponsored activity can face difficulties if their planned activities constitute a crime against information systems. Examples that may lead to difficulties include:

- uninvited remote analysis of security methods employed on third-party servers or security certificate infrastructures;
- uninvited remote analysis of third-party WiFi equipment;

- uninvited analysis of third-party LAN infrastructure;
- invited stress testing of live WAN environments, to the extent that this degrades performance of infrastructure operated by third parties who are unaware of the testing plan;
- analysing malware and testing anti-malware methods;
- analysing botnet components and performance;
- producing or distributing security testing tools; and
- various covert intelligence-gathering techniques.

With respect to testing tools specifically, the law tends to criminalise production or distribution only when the state can prove an intent to facilitate other violations of the law. This criminal act may have less to do with the operational characteristics of the testing tool than the subjective intention of the person who is producing or distributing it.[99]

In some states, researchers might be able to demonstrate a lack of criminal responsibility for these acts under some type of *de minimis* exception, if one is available (see the discussion in Section 3.5.2).[100]

Some may rest on the belief that 'legitimate' researchers will be saved from criminal liability as a result of state discretion to refrain from investigating or prosecuting *de mimimis* criminal acts, judicial or jury intervention to find accused parties not guilty, or if found guilty, through the imposition of only a token punishment. This situation is rather unsatisfactory for practitioners who attempt to assess potential criminal liability arising from an otherwise carefully risk-managed research or development effort.[101]

Even if practitioners find appropriate exceptions under relevant laws concerning crimes against information systems, they must also be careful to consider whether their actions would constitute crimes under other laws such as generalised privacy or data protection laws.

3.5.6 Self-help disfavoured: software locks and hack-back

'Self-help' refers to the practice of attempting to enforce legal rights without recourse to state authority. A routinely cited example is the re-possession of movable property by a secured lender from a borrower in default of payment of obligations. (For example, repossessing an automobile.)

Public policy is generally suspicious of self-help mechanisms, as they involve non-state actors exercising powers normally considered to be the exclusive province of the state. Laws that enable such actions often impose multiple conditions that limit the actor. In the context of cyber security, practitioners have occasionally designed or adopted methods that might be classified as self-help.

These actions come with the risk of potentially violating criminal law. Persons pursuing these strategies should also remain aware of potential tort liability (see Section 3.7).

3.5.6.1 Undisclosed software locks

Various technologies serve to limit the use of software. Implementing a system that clearly discloses to a user that operation requires the prior entry of a unique activation key is normally non-contentious, and is actively encouraged by certain aspects of copyright law (see Section 3.8.2.1). Similarly, SaaS providers usually do not face any sanctions when suspending access to a customer who terminates the service relationship or fails to pay their service fees.[102]

Problems arise when a supplier (for whatever reason, including non-payment of promised license or maintenance fees) installs a lock mechanism into a software product after the fact without customer agreement. Also problematic are instances where software sold as a product contains an undisclosed time-lock device which later suspends functionality (in the event of non-payment or otherwise). These types of undisclosed or *post-facto* interventions have a history of being prosecuted as crimes against information systems and are otherwise criticised as being against public policy, whether or not the vendor in question held a legitimate right of action against the end user for non-payment of licence fees [187, 188].

3.5.6.2 Hack-back

Hack-back is a term used to describe some form of counter-attack launched against cyberspace infrastructure from which an attack appears to have originated. This strategy is often considered in the context of an attack which appears to originate from a foreign state, and cooperation from the foreign state is deemed unlikely or untimely. Hack-back actions might consist of a DoS attack, efforts to intrude into and disable the originating infrastructure, etc.

Hack-back activity, on its face, falls squarely within the definition of crimes against information systems [180]. Such an action might be prosecuted as a crime by the state where the person conducting the hack-back is located, the states where the machines used to conduct the hack-back are located, or the state in which the hack-back target is located. In addition to the risk of criminal prosecution, a hack-back (if sufficiently aggressive) could serve as the basis under international law for the state of the hack-back target to take sovereign counter-measures against the person conducting the hack-back or against other infrastructure used to conduct the hack-back operation – even if the hack-back itself is not directly attributable to host state (see Section 3.12).

Many have debated adopting exceptions in law specifically to enable hack-back by non-state actors [180, 189, 190, 191].[103] These types of proposed exceptions have not yet found favour with law makers.

3.6 CONTRACT

[90, 91]

The term ‘contract’ describes a (notionally) volitional legal relationship between two or more persons. One extremely broad definition of contract is simply, ‘a promise that the law will enforce’ [192].

Unfortunately, the word ‘contract’ is often used colloquially to describe a communication that embodies and expresses contractual promises (e.g., a piece of paper, email, or fax). This confusion should be avoided. A contract is a legal relationship, not a piece of paper. In some

circumstances, applicable law may exceptionally impose a requirement that some contract obligations must be embodied in a specified form (see Section 3.10).

This section will discuss a few contract topics of recurring interest to cyber security practitioners.

3.6.1 Online contracts: time of contract and receipt of contractual communication

The definition of 'contract' above immediately begs a follow-up question: how does one distinguish a legally enforceable promise from other types of communication? Although different legal systems have varying approaches to defining a contract, the elements required by law can be classified into two categories: sufficiency of communication, and indicia of enforceability.

As an example, under the law of England a contract usually exists only when the parties have communicated an offer and an acceptance (collectively constituting sufficiency of communication), supported by consideration and an intention to create legal relations (collectively constituting indicia of enforceability).

Sufficiency of contract communication is a recurring issue when designing and implementing online transaction systems. Understanding the precise time when a contract comes into existence, the so-called contractual trigger, is important in risk-managing the design of online transaction systems [90, 91].[104] Prior to the existence of a contract the parties generally remain free to walk away. Post-contract, however, the parties are legally bound by promises made.

System designers should consider four successive moments in the contract communication process:

1. the time at which Alice transmits her offer[105] to Bob;
2. the time at which Bob receives Alice's offer;
3. the time at which Bob transmits his acceptance to Alice;
4. the time at which Alice receives Bob's acceptance.

Most common law systems would, by default, place the time of contract formation for online transactions into the last of these four times – the moment that Alice receives Bob's acceptance.

Practitioners are urged not to conflate these four distinct moments in time, even when they appear to be instantaneous. System designers should consider the impact of a lost or interrupted transmission and, accordingly, technical design should be carefully mapped onto relevant business process.[106]

A perennial question in the design of online systems concerns the precise point in time at which it can be said that Alice or Bob has 'received' a communication. The European Union attempted to address this in article 11 of the Electronic Commerce Directive 2000 [193]. This mandates adoption of a rule that 'orders' and 'acknowledgements' of orders[107] are generally deemed to have been received at the moment they become accessible to the receiving party (e.g., when the acceptance is received in Alice's online commerce server log or Bob's IMAP file).[108] This rule can be varied by contractual agreement in B2B commerce systems.

Differences in approach are worthy of investigation depending on the relative value of transactions supported and which state(s) contract law(s) might be applicable (see Section 3.6.7).

3.6.2 Encouraging security standards via contract

Contracts can serve as a mechanism to encourage the implementation of security standards. This can arise in a wide variety of contractual relationships.

3.6.2.1 Supply chain

A common contract technique is to incorporate terms within a procurement agreement that attempt to mandate some form of compliance by a supply chain partner with specified security standards: whether published standards such as ISO 27001, or *sui generis* standards adopted by the contracting parties. Although these contract terms can take many different legal forms (e.g., warranty, representation, undertaking, condition, mandate to produce evidence of third-party certification, access and audit rights etc.) the general principle is that these contract terms have become a common mechanism that is used in an attempt to influence the security behaviour of supply chain partners.

The value of these clauses in managing supply chain behaviour, however, is worth a closer examination. Consider the risk-weighted cost to a contracting party of breaching the terms of such a clause (introduced in Section 3.1.5 as R). In a legal action for breach of contract, the enforcing party normally remains responsible for proving that the breach caused financial harm, as well as the quantum of financial harm suffered by the enforcing party as a result of the breach (see Section 3.6.5). In the case of a breaching party's failure to comply with an obligation to maintain a third-party security certification, for example, it might be difficult or impossible for the enforcing party to prove that any financial harm flows from such a breach (thus effectively reducing the Q term to zero).

A sometimes-overlooked value of these contractual clauses arises well before the agreement is made. The process of inserting and then negotiating these clauses can operate as a due diligence technique. A negotiating party obtains information about the maturity and operational capability of the proposed supply chain partner during negotiations.

3.6.2.2 Closed trading and payment systems

Many high-value or high-volume electronic trading or payment platforms[109] require persons to enter into participation contracts prior to using the platform. These systems may be generally referred to as 'closed' systems: they constitute a club that must be joined contractually to enable members to trade with one another. These membership contracts typically adopt comprehensive rules concerning forms of communication, connected equipment, and the timing for finality of transactions. They also typically specify the adoption of certain security standards, authentication protocols, etc. The membership contract is thus a private law mechanism that is used to enforce certain security standards among the members. (See also Section 3.10.2.)

Breaching the terms of the membership contract might jeopardise the subject matter of the agreement itself – the finality of trades or the ability to collect payment. As an example, a merchant collecting payment via payment card that fails to comply with the authentication procedures mandated by its merchant acquirer contract might face a loss of payment for a

transaction even though it has delivered (expensive) goods into the hands of a person who has committed card fraud. Faced with such drastic financial consequences, the contracting parties may work exceptionally hard to meet the mandated authentication standards.

Perhaps the most well-known example of a widespread standard implemented using contract is PCI DSS adopted by the payment card industry. Failure to comply with this standard puts at risk a party's ability to receive payment. While there is some debate about the degree to which this standard has been effective, it is difficult to deny that it has had some impact on raising the standard of security practices employed by many merchants when handling card transaction data – especially those that previously seemed to approach the subject with a cavalier attitude.

3.6.2.3 Freedom of contract and its limitations

When considering using a contract as a means of regulating security behaviour, one must consider that the law can and does interfere with or otherwise limit the enforceability of some contract terms.

When considering PCI DSS standards, for example, the US Fair and Accurate Credit Transactions Act of 2003 [194] in Section 113 mandates specific truncation rules concerning payment card numbers displayed on printed receipts provided at the point of sale.[110] Thus, merchants subject to US law must consider these public law requirements as well as PCI DSS, and those who wish to modify the PCI DSS standards should do so in a manner that is sympathetic to the external requirements imposed on these merchants by US law. (Some states have taken the additional step of adopting the terms of PCI DSS into their law.)

In the case of funds transfer services, public law also establishes a framework to balance the rights and responsibilities of providers and users of payment services which includes considering the adequacy of authentication mechanisms. Examples can be found in articles 97 and 4(30) of the EU Second Payment Services Directive (PSD2), as implemented in the laws of member states [195], and article 4A §202 of the Uniform Commercial Code, as implemented in the laws of US states [196].

Limitations on the freedom of parties to deviate from public law norms in contract are further discussed in Sections 3.6.3 and 3.6.4.

3.6.3 Warranties and their exclusion

The term 'warranty'[111] describes a contractual promise concerning the quality or legal status of deliverables, the adequacy of information provided by a party, the status of a signatory, etc.

The contract laws of individual states normally imply certain minimum warranties into contracts concerning the quality of the goods and services supplied. The types of quality warranty most commonly imposed include:

- *Objective quality of goods.* The product vendor promises that the goods delivered will be objectively satisfactory to a normal purchaser given all of the circumstances of the transaction.[112]
- *Subjective quality of goods.* The product vendor promises that the goods delivered will be sufficient to meet the subjective purpose of an individual purchaser, whether or not the goods were originally manufactured for that intended purpose.[113] For this warranty

to apply, the purchaser is normally required to disclose the purchaser's specific purpose in advance to the vendor. As a result, this term is rarely discussed in the context of standard online commerce systems, which often do not allow unstructured communication between vendor and purchaser concerning intended use cases.

- *Objective quality of services.* The service provider promises that it will exercise due care in the process of service delivery.

Upon consideration, a significant distinction emerges between the quality of goods and services warranties. Compliance with the goods warranties is assessed by examining the goods supplied. A warranty that goods will be objectively satisfactory is breached if the goods are poor – without regard to the care taken by the vendor in manufacturing, sourcing, or inspecting goods. By contrast, a warranty that a service provider will take due care is assessed by examining the service provider's actions, qualifications and methodology. It is possible for a service provider to comply with a warranty of due care and yet produce a deliverable which is demonstrably poor or inaccurate. (The basis of this distinction between product and service is becoming increasingly difficult as persons place greater reliance on cloud services as a substitute for products. (See also discussion in Section 3.7.2.)

Although various laws imply these standard warranties into contracts as a matter of course, it is commonplace – nearly universal – for suppliers of information and communications technologies and services to attempt to exclude these terms by express agreement. Efforts to exclude these baseline warranty protections are viewed with suspicion under the contract laws of various states. As a general proposition, it is more difficult and often impossible to exclude these baseline protections from standard form contracts with consumers. In the context of B2B contracts, however, the rules allowing these exclusions tend to be more liberal.

Information and communications technology vendors normally exclude these baseline implied warranties and replace them with narrowly drawn express warranties concerning the quality of deliverables.[114] The relative utility of these express warranties provided by ICT vendors is questioned with some regularity, especially as regards commercial off-the-shelf software or hardware. It remains an open question to what degree these warranty standards encourage or discourage developer behaviours in addressing security-related aspects of ICT products and services [88].

3.6.4 Limitations of liability and exclusions of liability

Parties to contracts often use the contract to impose both limitations and exclusions of liability that arise from the contracting relationship. An exclusion of liability refers to a contractual term that seeks to avoid financial responsibility for entire categories of financial loss arising as a result of breach of contract, such as consequential loss, loss of profit, loss of business opportunity, value of wasted management time, etc. A limitation of liability, on the other hand, seeks to limit overall financial liability by reference to a fixed sum or financial formula.

The possibility of imposing and enforcing contractual limitations and exclusions of liability creates a powerful incentive for vendors to draft and introduce express terms into their contractual relationships with customers. The contract becomes a risk-reduction tool. As a result, these exclusions and limitations are ubiquitous in contracts for ICT goods and services.

As with the exclusion of implied warranty terms, limitations and exclusions of liability are viewed with suspicion under most systems of contract law. Once again, limitations and

exclusions of liability are most heavily disfavoured when contracting with consumers. Rules allowing these exclusions and limitations tend to be more liberal in B2B arrangements.

There is a wide variation among and between jurisdictions concerning the enforceability of these limitations and exclusions. As a general proposition, civil law jurisdictions disfavour these limitations and exclusions more than common law jurisdictions. Requirements of form (see Section 3.10.2) are common, and many legal mechanisms limit the enforceability of such terms.

It remains an open question to what degree the relative enforceability of these contractual limitations and exclusions encourages or discourages developer behaviours in addressing security-related aspects of ICT products and services [88].

3.6.5 Breach of contract & remedies

When considering the obligations imposed by contract, it is also important to consider the legal consequences of breaching a contract. (See Section 3.1.5.) A 'breach' of contract is simply a failure to fulfil a promise embodied in the contract. Breaches exist on a spectrum of severity. An individual breach of contract might be considered *de minimis*, moderately serious, very significant, etc.[115] The severity of breach can and often does result in different remedies for the injured party.

In the event of a breach of contract, various remedies provided by courts to non-breaching parties typically fall into the following categories:[116]

- *Damages.* Order the breaching party to pay monetary damages to the non-breaching party that are sufficient to restore the net financial expectation that the harmed party can prove was foreseeably lost as a result of the breach. This is the most common remedy available. A non-breaching party is often obliged to take steps to mitigate financial harm, and failure to mitigate can serve to reduce an award of damages accordingly.
- *Recision.* Declare that the contract is at an end and excuse the non-breaching party from further performance. This is a more extreme remedy, normally reserved for cases in which the breach is very severe. Alternatively, the terms of the contract might specifically legislate for the remedy of recision under defined circumstances.[117]
- *Specific performance.* Order the breaching party to perform their (non-monetary) promise. This is also considered an extreme remedy. This remedy is often reserved for situations when the breaching party can take a relatively simple action that is highly significant to the non-breaching party (e.g., enforcing a promise to deliver already-written source code or to execute an assignment of ownership of copyright that subsists in a deliverable).
- *Contractually mandated remedies.* The contract itself may specify available remedies, such as service credits or liquidated damages. Courts often treat these remedies with suspicion. The law concerning enforceability of private remedies is complex and varies significantly from jurisdiction to jurisdiction.

The remedies described above are normally cumulative in nature. Thus, a party can both request recision and claim for damages as a result of a breach.

3.6.6 Effect of contract on non-contracting parties

One potential limitation of the utility of contracts is that enforcement may be limited to the contracting parties alone.

In the context of seeking a remedy for breach, the rule of privity of contract (generally found in common law systems) normally restricts contract enforcement solely to the contracting parties. If Alice and Bob enter into a contract and Alice breaches, under the doctrine of privity Bob is normally the only person who can take legal action against Alice for breach of contract. Charlie, as a non-party, cannot normally take action against Alice for breach of contract even if Charlie has been harmed as a result of the breach. Charlie may, however, be able to take action against Alice under tort law, as discussed in Section 3.7. In complex supply chains, Bob might be able to assign the benefit of the contract rights (such as warranties) to Charlie. (Even in common law systems, there are circumstances in which parties can expressly vest contract rights in the hands of third parties.)

If Alice is a supplier of services and wishes to limit her potential liability to persons who rely on the outputs of these services, a contractual limitation of liability might not be effective against a non-contracting person like Charlie who relies on her service but is not in privity of contract with Alice. This inability to limit liability to non-contracting parties is a recurring consideration in the development of trust services, in which third parties who rely on trust certificates may have no direct contract relationship with the certificate issuer. (See the discussion at Section 3.10.)

3.6.7 Conflict of law – contracts

Deciding which state's law will apply to various aspects of a contract dispute is normally vested within the jurisdiction of the court deciding the dispute. The rules used to decide this question can and do vary from state to state. Within the European Union, these rules have been harmonised for most types of contract – most recently through the mechanism of the 'Rome I' Regulation [197]. Individual US states, by contrast, remain free to adopt their own individual rules used to decide whose law should be applied to aspects of contract disputes. Even with these variations some useful and generalisable principles can be identified.

Express choice by the parties. It is widely accepted that persons who enter into a contract should have some degree of freedom to choose the law that will be used to interpret it. (Rome I, Art 3 [197].) Various policy justifications are available, often built upon notions of freedom of contract. If parties are free to specify the terms of their contractual relationship, this argument suggests that the same parties should be free to incorporate within the agreement anything that assists to interpret the terms that have been agreed – including the substantive system of contract law used to interpret the agreement.

Absence of an express choice of law by the parties. When parties connected to different states do not make an express choice of law in their contract, the court is faced with the dilemma of deciding whose law to apply to various aspects of the contract dispute. In the European Union, rules determining the applicable law in the absence of choice are found in Rome I, Art 4. Of particular interest to those who deal with online contracting systems, are the following default rules in the absence of a clear choice by the parties:

- A contract for the sale of goods or supply of services will be governed by the law of the place where the seller or service provider has its habitual residence. Art 4(a)-(b).

- A contract for the sale of goods by auction shall be governed by the law of the country where the auction takes place, if such a place can be determined. Art 4(g).
- A contract concluded within a multilateral system which brings together or facilitates the bringing together of multiple third-party buying and selling interests in financial instruments in accordance with non-discretionary rules and governed by a single law, shall be governed by that law. Art 4(h).

Thus, we see in European law a baseline policy preference to apply by default the law where the vendor or market maker is resident, over the law where the buyers or bidders may be resident.

Contracts with consumers. When one of the parties to a cross-border contract is a consumer, the rules are generally modified to provide additional protection for the consumer. In disputes in a European Union forum court, for example, if the cross-border vendor of products or services pursues their business activity in the place of the consumer's residence, or 'directs such activities to that country or to several countries including that country', then the following special rules usually apply:

- If there is no express choice of law in the contract, the applicable law will be the law of the consumer's habitual residence. Art 6(1).
- If some other law has been expressly chosen, that choice of law cannot deprive the consumer of legal protections mandated by the law of the consumer's residence. Art 6(2).

Although the specific examples above are drawn from European legislation, they represent principles that regularly occur in other states that face conflict of law issues in consumer contract disputes.

3.7 TORT

A tort is any civil wrong other than a breach of contract. Unlike contractual liability, tort liability is not necessarily predicated upon a volitional relationship between the person who commits a tort (a 'tortfeasor') and the person harmed by that tortious action (a 'victim').

This section will address a few of the more common tort doctrines that should be considered by cyber security practitioners. Two substantive torts of interest (negligence and product liability) will be examined in some detail together with a series of more general tort doctrines such as causation and apportionment of liability. Rights of action granted to victims under other legal subject matter regimes (e.g., data protection, defamation, intellectual property, etc) are also characterised as tort actions, and general tort concepts (see Sections 3.7.3, 3.7.4, 3.7.5 & 3.7.6) often apply to these as well.

3.7.1 Negligence

Most legal systems recognise the idea that persons in society owe a certain duty to others in the conduct of their activities. If a person fails to fulfil this duty, and the failure causes harm to a victim, the victim is often given a right to take legal action against the tortfeasor for financial compensation.

3.7.1.1 Duty of care: how far does it extend

Legal systems implicitly acknowledge that a person is not always responsible to everyone all of the time. Some limitation on the scope of responsibility is normal. The courts of England, for example, have said that one person (Alice) owes a duty of care to another (Bob) in respect of a given activity if three conditions are fulfilled:

1. Alice and Bob are somehow proximate to one another in time and space;
2. it is reasonably foreseeable to Alice that her action (or inaction) could cause harm to persons in a position similar to Bob; and
3. with respect to Alice's action (or inaction), on the whole it seems fair and reasonable for persons like Alice to be responsible to persons in a position similar to Bob.

Although this three-pronged rule is not presented as a multinational norm, it illustrates the general proposition that the scope of civil responsibility owed to others as a result of negligence is limited.[118]

'Foreseeability' of harm is used routinely as a mechanism to limit the scope of liability in negligence law.[119] Foreseeability is normally measured by reference to whether or not an objectively reasonable person would have foreseen harm. A tortfeasor is not excused from liability due to failure of imagination, failure to plan, or an affirmative effort to avoid considering potential victims.[120]

This raises a number of related questions about possible duties of care in the context of cyber security, some of which are set out in Table 3.2. The purpose of Table 3.2 is merely to consider some of the types of relationship that might create a duty of care under existing law.

Negligence laws are tremendously flexible. As harm caused by cyber security failure becomes increasingly foreseeable, it seems likely that courts will increasingly interpret the concept of duty of care to encompass various cyber-security related obligations owed to a broader group of victims [199].

The concept of 'duty of care' does not normally depend on the existence of any business or contract relationship between tortfeasor and victim. As a commonly understood non-security example, automobile drivers are said to owe a duty of care to other drivers, to bicycle riders, to pedestrians, and to others who are expected to use roads and pathways. One might therefore consider the extent to which those who supply software on a non-commercial basis, such as open source security software, might be found to owe a duty of care to those persons who foreseeably rely upon such software.

When this potential tortfeasor:	Conducts this activity in an unreasonable manner:	Consider whether the potential tortfeasor owes a duty of care to these potential victims:
Retail merchant.	Maintaining security of payment card details supplied by customers at point of sale.	Card holders; Merchant banks; Card issuing banks. [198]
Email service provider.	Managing the security of email servers and related services; Making decisions about the types of security to be adopted.	Service subscribers; Subscribers' third-party email correspondents; Persons named in email correspondence.
Business enterprise.	Managing the cyber security of enterprise IT or OT; Making decisions about the types of security to be adopted.	Its own staff members; [199][121] Its counter-parties and supply chain partners; Unrelated third parties suffering harm when malicious actors compromise the enterprise's security measures and use the enterprise's IT to launch onward attacks; Unrelated third parties who suffer harm when compromised OT causes personal injury or property damage.
Developer of web server software.	Implementing standard cryptographic communication protocols.	Merchants that adopt the web server software for online commerce; SaaS providers that adopt the web server software for the provision of various services to customers; Customers submitting payment card details; Business customers that submit sensitive business data to a SaaS provider that adopted the server software; Business enterprises that adopt the server within their IT or OT infrastructure.
Trust service provider.[122]	Registering the identity to be bound to a certificate; Issuing certificates; Maintaining the trust infrastructure.	Customers who purchase certificates; Third parties who place reliance on these certificates; Third parties who operate equipment which (without their knowledge) places reliance on these certificates.
Web browser developer.	Selects root trust certificates for installation into its web browser.	Natural persons who use the web browser.

Table 3.2: Illustration of potential duty of care relationships

3.7.1.2 Breach of duty: measuring reasonableness

If a person (Alice) owes a duty of care to another person (Bob) in the conduct of a given activity, the question arises whether or not Alice has breached (failed to fulfil) her duty to Bob. The formulation of these two elements together – 'breach of a duty of care' – is normally synonymous with 'negligence'.

A typical standard used to assess conduct is to examine whether or not Alice has acted in an objectively reasonable manner. In classic negligence law persons like Alice are not held to a standard of perfection. Liability is based upon fault. In assessing fault, courts often make use of rhetorical devices such as the objectively 'reasonable person' similarly situated.

As a framework for measuring conduct, the reasonable person standard has proven remarkably resilient and flexible over time. Cyber security practitioners often converge with opinions on whether a given cyber security-related action (or inaction) was objectively reasonable or unreasonable. Changes in technology, development of new methods etc. can all serve to revise opinions on the definition of what constitutes 'reasonable' security conduct.

There is a temptation to conflate 'reasonable conduct' with efforts to define so-called 'best practice'. Rapid advances in information technology (e.g., the falling cost of processing capability) routinely alter the cyber security landscape. Disruptive changes in the environment (e.g., the move to the cloud, the emergence of big data, the birth of the Internet of Things) can rapidly de-stabilise received wisdom.

The highly respected US Judge Learned Hand warned of this in two famous decisions from the mid-twentieth Century. Responding to an operator of an ocean-going cargo vessel that argued that the vessel's lack of a working radio did not constitute 'unreasonable' conduct, Judge Hand observed in *The T.J. Hooper* case in 1932 that 'common practice is not the same as reasonable practice' [200, 201]. Although the vessel operator conformed with common industry practice of the 1920s, Judge Hand clearly expressed the idea that changes in technology and the surrounding environment should spur re-examination of methods and activities.

Fifteen years later in the 1947 *Carroll Towing* case, Judge Hand announced a definition of reasonable conduct that may be helpful in assessing whether or not the time has arrived to adopt a given method of operation. He reasoned that when the burden (cost) of taking a given precaution is less than: (1) the probability of loss in the absence of that precaution, *multiplied by* (2) the amount of the loss to be avoided, then the 'reasonable' action is to adopt that precaution [201, 202, 203].[123] His decision sets out a type of cost-benefit test, although in this case the cost (the 'burden' of the potential precaution) that would be incurred by Alice (the person who owes a duty of care) is compared with a benefit (i.e., reducing the risk-weighted cost of a potential loss) enjoyed by Bob (the person or set of similarly situated persons to whom that duty of care is owed).[124]

The doctrine of 'negligence, *per se*' is sometimes adopted by courts to assess conduct. Using this doctrine, a victim argues that the tortfeasor's conduct should be found to be unreasonable because that conduct violated a public law or widely-adopted technical standard. If Alice makes unauthorised access to Bob's computer (a crime, as discussed in Section 3.5) and causes damage to that computer or other parts of Bob's infrastructure as a result, Bob could plead that Alice is negligent, *per se*. This doctrine has already been pleaded together with standard negligence claims in legal action arising from cyber security-related incidents [198]. This doctrine may become increasingly useful to victims as a result of increasing standardisation and regulation in the field of cyber security.[125] The mere act of defining and adopting security

standards may therefore influence courts as they seek technical frames of reference for assessing the 'reasonableness' of conduct.

Another doctrine that may be useful in analysing cyber security failures is that of '*res ipsa loquitur*' (i.e., 'the thing speaks for itself'). Using this doctrine, a victim who might otherwise have difficulty proving the precise nature of the action that caused harm, claims that the most appropriate inference to be drawn from the surrounding circumstances is that the accused tortfeasor bears the responsibility. This doctrine tends to be used against persons who are supposed to maintain control over risky or dangerous processes that otherwise cause harm. A typical example might include legal action against a surgeon after a surgical instrument is discovered in the body of a post-operative patient, or a wild animal escapes from a zoo. Irrespective of the victim's ability to prove a lapse of caution, the most appropriate inference to be drawn from the circumstances is a lack of due care. (Hence, the thing speaks for itself.) In the field of cyber security, one might imagine a case in which this doctrine is applied in a legal action against a person who creates a new form of malware for research purposes, only to lose containment.[126]

Doctrines similar to negligence *per se* and *res ipsa loquitur* might be defined in some legal systems as rules concerning the reasonability of conduct, or they might be defined as rules of evidence – relieving a victim of some or all of their burden to prove unreasonable conduct, or shifting the burden to the alleged tortfeasor to prove reasonable conduct. (See the discussion of evidence in Section 3.1.4.)

Although they are not normally considered under the rubric of negligence law, other laws which influence cyber security practice define 'reasonable' conduct within their sphere of competence. An example is found in the law of funds transfer expressed in article 4A §202(c) of the Uniform Commercial Code as adopted by US states [196].

3.7.1.3 The interpretation of 'fault' differs by place and changes over time

Although the framework presented above for defining 'fault' is generally well-received by legal systems in most developed economies, this should not be mistaken for agreement on how to interpret or apply these standards.

The interpretation of both 'duty of care' and 'reasonable' behaviour can vary significantly from state to state. This should not be surprising, as both concepts are social constructs anchored by opinions about risk and responsibility that prevail in a given society at a given time.

The interpretation of 'duty of care' has (with some exceptions) mostly expanded over the past century as the increasingly complicated and interconnected nature of modern life creates more opportunity for the actions of one person to harm others. Similarly, the interpretation of 'reasonable' has generally moved in the direction of requiring more care, not less. These interpretations can be expected to change within the working life of a practitioner, especially as the harm caused by cyber security failure becomes increasingly foreseeable, better understood, and easier to prove with new forensic tools.

Similarly, practitioners are cautioned that potentially tortious acts committed in one state might be assessed by the interpretation of standards of care adopted by another, more demanding, state. (See the discussion in Section 3.7.6.)

3.7.2 Strict liability for defective products

In the second half of the twentieth Century, a number of states with developed industrial economies adopted rules of strict liability for defective products.[127] This liability regime provides a right of action for those who suffer personal injury, death, or property damage, caused by a defective product. A product is usually deemed to be defective when it fails to provide the safety that a reasonable person would expect under the circumstances. Depending on the specific law in question, strict liability typically attaches to persons who produce, import or sell defective products or component products. Liability can attach to a tortfeasor who has no pre-existing relationship with the victim.

In this type of liability, the focus of analysis shifts away from any notion of 'fault' by the tortfeasor and moves instead to an examination of the allegedly defective product. Liability is generally assessed without regard to the degree of reasonableness used in producing, examining, or selecting products for sale. This type of strict liability is found throughout the laws of the states of the US and is incorporated into EU member states' domestic laws as mandated by Directive 85/374 [204, 205].

Most authorities believe that software, as such, does not fit within the various definitions of 'product' applicable under such laws.[128] Even so, under currently-existing product liability law a defect in a software component can be the source of a defect in a product into which it is installed. Liability of this sort arising from cyber security failures will probably increase as physical control devices are increasingly connected to remote data services, presenting more cyber security-related risks to life and limb.

A cyberspace-connected product (e.g., an autonomous vehicle,[129] an industrial control system, a pacemaker, a vehicle with fly-by-wire capability, a remotely operated home thermostat) that fails to deliver appropriate safety, is defective whether the safety is compromised through failures in electrical, mechanical, software, or security systems. Thus, strict product liability could be implicated in cases of personal injury or property damage whether the safety of the connected device is compromised through errors in operational decision-making (e.g., an autonomous vehicle chooses to swerve into oncoming traffic after misinterpreting road markings) or errors in cyber security (e.g., a flawed or improperly implemented authentication scheme permits a remote hacker to command the same vehicle to divert into oncoming traffic, to open the sluice gates in a dam, or to alter a home thermostat setting to a life-threatening temperature).

In its comprehensive 2018 evaluation of European product liability law, the European Commission referred extensively to the increased role of software and other so-called 'digital products' in modern commerce [206]. The Commission openly questioned the extent to which digital products (e.g., software as a product, SaaS, PaaS, IaaS, data services, etc.) should be redefined as 'products' under product liability law and thus subjected to strict liability analysis when defects cause death, personal injury, or property damage [207]. This is an area of law that could change significantly in the medium-term future.

3.7.3 Limiting the scope of liability: legal causation

The primary purpose of tort law is to compensate victims for harm suffered. A victim can normally only bring a legal action against a tortfeasor if the victim can prove that the relevant tortious action was the cause of a legally cognisable harm suffered by the victim. Put simply, people may act negligently without tort liability – if their behaviour causes no harm.

Causation is one of the more difficult concepts to define in law. Different authorities take different views about when it is appropriate to hold a tortfeasor responsible when it is claimed that tortious action A has produced harm B. The victim is often required to prove causation-in-fact as a first step. This concept is also expressed as 'but for' causation, because it can be tested using the logical statement: 'But for tortious action A, harm B would not have occurred.' Liability can sometimes be eliminated by showing that a given harm would have occurred independently of a tortious act [208, 209].

Causation-in-fact, however, is often not sufficient on its own to prove liability. Difficulties arise when analysing more complex chains of causation where tortious action A causes result X_1, which in turn causes result X_2, ..., which in turn causes result X_n, which in turn causes harm B.[130] As the link between A and B becomes increasingly attenuated, policy makers and judges struggle to define the limits of responsibility of the person committing the tortious act. Similar difficulties arise when the 'last cause' in a combination of negligent acts causes harm that is significantly disproportionate to the individual negligent last act, as a result of more serious lapses of judgment by prior actors. Approaches adopted to resolve this issue include limiting the responsibility of the tortfeasor to harm that is reasonably foreseeable [210].[131]

The narrower definition of causation required by tort law may be referred to using terms such as 'legal causation' or 'proximate causation'.

Proving that a specific harm was caused by a specific cyber security incident can be extremely challenging. To take a common example, a natural person whose identification data has been compromised may find it difficult or impossible to prove that the data lost in a given data breach event are the source of the data subsequently used by malicious actors to carry out fraud through impersonation. Data breach notification laws help to redress the imbalance of evidence available to victims in these cases, but even then, the victim must prove a causal link from a specific breach to the fraud event. A notable exception is financial loss incurred following a breach of payment card data, as the causation of subsequent fraud losses can be easily inferred from a contemporaneous data breach event. These cases create other challenges as discussed in Section 3.7.4.

3.7.4 Quantum of liability

Consideration of the impact of tort law on cyber security related activity is incomplete without considering quantum of liability. (See Section 3.1.5.) Different states have different approaches to defining what constitutes legally cognisable harm for purposes of tort law. A victim is normally required to prove the financial value of harm caused by a tortious act.

In cases involving personal injury, the value of the harm is often calculated by reference to easily understood measures such as: loss of salary suffered by the victim due to their inability to work, costs incurred by the victim for medical treatment, rehabilitation, or nursing care, costs of installing accommodation facilitates for a permanently injured victim, etc. Some states also allow compensation for harm in personal injury cases that is more difficult to quantify such as pain and suffering, emotional distress, etc.

A recurring issue in negligence cases concerns whether or not a victim can recover for so-called pure economic loss. There is a divergence in the law on this question. A leading case in England concerned the economic loss caused by a poorly considered credit reference provided by a bank to its customer. Although the loss (the customer's subsequent inability to collect a trade debt from its insolvent client) was purely economic in nature, the English court decided it should be recoverable because the bank professed special skill (financial awareness), and the victim relied on the flawed statement to its detriment [211].[132]

A growing number of cases have been brought on the basis of negligent cyber security which claim losses other than personal injury or property damage. Some courts have already exhibited a willingness to award damages under the law of negligence to victims whose losses are purely economic in nature [199].[133] Other legal actions (settled by the parties before trial) have involved substantial claims for economic losses based on negligent cyber security.[134]

Proving legally cognisable harm can be challenging for some victims who might otherwise wish to take legal action based on cyber security failures. One example concerns the loss of privacy. There is a lot of argument about how to quantify (financially) the harm caused by breach of privacy unless the victim has some business or economic interest that is directly harmed as a result.[135]

Another common example concerns the loss of confidentiality of financial authentication methods such as payment card details. Card holders would have difficulty proving harm to the extent that fraudulent charges are refunded by the issuing bank. Of course, the issuing bank will then be able to demonstrate financial harm as a result of refunding these monies, plus a *pro rata* portion of the costs incurred issuing replacement cards earlier than planned.

In response to these types of difficulties in proving harm, some states have adopted specific laws that provide a schedule of damages that can be claimed without the need to quantify harm. An example is found in the State of Illinois Biometric Information Privacy Act, which provides that any party aggrieved by a violation of the act can take legal action and recover US$1,000 per violation (for negligent violations) or US$5,000 per violation (for intentional violations) of the law's mandates.[136] Similarly, US copyright law allows some rights owners to recover minimum damages using a statutory tariff.

Some jurisdictions, notably member states of the United States, are prepared to award 'punitive damages' (known in some jurisdictions as 'exemplary damages') in some tort cases. These awards are intended to punish and deter bad behaviour. These awards can be disproportionate compared to the underlying award for the harm suffered by the victim. Examples where a court might award punitive damages most commonly include cases where the tortfeasor demonstrates a pattern of repeated poor behaviour, or the tortfeasor has made relevant operational decisions with gross indifference to human life or human suffering.

3.7.5 Attributing, apportioning and reducing tort liability

This section discusses a few miscellaneous legal doctrines that are important to consider when attempting to assess risks of tort liability.

3.7.5.1 Vicarious liability

There are circumstances when the liability of a tortfeasor can be attributed to a second person. The situation commonly encountered in cyber security is liability for the tortious act of an employee attributed to their employer. This type of vicarious liability applies when the tort is committed during the course of an employment relationship.

Vicarious liability is strict liability. Once a victim proves that the employee committed a tort which caused relevant harm and then proves that the tort was committed within the course of employment, the employer becomes strictly liable for that underlying tort. Pleas by the employer about taking reasonable precautions, mandating reasonable training, due diligence when hiring or the employee's deviation from employment standards, are generally ineffective against claims of vicarious liability.

The Court of Appeal in England in 2018 affirmed a vicarious liability claim brought in a data protection tort action. In *Wm Morrison Supermarkets PLC vs Various Claimants*, the data controller Morrison was sued by various data subjects after a disgruntled internal audit employee published salary data of 100,000 employees in violation of data protection law.[137] The secure handling of salary data fell within the field of operation with which the employee was entrusted and therefore the tort was committed by the employee within the scope of the employment relationship, thus leading to vicarious liability [176]. At time of writing, this decision is pending appeal in The supreme court of the United Kingdom.[138]

The only reliable method to avoid vicarious liability is to encourage employee behaviour that limits or avoids tortious activity. This is worthy of consideration by those who develop and enforce acceptable use policies, staff security standards, employment policies, etc.

3.7.5.2 Joint and several liability

In cases where more than one tortfeasor can be said to have caused harm to a single victim, tort law often imposes joint and several liability. The doctrine is simple: any jointly responsible tortfeasor could be required to pay 100% of the damages awarded to a victim. Although the tortfeasor satisfying the victim's financial claim may have the right to pursue compensation (a.k.a. 'contribution') from other tortfeasors, this becomes problematic when the joint tortfeasors have no financial resources or are resident in foreign states where there is no effective method of enforcing such rights.

Practitioners may wish to consider the impact of this rule when working with supply chain partners or joint venturers that are small, do not have much capital, or are resident in a foreign state where enforcement of domestic judgments may be problematic.

3.7.5.3 Affirmative defences

Tortfeasors are sometimes able to take advantage of certain affirmative defences to tort claims. A tortfeasor who is able to prove the relevant elements of these defences can reduce, or sometimes eliminate, their liability.

In the context of negligence, a common category of defences includes 'contributory negligence' or 'comparative fault' of the victim. In this type of defence, the tortfeasor attempts to prove that the victim's own negligence contributed to their harm. Depending on which state's tort law is applicable to the claim, a successful defence can reduce or eliminate liability to the victim.

Another category of defence that can be useful in various cyber security contexts include 'assumption of risk' or 'consent'. In this type of defence, the tortfeasor avoids liability by proving that the victim was aware of, or knowingly consented to, the risks that ultimately caused the harm. This type of defence can be especially useful for those who supply cyber security services that risk damage to client infrastructure, such as penetration testing. Practitioners often draft commercial engagement documents with a view to attempting to satisfy one of these defences in the event that something goes wrong during the engagement.

As regards strict product liability, many states offer a so-called 'state of the art' defence. Where this defence is allowed, a party can avoid strict liability by proving that a product, although defective, was produced at a time when the technological state of the art would not have enabled discovery of the defect. It is debatable how this defence might apply to products made defective as a result of cyber security flaws.[139] Of greater significance, perhaps, is the affirmative defence against strict liability for a defective product available if the defending party can prove that the defect is present due to compliance with laws or regulations concerning product design.[140]

3.7.6 Conflict of law – torts

Deciding which state's law applies to various aspects of a tort dispute is normally vested within the juridical jurisdiction of the forum court deciding the dispute. The rules that are used to decide this question can and do vary from state to state. Within the European Union, these rules have been harmonised for most torts through the mechanism of the 'Rome II' Regulation [212]. Individual US states, by contrast, remain free to adopt their own individual choice of law principles when deciding whose law should be applied to aspects of tort disputes. Even with these variations some useful and generalisable principles can be identified.

Broadly speaking, courts that examine tort claims between persons in different states tend to adopt one of two methods most often used to decide whose law to apply: apply the law of the place where the tortious act originated or apply the law of the place where the injury was suffered. Historically, it might have been difficult to find cases where these two events occurred in different states. Modern commerce, however, has produced a number of cases where the two events can be widely separated by space and time.[141]

In disputes heard in courts throughout the European Union, the applicable law in a tort action (with some exceptions) is the law of the place where the damage was suffered. (Rome II, Art 4(1).) In cases of product liability, the rules are slightly more complex and the applicable law might be the place of the injured party's habitual residence, the place where the product was acquired, or the place where the damage occurred. (Rome II, Art 5.)

The above rules provide a reasonable indicator of the risk that cyber security failures occurring due to actions performed in State A, and subsequently causing harm to persons in State B, could easily become amenable to liability analysis under the tort law of State B. Thus practitioners (and their employers) might be held to a higher standard of care imposed by a foreign state where victims of negligent cyber security or defective IoT products are found. (See, for example, the discussion of liability in Section 3.10.4.)

3.8 INTELLECTUAL PROPERTY

[90, 91]

The complexity of intellectual property law prompted a nineteenth-century US jurist to comment that this subject is closest to 'the metaphysics of law'.[142] Metaphysical or not, intellectual property can serve to constrain or encourage actions by cyber security practitioners. This section will summarise some points where the two fields intersect.

3.8.1 Understanding intellectual property

Intellectual property rights are negative rights – they convey the right to demand that other persons cease a prohibited activity. The nature of the activity to be prohibited is defined in the law establishing that right. Ownership of intellectual property normally conveys a right of action against others who transgress one or more acts prohibited by the relevant property right.

Intellectual property rights do not give the affirmative right for the owner to take any action imaginable with the subject matter. A given action (e.g., combining one's own code with others, abusive intellectual property licensing practices) could infringe third-party intellectual property rights or trigger liability under competition law, among others.

Registered intellectual property rights (e.g., patents and registered trademarks) are granted on a state-by-state basis following application to an appropriate state agency, often following examination by state officials. Unregistered intellectual property rights (e.g., copyright) usually spring into existence without any need for intervention by state officials.

The term 'public domain' often causes confusion. In the field of intellectual property law, 'public domain' refers to a work in which no current intellectual property right subsists. By contrast, the phrase 'public domain' is also used colloquially to indicate a lack (or loss) of confidentiality. To distinguish these two, if a confidential original written work is subsequently published the contents become publicly known. Confidentiality has been lost. This work, however, may still be protected by copyright unless these rights are expressly relinquished. In contrast, if a person who writes software then declares that they are placing the code 'in the public domain' this statement is often treated as an irretrievable relinquishment of copyright. The term should be used with care.

3.8.2 Catalogue of intellectual property rights

This section will describe some of the intellectual property rights most likely to be encountered by cyber security practitioners. Additional intellectual property rights that may be of interest to practitioners, but which are not addressed in this section, include protections for semiconductor topographies, the EU *sui generis* right to prevent the extraction or reutilisation of the contents of a database, and registered and unregistered design rights.

In many circumstances, contract rights (especially licensing agreements) supplement intellectual property rights and may be treated informally as a type of intellectual property. To make matters more confusing, persons in business often use the phrase 'intellectual property' in an expansive and colloquial fashion to refer to any work product or process that is the result of intellectual effort - whether or not it incorporates legally recognised and enforceable intellectual property rights. This section deals only with legal rights, as such.

3.8.2.1 Copyright

Copyright[143] is an unregistered right[144] that springs into existence on the creation of a sufficiently original work. Copyright subject matter includes literary works, which for this purpose includes software code (both source and executable code). This makes copyright especially important for the developers and users of security products embodied in software.

The scope of copyright is generally said to be limited to the expression of an idea rather than the idea itself. Thus, copyright in software code normally protects only the code as written and not the functionality of the resulting software product. Protection of functionality is usually the province of patent rights.

The term of copyright is, by ICT standards, extremely long. Literary works are normally protected for the life of the author plus 70 years following their death. While the term of copyright protection granted to computer software may be less than this, it remains sufficiently long that the expiration of the copyright term is unlikely to apply to any relevant software encountered by a security practitioner within their lifetime.

Infringement of copyright normally consists of acts such as copying, transmitting, displaying or translating a significant part of the protected work. Proving that one work infringes the copyright embodied in a second work requires proof of copying. Copying can be inferred from sufficient points of similarity between the two works – there is no need to prove knowledge of copying by the accused. A plethora of forensic techniques have been developed over the course of decades to assess infringement of software source code.

Liability for copyright infringement can sometimes be avoided through various 'fair use' or 'fair dealing' limitations. These are defined differently from state to state.[145]

The scope of copyright protection was expanded at the turn of the twenty-first century to encompass the right to take legal action against persons who interfere with the technological measures used to protect copyright works [213] at Art 11.[146] This was intended to provide additional legal rights of action against those who circumvent technologies such as digital rights management systems (see the discussion in Section 3.8.4.) [214].

3.8.2.2 Patents

A patent is a registered intellectual property right, granted on a state-by-state[147] basis following application and examination. Patents are meant to protect an invention that is novel and that also includes an additional distinguishing characteristic variously described by states as an 'inventive step', a 'non-obvious' character, or something similar. This inventive step requirement is a policy device used to limit patent protection to inventions that are significant in some fashion, rather than trivial.[148] Novel inventions that would have been obvious to a person skilled in the relevant technical art are normally denied patent protection.

States expressly define additional subject matter that may not be claimed as a patented invention. Common exclusions of special interest to security practitioners are software, *as such*, and an idea or mathematical formula, *as such*.[149] Inventions that embody these, however, can be patentable subject matter in appropriate circumstances.

The US patent system has changed its approach to software patents in the past few decades and is increasingly receptive to them. Even states that notionally reject the concept of software patents regularly grant patents on inventions that are embodied in software. In other words, software patents (crudely speaking) are a regular feature of the ICT domain.

Cyber security-related inventions that appear on their face to be purely mathematical or algorithmic (e.g., cryptographic methods) can be the subject of patent protection as embodied in various devices – including software-enabled devices. Aspects of historically significant cryptography inventions have been protected by patents, including DES, Diffie-Helman, and RSA [215]. Although the patents on these breakthrough cryptographic inventions have now expired, the field of cyber security innovation remains awash with patents and pending patent applications [216, 217, 218].

The price of a patent is paid in two forms: money and public disclosure. Applications are expensive to prosecute and expensive to maintain. The process of navigating international application and examination is sufficiently complex that (expensive) expert assistance is always advisable, and often critical to success. In addition to application and examination fees paid to states, those who are granted a patent are then required to pay periodic fees to maintain the patent throughout its life.

Beyond the monetary cost, public disclosure is a core feature of the patent system. The patent application must disclose how the invention works in a manner that would enable a skilled technical practitioner to replicate it. The application and the granted patent, together with examination correspondence,[150] is normally published to enable future study.[151]

The term of a patent is normally 20 years from the date of application. Patents are typically subjected to an examination process which can take years to conclude. When a patent is granted, the right holder is normally given the right to take legal action retrospectively for infringements that took place after the application but before the grant, even if the infringement happened prior to the publication of the application.[152] The validity of a patent can be challenged post-grant and this is a common method of defending against infringement legal actions.

Infringement of a patent normally consists of acts such as manufacturing, distributing, importing, exporting or selling a product or service that embodies the claimed invention. Proving infringement involves a forensic comparison of the accused device or service with the invention as claimed in the granted patent. There is no need for a right holder to prove that the invention was copied from the patent or from any product. Many people who infringe

ICT-related patents do so initially without any awareness of third-party products or patent rights.[153]

3.8.2.3 Trademarks

Trademarks are usually registered[154] intellectual property rights, granted on a state-by-state basis following application.[155]

A trademark is a symbol or sign used to distinguish one person's business or products from another's. The most common trademarks consist either of words or figures.[156] Trademarks are granted within defined use categories, meaning that it is possible for two different persons to have exclusive rights for the use of the same symbol in different lines of business. The purpose of trademarks is to reduce the possibility of confusion for those who procure goods or services, and to protect investment in the reputation of the enterprise supplying those goods or services.

Trademarks are normally registered for a period of 10 years, although these registrations can be renewed indefinitely.[157]

Infringement of a registered trademark normally consists of displaying an identical or confusingly similar mark in combination with products or services that fall within the registered scope of exclusivity.[158] Proving infringement involves comparing the accused sign with the registered trademark and assessing whether the two are identical or confusingly similar. There is no requirement to prove that the accused party has actual knowledge of the registered trademark.[159]

Infringement of trademark can occur through the use of a domain name identical or confusingly similar to a registered mark. This creates well-known tensions, as domain names are (by definition) globally unique, while trademarks are not. To prove that the use of a domain name constitutes infringement of a registered trademark, a rights owners must normally prove that the domain name is identical or confusingly similar to the mark, and that the domain name is used in the supply of goods or services within the scope of exclusive use defined in the trademark registration.

Certification marks are a type of trademark that is used to demonstrate conformity with a given standard.[160] These marks are registered by a standards body, which then grants licences to use the mark subject to compliance with the relevant standard. Any person who supplies relevant goods or services bearing the mark that does not conform with the relevant standard risks legal action for trademark infringement.

A collective mark is a trademark that is used to identify the members of an association, such as a professional society. Having registered the relevant collective mark, the society can take action against those who use it without authorisation, and revoke authorisation from those whose membership has ceased.

3.8.2.4 Trade secrets

Trade secrets were traditionally protected under general tort law, giving persons who attempted to keep their secrets the right to take legal action against those who inappropriately obtained, used or disclosed these secrets. As the twentieth century progressed, a trend emerged to increase the legal protection of trade secrets. The position of individual US states has been significantly harmonised since the 1980s, and the US federal government adopted the Economic Espionage Act 1996 as a national trade secret law to deter trade secret theft [219, 220]. The European Union significantly harmonised its approach to trade secrets with effect from 2018 [221].

The subject matter of a trade secret is generally regarded as information that is secret, is valuable because it is secret and remains secret due to the reasonable efforts of the secret keeper. Subject matter can include information as diverse as an ingredients list, a method of manufacture, a customer list, an algorithm or details of a patentable invention prior to patent application and publication. Examples of current trade secrets in ICT include the finer details of Google's PageRank algorithm and various proprietary cryptographic algorithms.

Maintaining confidentiality is a core element of protecting a trade secret. Trade secrets can be protected indefinitely so long as secrecy is maintained.[161] Unfortunately, loss of trade secrets through acts of cyber industrial espionage is believed to be widespread and should be a source of major concern for cyber security practitioners [222]. Loss of confidentiality of patentable subject matter can be especially damaging, as publication of inventive details by a third party prior to patent application normally destroys patentability (as the invention then ceases to be 'novel').

Owners of trade secret rights can normally take legal action against persons who misappropriate their secrets. In some circumstances, owners of a trade secret can also take legal action against third parties who receive a trade secret from a mis-appropriator (see the discussion in Section 3.8.4.2).

3.8.3 Enforcement – remedies

Consideration of the impact of intellectual property law is incomplete without also considering remedies available to a successful litigant. (See Section 3.1.5.)

3.8.3.1 Criminal liability

In certain egregious circumstances, infringement of intellectual property – especially copyright and trademark – can be prosecuted as a crime. These prosecutions usually require proof that the infringing party was aware of the infringement and are often based on a pattern or practice of infringing these rights, *en masse*.

Those who violate legal prohibitions against anti-circumvention technologies for commercial advantage or financial gain, face a maximum sentence under US copyright law of 5 years for a first offence and 10 years for a second offence.[162] Under British copyright law a person who manufactures, imports, distributes, etc., a device intended to circumvent these protections faces a maximum sentence of 2 years.[163]

Some states classify the knowing misappropriation of a trade secret as a crime. The US adopted a national trade secret criminal law in 1996 [220]. These laws can serve as a basis

(not necessarily the only one) for the criminal prosecution of industrial espionage activity. Some states do not define misappropriation of trade secrets as a crime.[164]

3.8.3.2 Civil liability

A rights owner is normally able to take legal action against a person for infringement of intellectual property. Remedies for infringement normally include monetary damages, which may be calculated by reference to a so-called reasonable royalty, a statutory tariff or a demand that the infringer make an account of any profits – a demand to pay to the rights owner the economic benefit gained from the infringement.

Civil remedies may also include orders to seize, and perhaps destroy, products that infringe intellectual property rights. These orders are especially useful when interdicting shipments of 'knock-off' goods that embody trademark or copyright infringements.

With respect to trade secrets, persons in the US who suffered misappropriation of a trade secret traditionally brought legal action under the relevant law of their individual state. In 2016, the US national government adopted the 'Defend Trade Secrets Act 2016' amending the Economic Espionage Act to authorise private rights of action under federal law for the misappropriation of trade secrets [223].

A common civil remedy for the infringement of intellectual property is a court order addressed to the relevant infringing party to cease any ongoing infringing activity. In the context of patent enforcement, this can be especially devastating as an enterprise finds itself unable to continue manufacturing or selling an infringing product. In the context of trade secret misappropriation, this might include an order to cease manufacturing products employing the trade secret or an order prohibiting the publication of the trade secret. (Imagine how these would increase the Q value described in Section 3.1.5.)

In an online context, such orders might demand that content suppliers or server hosts take down content that infringes copyright or a trademark. Parties who enforce patents have sought orders to force a service provider to stop the operation of infringing services delivered via an online environment [224].

3.8.4 Reverse engineering

Reverse engineering, 'the process of extracting know-how or knowledge from a human made artefact', has generally been recognised as an accepted practice although treated differently within various categories of intellectual property law [225, 226]. Reverse engineering has historically been viewed as the flip-side of trade secret misappropriation. While trade secret law prohibits the misappropriation of a trade secret (e.g., industrial espionage, bribery etc.), the scientific study of a device sold and purchased in a public sale in an effort to learn its secrets has generally been viewed as 'fair game'. If a trade secret is successfully reverse engineered in this fashion and published, it ceases to be a trade secret.

Since the turn of the twenty-first century, however, the legal treatment of reverse engineering seems to have shifted following the adoption of laws prohibiting interference with anticircumvention technologies, generally making these activities more difficult [227, 228].

Most difficulties arise in the context of reverse engineering software products. Software licenses often contain onerous restrictions, including some limitations on reverse engineering generally and/or reverse compiling specifically. European law generally prohibits any restriction

on the ability of an authorised software user to observe and study the functioning of this software, and also grants these users the limited right to reverse compile specifically for the purpose of gaining interoperability information [229].

Pamela Samuelson has produced a useful comparative summary of this confusing landscape [230].

3.8.4.1 Circumventing copyright technological protection measures

Following the expansion of copyright law to prohibit the circumvention of technological protection measures, those who wish to meddle with these measures do so at their peril. The implementation of these laws provides some exceptions to liability for research in specified circumstances, although the precise circumstances vary. Each exception relied upon must be examined with care.

British copyright law, for example, includes a specific exemption to liability for circumventing protection measures in copyright works *other than a computer program*, for persons conducting research into cryptography, 'unless in so doing, or in issuing information derived from that research, he affects prejudicially the rights of the copyright owner' (CPDA s.296ZA(2)). In other words, one of these researchers might face peril under the law if they were to publish details that made it possible for others to circumvent the protection measures. There is no such general exception in British law for cryptography research involving the circumvention of measures on computer programs (CPDA s.296).

3.8.4.2 Testing a proprietary cryptographic algorithm

Security researchers hoping to test the strength of a cryptographic system normally require access to the relevant algorithm. This arises naturally from Kerckhoffs's Principle and is well-known to cryptographers. A person who wishes to test the security capabilities of an algorithm encounters practical difficulties when the manufacturer of the product employs a proprietary algorithm protected by trade secret and does not wish to disclose it for testing.

In the *Megamos Crypto* case *(Volkswagen v Garcia)*, the cryptographic product under examination (a special purpose processor chip used in automobile engine immobilisers and keys) was manufactured under license by the algorithm's developer. The testers (academic researchers) did not reverse engineer this product, which could have been accomplished using an expensive chip slicing technique. They chose instead to recover the algorithm by reverse engineering third-party software (Tango Programmer) that implemented the Megamos algorithm [231].

The researchers intended to publish the results of their analysis, which would have disclosed the algorithm. Parties who had an interest in the trade secret status of the algorithm brought legal action in the English courts to halt publication. The English high court was confronted with a request to prohibit publication of the research pending a full trial on the merits. The court seemed to accept that if the researchers had recovered the algorithm from the product itself using the chip slicing technique, there would be no case to answer. But the court found that there was a possibility that the third-party Tango Programmer software may have existed only as a result of trade secret misappropriation, and that the researchers should have been aware of this. The court issued a preliminary injunction prohibiting publication [232, 233]. The case was settled before trial commenced, and the researchers eventually published a version of their paper having redacted a component of the algorithm [234].

3.8.5 International treatment and conflict of law

The existence of intellectual property rights and assessment of first ownership are normally measured by reference to the place where these rights come into existence [212].

After creation in one state, the existence of copyright is generally recognised in most states around the world by the operation of various copyright treaties [235]. If an author writes some software while resident in State A, the copyright laws of State A are normally viewed as the source of authority for identifying the existence and first ownership of that copyright, while treaties oblige most other states to enforce that copyright within their territories (subject to limitations or exclusions granted by those states).

Grants of registered intellectual property rights (e.g., patents and registered trademarks) are made on a state-by-state basis. When identical or confusingly similar trademarks are registered in different states to different owners, the rights of each owner are equally valid within their respective registered territory. This can cause confusion when a trademark owner in one state makes an accusation of infringement against the owner of a second, nearly identical, trademark in another state [236].

Infringement, and defences to infringement, are normally assessed by reference to the law of the place where the intellectual property is infringed [212]. In cases of copyright, the courts show a persistent willingness to apply the rules (and limitations) imposed by their domestic copyright laws with respect to works that are distributed or displayed in-state via the Internet [78, 114]. The courts are also willing to enforce domestic patents against domestic instantiations of claimed inventions delivered as part of a global service offering [224].

3.9 INTERNET INTERMEDIARIES - SHIELDS FROM LIABILITY AND TAKE-DOWN PROCEDURES

[90, 91]

During the 1990s, policy makers around the world adopted special exceptions to shield certain communication service providers from liability for online content in prescribed circumstances. The changes were made in response to early cases that held these communication service providers liable under then-existing interpretations of content liability laws including copyright and defamation. These shields may be structured as affirmative defences, meaning that the use of the shield rests upon the ability of an accused to prove that they are entitled to be treated under the relevant exception.

In the European Union, these exceptions to liability were generally mandated by articles 12-15 of the Ecommerce Directive. These provide generalised liability shields in respect of 'mere conduit', 'hosting' and 'caching' services [193].[165] These principles are transposed into member state law in the usual fashion.

In US law, various shields from liability arising under copyright, defamation etc., have been adopted on a subject-by-subject basis.[166]

The widest scope of exemption from liability is normally afforded to those whose service consists of acting as a mere conduit for data.[167] These carriers are often exempted from liability without exception, although they may be ordered to filter traffic as part of a court-ordered enforcement plan [114].

Those who provide a service that consists of nothing more than hosting data are often exempted from content-related liability, unless and until they have reason to know that their infrastructure is hosting illicit content.[168] At this point, they often have an obligation to take down offending content 'expeditiously'. Confusion over how to implement this obligation resulted in changes to some laws which now specify in detail how take-down notices should be sent to hosting organisations, and how hosting organisations are required to reply to these notices.[169]

The topic of shielding service intermediaries from liability is not without controversy. Policy makers re-examine these liability exception provisions from time to time [237, 238, 239, 240]. In 2018, the US Congress amended the main US content liability shield so that it no longer protects any person in cases arising from claims of promoting prostitution or acting in reckless disregard of sex trafficking.[170]

3.10 DEMATERIALISATION OF DOCUMENTS AND ELECTRONIC TRUST SERVICES

As the age of ecommerce developed, concerns grew about how to transpose traditional methods for assuring information authenticity and integrity (e.g., signatures, seals, and indelible ink) into a form that could be used in fully electronic and online environments. Security technology experts responded with an array of new technologies (often based on PKI) intended to address these concerns.

This, in turn, prompted a series of legal concerns which potentially interfere with the utility of such technologies. These broadly fit into three categories. The first relates to the admissibility of electronic documents into evidence in legal proceedings. The second category are laws that threaten legal enforceability of communications made in electronic form. The third category relates to uncertainty about rights and responsibilities in the provision and use of identification trust services.

3.10.1 Admission into evidence of electronic documents

The admissibility[171] of electronic data as evidence into legal proceedings, once the subject of much suspicion by courts, has now become commonplace. Policy makers and judges have become increasingly confident as practitioners have developed forensic techniques to assure the authenticity and integrity of this data. Occasionally, local rules mandate special procedures for admitting electronic evidence. While forensic disputes about the weight to be accorded to this evidence persist, this is conceptually no different from arguments that might arise about any other type of recorded evidence.

3.10.2 Requirements of form and the threat of unenforceability

A requirement of form is any obligation imposed by applicable law that a given communication will be enforceable if and only if it takes a prescribed form. A failure to comply with an applicable requirement of form creates the risk that the subject matter of the communication will become unenforceable in whole or in part.

Different states have adopted differing requirements of form over the course of centuries in response to whatever policy issue was ascendant at the time. As a result, these requirements are remarkably diverse and can arise in a wide variety of circumstances.

Examples of requirements of form adopted by various states include rules demanding that, in order to be enforceable:

- certain legal notices must be delivered 'in writing';
- certain types of commitment must be in writing and 'signed' by (or 'executed under the hand of', etc.) the party against whom enforcement is sought;
- certain submissions to a state agency must be made using a specified form;
- certain contract clauses or notices that seek to restrict liability must be presented in a prominent fashion (e.g., all in uppercase letters, bold or italic font, etc) to the party against whom they are to be enforced;
- certain contract clauses that seek to restrict liability must be initialled by the party against whom they are to be enforced;
- a last will and testament must be delivered in writing and signed by the testator in the presence of a prescribed number of witnesses, who must also sign the document; and
- a document transferring title to certain types of property must be signed in the presence of a state judicial official, who must then affix an official seal to the document.

The examples above are merely intended to acquaint the practitioner with some of the more common types of requirement adopted within different laws by different states. Some states and some laws impose relatively few requirements, while others aggressively adopt a variety of such requirements.

Electronic trading systems developed as early as the 1960s (see Section 3.6.2.2) managed to work around many such problems. Requirements of form were overcome using a framework contract. Participants enter into written agreements (with wet-ink-on-paper signatures or following whatever other requirement of form might be imposed on these contracts) which constitute the foundation of the contractual relationships between participants.[172]

Newer trading platforms built on open standards, often directed to both businesses and consumers, made early gains by trading in subject matter (e.g., the sale of books and other small consumer goods) where contracts could be concluded, and payments settled, with few if any challenges based on requirements of form.[173]

There is a broad international consensus that it should be possible to create and conduct business relationships in an online environment. In 1996, the United Nations formally encouraged all states to enable online trading relationships [241]. Many states around the world contemporaneously adopted a variety of laws and regulations designed to enable the online conduct of various types of transactions, trading relationships, administrative reporting, court

filings, etc. Many were adopted in the specific context of enabling digital signatures and trust services, as discussed in Section 3.10.3.

The legal enforceability of communications related to other subject matter, especially topics such as the disposition of a deceased's estate and the transfer of title to immovable property, have been slower to transition to electronic platforms. These often retain significant requirements of form that make the electronic implementation of relevant communications impracticable unless and until states decide to amend their laws.

3.10.3 Electronic signatures and identity trust services

The emergence of modern ecommerce was contemporaneous with the emergence of identity trust services, specifically those that issue digital certificates that bind the identity of a person with a given public key in a PKI.

As engineering standards for these identity trust services began to emerge, two related legal questions surfaced for consideration by anyone who wished to provide or make use of these services:

- the extent to which a digital 'signature' produced using such a system would be accorded legal equivalence with a wet-ink-on-paper signature; and
- the nature of rights and responsibilities of various persons involved in the maintenance and use of these systems.

The question of legal equivalence for signatures is merely a sub-set of the more general problem of requirements of form discussed in Section 3.10.2. To the extent that various laws impose a requirement to 'sign' a communication, many states have adopted laws to provide legal equivalence to electronic signatures in most, but not all, circumstances.

The question of rights and responsibilities of persons involved in trust service arrangements is significantly more complex [242].

Consider first the potential liabilities of a certificate issuer in a standard three-corner operational model.[174] The law of negligence (see Section 3.7.1) immediately creates a number of challenges for any person operating as a certificate issuer, among them: what is the nature of the duty owed to a third party relying on a certificate; what are appropriate standards of care in this new operational model; and what harm is foreseeable when errors occur? Specific liability scenarios range from a system-wide disaster caused by the undetected compromise of a root certificate or technical flaw in the authentication mechanism, to the occasional (although recurring and perhaps inevitable) cases of improperly issuing a certificate following the misidentification of a signatory.

Consider also the potential liabilities of a signatory or a third party who relies on a certificate. Early policy debate focussed significantly on the degree to which signatures should be binding on the relevant signatory – especially when that person may have lost control of the signature creation device. This is a surprisingly old issue in law, commonly encountered in the context of cheque-signing machines, signature stamps, and the like, adopted by businesses, financial institutions, medical professionals, etc. Much of the policy debate over this issue appears now to be concentrated on the subject-matter laws governing specific use cases, such as those adopted to regulate electronic payment services [195, 196].

A lack of certainty over these issues has caused certificate issuers to seek out methods to

limit or otherwise rationalise their liability. A common strategy for limiting liability, entering into contracts which include limitation clauses, faces a significant problem. Forming a contract between the issuer and the relying person normally requires the communication of offer and acceptance between these persons (see Section 3.6). Most systems were designed to enable reliance without the constant intervention of presenting a user with new terms and conditions every time a relying party encountered a new certificate vendor. Similarly, certificate issuers might wish to warn relying parties about the scope of appropriate subject matter for which the certificates might be used.

The technology development community attempted to address these concerns by incorporating in the certificates specific data fields designed to communicate reliance limits and scope of use limits.[175] This strategy faced a different set of legal challenges. In practice, the certificates tend to be buried in rarely-accessed segments of the user interface. Further, a vast number of end users whose machines might be relying on these certificates would likely fail to comprehend the data presented in them, as certificate data tends to be presented in a highly technical fashion. In these circumstances, significant doubt has emerged about the ability to create an enforceable limitation of liability between the certificate issuer and the relying third party.[176]

States and legal experts intervened with a variety of recommendations, and then laws, attempting to address these issues [243, 244, 245, 246].[177] These laws, often identified with the term 'digital signature' or 'electronic signature' in their titles, typically adopted some combination of the following policy interventions:

- mandating the acceptance of electronic signatures as legal evidence;
- mandating the legal equivalence of electronic signatures that meet certain minimum technical characteristics to assure message authentication and integrity;
- instructing judges that electronic signatures (even those which provide little or no technical assurance of authentication or integrity) cannot be refused legal equivalence merely because they take an electronic form, but leaving open the possibility of denying equivalence for other reasons;
- imposing on a certificate issuer a duty of care owed to third parties who rely on certificates;
- reversing the burden of proof for negligent operation of a certificate issuer enterprise, so that an injured third party is no longer required to prove negligence but instead the certificate issuer is required to prove non-negligent operation;
- establishing frameworks for regulation to encourage higher technical and non-technical standards of care in the operation of a certificate issuance business;
- providing to certificate issuers the ability to limit their financial liability by presenting the limitation in the certificate itself, whether or not the relying third party actually sees this limitation; and
- providing to certificate issuers the ability to exclude liability for certain subject matter by presenting the exclusion in the certificate itself, whether or not the third party actually reviews this exclusion.

There is some degree of variance between states on how they have chosen to address these issues. Not all states have adopted all of these interventions. Many of these interventions are subject to a variety of additional conditions or limitations. A recurring theme concerns

the unwillingness of law makers to reduce rights otherwise afforded by consumer protection laws.

While some of these laws are general in nature, others are more narrowly drawn to address specific subject matter. In some cases, the law delegates authority to a regulatory body to adopt specific secondary legislation and/or technical standards on a subject-specific basis. Any practitioner who hopes to develop a platform in an area where requirements of form are commonplace must research and review applicable laws and regulations to reduce enforceability risk.

While much debate and discussion has focused on certificate issuers, signatories, and third parties who rely on certificates, another actor in this domain is more often overlooked: the person who selects which certificate issuers should be trusted by default. This 'certificate issuer selection' role is routinely undertaken, for example, by producers of consumer web browsers. This is perhaps inevitable, as the vast majority of end users would have no rational method of discriminating between good-quality and poor-quality certificate issuers. This raises the question of defining what duty of care these certificate issuer selectors might owe to end users.[178]

3.10.4 Conflict of law – electronic signatures and trust services

The nature of electronic signatures and trust services invariably implicates conflicts of law when relevant parties are in different states. Consider a certificate issuer located in State A, a signatory located in State B who procures a certificate and uses it to create digital signatures, and a third party relying on the certificate located in State C.

Assessing the legal equivalence of the signature can become complicated depending on which law imposes a relevant requirement of form that mandates a 'signature'. In the case of documents that purport to transfer title to immovable property, the legal equivalence question will almost certainly be answered by reference to the law of the state where the immovable property is located without any regard to the location of certificate issuer, signatory, or third party relying. (I.e., this could be a fourth State D.) The state where the immovable property is located is, in nearly all circumstances, the only one that can credibly assert enforcement jurisdiction over a title dispute as it is the only sovereign that could seize the property. In matters of a simple contract between a non-consumer signatory and non-consumer third party, the European courts should be willing to find formal validity of the contract if it meets the requirements of validity applied by the law of the state chosen by the parties to govern the contract, the law of State B, the law of State C, or possibly the law of either party's habitual residence if different from any of these (Rome I, Art 11) [197]. Where consumers are involved, the European courts would only find such a cross-border contract valid if it was deemed valid under the law of the consumer's habitual residence.

Determining the applicable law concerning limitations of liability is similarly complex. Consider, for example, the ability of a certificate issuer to rely on a limitation of liability granted under the trust services or digital signature law of State A. If the third party in State C brings a negligence action against the certificate issuer, the applicable tort law may well be the law of State C (see Section 3.7.6). The law of State C may not recognise the liability limitation otherwise granted by State A law, especially in cases where injured persons are acting as consumers. In other words, the value of any liability exclusion or limit granted by such laws becomes questionable when relationships cross borders.

3.11 OTHER REGULATORY MATTERS

This section will briefly address additional miscellaneous regulatory topics that a cyber security practitioner might be expected to encounter.

3.11.1 Industry-specific regulations and NIS Directive

A wide variety of single-industry regulators have embraced cyber security within the framework of their larger role regulating subject industries [247]. Many financial services regulators, for example, in their role as regulators of operational risk in financial services, have always had some degree of subject matter jurisdiction over cyber security operations. Details of cyber security risk management have increased in prominence within financial services regulation and can be expected to continue to feature prominently [248].

Similarly, within professions that owe legally mandated duties of confidentiality to clients or whose clients enjoy legally mandated privileges prohibiting disclosure of client-professional communications (e.g., lawyers and physicians) professional regulators have become increasingly attuned to problems of cyber security.

Many of these various regulations include obligations to report or disclose security breaches. Such disclosure requirements operate in addition to any obligations imposed by data protection law (see Section 3.4.7). This creates a potentially confusing landscape when considering disclosure obligations [249].

As states have begun to focus more heavily on cyber security risks, existing regulators have been encouraged to bring cyber security into their supervisory and regulatory frameworks especially in the context of critical national infrastructure.

In the European Union this has been accelerated by the adoption of the EU directive on network and information systems (NIS Directive) [250]. article 14 of the Directive requires member states to ensure that 'operators of essential services':

- 'take appropriate and proportionate technical and organisational measures to manage the risks posed to the security of network and information systems which they use in their operations';
- 'take appropriate measures to prevent and minimise the impact of incidents affecting the security of the network and information systems used for the provision of such essential services, with a view to ensuring the continuity of those services'; and
- 'notify, without undue delay, the competent authority or the CSIRT of incidents having a significant impact on the continuity of the essential services they provide'.

The UK implementation devolves responsibility for regulatory oversight to relevant competent authorities - instructing existing industry regulators to adopt and enforce cyber security obligations set out in the Directive [251].

Efforts to use regulation to heighten cyber security in society continue to take many different forms. Debate continues about which models of regulation are most effective [252].

3.11.2 Encouraging increased cyber security for products and services

The emergent Internet of Things and the accompanying growth of cloud-based services create increased risks from cyber security breaches to both consumers and business enterprises. Policy makers have begun to adopt legal frameworks for certification of compliance of products and services with various cyber security standards.

In the European Union, certification activity is expected to operate within the framework of the EU Cyber Security Act [253]. (See also discussion of certification marks used by public and private standards bodies in Section 3.8.2.3.)

Relevant security standards may emerge from a variety of sources [254, 255].

3.11.3 Restrictions on exporting security technologies

States have long imposed restrictions on the export of goods intended for use in armed conflict. These laws grew significantly during the Cold War, as Western bloc states sought to restrict the flow of defence technologies to the Eastern bloc.[179] These export limitation regimes also apply to 'dual use' goods: sensitive products that have legitimate uses in both peace and war. Although a surprisingly wide variety of dual use products (and services) have been caught under the terms of such restrictions, those that are caught because they embody certain cryptographic functions have been especially contentious in the field of cyber security.

Prior to the 1990s, the US (and other states) regulated the export of strong cryptographic products with an extremely broad brush. Export prohibitions were framed in such expansive language that almost any export required prior government licence. At the beginning of the 1990s, the implementation of strong cryptography in software for general purpose computers, the growing body of non-governmental research work into cryptography, the availability of the Internet as a means to distribute know-how and source code, and the increasing pressure for reliable standards-based cryptographic implementations in support of cyberspace infrastructure, collided with these same export restrictions.

In the US, a series of legal actions under US free speech law (i.e., the First Amendment to the US Constitution) were brought challenging the validity of export regulations as applied to cryptographic software. The argument presented proceeds, in essence, as follows: source code is expressive, expressive content is protected speech, therefore source code is speech, the export regulations are therefore a governmental prior restraint on speech, as a prior restraint the regulations must be extremely narrowly tailored to address a clear danger, but the regulations are in fact very broadly drawn and therefore do not meet constitutional muster. The US courts struggled with the concept of whether source code was protected 'speech'. Eventually, in *Junger v Daley* (2000), the US Court of Appeals for the 6th Circuit held that source code was speech and found the US export regulations unconstitutionally over-broad [256].[180] No doubt in response to this and similar legal challenges in other US Circuits, combined with heavy lobbying by the ICT industry, the US government issued revised export regulations to create significantly more limited restrictions on cryptographic exports [257].

Many states including the US continue to maintain export restrictions on certain dual use products, including some implementations of cryptographic technology. Anyone engaged in the production of these products should review applicable laws carefully, as violations can be prosecuted as crimes.

3.11.4 Matters classified as secret by a state

Practitioners who are employed or engaged by states are routinely subject to laws that mandate secrecy of certain information classified as secret by those states. Most commonly, this arises in an environment where the disclosure of relevant secrets could harm the defence of the state, the integrity of a police investigation, the safety or efficacy of persons conducting state sponsored espionage activity, etc.

These laws can sometimes be used to intervene and classify as secret the research and development work of third parties. Practitioners may also come within the remit of these laws when state security officials choose to disclose certain classified information relating to cyber threats.

These laws tend to authorise extremely severe criminal penalties for those who violate them.

3.12 PUBLIC INTERNATIONAL LAW

[87]

Public international law[181] is the body of law that regulates relationships among and between states, which for this purpose includes international governmental organisations but excludes constituent states of a federal state. Sources of public international law include treaties, widely accepted international norms and customs, and decisions of international tribunals.

Only states are said to have 'standing' to enforce claims arising under public international law. Non-state persons are normally unable to take legal action against states for violation of public international law. A non-state person may hold the right to take legal action against their home state for failure to implement obligations imposed upon the state by public international law, although states normally must affirmatively grant these rights to non-state persons [138].[182]

Similarly, international law normally seeks to regulate the behaviour of states rather than the actions of their residents or nationals.[183] Cyber operations undertaken by a non-state person in State A against persons or infrastructure in State B normally do not constitute a violation of international law, unless the action can be attributed to State A or to some other State C (see Section 3.12.1.) This action by a non-state person could, however, serve as a legal justification under international law for State B to take some form of proportionate countermeasure against such persons or equipment in State A as an act of self-defence ([87] at R.4, cmt.2).

It has become widely accepted that principles of public international law should apply to actions taken with respect to cyberspace [85, 86, 87, 258]. Having said this, states can and do diverge in their opinions of how these principles should be interpreted in the context of specific types of cyber operation. At the time of writing, the most comprehensive and most widely accepted published source of analysis on the application of international law to cyber operations is the restatement of international law found in the Tallinn Manual 2.0 [87].[184]

3.12.1 Attributing action to a state under international law

Attribution is the process of determining if a state is legally responsible under international law for a given action. A series of international law doctrines are used to define when a state is responsible for a given act ([87] at R.14-19).[185]

A given action might be legally attributed to a state if, for example:

- the action is undertaken by an agent or officer of the state (such as an active on-duty member of the state's military or police); or
- the action is undertaken by a non-state person (such as a technology services provider) under the direction, or with the active encouragement, of state officials.

In extreme circumstances, if illicit activity is regularly initiated by non-state actors from cyber infrastructure inside the territory of a state, and that state remains willfully blind to that activity or otherwise fails to exercise appropriate due diligence in attempting to identify and restrain the illicit activity, then it may be possible to attribute the illicit activity to that state. This theory is not without controversy ([87] at R.6-7; R.14, cmt.3) [259].

3.12.2 State cyber operations in general

Public international law is founded on the principle of territorial sovereignty.[186] A state is said to be sovereign within its own territory, and also has the right to conduct activities outside of its territory consistent with international law.

International law generally prohibits one state from violating the sovereignty of another ([87] at R.4). States are, however, entitled to take appropriate countermeasures in response to a second state that has violated the obligations it owes under international law to the first ([87] at R.20-26). Countermeasures are actions that would normally violate international law that are directed against the second state in an effort to encourage it to comply with its obligations under international law.

Countermeasures must be proportional to the complained-of violation of international law by the second state. Countermeasures in response to an illegal cyber operation might consist of cyber or non-cyber responses. Thus countermeasures responding to a cyber operation which violated international law could legitimately consist of so-called 'kinetic' responses, cyber operational responses, or economic sanctions [260]. This raises a recurring challenge when attempting to understand how to assess the relevant 'proportionality' of non-cyber countermeasures to a cyber operation that violates international law [261].

A cyber operation of one state directed against another state is normally contrary to the principles of international law if it interferes in the affairs of the second state ([87] at R.66). A cyber offensive operation such as a DDoS operation, for example, would constitute interference if used to coerce the second state in a manner designed to influence outcomes in, or conduct with respect to, a matter reserved to the target state ([87] at R.66, cmt.8&19). The outcomes in question need not be physical in nature, and can include domestic political outcomes ([87] at R.66, cmt.20).

A cyber operation of one state directed against another state is normally contrary to principles of international law if it constitutes a use of force or threat of same ([87] at R.68-70).

A state that is the victim of an 'armed attack' is entitled to take proportionate countermeasures, including the use of force ([87] at R.71). Actions that constitute an armed attack are a sub-set

of acts that constitute 'use of force' ([87] at R.71, cmt.6). Finding the dividing line between them is challenging and is generally measured by reference to the scale and effects of the complained-of act. In the context of discussing the Stuxnet operation, for example, the Tallinn Manual 2.0 experts agreed that if Stuxnet were to be attributed to a state it would constitute a use of force; but they were divided on whether the scale and effects of the operation were sufficient to constitute an 'armed attack' ([87] at R.71, cmt.10).

Because of the uncertainty over when the scale and effects of a cyber operation are sufficiently severe to constitute an armed attack, it has been suggested that some states have adopted a strategy using this uncertainty to conduct cyber operations in a 'grey zone' somewhere between peace and armed conflict [261].

3.12.3 Cyber espionage in peacetime

Cyber espionage, *per se*, during peacetime is not generally considered a violation of international law ([87] at R.32) [142, 143, 262].[187]

Cyber surveillance and evidence gathering activities conducted from within the territory of one state against persons or equipment in another state would therefore not necessarily constitute a violation of international law. Espionage methods, however, could easily violate the domestic criminal law of the second state (e.g., obtaining unauthorised access to computers). Furthermore, methods that support espionage by harming equipment within the territory of the second state would constitute a violation of that state's sovereignty and (if sufficiently damaging) could amount to a use of force.[188]

A specific example of this principle applies to state efforts to tap submarine communication cables for the purpose of intercepting communications. If a state covertly taps cables in international waters without significantly interrupting their functionality, this very likely does not constitute a violation of international law. If one state places the tap within the territorial waters of a second state, however, the operation constitutes a violation of the second state's sovereignty ([87] at R.54, cmt.17).

3.12.4 Cross-border criminal investigation

Actions by one state that constitute the exercise of police power within the territory of another (unrelated)[189] state normally constitute a violation of that state's sovereignty under international law. This is easy to see in cases where police powers involve the use of force in person, such as searching physical premises, or arresting or interrogating a suspect located in the second state.

Acts of surveillance conducted from within the territory of one state that do not involve physical contact by that state's agents with the territory of another state are more complex to analyse. While state remote surveillance actions on their own might not constitute violations of international law (see Section 3.12.3), state evidence gathering methods (such as covertly taking remote command of a botnet controller or other equipment located on the territory of another state) can constitute a violation of the other state's sovereignty under international law ([87] at R.4, cmt.18) and may also constitute the commission of a crime under the other state's domestic law.

Nonetheless, it is well documented that remote cyber surveillance and evidence gathering activities are conducted by the law enforcement agencies of various states from time to time

with the express or implied authorisation of the investigating state, and directed against cyber infrastructure located in another, non-consenting, state [92].

3.12.5 The law of armed conflict

Upon commencement of armed conflict, the conduct of activity within the context of that conflict is said to be governed by the 'law of armed conflict' (also known variously as the 'law of war' and 'international humanitarian law'.)[190] State cyber operations conducted as part of an armed conflict are assessed by reference to the law of armed conflict. The Tallinn Manual 2.0 addresses this topic in some detail ([87] at Part IV, R.80-154).

This field is the subject of extensive study and debate by the military leadership of many states, which invest significant time and effort producing legal guidance for their military commanders concerning that state's interpretation and implementation of the law. Some of these are published and available for public review [263, 264, 265]. The US DOD Manual specifically addresses cyber operations [264].

The precise meaning of 'armed conflict' is subject to some disagreement, although it is widely understood that armed conflict can (and often does) exist in the absence of any formal declaration of war ([87] at R.80, cmt.2). During the course of armed conflict, some international legal obligations (e.g., the 1944 Chicago convention on civil aviation) are suspended or otherwise altered as between belligerent states engaged in hostilities.

Key principles that underpin the law of armed conflict include:

- *Military necessity*: a state may use such force as is necessary to defeat an enemy quickly and efficiently, provided that such force does not violate other principles of the law of armed conflict.
- *Humanity*: a state may not inflict suffering, injury or destruction that is not necessary to accomplish a legitimate military purpose.
- *Distinction (aka Discrimination)*: a state must endeavour to distinguish military persons and objects from civilian persons and objects. This imposes an obligation upon a belligerent state to distinguish its own military person and objects from civilian persons and objects, as well as working to distinguish enemy state military and civilian persons and objects.
- *Proportionality*: a state may not act in a manner that is unreasonable or excessive.

A recurring issue of discussion among experts concerns what is required to treat a cyber operation, on its own, as an 'attack' under the law of armed conflict. The Tallinn Manual 2.0 refers to such an operation as a 'cyber attack', which the expert group defines as a cyber operation 'that is reasonably expected to cause injury or death to persons or damage or destruction to objects' ([87] at R.92).[191] The characterisation of a cyber operation as a cyber attack under international law is critical, as the law of armed conflict limits how states carry out such attacks.

Belligerent states are to avoid targeting their attacks (which would include cyber attacks) against civilian persons or objects ([87] at R.93, 94 & 99). Exceptions arise with respect to civilians who participate in armed conflict ([87] at R.96, 97).

Although the principles of the law of armed conflict are not significantly in dispute, how to interpret and apply these in the context of specific cyber operations raises a series of recurring

questions. For example, many legal experts take the view that the principle of not targeting cyber attacks against civilian objects applies only to tangible objects and that intangible data, as such, does not fall within the legal definition of 'object' [266]. This was the view of the majority of the Tallinn Manual 2.0 expert group, although a minority of that group felt that intangibles such as data should count as 'objects' if the effect of damaging or altering such data was sufficiently significant ([87] at R.100, cmt.6). There is wider agreement, however, that an operation that targets and alters data, which in turn causes injury to persons or damage to property, does rise to the level of cyber attack ([87] at R.92, cmt.6).

Another difficulty in application concerns the intermingling of military and civilian cyber infrastructure. Under the law of armed conflict, if an object is used for both military and non-military purposes it becomes a military objective ([87] at R.101). This leads to the possibility that significant components of public cyber infrastructure, including dual-use data networking and cloud services infrastructure, could be characterised as a legitimate target of cyber attack in time of armed conflict (subject to other legal limitations such as the need to respect the principles of humanity and proportionality) ([87] at R.101, cmt.4-5). Some have argued that such outcomes point to the need to reconsider how public international law should operate in this context [266].

3.13 ETHICS

Cyber security practitioners often find themselves operating in positions of trust, where special knowledge and skills potentially give them asymmetric power to influence or disrupt the affairs of their clients or other members of the public. Those who act outside of a specific client relationship, such as product developers and academic researchers, exercise special skills in a manner that could cause harm to a wide variety of third parties. Practitioner activities often take place behind closed doors away from the glare of public scrutiny. This is a volatile mix. Ethical norms might assist in curbing behaviours that abuse positions of trust or otherwise present significant risk to the public.

Early cyber security ethics guidance focused significantly on legal risk management such as liability arising under intellectual property, data protection and privacy laws [267]. Although practitioners should remain aware of laws that apply to their actions, compliance with the law, on its own, may be insufficient to guide a practitioner to ethical action.[192]

As a practice that is generally conducted in the absence of formal state professional regulation, it is difficult to identify generalisable norms that are expected to apply to activities undertaken by security practitioners.[193] This section will survey some of the recurring issues and potential sources of guidance.

3.13.1 Obligations owed to a client

A review of some obligations normally owed by regulated professionals to clients may be helpful as societies (and various nascent professional bodies) continue to develop approaches to obligations that should be owed by cyber security practitioners to their clients.

At the very least, one can identify various duties of care that arise under contract or tort law to conduct services and other activities that involve risk in a reasonable fashion and with appropriate expertise. Product designers similarly owe various legal obligations under the normal rules of tort law.

Regulated professionals are normally expected to act in the best interests of their clients, to avoid conflicts of interest and to maintain the confidentiality of client affairs. While affirmatively adopting these types of obligation by contract is often non-controversial, difficulties can arise when a security practitioner and client disagree about the most appropriate course of action in specific circumstances.

Challenges can arise with respect to non-mandatory disclosure of evidence to interested third parties.[194] If a practitioner discovers evidence of wrong-doing and there is no supervening legal obligation to report that evidence, the practitioner and client might disagree concerning the disclosure of such evidence to interested third parties such as relevant police authorities, CERTs or tort victims.

These cases can be difficult to navigate. In cases of evidence of economic crimes directed against the client (e.g., petty theft), the client may view public disclosure as more damaging than handling the matter solely on an internal disciplinary basis. In cases where an employee is found to have harmed a third party through tortious action such as negligence, disclosing this evidence to the victim may work to the financial detriment of the client's company through vicarious liability.

Other difficult cases arise when the interests of the practitioner and their client are not aligned. Some professional ethics systems, for example, allow a regulated professional to disclose some parts of a client's confidential information as part of a legitimate bill collecting activity (e.g., by filing a legal action for breach of contract relating to the delivery of confidential services). Such disclosures must normally be limited to information that is necessary to pursue collection and may come with obligations to seek appropriate protective orders from the courts.

Actions by a practitioner that interfere with the proper functioning of their client's infrastructure in an effort to exercise undue influence over the client are unsavoury at best, and might cross a line into criminal conduct at worst. An express or implied threat of such action seems no better.

It remains to be seen whether cyber security practitioner-client relationships will become the subject of formal state regulation or licensure in due course.

3.13.2 Codes of conduct

Various professional bodies have published codes of conduct and ethical guidelines for cyber security practitioners. Many of these refer to high-level ethical principles without the more detailed guidance that is necessary to assist practitioners with interpretation of the principles.[195]

Examples of two more recent and carefully considered codes of conduct are presented below for consideration. One is framed as a code of general applicability and one is built around a defined business process.

The Association for Computing Machinery can trace its history to the mid-twentieth century and maintains a global membership of more than 100,000 [268]. The ACM Code of Ethics and Professional Conduct was extensively revised in 2018 to take account of the impact of data connectivity [269]. The revised ACM Code provides multiple points of guidance relevant to the field of cyber security. The ACM also provides supplementary materials to assist in understanding and applying the Code [270].

The ACM Code clearly demonstrates the difficulties of balancing ethical imperatives. In its admonition to avoid harm (Section 1.2), it states there is an 'additional obligation to report any signs of system risks that might result in harm'. While the Code addresses the possibility of 'whistle-blowing' as a reporting technique in appropriate circumstances, it also cautions that 'capricious or misguided reporting of risks can itself be harmful. Before reporting risks, a computing professional should carefully assess relevant aspects of the situation'.

By contrast, CREST was established in the UK in the early twenty-first century originally as a membership body for firms that supply penetration testing services.[196] At the time of writing, it has more than 180 accredited member firms [271]. Penetration testing typifies a service that should be of significant concern to the public: information asymmetry means clients are generally unable to distinguish good practice from bad, services are supplied confidentially away from public scrutiny, and practitioner errors can cause disproportionate harm to clients or third parties. The CREST Code of Conduct for CREST Qualified Individuals provides guidance on numerous topics relevant to delivering these services including service methodology, ethical business practices, and obligations owed to clients [272]. The CREST Code also provides a client complaint mechanism and the organisation reserves the right to expel from membership those who fail to adhere to the CREST Code. To the extent that clients mandate that suppliers of relevant services maintain CREST membership, these mandates may slowly migrate CREST from a purely voluntary membership association into a *de facto* regulator of those who supply these services.

By historical standards, cyber security presents a relatively new set of methods and processes which are at best poorly understood by the public. Generalised codes like the ACM Code are helpful, as they guide a community of persons with relevant technical expertise who may work in fields as diverse as research and development or security management. Service-specific codes like the CREST Code are helpful, as they focus clearly on specific high-risk services. Codes of conduct will undoubtedly continue to develop as the impact of cyber security practitioner activity on the public becomes better understood.

3.13.3 Vulnerability testing and disclosure

The process of searching for, finding, disclosing, and acting in response to, security vulnerabilities causes recurring ethical (and legal) issues [273, 274].

3.13.3.1 Testing for vulnerabilities

Practitioners who test for security vulnerabilities should consider carefully the nature of their activities. The mere act of studying and analysing objects such as tangible hardware products, locally resident licensed software, or published cryptographic primitives and communication protocols, is normally uncontroversial. It is difficult to draw a line of causation from the mere act of analysis to public harm.

Practitioners should be careful, however, to consider the source of the security object under study. There may be a distinction, for example, between reverse engineering a silicon chip to discover the functionality of a trade secret cryptographic scheme and reverse engineering third-party software of suspicious provenance that embodies the same secret methodology. Although the first might be generally permissible, the second may constitute a violation of trade secret rights and result in liability or restrictions on limiting the ability to publish results [232, 233] (see the discussion in Section 3.8.4.2 and Note 201).

When vulnerability testing is conducted remotely, the testing methods can raise additional issues. Practitioners must first remain cognisant that unauthorised efforts to gain access to a computer system are often defined as a crime (see Section 3.5). As stated in the ACM Code Section 2.8, 'A system being publicly accessible is not sufficient grounds on its own to imply authorization' [269].[197] If practitioners are testing in response to a 'bug bounty' program, they should review carefully the terms of the program to assure that they are not exceeding the scope of authorised testing activity.

Practitioners should also consider the potential impact of their testing methods on the stability of public or private infrastructures, including those that are the target of testing as well as intermediary and third-party systems.

3.13.3.2 Disclosure of vulnerabilities

Those who find security vulnerabilities face a choice of what to do with their new-found knowledge. Choices exist on a spectrum from making no disclosure, to publishing every detail immediately to the world at large. In between these two extremes lie an array of possibilities.

Those who make no disclosure choose to do so for different reasons. Some wish to make no disclosure in an effort to avoid complicated problems of ethics and potential liability. It is difficult to reconcile this position with the ethical principle expressed in the ACM Code Section 2.8 'to report any signs of system risks that might result in harm' [269].

Some who operate in support of state security agencies may wish maintain the secrecy of a vulnerability after deciding that the risks and benefits of disclosure outweigh the risks and benefits of maintaining secrecy [275, 276, 277, 278].[198] The ethics of this 'equities' balancing process is a topic of continued debate [277, 279].

Finders who choose to make an immediate full public disclosure of vulnerabilities without any prior warning to any effected person may do so for a variety of reasons. Some suggest that this is the only certain method of encouraging remediation efforts by developers. Some do

not wish to invest in the time-consuming process of staging the private and public disclosures described below. Some fear that engaging with developers will prompt a legal intervention prohibiting disclosure.[199] It is difficult to reconcile these arguments with the guidance from the ACM Code to minimise harm.

Many practitioners follow a principle known as 'responsible disclosure'. The idea is to disclose first on a confidential basis to a person or group of persons who may be able to remediate or mitigate the impact of the vulnerability. After a period of time has elapsed, the finder might then proceed to a second stage of public disclosure. Second-stage public disclosure is often justified by the practitioner on the theory that publication will enable other practitioners to study and avoid similar vulnerabilities, and/or incentivise product and service providers to remediate the vulnerability.

At the time of writing, there appear to be no generally agreed principles on the proper conduct of responsible disclosure. Points of divergence include:

- how to manage private disclosure when the vulnerability forms part of a widely adopted industry standard;
- how to manage private disclosure when the vulnerability is found in a product which forms a component or sub-component in downstream products;[200]
- defining the appropriate length of time between private and public disclosures;
- defining what circumstances, if any, mandate an indeterminate delay to public disclosure; and
- defining how to respond if the relevant vendors or purchasers of compromised products disagree with the finder about the wisdom or timing of public disclosure.

Public disclosure of vulnerabilities could also create tortious liability for a disclosing finder, especially if the process or sequence of disclosures is poorly managed or the vulnerability is misdescribed.

Practitioners who seek to justify publication on the basis that it fits generally within the rubric of 'responsible disclosure' may receive a poor reception from state authorities [232, 233].[201]

Various efforts to obtain financial benefits from disclosing a vulnerability also lead to debate. Accepting a financial reward from a vendor pursuant to a published 'bug bounty' programme seems now to be widely accepted, especially as the vendor controls the terms of the programme [280]. Other more controversial tactics to monetise findings include:

- requesting a financial 'bug bounty' from a vendor as a condition of disclosure when the vendor has no existing bug bounty programme in place;
- selling knowledge of the vulnerability to a third-party broker, who then re-sells the information; and
- engaging in market trade activity (e.g., short-selling publicly traded shares of a vendor) in an effort to profit financially from advance knowledge that a vulnerability will soon be published [281].

Practitioners who find vulnerabilities during the course of their work as security researchers must further consider the extent to which they may be accountable to their employer or funder for any financial benefits obtained.

3.13.3.3 Facilitating and acting on vulnerability disclosures

Product and service vendors should consider how they can facilitate and then act upon vulnerability disclosures in a manner that minimises harm to customers and third persons.[202]

Key principles to facilitate proper vendor handling of vulnerability disclosures include: publishing acceptable methods for finders to disclose vulnerabilities to the vendor, working diligently to verify the vulnerability once it is disclosed, developing remediation or mitigation strategies, disseminating fixes, working with supply chain partners, and providing feedback to finders.

A framework to develop specific vendor policies and processes can be found in ISO/IEC 29147 (the process of receiving and handling information about vulnerabilities) and ISO/IEC 30111 (the process of verifying and remediating vulnerabilities) [282, 283]. State agencies have also published varying guidance on this topic, which is revised from time to time [284, 285].[203]

3.14 CONCLUSION: LEGAL RISK MANAGEMENT

Section 3.1.5 introduced one way for a given Bob to think about the risk of legal action from a given Alice. There were many points implicit in that introduction, however, including Bob's awareness of: Alice, her right of action, as well as details of applicable substantive and procedural law. On its own, the function presented is most helpful after-the-fact - after Bob receives a threat of legal action from Alice. The purpose of this Section is to consider legal risk management before-the-fact.

Anyone seeking to understand legal risk often begins with an information deficit. Simply learning about the many laws and regulations that can or should influence the operation of a single enterprise can be prohibitively time-consuming and expensive.

This problem multiplies according to the number of jurisdictions with which a person may be dealing – a significant challenge if cyberspace truly enables contact with every jurisdiction in the world. In the modern era of more than two hundred sovereign states recognised under public international law, plus hundreds of states that are members of a federal or similar structure, plus untold tens (or hundreds) of thousands of additional municipal jurisdictions with varying degrees of law making and enforcement authority, merely discovering the content of applicable laws and regulations and assessing enforcement risks can be a monumental task. Private law obligations imposed by contract and (potentially voluntary) self-regulatory systems complicate matters further. In a field where multinational contacts and relationships are commonplace, considerations of the effective limits of state power are also appropriate.

What follow are a few subjects for consideration when constructing a legal risk management framework.

Identify subject matter areas of greatest risk. The nature of the activities undertaken by a person helps to identify which laws and regulations will be of most significance to that person. For example, banks, telecommunications infrastructure providers, and providers of medical and legal services are always cognisant of their need to seek and maintain appropriate licenses for their activities. Providers of gaming (gambling) services are also very attuned to the wide variation of laws that apply specifically to their operations. And all businesses are extremely aware of the need to understand tax reporting, collection, and payment obligations.

Consider the impact on human life. A strict cost-benefit analysis may be useful when making operational decisions, but becomes problematic where matters of human life and safety are

concerned. Laws and regulations adopted to protect human life and compensate for personal injury should be accorded special respect. A blatant disregard of such rules raise significant moral and ethical concerns and can also result in exceptional or punitive measures when these rules are enforced.

Conduct due diligence that is aligned with identified risks. Nobody instructs a lawyer to 'Go and find every law in the world that arguably might apply to anything I do.' A typical due diligence strategy involves first identifying and investigating the laws that could destroy or bankrupt an enterprise. Other laws and regulations may become increasingly significant as an enterprise grows or changes character. Foreign laws become increasingly significant as the enterprise makes increasing contact with new jurisdictions.

Consider the practical limits of territorial enforcement jurisdiction. In the era of online commerce, some enterprises become paralysed with fear about the potential legal obligations to hundreds of states whose residents might gain access to site content. Those that remain paralysed may go out of business. Most of the others try to adopt pragmatic approaches that include good faith efforts to filter content or otherwise block access to residents of states that characterise one's products or services as illicit.

Consider the relative cost of breaching a (non-criminal) legal obligation. Committing a crime is different from failing to honour a civil obligation. There are times when the cost of answering a civil legal action is less than the cost of compliance. Most commonly, this occurs in the context of a commercial contract which has become uneconomic to perform, or a civil regulatory requirement with a fixed financial penalty. In appropriate circumstances, a person might reasonably conclude that repudiating its civil obligation (and accepting the risk of a legal action for money damages) is less expensive than fulfilling the corresponding obligation.

Consider the risks to one's own personal reputation, safety and liberty. Cyber security practitioners are sometimes confronted with situations where they are tempted, or instructed, to violate criminal law. Those who face this circumstance should remember that they may personally suffer the consequences of their actions, irrespective of whatever incentive has been provided by an employer or client.

Consider the likelihood of enforcement. There are times when persons who have legal rights choose not to enforce them. For example, the risk of legal action from an individual natural person who has suffered a *de minimis* loss as a result of a minor business tort may be diminishingly small. If the rights of thousands or millions of these persons can be joined together in a class action lawsuit, however, the risk increases significantly.

Consider the challenges of collecting, preserving, and presenting evidence. Efforts to enforce legal rights and efforts to defend against claims all hinge on one's ability to prove, or to rebut an adversary's efforts to prove, the underlying facts in dispute. Consider what issues will require proof when an adverse party seeks to enforce a legal right, and how one can collect and preserve evidence to an appropriate forensic standard in anticipation of the need to defend against this effort. Practitioners are also cautioned to explore the parameters of any applicable document or data retention policy, which involves the routine and regularly scheduled destruction or deletion of documents. While the routine destruction of documents in accordance with a carefully defined governance policy is usually permissible, these procedures normally must be suspended to the extent that documents may be relevant to any legal action that has commenced or been threatened. Any attempt to destroy evidence that might be relevant to such a legal action often constitutes a violation of the law and can result in severe consequences.[204]

Consider vicarious liability. The only certain way to reduce vicarious liability is to influence employee behaviour to reduce the number of acts that are tortious or otherwise violate applicable regulations. Internal governance documents intended to reduce liability to third parties should therefore be written with the goal of influencing this behaviour.

Consider localising risky activities in separate limited liability legal persons. Lawyers routinely counsel business clients concerning the creation and structuring of separate legal persons in an effort to contain liabilities within defined pools of investment capital. This is a complex subject that requires careful navigation.

Consider risks that are external to the legal system, per se. In some circumstances, the greatest risk arising from a legal action or a threat of legal action has almost nothing to do with laws or regulations, as such. Consider, for example, the impact of a potential legal action on the reputation of an organisation or the impact of an adverse finding on the maintenance of relevant state licenses to conduct business.

Consider changes to law or enforcement policy that are likely to arise. Societies and policy makers are generally becoming more aware of the impact of cyber security. As this awareness increases, states and their agents may increase enforcement activities, re-examine assumptions about existing policy, and intervene rapidly with amended or new laws.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	Goldsmith & Wu [78]	Tallinn Manual 2.0 [87]	Rowland, Kohl & Charlesworth [90]	Murray [91]	Walden [92]	Holt, Bossler, & Seigfried-Spellar [93]	Clough [94]	Handbook on EU data protection law [153]	GDPR Guidelines [154]
3.2 Jurisdiction	X		X	X	X	X	X		
3.4 Data protection			X	X				X	X
3.5 Computer Crime					X	X	X		
3.6 Contract			X	X					
3.8 Intellectual property			X	X					
3.9 Internet intermediaries			X	X					
3.12 Public international law		X							

NOTES

[1]Civil procedure governs process-related matters in non-criminal legal proceedings, such as the form of pleadings submitted to the court (including the size and shape of paper on which they are written), time allowed for defensive pleadings and replies, methods of serving notice on parties, expanding or contracting the number of parties in a law suit, the scope of mandatory disclosure of potential evidence, case management orders, methods of appealing adverse decisions, etc. Examples of these rules can be found in the [England and Wales] Civil Procedure Rules, Title 28 of the United States Code, and the [US] Federal Rules of Civil Procedure.

[2]Criminal procedure governs process-related matters in criminal proceedings. Because criminal proceedings place at risk the individual liberty of the accused, these rules are heavily influenced by human rights law, can be significantly different from civil procedure, and thus are often maintained separately from civil procedure rules. Examples of these rules include the [UK] Criminal Procedure and Investigations Act 1996, the [US] Federal Rules of Criminal Procedure, etc.

[3]Rules of evidence govern the presentation and examination of evidence before a tribunal. This can include matters such as the prohibition of some categories of hearsay evidence, presentation of so-called 'computer evidence', introduction and examination of expert testimony, permissible examination and cross-examination techniques, etc.

[4]Cyber security practitioners are not alone in this respect. Highly experienced lawyers routinely require guidance from 'local counsel' who are retained specifically to assure compliance with these rules when attempting to manage multi-state disputes.

[5]See Note 13

[6]Anecdotal evidence gathered by the author over many years of ICT-focused international commercial legal practice, however, strongly suggests that many of the norms expressed in this knowledge area are also reflected in systems of civil law (see Note 14). There is no claim that the norms presented here would necessarily be found in other systems of domestic law such as those founded on religious doctrine or *sui generis* customary law.

[7]Practitioners might wish to think of them as 'Actual Alice' and 'Actual Bob' as distinguished from 'Device Alice' and 'Device Bob'.

[8]Readers ignore the Notes at their peril.

[9]While jurists and legal scholars tend to agree on the process of legislative amendment to law, the idea that evolution in societal values can lead to changes in the interpretation of un-amended law is not universally

accepted. Depending upon the system of law in question, some jurists and legal scholars take the view that some laws represent unchanging (perhaps even universal) values and reject any other notion as inappropriate or heretical. This disagreement also exposes a recurring tension in defining and maintaining the division of legislative from judicial authority. For the security practitioner this debate can be simplified as follows: by whatever mechanism, law changes over time.

[10]The nature and challenges of legal scholarship have been nicely summarised by David Feldman, Q.C. [286].

[11]The pace of change in various laws depends upon how deeply rooted they are in social values. Certain foundational principles concerning the administration of justice (e.g., the right to notice and the right to present one's case to a tribunal), are so slow to change that they appear within the span of a single generation to be immutable. Other types of law (e.g., tax law) are amended continually.

[12]Indeed, the relative utility of a system of law arguably depends upon this characteristic of predictability of outcome. A contrary, even nihilistic, view of law and legal analysis is found in the academic school of critical legal studies. A good and accessible example is the scholarship of Girardeau Spann [287].

[13]Common law systems are those derived from the law of England. These are the foundation of legal systems in states that had close historical connections with Great Britain, including England, Wales, Ireland, Australia, New Zealand, Singapore, most of the constituent provinces of Canada, most of the constituent states of the United States, etc. As a result, this system of law is nearly ubiquitous in anglophone territories.

[14]Civil law systems are those derived from a mix of Germanic, Napoleonic, and/or Nordic laws. These are the foundation of legal systems throughout Europe (with the exception of a few common law jurisdictions) and in states that had close historical connections with continental Europe. These include European states such as France, Germany, Italy, Sweden, and Russia, and non-European states such as Argentina, Brazil, Mexico, and Japan. (In the case of Japan, the late 19th century Meiji State selected the civil law of Germany as the primary basis for its legal modernisation programme [288].)

[15]See discussion of 'code' in the text accompanying Note 20

[16]In the author's experience, mistaking a 'bill' (not law) for a 'statute' (law) is not an uncommon occurrence among cyber security practitioners who are unaccustomed to legal research. This is especially easy for the unwary who stumble across the annual mountain of bills introduced by members of the US Congress which never become law.

[17]As a limited narrow exception, some states adopt the practice of examining legislative history (such as the series of draft bills as they were amended in the process of debate to become law) as a means of helping to interpret the law as finally adopted.

[18]The status of European Union legislation in the United Kingdom after Brexit is complex. The UK has adopted legislation that will generally continue within the body of UK domestic law those pre-Brexit EU laws that are most relevant to cyber security (e.g., data protection regulation) unless and until the UK Parliament decides to diverge from EU legal principles.

[19]In the context of a system of federal states, 'foreign state' can include another member state of the federation. Thus, courts of the State of New York would regard interpretations of law issued by courts of the State of California as the decisions of a foreign state. As such, they would not constitute binding authority in New York State courts, although they might have value as a source of persuasive authority. This discussion should not be confused with the subject of enforcing foreign judgments (see Section 3.2.4).

[20]For example, the United States Code (U.S.C.) (a collection of otherwise disparate Acts of the US Congress organised into code form by editors who then revise the code as further legislation is adopted or amended), and the Bürgerliches Gesetzbuch (BGB) (the comprehensive code of German civil law adopted en masse at the start of the 20th century and amended from time to time).

[21]For example, the Code of Federal Regulations (C.F.R.) (a codified form of US secondary legislation).

[22]For example, the Uniform Commercial Code (U.C.C.) (a set of model laws produced as a joint project of the Uniform Law Commission and the American Law Institute, which has in turn been adopted with some local variations by most constituent states of the United States and has thus become extremely influential).

[23]This last category can sometimes suggest the future development of law, as states may decide to mandate compliance with codes that began life as suggested rules. Similarly, courts may use advisory codes as a way of interpreting responsibilities such as the requirement to act 'reasonably' in assessing negligence liability.

[24]For example, The Tallinn Manual (a restatement of public international law applicable to cyber operations) and the Restatement (Third) of Torts: Products Liability [87, 204].

[25]This can be simplified with the observation, 'There's no such "place" as cyberspace'.

[26]Some creative arguments against this result include attempting to recharacterise cyberspace as 'territory' that exists separately from sovereign states, thus attempting to describe a universal and harmonised set of legal principles that should be applicable to all uses of cyberspace globally, and in some cases rejecting the authority of sovereign states to intervene in cyberspace-related activities. The best of these constitute interesting experiments in legal philosophy [289].

[27]The relative merits of defining an artificial intelligence as a person for legal purposes have been considered by legal scholars from time to time [290, 291].

[28]A variety of other legal doctrines might create liability for persons as a result of actions directed by an artificial intelligence [292].

[29]Various financial fraud crimes are often defined in this fashion, for example, requiring proof that the accused had a specific intention to deceive (*scienter*). Many of the computer crimes discussed in Section 3.5.1 may not require such proof.

[30]Criminal intent (or its lack) should be distinguished from circumstances where the law expressly provides a defence such as 'public interest', 'public necessity', or 'self-defence'.

[31]'Civil law' in this context, meaning non-criminal law, should not be confused with the term " as a means of classifying systems of law such as are found in the states of continental Europe. See Note 14.

[32]Principles of human rights law designed to guarantee a fair trial for Alice often force people like Bob to delay their civil action until the relevant criminal prosecution is concluded. The difference in standards of proof, it is entirely possible for Alice to be found 'not guilty' of the alleged crime and still be found liable for the alleged tort.

[33]The law of the United Kingdom expressly prohibits introducing the content of such intercepted communications as evidence in court proceedings.

[34]This definition of 'proof' stands in sharp contrast to a 'mathematical proof'. In the field of mathematics, to 'prove' something means to establish undeniability as a logical necessity – to establish the truth of a proposition beyond any dispute. (For example, proof of Pythagoras' theorem.) By contrast, the proof of a real-world event in a court of law never results in absolute certainty. A fact finder in a legal proceeding must reach a conclusion on less than total certainty. A technology journalist eloquently summarised this when he stated, 'The purpose of law is not to achieve philosophical or mathematical truth, but to take a messy reality and achieve workable results that society can live with' [293].

[35]An 'affirmative defence' is contrasted with other types of defence where the party pursuing a right of action continues to carry the burden of proof. For example, in a criminal prosecution for murder under English law if the accused claims 'self-defence' it remains the responsibility of the state to prove beyond reasonable doubt that the accused is NOT entitled to the defence. By contrast, a claim of 'insanity' is an affirmative defence under English criminal law. The burden of proving insanity falls on the accused, although the standard of proof required is merely the 'balance of probabilities' [294].

[36]There are many possible criticisms of this approach to explaining legal risk analysis, including the focus on 'cost' as a determinant of risk. Factors beyond the mere financial are considered in Section 3.14.

[37]Although courts use this same phrase to describe the two standards of proof, they remain free to define them differently in the context of interpreting two different laws adopted for two different purposes.

[38]This term can also be used to describe subject matter and territorial authority of legal persons created by treaty between states, such as international governmental organisations (e.g., the ITU) and special purpose multi-state municipal organisations (e.g., The Port Authority of New York and New Jersey, and the Washington [DC] Metropolitan Area Transit Authority).

[39]By contrast, 'subject matter jurisdiction' refers to the scope of the subject matter that can be addressed by a given entity. For example, a single state might choose to divide its court system into two parts: one that addresses only criminal complaints and one that addresses only civil matters. While the territorial jurisdiction

of both courts might be identical, the subject matter jurisdiction is clearly different. Similarly, the scope of authority delegated to individual regulatory agencies, ministers of state, etc., constitute a type of subject matter jurisdiction for that agency.

[40]A good introduction to the principles of juridical jurisdiction for civil cases is found in the recast Brussels I Regulation, which presents the rules normally applicable to civil matters in courts located within the European Union [295].

[41]To take an admittedly whimsical fictional example from the Wild West, in the film *Silverado* Sheriff Langston (portrayed by John Cleese) discontinues his hot pursuit of criminal suspects through the wilderness after a sniper opens fire on his posse. He justifies his action explaining wryly to his companions, 'Today my jurisdiction ends here' [296]. Sheriff Langston's quandary illustrates the relationship between state power and enforcement jurisdiction. Non-fictional examples that explore the territorial limits of state enforcement power are readily available, albeit controversial, and in some cases are the subject of diplomatic and international legal dispute.

[42]See various US statutes criminalising acts of homicide against US nationals while overseas codified at 18 U.S.C. §2332.

[43]Reasons this activity might not be illegal under the law of the first state include, most obviously, where the first state has not adopted any law to criminalise the complained-of hacking activity. Alternatively, the first state may criminalise the activity in normal circumstances but officially warrants the cyber operation pursuant to the lawful domestic authority of the first state. In this second scenario, the person undertaking the operation would normally be immune from criminal prosecution in the first state but subject to criminal prosecution in the second. This discussion focuses solely on liability of the relevant non-state person undertaking the cyber operation. The liability of states to one another for such operations is addressed in public international law (see Section 3.12).

[44]The subjects of espionage and remote evidence gathering are discussed in Sections 3.12.3 & 3.12.4

[45]The 1998 dispute over legal control of DNS root servers, and its informal albeit dramatic resolution, is recounted by Goldsmith and Wu and criticised by Froomkin among others [78, 297].

[46]A bank in this situation faces the practical problem of two competing states making conflicting demands: one ordering payment, and a second prohibiting payment. Taking the analysis one step further, imagine what could happen if (in an effort to avoid adverse enforcement actions by the United States) the London branch of a US bank refused to comply with the judgment of an English court. This bank might jeopardise its ability to conduct regulated banking activities in London. Presumably, the depositor could also ask English courts for enforcement assistance by demanding the seizure and forfeiture of funds held by such defaulting US banks on deposit with other banks in the UK. The depositor could also take the judgment and request enforcement by third-party states where the US bank also held funds on deposit. These are the types of analysis that arise when a non-state person considers the risk of potentially conflicting state mandates.

[47]In the context of the US federal system, each member state of the US is normally required to enforce civil judgments issued by courts of other member states under the Constitutional mandate to give 'full faith and credit' to acts of other US states. (US Constitution, Art IV, Sec 1.) A similar rule applies in the European Union by operation of Chapter III of the (recast) Brussels I Regulation [295].

[48]To avoid a point of occasional confusion, if a suspect is travelling from State A to State C and they are transiting State B, the police of State B are normally able to arrest that suspect upon arrival in State B. This continues to hold true even if the suspect does not request entry to State B. Criminal suspects can be, and have been, arrested in the international transit areas of airports.

[49]Lessig's phrase 'code is law' has been the subject of widespread discussion among both technologists and lawyers [298]. Lessig highlights the close interrelationship between technological controls (computer code) and human governance controls (legal code). Both are mechanisms that can serve to limit how systems are used. Each mechanism has different utility. And ultimately, each mechanism may influence the other. However, the phrase has been interpreted by some to mean that 'whoever writes the computer code is essentially making the law'. The history of how laws are enforced with respect to Internet-related activity strongly suggests that this interpretation is (at best) terribly misleading and (at worst) misunderstands the direction of causality between computer and legal code.

While technologists certainly enjoyed first-mover advantage in choosing the design of the underlying architecture of the Internet and related applications, law makers - and the societies for whom they serve as a proxy - have responded strongly with their own opinions about how systems should work. As one author has wryly

observed, societal opinion seems to have moved 'from "Code is Law" to "Law is Law"' [143]. In other words, one should not assume that the persons who write the (computer) code are the same persons who create the legal norms to be enforced.

[50]The significant role undertaken by various platform operators in filtering content on a state-specific basis and supplying similar geo-filtering tools to their content supplying customers is often overlooked in policy debate.

[51]An example of collaborative filtering is found in the work of the Internet Watch Foundation. Among other things, the IWF maintains a URL database of sites known to host images of child sexual abuse. This database is used by various service providers to restrict access to these sites [299].

[52]The opinion of Judge Lynch, concurring, is especially interesting as he wrote to highlight many of the more unsettling and counter-intuitive policy aspects that would result from the judgment and 'to emphasize the need for congressional action to revise a badly outdated statute'.

[53]Although the Microsoft case was dismissed prior to judgment, the extensive collection of briefs filed with the US supreme court by a variety of interested third parties constitutes a treasure trove of analysis and advocacy on this topic. It remains possible that a future dispute might be brought in the US courts to challenge the authority of the US Congress under the terms of the US Constitution to extend jurisdiction in this fashion. While the outcome of any such future challenge is debatable, it seems unlikely to succeed.

[54]The precise meaning of 'lawful and voluntary consent' in article 32b of the Budapest convention has prompted much discussion. One area of repeated concern is the acceptance by some states of criminal plea bargaining techniques as a means of obtaining consent from criminal suspects [122, 300]. 'Consent' is a challenging subject in law, generally [301]. See also Section 3.4.2.

[55]Although people most often discuss the issue of data sovereignty in the context of compelled disclosure of data, other state interventions may also be possible such as compelled data alteration or deletion, or compelled service interruption.

[56]Methods used in an effort to mitigate this risk using cryptographic technology, database sharding or replication over servers in multiple states, etc. are outside the scope of this knowledge area.

[57]The Regulation, of course, does not interfere with any data localisation rules imposed for reasons of state security as this subject area falls outside the regulatory subject matter jurisdiction of the European Union.

[58]The discussion in the Section focuses primarily on the privacy rights of natural persons. States can and do apply these or similar rights to legal persons, although the privacy rights of legal persons may be less than those accorded to natural persons in some circumstances.

[59]To understand the legal context of the international instruments cited, see the introductory discussion of public international law in Section 3.12.

[60]The conditional nature of the right expressed in article 7 is explained in an accompanying report [302, 303].

[61]In the US legal system, for example, the Fourth Amendment to the US Constitution provides a set of rights that limit only state actions, while the California Constitution grants a general right of privacy effective against state and non-state actions. Both the US and its constituent states have promulgated a large number of laws that regulate intrusions under various conditions. The landscape is complicated.

[62]Examples made possible by the emergent mobile app economy include processing data concerning personal contacts, calendar and scheduling information, banking data and authentication credentials, personal notes and communications, browsing and shopping history, intimate relationship data, and a variety of health-related data from heart rate and exercise patterns to menstruation data. Each new data set presents a question about the 'normal' expectation of privacy when using these services, and the permissible scope of intrusion by state and non-state persons.

[63]In the referenced case of Smith v Maryland, the US supreme court decided in 1979 that compelled disclosure of customer dialling records did not constitute a 'search' for purposes of the Fourth Amendment to the US Constitution as the customer had no expectation of privacy in the list of numbers dialled [141].

[64]Compare the information that can be inferred after discovering that a target of investigation has navigated to a URL string such as 'web.example.com/politicalpartyname/how-to-renew-my-membership.htm' with the discovery that the same person dialled a phone number such as '1-202-555-7730'. In this example, the URL meta-

data leads to a strong inference of communication content and the probable ability to reconstruct accessed content precisely.

[65]The US supreme court, for example, decided in 2018 that a cell phone customer has a reasonable expectation of privacy in location data and therefore the state-compelled disclosure of this data constitutes a 'search' for Fourth Amendment purposes [304]. In Europe, customer location data has been expressly protected under privacy and data protection laws for some time.

[66]Consider, for example, the capability of various de-anonymisation techniques that can be applied to metadata as well as the reported growth of metadata analysis in the field of signals intelligence.

[67]There are, of course, risks associated with the implementation of these facilities and examples of how they have been abused in violation of applicable law. Anti-abuse measures can and should be founded on both technological and organisational controls.

[68]The complexity facing a multinational services provider in complying with lawful interception obligations is well illustrated in Vodafone's published transparency report, which includes a lengthy summary of the relevant laws they face in 28 states [305].

[69]In an effort to navigate potential restrictions on reporting new types of interception, some service providers adopted the practice of publishing so-called 'Warrant Canaries' – a statement published on a recurring basis that no interception warrants of a given type had been received. The theory behind this practice was that a subsequent failure to re-publish the Canary statement would allow the public to infer (without the communication provider expressly stating) that state-warranted interception had commenced. This practice seems to have fallen into disfavour, probably aided by the sometimes-debatable legal status of the practice plus additional state legal interventions that made this strategy more difficult or impossible to carry out within the terms of applicable law. See, e.g., 50 U.S.C. §1874(a) [306].

[70]In the US, some courts have held that efforts to compel disclosure of passwords triggers scrutiny under human rights law as it forces a suspect to give testimony against himself, while mandatory presentation of a fingerprint to unlock a device does not trigger this same legal objection [148]. This is an area where legal standards remain murky and the topic is ripe for further dispute and development [149].

[71]An example is s.49 of the (UK) Regulation of Investigatory Powers Act 2000.

[72]Practitioners should be careful to distinguish between activities such as developing a communications protocol, writing software that implements a protocol, supplying such software to the public, and supplying a service that implements a protocol. A quick test that may assist in clarifying a person's status is to ask this question: 'Would the relevant communications service continue to operate if the processes administered by this person ceased to function?' Thus, a person who supplies IMAP services, SMTP services, or a key distribution service to support end-to-end encryption for a communications app is more likely to be classified as a communications service provider under relevant legislation than a person who merely writes software that implements a protocol. Details of applicable laws diverge significantly and must be investigated on a state-by-state basis.

[73]For example, Brady v Maryland [307]. This is less likely to occur to the extent that the law of a state (such as the UK) prohibits use of intercepted communications as evidence in legal actions [150] at s.56.

[74]The US exclusionary rule has been hotly debated for more than half a century.

[75]Activities undertaken by states in defence of state security generally fall outside the prescriptive jurisdiction of the EU. Member states may choose individually to apply similar principles in the state security context [170], Part 4.

[76]Practitioners must not lose sight of this regulatory purpose. When assessing risks of various processing activities and security arrangements in the context of data protection law compliance, the risk to be assessed is normally the risk of harm to data subjects - living human beings. Risks faced by the processing enterprise (including risks of non-compliance with data protection law) should be evaluated separately. A similar observation can be found in the context of the *Carroll Towing* case discussed in Section 3.7.1.2 and Note 123.

[77]The attempt to delineate 'pseudonymous' from 'anonymous' data is a subject of significant discussion [308]. While details of de-anonymisation methods are beyond the scope of this knowledge area, examples seem readily available [309].

[78]For example, the term 'personally identifiable information' is defined for the purposes of US bankruptcy law at 11 U.S.C. §101(41A) and defined differently for the purposes of a federal law prohibiting the disclosure of

video rental histories at 18 U.S.C. §2710(a)(3).

[79]This is a natural result of the US approach to this subject, which is to adopt narrowly drawn *sui generis* laws that specifically focus on individual use cases. The cited decisions are drawn from examples of the US courts interpreting a 1988 law originally adopted to restrict the disclosure of video rental records as they were kept in the 1980s. The courts were called upon to interpret this ageing statute in the context of online streaming service records in 2015. Practitioners may be interested to note that as US courts are charged with responsibility to interpret the will of the US Congress when resolving these cases, they seem unaware of (or perhaps uninterested in) the technical definitions of PII offered by the ISO and NIST publications [162, 163].

[80]In practice, there may be a strong temptation, and corresponding pressure, to assume that the absence of obvious personal identifiers in a data set means that no personal data are present. A better approach is to appreciate that data protection law tends to measure obligations in proportion to the risks presented by any given processing activity. Data sets containing personal data without obvious personal identifiers might therefore present a lower risk when processed, thus making compliance less onerous in such cases.

[81]Consent is perhaps one of the least well understood, and hotly debated, terms in data protection law. In addition to many sources of guidance published by public authorities on this topic, practitioners who wish to explore this concept in depth might take inspiration from outside the body of data protection law [301].

[82]The notifications discussed in this section are distinguished from separate requirements, if any, to share security breach information with relevant industry-specific regulators or security coordination authorities (see Section 3.11.1).

[83]By 2010, 46 US states had adopted legislation mandating some form of personal data breach notification to effected persons [310].

[84]Mandatory obligations to communicate personal data breaches to data subjects irrespective of the risk of harm has been criticised on a number of grounds, including: data subjects become overwhelmed by communications and are unable to distinguish the degree of risk presented by any individual breach, communicating to a large set of data subjects is extremely resource-intensive, and communicating to data subjects could interfere with the ability of police authorities to investigate the breach.

[85] The UK ICO explained the proposed fine in its Statement of July 8, 2019: 'The proposed fine relates to a cyber incident notified to the ICO by British Airways in September 2018. This incident in part involved user traffic to the British Airways website being diverted to a fraudulent site. Through this false site, customer details were harvested by the attackers. Personal data of approximately 500,000 customers were compromised in this incident, which is believed to have begun in June 2018. The ICO's investigation has found that a variety of information was compromised by poor security arrangements at the company, including log in, payment card, and travel booking details as well name and address information.'

[86] The UK ICO explained the proposed fine in its Statement of July 9, 2019: 'The proposed fine relates to a cyber incident which was notified to the ICO by Marriott in November 2018. A variety of personal data contained in approximately 339 million guest records globally were exposed by the incident, of which around 30 million related to residents of 31 countries in the European Economic Area (EEA). Seven million related to UK residents. It is believed the vulnerability began when the systems of the Starwood hotels group were compromised in 2014. Marriott subsequently acquired Starwood in 2016, but the exposure of customer information was not discovered until 2018. The ICO's investigation found that Marriott failed to undertake sufficient due diligence when it bought Starwood and should also have done more to secure its systems.'

[87]As Ian Walden notes, 'classifying certain subject matter as criminally illegal can be a highly contentious matter, raising complex definitional issues, questions of causation, and human rights concerns, specifically rights to privacy, freedom of expression, assembly, and association' [92] at para 3.95.

[88]The problem is presented in the well-known case of *R v Gold and Schifreen* [311]. The accused were arrested in the UK in 1985 after they had obtained a system password for an early email system and used it to access an email account assigned to a member of the British Royal Family. Although the accused were originally convicted following trial in 1986 for violating the Forgery and Counterfeiting Act 1981, the House of Lords (at that time the court of last resort in the UK) quashed the conviction in 1988 holding that the complained-of action was not a violation of the 1981 statute.

[89]Bruce Sterling provides an interesting history of early computer crime investigation and prosecution efforts in the 1980s by the US authorities, and colourfully describes how they sometimes missed their intended target

[312]. Clifford Stoll also describes the contemporaneous challenges he encountered as a private citizen attempting to investigate computer intrusion, complaining that he often could not find law enforcement officials able to assist him [313].

[90]A catalogue of US state computer crime statutes is maintained by the National Conference of State Legislatures [314]. A very useful and oft-cited survey of US state laws was compiled by Susan Brenner [315].

[91]Both the Budapest convention and Directive 2013/40 allow states a certain degree of flexibility in the detail of their domestic laws, and many contracting states have declared reservations against certain provisions of the Budapest convention.

[92]Confusingly, the verb 'to hack' is also used to describe non-criminal, often informal, ICT research and development activities that are pleasingly clever or demonstrate a previously unknown characteristic of an object. This positive connotation of the term now extends beyond the realm of ICT development, as can be found in emerging phrases such as 'life hack' and 'hackathon'.

[93]The role of prosecutorial discretion is one possible explanation for the lack of a *de minimis* exception in the definition of computer crimes. See the discussion in Sections 3.5.3 and 3.5.5.

[94]This was discussed after the Click television programme's 'Botnet experiment' was broadcast on the BBC in March 2009, in which the show's producers procured and then commanded the actions of such a botnet, albeit with an avowedly benign intention [316, 317, 318, 319].

[95]In some rare instances, non-state persons are granted the right to bring a criminal prosecution when state officials have chosen not to do so.

[96]US Federal Courts undertake an algorithmic approach in calculating recommended criminal sentences pursuant the US Federal Sentencing Guidelines [320]. Under these guidelines, crimes against information systems are classified as 'economic' crimes and sentences may be significantly enhanced based upon value of damage caused by the criminal activity [180]. (Details of the calculation are set out in [320] at §2B1.1, taking note of the various interpretive rules applicable to violations of 18 U.S.C. § 1030, *et seq*.) Although federal judges are required to take this calculation into account when passing sentence, they may deviate from the sentencing guidelines subject to whatever limits may be mandated by Congress in the substantive criminal statute.

[97]This becomes more obvious when considering intrusion efforts against industrial control systems such as those that operate dam sluice gates, national electricity power grids, steel mills and foundries, automobiles, oil tankers, pipelines, and nuclear power generation facilities.

[98]Historically, the Computer Misuse Act 1990 did not contemplate the idea of state-warranted intrusion into information systems. This express exception to criminal liability under the 1990 Act first appeared in the Regulation of Investigatory Powers Act 2000, the predecessor of the Investigatory Powers Act 2016.

[99]Such proof would most likely consist of asking the fact finder to draw reasonable inferences from the circumstances surrounding any given act of production or distribution.

[100]Some have argued that conducting 'legitimate' security research activities should be shielded from criminal (and perhaps civil) liability if appropriate conditions are met [321, 322]. Similar arguments have been advanced in the cause of security-related journalism. These policy arguments have not yet found overwhelming support from various state law makers, although the debate is not well advanced.

[101]It has been suggested that persons who engage in security research and development activity that might otherwise constitute a *de minimis* violation of computer crime laws might enter into formal or informal agreements with law enforcement or state security officials to receive an assurance of non-prosecution. Risks to the practitioner include potential misunderstanding with state officials, potential inability to enforce the non-prosecution agreement, or collateral legal risk such as tort liability [189]. Risks to a state pursuing this strategy include the possibility that such an agreement might be used to attribute responsibility to the state under public international law for the actions of such researchers or developers (see Section 3.12.1).

[102]In some systems of contract law, however, a service provider may be required to give customers additional time to pay or special notices before services can be suspended. Suspension of services in circumstances that would cause a threat to human life and welfare (e.g., energy services supplied in a freezing cold climate) are often separately regulated and can be suspended only in accordance with strict rules.

[103]Orin Kerr's US casebook contains a helpful collection of citations and arguments on this topic ([180] at ch.2G).

[104]Indicia of enforceability are generally beyond the scope of this work. It is both difficult to describe generalisable multinational legal norms about these, and this topic is of lesser concern to cyber security practitioners.

[105]The term 'offer' must be considered with care and distinguished from less significant forms of communication such as an 'invitation to treat' or 'invitation to tender'. While some ecommerce systems make contractual offers to a wide range of potential customers, the most common design is for the vendor to publish invitations to treat – essentially asking customers to make an offer when placing an order. This generally shifts who has control over the time of contract creation back into the hands of the online vendor – an often-useful risk management device.

[106]Practitioners skilled in computer science might wish to draw inspiration from the Two Generals Problem.

[107]In this context, 'order' refers to a communication by a potential customer to a supplier seeking a contract. In practice, an order usually constitutes either an offer or an acceptance depending on the terms and conditions applicable to the relevant online platform. In the field of B2C online commerce, it has become common practice for an order to be defined as a contractual offer – capable of being accepted or rejected by the online supplier.

[108]The rule embodied in article 11 is a rather muted result of a European debate in the 1990s concerning whether to harmonise the time of the contractual trigger in online commerce. Law makers, facing a wide variety of contract rules which are beyond scope of this knowledge area, ultimately chose not to harmonise this aspect of law. The resulting version of article 11 is limited to this question of defining the time of receipt of electronic orders and acknowledgments.

[109]For example, financial transaction systems such as SWIFT, airline reservation systems such as Amadeus, Galileo, etc.

[110]Codified in 15 U.S.C. §1681c(g).

[111]This knowledge area will not seek to differentiate between a contractual warranty and a contractual condition. Although these create different rights in the hands of a party suffering a breach, the topic is beyond scope of this knowledge area.

[112]Under English law, this is normally styled as the condition of 'satisfactory quality' and was formerly known as the condition of 'merchantable quality'. Under the law of most US states is it styled the warranty of 'merchantability'. Civil law systems adopt a similar concept.

[113]In the law of England and most US states this is styled 'fitness for purpose'. Once again, in England this is said to be a contractual condition and in US states it is generally a warranty.

[114]Examples of typical language found in contracts for the supply of software include, 'Vendor warrants that the software will comply with the Documentation for a period of 60 days following delivery.'

[115]These are not legal terms of art, but merely used to illustrate the variable degree of breach severity.

[116]The names of the remedies are drawn from common law practice. Other legal systems may employ different terms and/or grant alternative remedies.

[117]Those who deal regularly with procurement agreements might find this concept expressed in clauses that specify the right to terminate a contract following 'material breach', 'material breach that causes significant harm', 'a series of minor breaches that collectively create material harm', etc. The definition of the trigger is limited only by the imagination of the drafter, although some legal systems impose limits on the effectiveness of these clauses.

[118]The leading case on this issue in England in the early twentieth Century concerned the duty of a person who bottles beverages owed to those persons who eventually drink them. The advent of the modern economy created supply chains in which the producer and consumer had no direct business relationship, where products change hands multiple times before being consumed. Applying an earlier version of the rule described above, the English court (acting in its capacity as a common law policy maker) stated that the bottled beverage was itself the proximate link between producer and consumer, and that a producer of such a drink could readily foresee the harm caused by the adulteration of the bottle's contents.

[119]A well-known, bordering on comic, example can be found in the 1928 Palsgraf case [323]. (See also discussion of causation in Section 3.7.3.)

[120]This last category is mentioned because of the occasionally encountered practice where a person attempts to avoid liability by purposefully avoiding knowledge of risk. This strategy is unlikely to defeat a claim of neg-

ligence and may even exacerbate liability in jurisdictions that award punitive damages. (See the discussion in Section 3.7.4.)

[121]In the cited 2018 *Dittman* case, the Pennsylvania supreme court announced that the common law of Pennsylvania imposes a duty of care on employers to safeguard the electronic storage of employee data. A mid-ranking appellate court in the State of Illinois reached the opposite conclusion in 2010 when interpreting the common law of Illinois [324]. In the US, negligence law may play an increasing role in defining responsibilities to safeguard personal data.

[122]See discussion in Section 3.10.3.

[123]Judge Hand surprised legal practitioners of the day by expressing this concept using a mathematical formula, stating that if $B < PL$ then the failure to adopt a given precaution constitutes negligence, where B is the burden (cost) of the method, P is the probability of loss in the absence of the method, and L is the amount of loss to be avoided. These two cases and one formula have been the subject of extensive comment, debate and analysis by generations of US lawyers and law students and it remains a useful framework to discuss risk and responsibility [201, 203]. Practitioners may wish to consider how changes in the cyber security environment over time (including the falling costs of some defensive technologies (B), as well as changing probabilities of harm to third parties (P) and the amount of losses they might suffer (L) as a result of changes in the surrounding environment such as migration to different infrastructures, ever-larger aggregations of 'big data', etc.) may influence liability. The speed at which these variables change may help to plan the frequency for reassessing decisions to reject proposed precautions. Yesterday's 'impractical' precaution may become tomorrow's 'must have' solution.

[124]A similar observation in the context of data protection regulation can be found in Section 3.4.1.

[125]In the referenced case, the negligence per se claim was based on an allegation that Target had failed to comply with a Minnesota law concerning the proper storage of credit card details [198]. (See also the discussion of this case at Section 3.7.4)

[126]The Morris worm might be an early example of this type of incident [313].

[127]This section is addressed primarily to 'design defects' and does not discuss 'manufacturing defects', in which individual products from a production run deviate from their specification due to sporadic intermittent errors in the manufacturing process.

[128]Even if a producer of software is not amenable to a claim founded on a theory of strict liability, it could still face a claim founded on a theory of negligence. A victim taking legal action against a software producer based on a theory of negligence would need to prove unreasonable conduct by the software producer.

[129]Self-driving automobiles in particular have prompted a significant amount of discussion as lawyers and law-makers consider both current liability rules, and potential amendments to these rules to enable this highly-anticipated technology [325, 326, 327].

[130]Such attenuated chains of causation are a familiar meme in science fiction stories about time travel. A non-fiction but entertaining exploration of highly attenuated chains of causation from scientific history is found in the work of journalist James Burke in various iterations of his BBC television programme, 'Connections'.

[131]See also discussion of foreseeability in Section 3.7.1.1.

[132]Such 'negligent mis-statement' cases are watched closely by professionals and other service providers in the business of supplying critical information-related services such as public accountants. This type of negligence theory is also of special interest to providers of trust services, as it potentially defines their liability to third parties who rely upon the accuracy of issued certificates. (See Section 3.10.3)

[133]In the cited *Dittman* case the supreme court of Pennsylvania, acting in its role as interpreter of the common law of Pennsylvania, held in November 2018 that employers owe a duty of care to their employees to maintain reasonable cyber security to safeguard employee data from loss [199]. In any similar incident in the EU, a tort action could be fashioned easily under a theory of breach of data protection rights.

[134]An obvious example is the various legal actions brought by financial institutions against Target following its well-known 2013 loss of card data incident. Plaintiff banks in at least one of the actions based their claim on various legal theories including negligence and negligence per se [198]. Settlements of this law suit and others brought by financial institutions against Target exceeded US$100 million [328, 329].

[135] *Compare* easily quantifiable losses resulting from breach of privacy such as loss of revenue from an exclusive agreement to publish the victim's wedding photographs in a specific newspaper, loss of salary as a result of victim's dismissal from employment etc., *with* more difficult-to-quantify harm such as the victim's embarrassment or shame.

[136] This provision is codified in Illinois law at 740 ILCS 14/20.

[137] The internal auditor was arrested and charged with criminal violation of data protection law, computer crime, and fraud. He was convicted and sentenced to eight years imprisonment.

[138] The supreme court of the United Kingdom granted leave to appeal on 15 April 2019. Hearing has been scheduled for late 2019, which suggests the potential for a decision sometime in 2020.

[139] *Compare* potential application of the state-of-the-art defence in the context of materials science where (for argument's sake) at the time of production there was no known scientific test for the later-discovered defect in the material, *with* the context of a software-induced product defect due to a previously unknown zero day exploit. The former might be said to have been undiscoverable, while the latter was merely undiscovered. It is debatable when a given exploit could be truly classified as having been 'undiscoverable'. This topic merits further study [330].

[140] It has been suggested anecdotally that some regulation of safety-critical systems can lead to weaknesses in that system's cyber security by limiting or foreclosing the possibility of adopting state-of-the-art security measures. A specific instance related to the author concerns a regulatory requirement that certain safety-critical control systems must be exhaustively tested by examining every possible state of the control device prior to use. Some defensive cyber security methods, especially those that adopt artificial intelligence or machine learning, are by their nature impossible to test to exhaustion in this fashion. This topic merits further study.

[141] A favourite example beloved of law professors involves the hypothetical case of the badly loaded rail car. The car may have been improperly overloaded in State A, but only produces injury after the train begins to descend a steep grade many hours later in State B.

[142] This is attributed to US supreme court Justice Joseph Story, who apparently used the phrase more than once [331]. A darker shadow was cast over the practice of intellectual property law by Lord Esher, MR, when in 1892 he observed, 'a man had better have his patent infringed, or have anything happen to him in this world, short of losing all his family by influenza, than have a dispute about a patent. His patent is swallowed up, and he is ruined. Whose fault is it? It is really not the fault of the law: it is the fault of the mode of conducting the law in a patent case' [332]. Little has changed in the intervening century [333].

[143] Moral rights arising under an author's rights (*droit d'auteur*) infrequently present challenges for security practitioners and is beyond the scope of this work.

[144] In United States law, copyright comes into existence automatically but must be registered prior to commencement of any US infringement proceedings.

[145] Limitations to UK copyright are codified in Chapter 3 of the Copyrights Designs and Patents Act 1988, ss.28, et seq. The US fair use exception and other limitations are codified in 17 U.S.C. §107, et seq.

[146] The implementation of this protection has been both inconsistent and controversial [214, 334]. It is codified in US copyright law at 17 U.S.C. §1201, et seq., and in UK copyright law in Part VII of the Copyrights Designs and Patents Act 1988 at ss.296, et seq.

[147] The European Union is in the process of adopting the Unitary Patent, a single patent right that applies throughout much, but not yet all, of the territory of the EU. The status and use of this new patent right continues to evolve.

[148] Inventors should not confuse this concept from patent law with various scientific or academic definitions of significant or trivial. A scientifically trivial step can still be 'inventive' in patent law [335].

[149] The phrase 'as such' should serve as a warning that loopholes are about to appear, as if by magic. They are.

[150] While copies of patents and published applications from many states are now easy to find online, correspondence with the patent examiners and the prosecution history is often more difficult to obtain and may require assistance from a legal practitioner. Once obtained, however, this can be very enlightening to any person who wishes to challenge *post-facto* the validity of a granted patent.

[151]A very limited subset of patent applications for inventions are subject to a state secrecy classification and are only published in a register of secret inventions.

[152]Anyone innovating in the ICT field faces a series of related challenges. The pace of ICT innovation is so fast, the intermingling of parallel innovative ideas so commonplace, the number of patent applications filed so large, and the prior art cataloguing ICT innovation so untidy, that it is difficult to produce any innovative ICT product that does not infringe some extant third-party patent, or published or unpublished application. A typical strategy adopted by large ICT developers is to file large numbers of patent applications on their own inventions, move to market as quickly as possible with new products, and then wait to receive suggestions of patent infringement from third parties in the hope of eventually defending against some of these threats or negotiating an acceptable cross-license arrangement.

[153]In the US patent system, awareness by the infringing party of patent rights triggers a special 'treble damages' rule: monetary damages awarded to the rights holder are multiplied by three with effect from the date of the infringer's awareness. This is why rights holders typically begin a US patent enforcement process by sending copies of their patents together with a 'we wish to make you aware' cover letter that does not expressly accuse the recipient of infringement. This, combined with the factors set out in Note 152, is why many ICT innovators assiduously avoid researching third-party patents and patent applications.

[154]Some states define 'unregistered' trademark rights which are similar in character to the English law tort of passing off.

[155]A Community Trademark is a single trademark that extends throughout the territory of the EU.

[156]In modern trademark practice, the relevant sign can consist of sounds or smells. A sound trademark likely to be familiar to cyber security practitioners is the 'Intel Inside' musical chime (US75332744, UK00002403603).

[157]Trademark UK00000000001 has been registered continuously in the UK from 1876 to date.

[158]Courts are divided on the question of whether meta-tags, not normally visible to end users, can constitute an infringement of registered trademarks. Even where meta-tags cannot be used to prove infringement, they can serve as useful evidence for other purposes such as demonstrating the knowledge or awareness of the tag author in related tort actions such as passing off.

[159]By contrast, in actions based on theories of passing off or unregistered trademark rights the complaining party is usually required to prove that the accused party has knowledge of the unregistered mark and is purposefully taking advantage of the reputation connected to that mark.

[160]An example that should be familiar to practitioners is the Wi-Fi logo (US75799630, UK00002209133) registered by Wi-Fi Alliance.

[161]A commonly-cited example of a long-standing trade secret is the formula for Coca-Cola, which remains the subject of much speculation.

[162]17 U.S.C. §1204(a).

[163]Copyrights Designs and Patents Act 1988, s.296ZB.

[164]The European Union Directive does not mandate the criminalisation of trade secret misappropriation [221]

[165]Note that the Ecommerce Directive does not apply to data protection law [193] at Art 5(b).

[166]For example, 17 U.S.C. §512 (shielding from copyright infringement), 47 U.S.C. §230 (shielding from liability those who block or screen offensive material, although not applicable as a shield against liability arising under obscenity, intellectual property or privacy laws.) Section 230 in particular has come under increasing scrutiny by US courts as more legal actions have been taken against social media service providers.

[167]Although these legal definitions are not specifically linked to technical definitions, this concept is approximately equivalent to providing services that consist of nothing more than carrying and routing traffic at the physical, data link and/or network layers of the TCP/IP protocol suite. A good definition of the concept is found in article 12 of the Ecommerce Directive [193].

[168]See article 14 of the Ecommerce Directive [193].

[169]The best-known procedure codified in law is probably found in US copyright law at 17 U.S.C. §512(c).

[170]The 'Allow States and Victims to Fight Online Sex Trafficking Act of 2017' is the result of the FOSTA-SESTA bills proposed in Congress. The narrowing of the liability shield is codified in 47 U.S.C. §230(e)(5). A legal action challenging this law as a violation of US freedom of speech principles was launched shortly after its passage and remains pending at the time of writing [336, 337].

[171]The ability to admit (to present) evidence to a court is a threshold question governed by the rules of evidence used by that court. Admissibility asks simply whether such evidence will be considered at all. If admitted into evidence, a fact finder must then assess the relative value or 'weight' of that evidence. Mason summarises the English law position on electronic documents in [338].

[172]The framework contract, in turn, would refer to a series of electronic trading rules (often called 'the rule book') that specified how electronic communications mapped onto legal obligations. Such 'EDI' systems were developed at a time when telecommunications bandwidth constraints meant that trading instructions were typically transmitted in extremely small payloads using text-based (e.g., ascii) highly structured messages. The rule book was used to translate between structured messages and legally significant communication, and served as the specification for software used to access the system.

[173]By underwriting the risk of many such transactions, the positive impact of the payment card industry to the growth and success of these platforms should not be underestimated.

[174]The 'three-corner' model for this purpose comprises only three persons: the certificate issuer who both identifies the signatory and issues the certificate, the signatory whose identify is bound to the certificate and the third party who relies on the certificate to identify the signatory. As each of these roles becomes divided among more persons, analysing the relationships and responsibilities becomes more complex.

[175]See, for example, X.509.

[176]These doubts arise from a variety of legal doctrines. For example, there may be failure to form a contract with a relying party because of failure to communicate the terms of contract. Courts might refuse to enforce limitations of liability presented in certificates or elsewhere due to public policy concerns such as the reasonability of the limitation. Note that these concerns are more easily addressed in the so-called 'two-corner' issuance model, where a signatory self-certifies its identity and serves as its own certificate issuer. In the two-corner model there is no 'third party' and the signatory may have a direct relationship with the relying party more easily enabling the imposition of liability limits.

[177]Stephen Mason's work in particular includes an extensive international catalogue of these laws [243].

[178]A similar analysis could apply in circumstances where enterprises order their members of staff to adopt and install trust certificates issued by the enterprise specifically to support SSL/TLS inspection. Such enterprises should consider the various types of liability that might arise as a result.

[179]States may also apply embargoes on most or all trade with specific states as part of a more general programme of sanctions.

[180]The precise status of software as speech for purposes of US free speech law remains somewhat murky. While US Federal courts seem willing to classify source code as protectable expression, they also appear to take its inherent functionality into account when assessing free speech rights. This in turn suggests that government intervention to restrict acts of distributing source code are more easily justified than restrictions on classic non-functional speech [257, 339, 340].

[181]The term 'pubic international law' is often referred to more simply as 'international law'. By contrast, the field of 'private international law' describes the process of determining which domestic state law(s) will be applied to various aspects of private law disputes such as tort and contract actions. Aspects of private international law, or conflicts of law, are considered in this knowledge areas in the context of individual substantive legal subjects.

[182]In the referenced *Halford* case the complaining party successfully argued that the United Kingdom had failed to provide her with privacy rights required by the European Convention on Human Rights as regards interception of communications by state authorities [138]. This case precipitated the adoption by the UK of comprehensive legislation regulating the interception of communications.

[183]A notable exception involves prosecution according to the principles of international criminal law such as crimes against humanity.

[184]The process of creating the Tallinn Manual was not without controversy, and even the conclusions expressed in the 'Rules' (which represent unanimous consensus among the large expert group) are not universally agreed [87, 341]. The Tallinn Manual itself helpfully provides extensive commentary that highlights circumstances where some states disagree with the unanimous views of the expert group, and other issues where the expert group itself did not achieve consensus. It is therefore not surprising that the Tallinn Manual 2.0 does not represent the official policy of the project's sponsors (NATO and its member states) or the various additional organisations whose experts participated in its creation and revision. Nonetheless, anecdotal evidence suggests that experts who advise all of these persons keep a copy of the Tallinn Manual close at hand and consult the work routinely.

[185]The term 'attribution' is often used in more than one sense. Practitioners should be careful to distinguish the legal doctrines used to analyse attribution from the process of collecting and presenting evidence intended to prove attribution. This section discusses only the former. The latter is more properly addressed in the field of forensics.

[186]The principle of territoriality and the exercise of state power is explored in the context of jurisdiction in Section 3.2

[187]Espionage during armed conflict is treated separately under the law of armed conflict. See, for example, ([87] at R.89.)

[188]See, for example, the discussion in Tallinn 2.0 of 'herding' target state communications to less secure infrastructure by interfering with more secure infrastructure ([87] at R.32, cmt.12).

[189]The qualifier 'unrelated' state is meant to distinguish circumstances where more than one sovereign state maintains concurrent jurisdiction over a single territory, as found in federal states.

[190]The term 'law of war' significantly predates the term 'law of armed conflict', but is now used less frequently especially as armed conflict often takes place in the absence of any formal declaration of war. 'International humanitarian law' (IHL) is a more recent term [265] at p.8, fn.5. The adoption and use of 'IHL' to describe this field is not without controversy [342].

[191]Although this should be obvious, the concept of 'cyber attack' used to discuss obligations under international law is significantly more narrow than the term 'attack' which is broadly defined for most other purposes in cyber security. Cyber security practitioners tend to use the term 'attack' to describe any effort to obtain unauthorised access to a system, resources, or information, or any malicious activity intended to collect, disrupt, deny, degrade, or destroy information system resources [343].

[192]In the case of offensive cyber operations undertaken at the direction of a state, compliance with 'all applicable laws' may indeed be impossible as the actions taken at the direction of a sponsoring state may constitute crimes under the domestic law of the target state. Similarly, in some circumstances a practitioner might complain that compliance with a legal obligation is, itself, ethically challenging [344]. This section does not attempt to resolve these issues.

[193]Practitioners should be mindful that if they are employed or engaged by a regulated professional firm (e.g., a legal, medical, or public accountancy firm) the practitioner may be obliged by applicable law to conform with the rules of the relevant regulated profession – especially on matters such client confidentiality or client's legal privilege to prohibit the disclosure of sensitive information. These practitioners must become familiar with the obligations imposed by the regulations that apply to that firm.

[194]This discussion does not address circumstances where applicable law mandates disclosure of this evidence to identified third parties, such as the data breach disclosure requirements imposed by GDPR. In such cases, a practitioner should be careful to take appropriate advice concerning their individual legal reporting obligations as distinct from obligations imposed upon their client, and to urge their client to investigate the client's own potential legal reporting obligations.

[195]Many early examples include mandates to 'comply' with the law, while others demand that a practitioner should 'be aware' of the law. Some include the concept of avoiding harm to others without discussing the subtleties of this proscription. Some speak of a general affirmative obligation to protect 'society', without identifying the nature of the obligation in practice, identifying the relevant society in cases where two societies are in conflict, or discussing the possible conflict between protecting society generally and a client individually. Some speak of obligations to protect 'infrastructure', without clarifying whose infrastructure is to be protected: public, private, first-party, third-party, domestic, or foreign. Many of these codes fail entirely to discuss specific obligations owed to a client and how to manage potential conflicts.

[196]CREST has subsequently added additional services to its certification process.

[197]See also Orin Kerr's extensive discussions of 'authorization' in the context of US computer crime statutes [182, 183].

[198]Some risks of disclosure might include the impracticability of patching or fixing the vulnerability. The benefits of secrecy might include a state security agency's ability to exploit the given vulnerability.

[199]This is a special threat for finders engaged in academic security research who face normal academic pressure to publish research results [321].

[200]While disclosing a unique vulnerability in a single online service to a single effected firm is simple, disclosing a vulnerability in a complex supply chain presents special problems. Disclosing first to downstream producers of finished goods or services focuses the disclosure on those who appear to have the most at risk from security failure, but who may not have the tools necessary to mitigate the threat. This downstream disclosure also creates a risk of alienating the upstream developer of the component – especially if the vulnerability is misdescribed. In the field of academic security research in particular, researchers often depend on good continuing relationships with the developer community. Disclosing first to the upstream developer creates a challenge if that developer is dilatory in remediating the vulnerability. Finders in this situation might consider the potential for a multi-step private disclosure process starting (perhaps) with the upstream party most likely to be able to understand and remediate the threat. Having disclosed and then provided an opportunity for that party to analyse or rebut the claim of vulnerability, the finder might begin additional private disclosures one step at a time down the supply chain to those who might take action to mitigate the threat. Second-stage public disclosure would then become a last step of many.

[201]Commenting on a decision by academics to publish vulnerability details more than nine months after private disclosure to an upstream security component vendor, but in the face of strong objections by a downstream purchaser who incorporated the compromised security product into their mass-produced automobiles, an English high court Judge noted, 'I think the defendants' mantra of "responsible disclosure" is no such thing. It is a self-justification by defendants for the conduct they have already decided to undertake and it is not the action of responsible academics.' *Megamos Crypto (Volkswagen v Garcia)*, Para 42 [232, 233]. Note that the academics in the *Megamos Crypto* case claimed that they were adhering to then-current responsible disclosure procedures published by the National Cyber Security Centre of the Netherlands, the state in which they conducted the bulk of their research.

[202]While the mere presence of a vulnerability in a product or service does not necessarily constitute vendor negligence, vendors who receive a vulnerability report should consider that a failure to act in a reasonable fashion following receipt of such a report could constitute an independent act of negligence.

[203]Some have attempted to address this issue by adopting legislation that would regulate the disclosure process. One (as yet) unsuccessful attempt in Latvia is helpfully described in [322].

[204]An example likely to be familiar to practitioners was the fate of the Arthur Andersen accounting firm in the early 21st century. The firm was an adviser to the Enron Corporation, and was accused by the US government of improperly destroying evidence related to the Enron investigation. A federal jury returned a guilty verdict in the firm's 2002 criminal trial [345]. Upon conviction, the firm was debarred from conducting public company audit work effectively ending its ability to operate as a going concern. The criminal conviction ultimately was overturned by the US supreme court in 2005 [346]. This came too late for the firm, which had ceased ongoing operations.

Chapter 4
Human Factors

M. Angela Sasse | Ruhr Universität Bochum & University College London
Awais Rashid | University of Bristol

4.1 INTRODUCTION: UNDERSTANDING HUMAN BEHAVIOUR IN SECURITY

In their foundational 1975 paper, *The Protection of Information in Computer Systems*, Jerome Saltzer and Michael Schroeder established ten principles for designing security [8]. Three of those principles are rooted in the knowledge of behavioural sciences:

- *Psychology:* the security mechanism must be 'psychologically acceptable' to the humans who have to apply it;
- *Human Factors and Economics:* each individual user, and the organisation as a whole, should have to deal with as few distinct security mechanisms as possible;
- *Crime Science and Economics:* the effort required to beat a security measure should exceed the resources and potential rewards for the attacker.

Nearly 100 years before Schroeder & Saltzer, the founding father of cryptography, Auguste Kerckhoffs formulated six principles for operating a secure communication system, with a key focus on human factors: Three of those were *"it must be easy to use and must neither require stress of mind nor the knowledge of a long series of rules"*.

Both of these foundational texts recognised that security measures cannot be effective if humans are neither willing nor able to use them. A good example is email encryption. We have had tools for encrypting email for over 20 years. Yet today, less than 0.1% of emails sent are end-to-end encrypted. This outcome was predictable since Whitten & Tygar found in 1999 that even well-motivated and trained people could not use email encryption correctly [347]. This situation has not yet changed substantially—although recent research offers insights into the means to do so [348, 349, 350].

Over the past 20 years, there has been a growing body of research into the underlying causes of security failures and the role of human factors. The insight that has emerged is that security measures are not adopted because humans are treated as components whose behaviour can be specified through security policies, and controlled through security mechanisms and sanctions. But the fault does not lie primarily with the users, as suggested by the oft-used phrase that humans are the 'weakest link', but in ignoring the requirements that Kerckhoffs and Schroeder & Saltzer so clearly identified: that security needs to be usable and acceptable to be effective. An example of this is the case of password policies. Adams & Sasse showed that password policies and mechanisms agreed upon by security experts did not work at all in practice and, consequently, were routinely bypassed by employees [351]. Naiakshina et al. showed that not only end-users have trouble with passwords but developers do as well. Developers need to be explicitly prompted to include security and, even when this is done, they often include outdated and faulty security mechanisms [352, 353].

The aim of this CyBOK Knowledge Area is to provide a foundational understanding of the role of human factors in cyber security. One key aspect of this is how to design security that is usable and acceptable to a range of human actors, for instance, end-users, administrators and developers. This knowledge area also introduces a broader organisational and societal perspective on security that has emerged over the past decade: the importance of trust and collaboration for effective cyber security, which can only be achieved by engaging stakeholders and negotiating security solutions that meet their needs [354]. This requires a set of skills that have traditionally not been part of the training provided for security experts and practitioners. This knowledge area aims to capture the knowledge to change that.

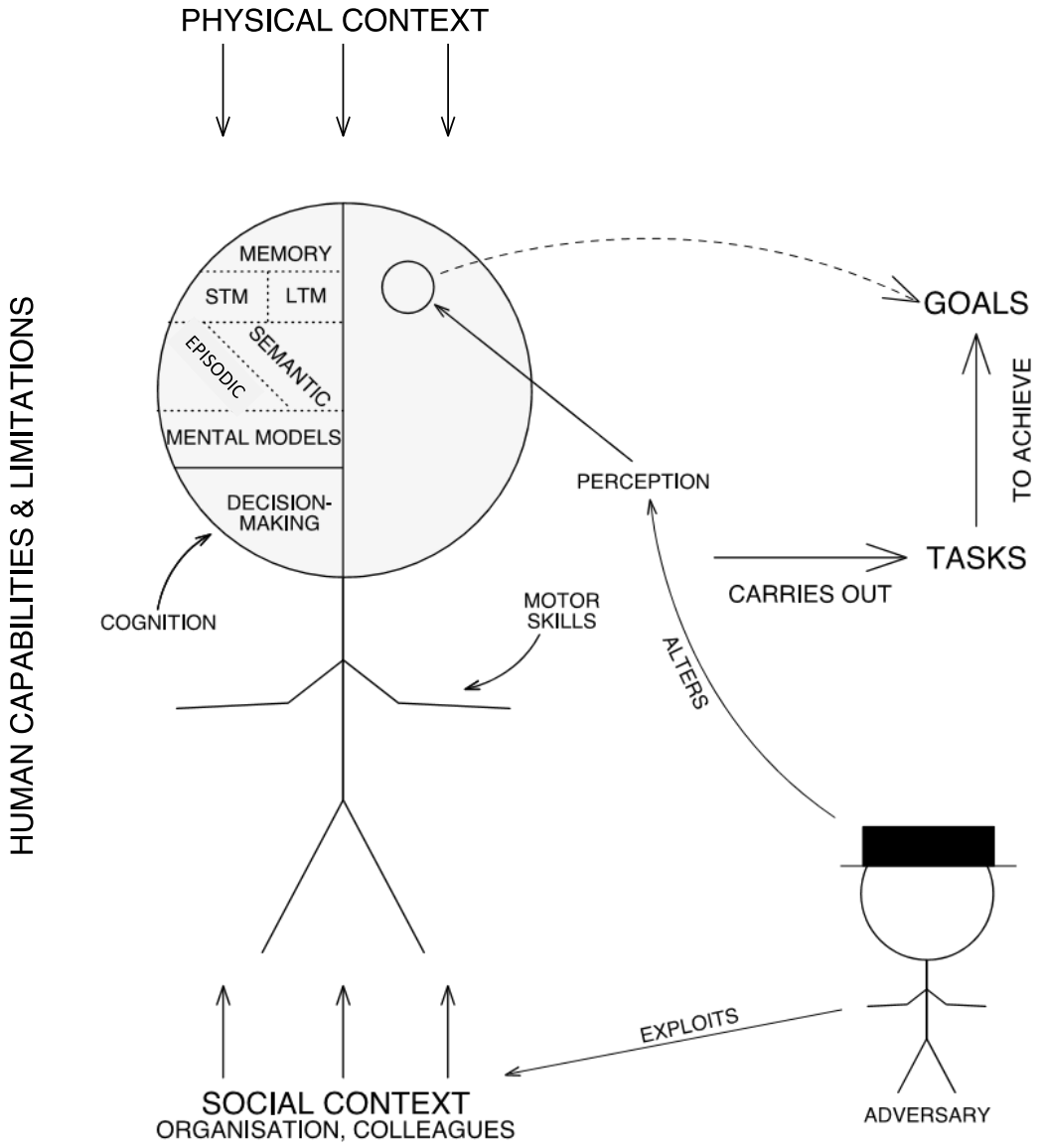


Figure 4.1: Human behaviour in context, showing internal factors and contextual ones that influence behaviour.

This knowledge area is organised (Figure 4.1) in a *starting on the inside, working outwards* manner: starting with the individual and internal factors that drive human behaviour (capabilities and limitations, mental models), moving onto aspects of the broader context in which interaction with security takes place. We will then consider the other immediate factors that have an impact: the behaviour of others around us, and especially how they handle security risks, users' emotional stances towards the organisation and how security behaviour can be successfully managed through design and a range of group and organisational factors. Note that human factors and usability in a security context can be distinguished from other contexts by the presence of adversaries or risk. As shown in Figure 4.1, the adversary may actively work to alter users' perceptions of the system's capabilities and boundaries as well as exploiting the specifics of social and organisational contexts (e.g., security policies, working practices, decision-making hierarchies) to impact security. Studying usable security through an active attacker model [355, 356] and raising users' awareness about security issues by incorporating such models, e.g. anti-phishing simulations [357, 358], is an on-going area of study. These mechanisms offer some protection, but require user time and effort. Therefore,

as we discuss later, the total security workload needs to be monitored so that productivity is not reduced and workarounds induced. Furthermore, they have implications in terms of users' trust in the organisation and completion of the primary (non-security) task at hand – the design of any such interventions or campaigns needs to consider and address these risks [358].

Note that we do not discuss the specifics of adversarial behaviours, as these are the subject of the Adversarial Behaviours Knowledge Area (Chapter 7). However, we will touch on any relevant elements where they relate to usability and human factors, for example, security awareness, training and anti-phishing. Usability considerations are equally important with regards to privacy controls and technologies. This discussion formulates part of the Privacy & Online Rights Knowledge Area (Chapter 5) and hence is not considered here any further.

4.2 USABLE SECURITY – THE BASICS

[359, 360]

When users do not behave as specified by security policies, most security practitioners think that the users are at fault: that they 'just don't understand the risks' or 'are just too lazy'. But research has shown that non-compliance, which we now refer to as 'rule-bending', is caused by people facing a stark choice between doing what is right by security, and reducing their productivity. Most choose productivity over security, because that is what the organisation also does.

A typical response to such rule-bending is security awareness and education, that is, *'fitting the human to the task'*. But Human Factors research established decades ago that, when we take all of the costs and the resulting performance into account, *'fitting the task to the human'* is more efficient. There is a role for security awareness and training (Section 4.4) but it should be thought of as one of the options but not the first resort. It cannot help humans to cope with security tasks that are impossible, error-inducing, or drain too many individual and organisational resources [361]. As the UK's National Cyber Security Centre (NCSC) policy puts it:

> 'The way to make security that works is to make security that works for people[1]'

In other words, security has to be usable. The ISO defines usability (ISO 9241–11:2018) as

> 'The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.'

And the criteria by which usability is assessed are:

1. *effectiveness:* the accuracy and completeness with which specified users can achieve specified goals in particular environments;
2. *efficiency:* the resources expended in relation to the accuracy and completeness of the goals achieved;
3. *satisfaction:* the comfort and acceptability of the work system to its users and other people affected by its use.

[1]https://www.ncsc.gov.uk/information/people-strongest-link

We can immediately observe that these criteria align with the principles articulated by Kerckhoffs and Saltzer & Schroeder's. But how to deliver this in practice?

4.2.1 Fitting the task to the human

From a practical point of view, making security tasks fit or usable means establishing a fit with four key elements [360]:

1. the capabilities and limitations of the target users;
2. the goals those users have, and the tasks they carry out to achieve them;
3. the physical and social context of use; and
4. the capabilities and limitations of the device on which the security mechanism is used.

We now examine each of these in turn, and how they apply to designing a usable security mechanism.

4.2.1.1 General human capabilities and limitations

There are general capabilities and limitations – physical and mental – that apply to most humans. Giving humans a task that exceeds their capabilities means we set them up to fail. When the demand they face is borderline, most humans make an effort to meet it. But this will come at a significant cost, which may ultimately prove to be unsustainable.

With general computing devices today, the physical capability that can be exceeded by security tasks is most likely the ability to detect signals: many security systems provide status messages, reminders or warnings. Humans can only focus their attention primarily on one task at any one time. That focus will be on their main activities, and many security mechanisms demand more time and attention than users can afford [359]. This means that changes in passive security indicators are often not noticed, in particular if they are on the edges of the screen. Asking users to check these indicators is setting them up to fail—even if they consciously try to do it, their focus will be drawn back to the main task. If security indicators need to be attended to, they should to be put in front of the person, and require a response. This will work, but only for infrequent and reliable indicators (see Alarm fatigue).

Alarm fatigue The brain stops paying attention to signals it has classified as irrelevant. They are filtered out before they reach the conscious processing level (Section 4.2.1.2). It means humans do not perform well on tasks where they have to screen for rare anomalies (e.g., in baggage screening and some forms of Closed Circuit Television (CCTV) monitoring). We need technology support and processes such as job rotation to get good performance. Alarm fatigue is a related phenomenon. Once alarms have been classified as unreliable, people stop paying attention to them. How high a false alarm rate (with which people can work) depends on the risk, the frequency at which false alarms occur, and the demands of the other tasks they have to complete. But even with a 10% false alarm rate, one can expect alarm fatigue. Once people start to dismiss alarms, it is hard to get them to take them seriously again. Moreover, once they dismiss one type of security warning as false, similar-looking or sounding ones will also be dismissed. Many security warnings today have far too high a false alarm rate and are thus dismissed. SSL certificate warnings, for instance, have a false-positive rate of 50% or more. So, it is not surprising that people ignore them, particularly if no secure alternative for completing the task is offered at the same time. Rob Reeder, when working at Microsoft, coined the handy acronym NEAT: warnings should be Necessary, Explained, Actionable, and Tested [362]. Add to that 'and have a false alarm rate of 10% or less' and one may have a chance of security warnings being effective.

A key mental capability is memory. There are several types of memory. The first distinction is between Short Term Memory (STM) and Long Term Memory (LTM). When one tries to memorise an item, it needs to go round the STM loop a few times before it is transferred into the LTM. STM is what is, for instance, used for one-time passwords, such as numeric codes displayed by tokens or displayed on another device.

STM and One Time Passwords (OTPs) The use of one-time PINs or passwords (OTPs) in security has increased as Two Factor Authentication (2FA) has become more common. We focus our attention on the number displayed and repeat it to ourselves (mentally or aloud). Then we turn our attention to the entry field, retrieve the item from the STM loop, and repeat it to ourselves while entering it. What is important to note is that this works for most people for strings of up to 6 characters, that is, a 6-digit number, because we can break them into 2 bits of 3 characters each. Codes that are longer overload the STM loop. People have to start looking forwards and backwards between the display to read the characters and enter them. This increases both the entry time and the likelihood of error. And mixing alpha-numeric characters also impacts performance.

Whether a user will be able to recall what is stored in LTM depends on how embedded it is: items retrieved frequently are well embedded, those that are not will fade over time. That means we can expect problems with infrequently used items that require unaided recall (aided recall, e.g., recognising one's own images in a set of images, is easier). LTM is divided into two distinct areas: general knowledge is stored in Semantic Memory (LTM-SM), whereas items connected to one's personal history are stored in Episodic Memory (LTM-EM): autobiographical memory. Items stored in LTM-SM fade faster than those in LTM-EM because, in the latter case, one stores not just the item, but the images and emotions connected to them.

LTM and passwords LTM-SM is divided into areas in which similar items are stored. When one tries to retrieve an item, the section in which it is stored is activated, and the items in the section compete to be retrieved – with those that have been retrieved most frequently 'coming to mind' first. This *interference effect* is quite powerful and disruptive, particularly because items one does not need any more (such as old passwords) keep lingering and compete with those that need to be recalled. Thus, managing a multitude of the same type of credentials is impossible, especially if several of them are used infrequently. People need coping strategies, be it writing them down, using a password manager or one-time credentials. We can agree that 123456 or P@SSword are not secure. But, since most users now have dozens of passwords, the insistence on *strong* passwords has created a humanly impossible task. Most people struggle if they have more than 2–3 passwords or PINs – and the longer and stronger they are, the more they will struggle.
The NCSC Password Guidance [a], therefore, recommends several ways of supporting people in managing large numbers of unique passwords: switching to 2FA solutions and/or password managers, and if it is not possible to do either, not expiring strong passwords on a regular basis. If a password has to be expired (e.g., because it has been compromised), a little time investment during the day the password has been changed can help. People can brute-force the old password out by repeating the new password around ten times immediately, and repeating that process three or four times at hourly intervals.

[a] https://www.ncsc.gov.uk/guidance/password-guidance-simplifying-your-approach

One important security criterion for knowledge-based authentication is that a credential should be difficult to guess. Due to human physical and mental characteristics, the selection of credentials is, however, often biased towards the familiar, or those that can be more easily distinguished from others.

1. With passwords, people try to pick ones that are easier to recall, e.g., those that have meaning for them such as memorable names or dates.
2. When users have to choose images as credentials, they prefer strong colours and shapes over more diffuse ones [363].
3. When these are pictures of humans, they will pick pictures of 'more attractive' people and those from their own ethnic background [364].
4. When the credential is a particular location within a picture, people prefer features that stand out [365].
5. With location-based systems, people pick memorable locations, for example, when choosing locations for a 4-digit PIN on a 5×5 number grid, they go for connected locations, anchored on an edge or corner of the grid [366].
6. The order of the elements of a credential is predictable, because there is a strong cultural preference, e.g., people who speak languages that read left-to-right will choose that order [366].
7. With finger swipe passwords on Android phones, people pick from a very limited number of shapes [367].

These human biases reduce the diversity (number of different passwords) in a password database, and increase the likelihood of an attacker guessing a password. To counteract this, security policies have barred *too obvious* choices. Whilst not allowing very obvious choices such as 'password' as a password and '0000' as a PIN is prudent, having too many restrictions

increases the workload associated with the password creation task (see Section 4.2.1.2). For instance, a password checker that rejects 5+passwords in a row as *too weak* will put users under considerable stress and most likely towards re-using a password.

Similarly, password strength meters are often used to guide and influence the user's password choices. For instance, Ur et al. [368] discussed the impact of various password meter designs on users' choice of passwords, as well as highlighting the increased workload for users and the frustration faced by them when faced with more stringent password meters. A recent work by Golla and Dürmuth [369] investigated the accuracy of 45 password strength meters including several deployed in practice, as well as academic proposals. Their work shows a degree of variation in terms of accuracy and, more critically, that this has not significantly improved over five years. So, even if we are to disregard the additional workload on users (not that we should), these approaches do not always have the level of accuracy required to effectively implement password policies. These considerations must be borne in mind when deploying solutions to *enforce* security policies.

Sometimes, the question is raised as to whether there is training to help users cope with recalling security credentials. Memory athletes use specific exercises to enhance memory performance. The writer Joshua Foer details in his bestseller *Moonwalking with Einstein* [370] that it requires a serious initial time investment (several months full-time) but also continuing training (at least 30 minutes a day), plus the time required to recall and enter the passwords (which in itself people find too much [371]).

We have, so far, discussed the capabilities and limitations that apply to most people. But, specific user groups will have additional needs that should inform the selection or configuration of security mechanism or processes. For instance, children and older citizens can have capabilities and limitations (e.g., motor skills) that differ from working age adults. People with larger fingers struggle to hit small targets accurately, such as the small keys on a soft keyboard. Cultural values and norms need to be considered. The physical and mental conditions of users also need to be taken into account. Not all users are able to operate equipment with their hands, read from screens, or hear audio. Conditions such as colour blindness affect sizeable numbers of people, so images used for graphical authentication need to be checked. Certain audio or video effects can harm users with conditions such as autism or epilepsy.

CAPTCHAs Some work on Completely Automated Public Turing test to tell Computers and Humans Aparts (CAPTCHAs) has investigated supporting users with sensory impairments, e.g., [372]. However, one needs to bear in mind that CAPTCHAs add more effort for the legitimate user, impeding the achievement of the intended goal, i.e., access. The usability limitations of these mechanisms that aim to 'verify' legitimate human users – and their contribution to security fatigue – must be considered [373, 374].

4.2.1.2 Goals and tasks

Human behaviour is essentially goal-driven. People perform tasks to achieve goals, at work: 'I want to get this quotation to our customer today', or in their personal life: 'I want to get the best utility deal for us'. To achieve these goals, people complete a series of tasks. To prepare a quotation, these would include working out the materials required and their cost, the person-hours required and their cost, the relevant fees, taxes etc. If a task has several steps or units, it can be decomposed into sub-tasks. For instance, working out the person-hours required on a job can be broken down into the following tasks:

1. identify all the worksteps that need to be completed,
2. work out what type of employee is required to complete each task,
3. how long each specific type of employee needs to spend on which task,
4. what preparations each type of employee may need to make.

These tasks are called primary or *production tasks* in human factors terminology, and designing the technology tools so people can complete these tasks effectively and efficiently is the most fundamental aspect of usability. To ensure people can complete tasks effectively, technology (and security) designers need to know the requirements for the tasks they perform:

Production and enabling tasks Production tasks are what people consider 'their job', and in many jobs, they may have spent years studying or training for them. At an organisational level, the production tasks performed by many individuals in an organisation add up to business processes that produce the goods or services. Anything that stops these processes or slows them down will cause the organisation significant problems. When we talk about '*resilience*' of an organisation, it is about the ability to keep those business processes going to produce the output. As well as production tasks, an organisation has tasks that do not directly contribute to business processes, but have been added to protect its ability to keep going in the long term: safety and, indeed, security are key enabling tasks. Some organisations get away with not supporting these enabling activities for a period of time and this explains the *grudge* with which some individuals and organisations view security. The fact that safety or security measures do not immediately contribute to the output and the bottom line explains why it is a grudge sale, particularly when individuals or organisations feel under pressure.

1. What output has to be produced so the goal is achieved? The task has to be completed effectively, e.g., if the quotation is not correct or not sent to the customer in time, the task is not completed effectively.
2. Are there constraints on time and resources? Business processes may set an upper limit on the time tasks can take, or the resources they can draw upon, such as, access to information or services for which the organisation has to pay.
3. Is the task performed frequently (several times a day) or infrequently (once a month)? The execution of tasks people perform frequently becomes 'automatic', whereas new or infrequently performed tasks are completed in a conscious, step-by-step manner (see Section 4.2.1.4). For frequently performed tasks, the design should optimise for speed and reduce physical effort (which could lead to fatigue). For infrequent tasks, the design should try to reduce mental effort by guiding the users and minimising how much they have to remember.

People focus on the production task, and enabling tasks are often experienced as an unwel-

come interruption or distraction. To stay with our authentication example[2]: an employee has to authenticate with a password to a database to find out the hourly rate of a particular specialist that the business charges. If she does this frequently, and can remember the password, it may only take a few seconds for her to recall and enter it. But if she has just returned from a vacation, cannot remember it and it takes 20 minutes to get through to a help desk to have it reset, and then she has to think up and memorise a new password – all before she can get to the database – the security task has suddenly become a massive disruption, and perhaps the effective completion of the production task is now under threat.

And note how one seemingly quick task of 'authenticate' (with 2 subtasks of 'recall password' and 'type password') has now spawned two further authentication tasks: 'recover password' and 'create password', both with multiple steps each.

Most workarounds to security mechanisms, such as, writing passwords down or sharing them, happen because people try to ensure effective production task completion (to protect business productivity). For instance, people often keep their own copies of documents that should be in an access-controlled repository, or clear-text copies of documents that should be encrypted, because they fear not being able to access them when they need them. Or when the repeated effort and disruption resulting from having to enter a password to unlock a screen gets too much, they install mouse-jiggling software to stop the screen locking and having to enter their password [371]. Even if a user knows the password well, the seconds it takes add up if it needs to be done dozens of times a day.

Therefore, to avoid security tasks being bypassed, we must design them to fit into primary tasks. We can achieve a good fit in a number of ways:

- Automating security, for instance, using implicit authentication to recognise authorised users, instead of requiring them to enter passwords many times over.
- If explicit human action is necessary in a security task, we should minimise the workload and the disruption to the primary task.
- Designing processes that trigger security mechanisms such as authentication only when necessary (see, for example, [375]).
- Design systems that are *secure by default*[3] so that they do not push the load of security configurations and management on to the users.

Workload can be physical (typing a password) or cognitive (remembering a password). Humans generally try to be efficient and keep both their physical and mental workload as low as possible. But, given a choice, most people will take an extra physical over extra mental workload, especially if the physical task is routine and can be done 'on autopilot' (See Section 4.3). Mental workload quickly becomes too much, especially if adjacent tasks require the same mental capability, such as memory.

Therefore, in order to design a security task that fits well, we need to know the production tasks, and consider the mental and physical workload. Before selecting a security measure, security specialists must carry out a workload audit:

1. What is the workload associated with the primary and secondary (security) task?

[2]We could equally consider other examples, for instance, *access control* where the user perspective is: 'I need to share information X with person Y' whereas access control policies take the approach: 'deny all and then enable specific access'.

[3]**https://www.ncsc.gov.uk/information/secure-default**

2. Are there performance constraints in the primary task (e.g., the time in which it has to be completed)?
3. Are there resource constraints (mental or physical capability, or external ones such as limited access to paid services)?
4. What is the impact of failing to complete the security task?

Workload measurement How can we measure the workload associated with a security task? A simple proxy is the time: how long does it take to complete the security task? Considering this before implementing a new policy or security measure would be an improvement on the status quo, whereby the impact of a policy or measure is only considered once it is causing problems. Once we know how long it takes, we need to determine if and where it disrupts primary activity. The assessment of whether the impact on the primary task is acceptable can be carried out informally, for instance, with experienced staff and line managers who know the production task well. A more formal assessment can be carried out analytically using the Goals, Operators, Methods (GOMS) method or empirically using the NASA Task Load Index (TLX).

As we have already discussed, people are hardwired to protect their productivity. They have a built-in awareness of how much time and effort they are spending on non-productive tasks, and an idea of how much non-productive activity is reasonable. They have what Beautement et al. called a *Compliance Budget* [376]. As the day progresses and enabling tasks add up, the likelihood that they will seem *too much* and be bypassed increases. Furnell & Thompson coined the term *security fatigue* [377] and the uphill battle to turn security from a grudge sale into a positive quality (Section 4.2.1.3) can be attributed to this.

Security is not the only enabling task employees face. Others include: safety, sustainability, diversity training, various regulatory regimes and so on, leading to *Compliance Fatigue*. Beautement et al. [376], recommend that security specialists have an open and honest discussion with line managers and business leaders about the time and budget available for enabling activities, and how much of it is available for security versus other enabling functions. Once that is known, the workload of the security tasks can be calculated and priorities identified – which security behaviours really matter for the key risks a particular group of employees face – and security tasks streamlined. Making security mechanisms smarter and less 'all or nothing' can also help reduce compliance fatigue. For instance, allowing authentication with an old password, or having 'break the glass' policies that allow but flag access by users who do not have permission reduces the likelihood of task disruption. And if users know they have access to efficient security recovery and support services, it will reduce the need for workarounds.

4.2.1.3 Interaction Context

Contextual Inquiry In modern work organisations, staff can work in many parts of the world, and in many different physical and social environments. It can be quite a challenge for a security expert to identify all the factors that could impact security and usability. Many usability professionals follow an approach called Contextual Inquiry [378]:

> 'The core premise of Contextual Inquiry is very simple: go to the user, watch them do the activities you care about, and talk with them about what they're doing right then.'

Contextual Inquiry uses a mixture of observation and interview to identify the primary tasks people are carrying out, and what makes them do this well.

Both the physical surroundings and the social environment in which people have to perform security tasks affect performance and security. Most working age people now interact with technology *on the move* more frequently than *at the desk* traditional working environments. This change in the context of use affects a number of security mechanisms, not least of being overheard when on the phone – the case of former CIA Director Michael Hayden being overheard giving an off-the-record interview on board a train being a particularly spectacular one[4]. The risk of being overheard is now addressed in many corporate training packages, but several security mechanisms are still in use that are vulnerable to being overheard, e.g., security questions such as date of birth, mother's maiden name. Using partial credentials only and entry via keypad increases security but also accentuates the mental and physical workload at the same time. Some attackers can also try to glean credentials via shoulder-surfing or hidden cameras. Overall, the use of a One Time Password (OTP) as part of a 2FA solution could offer protection *and* better usability.

The usability of security mechanisms can be affected by the following physical characteristics:

1. *Light:* In bright light, displays can be hard to see, which can affect graphical authentication in particular. Biometric systems such as iris and face recognition rely on input from cameras. Bright light can lead to glare, which means the images captured are not good enough to process.
2. *Noise* will most obviously interfere with the performance of voice recognition systems. But high levels of noise also impact human performance in general due to increased stress and, in turn, increased likelihood of error. Unexpected loud noises trigger a human startle response, which diverts attention away from the task.
3. *Ambient temperature* can affect the performance of both technology and humans. Fingerprint sensors can stop working when it is cold, and humans are slower at pointing and selecting. They may also need to wear protective clothing such as gloves that make physical operations of touchscreens impossible or difficult. Similarly, too hot an environment can lead to discomfort and sweat can interfere with sensors.
4. *Pollution* can impact equipment operated outdoors. This is a particularly concern for fingerprint sensors and touchscreens. The lipids left behind combine with the particles and the resulting dark grease can clog sensors or leave a clearly visible pattern on the touchscreen.

[4] https://www.theguardian.com/world/2013/oct/24/former-spy-chief-overheard-acela-twitter

The social context in which people find themselves strongly influences behaviour though *values*: shared beliefs about what is important and worthwhile, and *norms*: rules and expectations about actual behaviour. If the expected security behaviour is in conflict with day-to-day behavioural norms, we can expect problems. For instance, if an organisation values customer satisfaction, and employees are told to be friendly towards customers at all times, a security policy that requires staff to treat any customer enquiry as a potential attempt to extract information will not fit. Understanding the reasons underpinning non-compliance with security policies can shed light on these conflicts between security requirements and the primary task [379]. Trust is another key norm. Humans do not like to feel distrusted – and it has been shown that communicating distrust to employees encourages bad behaviour, rather than prevent it [380].

Other aspects need to be considered in order to understand how security beliefs, norms and coping strategies are shaped. For instance, users often get their knowledge from their wider social networks and these are also a source of support and help when they face usability challenges [381, 382].

4.2.1.4 Capabilities and limitations of the device

We have already discussed that the physical characteristics of a device may make interaction with security mechanisms difficult in certain circumstances. Some characteristics of the device can result in security mechanisms becoming difficult to use in any circumstance. Entering long and complex passwords on soft keyboards on a mobile phone takes far longer and is more error-prone than on a regular keyboard [383]. And while with frequent use on a keyboard, most people can become quite proficient at entering a complex password, performance does not improve when humans hit a basic limitation. What is particularly worrying from a security point of view is that (without colluding) a user population starts to converge on a small number of passwords that are easiest to enter with the minimum amount of toggles, which makes guessing a valid password easier for attackers [384].

Whilst 2FA has security benefits and reduces the need for strong passwords, not all 2FA solutions are usable by default. Many users find widely used 2FA tokens such as Digipass difficult. They appreciate the fact it fits into their wallet, but it is ultimately 'too fiddly' [385]. Also, over half of online banking users have accounts with more than one financial services provider. The fact that even those that use 2FA implement it differently (which token is used when it has to be used, and how the different elements of authentication are referred to (passphrase, passcode, key phrase) causes confusion for the users. Similarly, different implementations of Chip and PIN create slightly different variations in the task that catches users out, leading to human error (Section 4.3).

With increasing numbers of new devices appearing, from smart watches to home devices, and even smaller screen sizes and implicit interactions between users and devices through a variety of sensors and actuators, considering the ergonomics of security interactions [386] is ever more important. The risks arising from Bring Your Own Device (BYOD) cultures are discussed in the Risk Management & Governance Knowledge Area (Chapter 2).

4.3 HUMAN ERROR

[11, 387]

In over 30 years of research into accidents and safety, the psychologist James Reason worked out that virtually all mistakes people make are predictable [11]. They occur as a result of *latent failures* (organisation and local workplace conditions) and *active failures* (errors and violations by humans) in combination to allow the accident to occur. Figure 4.2 shows Reason's 'Swiss Cheese' model adapted for security. A security incident occurs because the threat finds its way through a series of vulnerabilities in the organisation's defences. A person may be the one who pushed the wrong button or clicked on the link and caused the incident. However, several other failures preceded this, leading to that person being put in a position where making what appeared to be the right choice turned out to be the wrong one.

> ***Latent usability failures in systems-of-systems*** One can also not assume that all systems are designed from scratch with usable security considerations in mind. Most often systems are, in fact, systems-of-systems (SoS), derived from composing otherwise independent systems that come together to orchestrate a particular service or task. Integration problems in SoS have been studied, e.g., [388] and one must consider the latent failures that arise due to the decisions made during integration. Poor usability and task fatigue represents a sufficient risk to the security of the SoS to warrant upfront investment in order to avoid latent failures.

The work of Reason and his fellow safety researchers [389, 390] led to organisations being held responsible for fixing upstream safety issues as they are discovered, rather than waiting for an accident to happen. The concept of a *near miss* describes a situation where safety issues become apparent, but an accident is avoided at the last minute. In most industries that are subject to safety regulations, there is an obligation to report near-misses and investigate any failure as soon as it is discovered – with a requirement to address the root causes identified through the investigation so that future failures are mitigated.

Applied to security, an employee not following a security procedure constitutes an active failure and should be investigated and fixed. If the investigation shows that the conflicting demands of production task and security lead the employee to disregard security, the conflict is an underlying latent failure that the organisation needs to address. Often security non-compliance is ignored until an incident occurs. Unlike security, safety does not have active adversaries with whom to contend. But many improvements could be made to current security practices by applying safety concepts (as discussed in Section 4.2.1.2).

As already mentioned in Section 4.2.1.2, tasks that people carry out frequently become *automatic*, whereas tasks they are doing for the first time or very infrequently are carried out in a conscious, step-by-step manner. The psychologist Daniel Kahneman, 2002 Nobel prize laureate in economics for his work on human biases in decision-making, described the two areas, System 1 and 2, and the way they work as, *Thinking Fast and Slow* [391]. One very important insight is that the majority of activities people undertake are carried out in System 1 mode, and this is what makes us efficient. If people carried out most of their activities in System 2 mode, they would not get much done. Exhortations to 'Take Five'[5] every time before clicking on a link are unrealistic when people get dozens of work emails with embedded links. Furthermore, if without clicking on that link or giving personal information, there is no way of completing the primary task, productivity comes under serious threat. Unspecific advice such as 'just stop

[5] https://takefive-stopfraud.org.uk/

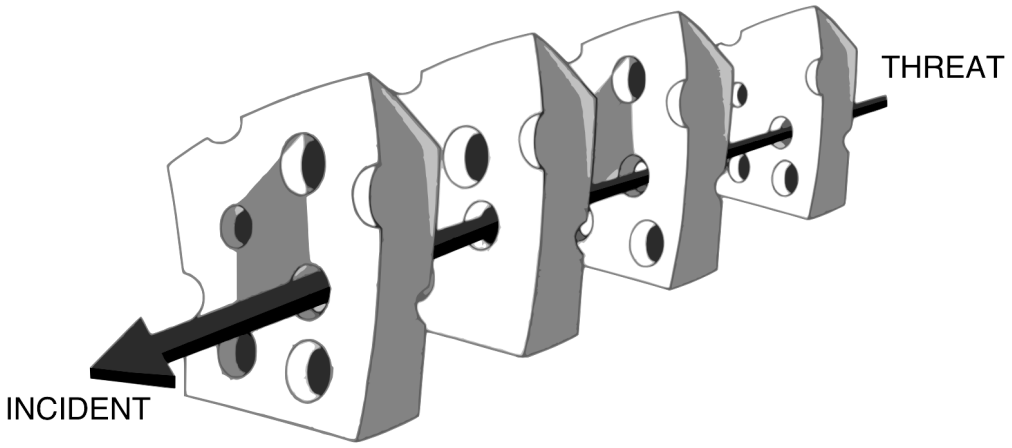


Figure 4.2: Security Version of Reason's 'Swiss Cheese' model. Holes are latent & active failures. When a threat finds one in successive layers then the threat succeeds. 'Cheese slices' are defences provided by security policies & mechanisms.

and think' rarely works because just stopping people in their tracks and without supporting them achieving their goals securely is not helpful. In addition, considering the workload of security measures, security experts need to consider the further impact that following their advice has on people's ability to complete their primary tasks, as well as the impact on the effectiveness of general communication between organisation and employees. The use of Domain-based Message Authentication Reporting and Conformance (DMARC), for instance, should enable employees to distinguish genuine internal communications from potential phishing attempts. The use of DMARC to provide a reliable indication of 'safe' senders can reduce the number of emails about which users have to be cautious. Even better, the provision of ultra-secure browsing technology, which is now available, means that clicking on links has no adverse technical consequences, so user education and training can focus on explaining social engineering and manipulation techniques.

When tackling complex problems, humans often have to combine both fast and slow processes, and there is an in-between *mixed-mode*, where task execution is not fully automatic: some of the behaviours are automatic, but one needs to stop and consciously work out which behaviour to select. Productivity costs aside, security experts suggesting people should 'stop and think' assume that 'slow mode' equals 'safe mode'. For instance, using slow mode can also lead to overthinking, to rationalising or explaining away evidence, to bringing irrelevant concerns to bear, focusing on the wrong goals (e.g., production goals), and to wasting large amounts of time and energy. In fact, each of these modes of operation comes with its own type of human error (Table 4.1).

Mode	Type of error	Cause	Security Example
Automatic mode (fast)	Slips and lapses	Recognition failure Memory failure Attention failure	"I forgot to check for the padlock before I entered my credit card details."
Mixed mode	Mistake I	Human chooses incorrect response	"I did not check for the padlock because websites on my iPhone are safe."
Conscious mode (slow)	Mistake II	Human does not know correct response	"I did not know to check for the padlock before entering my credit card details."

Table 4.1: Automatic, mixed mode and conscious workspace (based on [11])

Even in conscious mode, people try to be efficient, resorting to 'the closest thing they know', that is, they are most likely to choose behaviours they use frequently, or those that seem most similar to the situation they encounter. Attackers exploit this by creating very similar-looking websites, or incorporating security messages into their phishing emails.

Reason identifies four types of latent failures that are more likely to cause people to make errors.

1. Individual factors include fatigue (as discussed in Section 4.2.1.2), but also inexperience and a risk-taking attitude.
2. Human Factors include the limitations of memory (as discussed in Section 4.2.1.1) but also common habits and widely shared assumptions.
3. Task factors include time pressure, high workload and multiple tasks, but monotony and boredom are equally error-inducing because people shift their attention to diversions. Uncertainty about roles, responsibilities and rules also lead to incorrect choices.
4. Work environment factors include interruptions to tasks (as discussed in Section 4.2.1.2) and poor equipment and information. People are also particularly prone to error when rules and procedures change.

Task and work environment factors are clearly the responsibility of the organisation. There should be regular reviews of how well policies are followed. If they are not, the underpinning causes must be identified and addressed. The causes of near misses, mistakes that happened but did not lead to an incident, should be similarly used to identify and change the underlying causes. We also need to develop a better understanding of how humans respond when under stress conditions, e.g., in real-time when faced with an unfolding attack.

***'Never issue a security policy that can't be followed'* (Or: General MacArthur, *shadow security* and *security hygiene*)** The famous WWII military leader General Douglas MacArthur coined the phrase *'never give an order that can't be obeyed.'* He recognised the corrosive impact of a single order that cannot be followed in reality—it undermines the credibility of all orders and the superiors who issue them and seeds uncertainty and doubt. It is the same with security policies: when employees encounter security policies that are impossible to follow or are clearly not effective, it provides a justification for doubting all security policies. That is why *security hygiene* is essential. When policies are not being followed, security professionals must investigate, in a non-confrontational manner, why and if it is because they are impossible or too onerous to follow and re-design the solution. Kirlappos et al. pointed out that in most cases, employees do not show blatant disregard for security, but try to manage the risk they understand in the best way know how, what they call *shadow security* [379]. Their 'amateur' security solutions may not be entirely effective from a security perspective, but since they are 'workable', asking 'how could we make that secure' is a good starting point for finding an effective solution that fits in with how people work.

4.4 CYBER SECURITY AWARENESS AND EDUCATION

[392, 393]

Security practitioners often respond with security awareness, education and training measures when people do not follow security policies. But, in Section 4.3 we established that *security hygiene* must come first: if people keep being told that the risk is really serious and they must follow policy, but cannot do so in practice, they develop resentment and a negative attitude towards security and the organisation (which is counter-productive).

In practice, the three terms: awareness, education and training, are often used interchangeably but are different elements that build on each other:

Security Awareness. The purpose of security awareness is to catch people's attention and convince them security is worth the engagement. Given that many organisations face *compliance* and *security fatigue*, to quote Cormac Herley: *More Is Not The Answer* [359]: aiming a lot of communications will backfire. We need to capture people's attention, and get them to realise that (a) cyber security is relevant to them, that is, the risks are real and could affect them, and (b) there are steps they can take to reduce the risk and that they are capable of taking those steps. Crafting effective awareness messages is not an easy task for security professionals. Working with the communications specialists in an organisation can, therefore, help. They not only know how to craft messages that catch people's attention, but know how to reach different audiences via the different channels available to them, and integrate them into the overall set of communications to avoid message fatigue.

Security education. Once people are willing to learn more about cyber security, we can provide information about risks and what they can do to protect themselves against them. Most people currently have very incomplete and often incorrect mental models (see Section 4.4.2) on cyber risks. Transforming them into more accurate ones provides a basis on which to build cyber security skills. However, it is hard to ascertain whether the education leads to more accurate mental models or at least the ones that security professionals expect people to possess. This divergence must be borne in mind. For instance, Nicholson et al. [394] introduce the *Cybersurvival task* as a means to understand such divergence between security experts and employees in order to inform the design of security education programmes.

Security Training. Training helps people to acquire skills, e.g., how to use a particular security mechanism correctly, how to recognise and respond to a social engineering attack. In addition to showing people how to do something, we need to support the acquisition of skills by letting them practise the skills in a setting where they can 'experiment' with security decision-making and reflect on their perceptions and biases [395]. Parts of skill acquisition can be supported online, but, like all learning, it is much more likely to be successful when taking place in the context of a social community [396].

A common misunderstanding is that if people complete the three steps above and know what to do, they will change their behaviour. But knowing what to do and how to do it is not enough. As we discussed in Section 4.3, human activity is 90% automatic, driven by routines or habits stored in the long-term workspace. The new security behaviour needs to be embedded there but its place is occupied by an existing behaviour (similar to an old password). The adage that *'old habits die hard'* accurately describes the fact that until we manage to push the old behaviour out and the new behaviour becomes automatic, all our awareness, education and training efforts may not yield the changes in behaviour we are seeking. This is a challenging undertaking. Since productive activity needs to carry on while we change security behaviour (Section 4.2), we can only target 1–2 behaviours at a time, and embark on changing the next 1–2 only once these have become genuinely embedded. Nor should one conflate security awareness and education with security culture (cf. Risk Management & Governance Knowledge Area (Chapter 2)). These can be one element in developing a security culture but are not in themselves representatives of an effective security culture.

The RISCS White Paper *'Awareness is only the first step'* [392], presents a model of support (Figure 4.3 that organisations need to provide to achieve security behavioural change. It shows that the three steps we have discussed so far are only the first steps, and that a further four steps are required to achieve behavioural change. To support these additional steps, we can draw on a new generation of learning resources that have evolved. And such steps require investment from organisations - in terms of strategy, time, planning and resources.

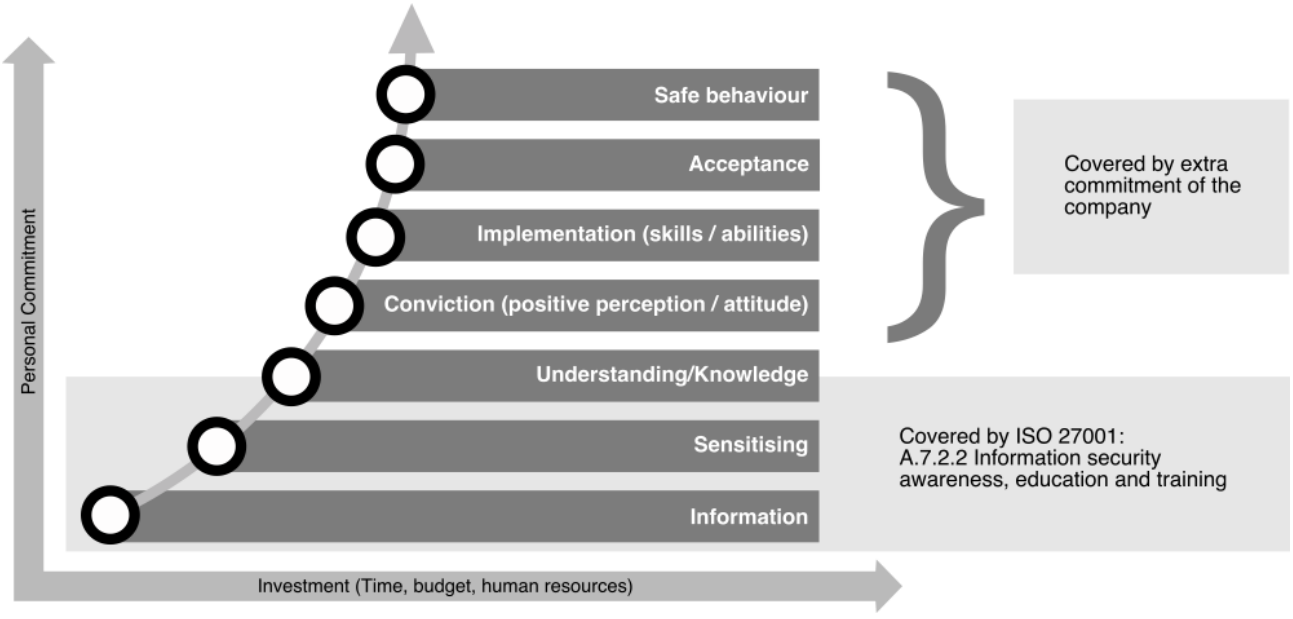


Figure 4.3: Behaviour change model from RISCS White Paper [392]

4.4.1 New approaches to support security awareness and behaviour change

Simulations and games are increasingly being used, both to make security awareness more attractive, and to help with more complex educational measures and behavioural change.

Anti-phishing simulations designed to teach employees not to click on suspicious links are probably the most widely used in organisations today. Their popularity stems from the fact that they provide the ability to measure the impact of interventions, and they tend to show a decrease in click rates in the short term. The argument is that the experience of having been phished is a 'teachable moment' that captures the employees' attention and persuades them to work their way through the education being offered. However, Fogg, who first introduced the concept of 'trigger moments' (referred to as Prompts in the most recent Fogg Behaviour Model, cf. Figure 4.4) is very clear that they will only lead to behaviour change if the person has a sufficient level of motivation to engage with the training provided, and the ability to apply the skills being taught. Joinson argues that certain emotional and contextual triggers employed by social engineering attackers are so targeted and powerful (for instance, a notification purporting to have information about traffic or public transport disruptions shortly before the end of the working day) that they cannot be prevented by training [397].

From a human factor perspective, anti-phishing simulations can be problematic: 1) because employees may perceive this as being attacked by their own organisation, which reduces trust [387] and 2) they may lead employees to become so reluctant to click on links that they do not act on genuine emails that may be important. These factors need to be carefully considered in the design of any such simulations [358]. Furthermore, as we discussed above, the use of mechanisms such as DMARC can reduce the number of suspicious emails on which users need to focus, enabling education and training to be geared towards explaining social engineering and manipulation techniques.

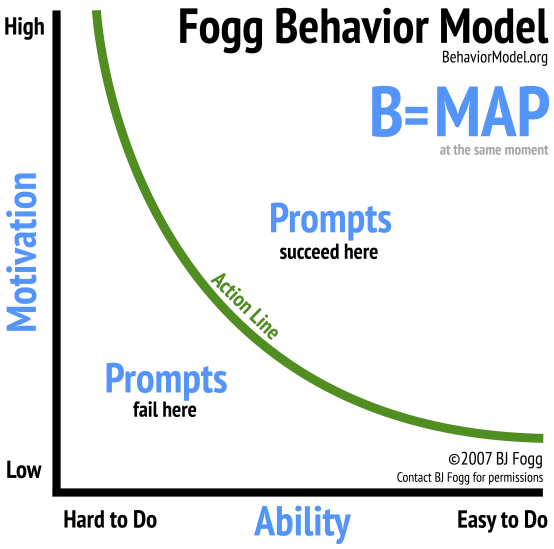


Figure 4.4: Fogg Behaviour Model has three factors: motivation, ability and triggers (**https://behaviormodel.org**)

Security awareness games Capture The Flag (CTF) games are designed to raise awareness of vulnerabilities, and how they can be exploited. The idea is that by seeing how they can use the vulnerabilities to attack a system, defenders learn to not incorporate them in their own systems. However, the focus is on training those charged with securing the organisation and not the wider set of users and employees.
There are tabletop card games aimed at providing security awareness to a wider user base within organisations, e.g., Ctrl-Alt-Hack [398], dox3d![a] and others are specifically targeted towards ICT specialists and developers, e.g., Microsoft's Elevation of Privilege[b]. There are also board games designed to be played by work groups to raise awareness of cyber security threats and the complexity of cyber risk decision-making, e.g., Decisions and Disruptions [395]. All of these games have the potential advantage of offering a social learning experience if played in a group context. But, if they are provided as one-off exercises, they are unlikely to have a lasting effect.
Overall, games and simulations have the potential to offer engaging new elements that can be deployed at different stages of the behaviour change model (see Figure 4.3) but they need to be part of a planned behaviour transformation programme, not one-shot interventions.

[a] https://d0x3d.com/d0x3d/about.html
[b] https://www.microsoft.com/en-us/SDL/adopt/eop.aspx

4.4.2 Mental models of cyber risks and defences

Much of the knowledge in the long-term workspace is organised in the form of mental models, mental analogues of devices with which people interact. They can range in detail from structural models (like blueprints) that experts have, to task-action models that enable non-experts to operate a device competently. A person with a task-action model of operation can drive a car, but only an expert with a structural model can diagnose faults and repair them. Clearly, we cannot expect non-security experts to understand all cyber risks in detail.

Wash argues that inadequate mental models of security make users vulnerable against

intelligent adversaries:

> 'These users believe that their current behavior doesn't really make them vulnerable, so they don't need to go to any extra effort.' [393]

Understanding users' mental models can provide insights into how users perceive particular security information, e.g. alerts [399] or specific tasks they have to undertake, e.g. deletion [381]. The question is: which models would be helpful? There are example mental models in the literature, for instance, physical security models, medical models, criminal models, warfare models and market models [400], which may provide a basis to communicate complex security issues to users. Perceptions of risk are also relevant in this regard. These, along with responsibility, are covered in the Risk Management & Governance Knowledge Area (Chapter 2) so are not discussed further here.

4.5 POSITIVE SECURITY

[401]

What is the goal of cyber security? When asked, most people's first response is along the lines of preventing cyber attacks or at least reducing the risk of attacks succeeding, or losses being too high. As Florencio et al. pointed out, vendors and those who want organisations to take security more seriously resort to a 'Fear Uncertainty and Doubt (FUD) sale' – creating fears of attacks and their consequences, Uncertainty about consequences and Doubt about organisations' ability to defend themselves – thus boosting the cyber security market and the sale of products [401].

> 'FUD provides a steady stream of factoids (e.g., raw number of malware samples, activity on underground markets, or the number of users who will hand over their password for a bar of chocolate) the effect of which is to persuade us that things are bad and constantly getting worse.'

Security practitioners today complain that most individuals and businesses do not take cyber risks seriously. The problem is that fear sales are not a good basis for security decision-making: when the resulting investment in security turns out not to be effective, decision-makers become skeptical about the benefits of cyber security. This, in turn, encourages the other side to ramp up the FUD, leading to a spiral of fear and grudging investment in security.

In order to defend from novel threats, companies need more than passive adherence – employees wanting to defend the organisation, and understanding and agreeing with the responsibilities they have been assigned in the defence. To achieve that, we must make security a proposition that is credible, so that people want to buy into it. Positive security offers more than protecting things we care about from negative consequences ('*freedom from*'). It enables us to engage in activities we value, and have experiences we cherish ('*freedom to*') [402, 403]. Roe argues that a positive conception of security will open ideas for new policy options and interventions, and encourage individuals or groups to become more involved in decision-making about security, and being part of delivering it [403].

Another key aspect of positive security is the language we use in connection with it. As a first step, we must stop the practice of demonising people who are unwilling or unable to follow security advice: calling these people 'The Weakest Link' implicitly blames them for not being able to make sense of, or comply with, security.

4.6 STAKEHOLDER ENGAGEMENT

[376, 404, 405]

4.6.1 Employees

From the research on human behaviour in cyber security over the past decade, one very clear theme has emerged: the importance of engaging in finding ways of making security work for employees. Communication and leadership are important in this regard. However, these aspects and others pertaining to organisational cultures are discussed in the Risk Management & Governance Knowledge Area (Chapter 2). Here, we focus on employees rather than organisational leadership and aspects, such as strategic board-level leadership of cyber security.

Lizzie Coles-Kemp and colleagues have developed an approach that takes employee involvement in improving security a step further. They use projective techniques (e.g., drawings and collages) to build representations of daily activity, and ground the discussion of security in these. Case studies [354, 379] show how this helps to identify the root causes of insecure behaviour that the organisation sees as undesirable, in many cases badly designed security (echoing the results of Beautement et al. [376]), but also more fundamental failings *of* the organisation to support the business and its individual tasks.

Creative security engagements (first mentioned by Dunphy et al. [406]) encourage participants (employees in the company context or consumers or citizens in wider engagement) to reflect on:

- their environment,
- the emotions they feel,
- the constraints they experience,
- the pressures they are under,
- the actions and tasks they perform when generating and sharing information.

One particular technique for creative engagements using Lego for the physical modelling of information security threats was developed by the EU Trespass Project[6]. This type of physical modelling bridges the gap between the typical diagrams (flow-charts and Unified Modelling Language (UML) diagrams, for example) with which security practitioners commonly work, and the everyday practices of the consumers who are affected by security design. Heath, Hall & Coles-Kemp [407] reported a successful case study of this method to model security for a home banking application, which identified areas where human intervention and support needed to be provided to make security work overall.

These studies provide examples of different ways of engaging with employees, consumers and citizens on security. They are part of a growing trend in research (cf. work on Productive Security [408]), moving away from the mechanistic approach of looking for traits within individuals that are conducive to the desired security behaviour, or trying to change behaviour by addressing or tweaking those traits. The fundamental focus of these approaches is about changing the design of security to align with user and organisational tasks to reduce workload

[6] **https://www.trespass-project.eu/**

and increase productivity for an organisation. The fact that it also leads to a more positive perception of security is a valuable side-effect.

4.6.2 Software developers and usable security

Zurko & Simon pointed out that unusable security affects not only general employees who may not have specific computing or security education but also those who have significant technical skills, such as developers and system administrators [409]. They also face increasing workloads and complexity, and make mistakes because the libraries and application programming interfaces (APIs) they draw on are not usable. Arguably, errors that these *technical* users make generally have a more significant impact than mistakes made by general employees, e.g., the Heartbleed vulnerability.

Developers and password security We noted above the usability issues of password and other authentication systems that have been studied extensively for end-users, highlighting problems and informing design decisions for better policies and motivating research into alternatives. However, end-users are not the only ones who have usability problems with passwords. The developers who are tasked with writing the code through which the passwords are stored must do so securely. Yet, history has shown that this complex task often fails due to human error with catastrophic results. If developers forget to 'hash and salt' a password database, this can lead to millions of end-user passwords being compromised. Naiakshina et al. [352, 353] conducted a randomised control trial with computer science students, as well as freelance developers, and found that, similar to end-users, developers also suffer from task-focus and they see security as a secondary task. None of the student participants, and only a small number of freelance developers, implemented any kind of security unless explicitly prompted to do so. Interestingly, of those participants who did implement some security measures, the students did better than the freelance developers, who on the whole used more outdated and incorrect cryptographic mechanisms to store their passwords.

A number of studies, e.g., Enck et al. [410] and Fahl et al. [404] have highlighted the extent to which vulnerabilities manifest in modern eco-systems centred on app development. It was notable that, of the 96 developers who were contacted by Fahl et al., a large number were willing to provide information, but only 13 were interviewed because their companies refused permission for them to do so. From the interviews, Fahl et al. found that developers had little to no security training and were under extreme pressure to complete the app quickly—and that was the reason for the mistakes that led to vulnerabilities.

Acar et al. [411] have studied the impact of online social networks, such as StackOverflow, on the security of code that developers produce. Two thirds of the developers who used StackOverflow or a textbook managed to produce a functionally correct solution within the allocated time, whereas only 40% of those using official documentation did. In terms of the security tasks, the results were reversed. Those using official documentation produced the most secure code and those using the StackOverflow the least. A traditional security response to this result would be 'use of StackOverflow should be forbidden.' But clearly, the productivity price developers and their organisations would pay would be a hefty one. For instance, recent work [412] has shown that developers utilise such forums to exchange information and offer mutual support for security problem-solving. That is not to say that such advice is always effective (as noted above) but the forums do provide a community of practice in which developers can share their problems and seek help. Banning such forums outright without replacing them with relevant support would, therefore, not address the crux of why developers

seek such support.

The usability challenges of cryptographic APIs and their documentation have been highlighted by Arzt et al. [413] and Nadi et al. [414], and tools proposed to support developers in their usage [415]. Recently, tools have also been proposed to improve the usability of static analysis, e.g. [416]. Green and Smith have synthesised insights from the existing body of research into a set of ten principles to make application programming interfaces for security and cryptography libraries more usable for developers [405]. Patnaik et al. [417] identify four *usability smells* that indicate that cryptographic APIs may not be fully addressing such principles, offering insights to library developers on the key areas on which to focus in order to improve the usability of their libraries.

The disconnect between developers and users also needs to be considered. Caputo et al. [418] highlighted that developers did not understand the impact of the lack of usability on individual performance and wellbeing, organisational productivity, or the effectiveness of security. They recommend that management must ensure that developers experience the results of the lack of security and usability directly – by having to deal with help desk calls, the impact of losses – and engage more. Recent work has provided insights into the role of strong organisational security cultures on developers' mindsets with regards to security [419] and how experts improve their security practices [420].

4.7 CONCLUSION

Humans and technologies do not exist in isolation. Humans conceive new technologies, design and implement them, and are also their users and maintainers. Cyber security is no different. Human behaviours shape cyber security (e.g., responses to phishing campaigns lead to anti-phishing filters or new security training). Equally, the design of cyber security (humans design those filters or training mechanisms) impacts people's interactions with systems and the security mechanisms designed into those systems (e.g., impedence to primary tasks or increased workload arising from security tasks). We must consider this symbiotic relationship throughout the conception, design, implementation, maintenance, evolution – and let's not forget, decommissioning – of cyber security mechanisms. Human factors must play a central role as, after all, the purpose of cyber security is to protect people, their data, information and safety. We must – as far as possible – fit the task to the human and not the human to the task.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[359]	[360]	[11]	[387]	[392]	[393]	[401]	[376]	[404]	[405]
4.2 Usable security – the basics	c4.2	c4.2								
4.2.1 Fitting the task to the human										
4.3 Human Error			c4.3	c4.3						
4.4 Cyber security awareness and education					c4.4	c4.4				
4.5 Positive Security							c4.5			
4.6 Stakeholder Engagement								c4.6	c4.6	c4.6

Chapter 5
Privacy & Online Rights

Carmela Troncoso | École Polytechnique Fédérale de Lausanne

INTRODUCTION

The pervasiveness of data collection, processing, and dissemination raises severe privacy concerns regarding individual and societal harms. Information leaks may cause physical or psychological damage to individuals, e.g., when published information can be used by thieves to infer when users are not home, by enemies to find out weak points to launch attacks on users or by advertising companies to build profiles and influence users. On a large scale, the use of this information can be used to influence society as a whole, causing irreversible harm to democracy. The extent of the harms that privacy loss causes highlights that privacy cannot simply be tackled as a confidentiality issue. Beyond keeping information private, it is important to ensure that the systems we build support freedom of speech and individuals' autonomy of decision and self-determination.

The goal of this knowledge area is to introduce system designers to the concepts and technologies that are used to engineer systems that inherently protect users' privacy. We aim to provide designers with the ability to identify privacy problems, to describe them from a technical perspective, and to select adequate technologies to eliminate, or at least, mitigate these problems.

Privacy is recognised as a fundamental human right [421]: "No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation". As such, it has been studied for many years from a socio-legal perspective with two goals. First, to better understand what privacy means for society and individuals. Second, to ensure that the legal frameworks that underpin our democracies support privacy as a right. The former studies proposed definitions such as privacy being 'the right to be let alone' [422], 'the right to informational self-determination' [423, 424] or 'the freedom from unreasonable constraints on the construction of one's own identity' [425]. Probably one of the best examples of the latter are the principles and rules associated with the European Data Protection Legislation [426] covered in the Law & Regulation Knowledge Area (Chapter 3). All of these conceptualisations are of great importance to define and understand the boundaries of privacy and its role for society. However, their abstract and context-free nature often makes them not actionable for system designers who need to select technologies to ensure that privacy is supported in their systems.

To address this gap, in this knowledge area, we conceptualise privacy in a similar way as security engineering conceptualises security problems [427, 428]. We consider that privacy concerns, and the solutions that can address them, are defined by the adversarial model considered by the designer, the nature of the information to be protected, and the nature of the protection mechanism itself. Typical examples of adversarial models can be: third-party services with whom data are shared are not trusted, the service provider itself is not trusted with private data of the users, or users of a service should not learn private data from other users. Typical examples of private data to be protected from these adversaries can be: the content of users' communications, their service usage patterns, or the mere existence of users and/or their actions. Finally, typical examples of protection means can be techniques that enable information availability to be controlled, such as access control settings, or techniques to hide information, such as Encryption.

This knowledge area is structured as follows. The first part, comprising three sections, considers three different privacy paradigms that have given rise to different classes of privacy technologies. The first is privacy as confidentiality (Section 5.1), in which the privacy goal is to *hide* information from the adversary. We revise technological approaches to hide both data

and Metadata, and approaches to hinder the adversary's ability to perform inferences using the data that cannot be hidden. The second is privacy as informational control (Section 5.2), in which the goal is to provide users with the means to *decide* what information they will expose to the adversary. We revise technologies that support users in their privacy-oriented decisions and techniques that help them express their preferences when interacting with digital services. Finally, we introduce privacy as transparency (Section 5.3), in which the goal is to *inform* the user about what data she has exposed and who has accessed or processed these data. We revise solutions that show users their digital footprint, and solutions that support accountability through secure logging.

The privacy requirements that define the privacy goals in the paradigms mentioned above are often context dependent. That is, revealing a particular piece of information may be acceptable in some environments but not in others. For instance, disclosing a rare disease is not considered a privacy concern in an interaction with a doctor but would be considered a privacy violation in a commercial interaction. Nissembaum formalizes this concept as *contextual integrity* [429], which explicitly addresses an information flow may present different privacy needs depending on the entities exchanging this information or the environment in which it is exchanged. We note that once the requirement for a flow are clear (including the adversarial model), a designer can directly apply the technologies described in this chapter.

The second part of the knowledge area is devoted to illustrating how privacy technologies can be used to support democracy and civil liberties (Section 5.4). We consider two core examples: systems for secure voting and to circumvent censorship. For the former, privacy of the votes is imperative for the functionality itself. For the latter, privacy of communication partners is necessary to ensure that content cannot be blocked by a censor.

We acknowledge that privacy technologies can be used in to support illicit (e.g., distribution of child pornography) or anti-social behaviors (e.g., cyberbullying), as described in the Adversarial Behaviours Knowledge Area (Chapter 7). While there exist solutions to selectively revoke the protection provided by privacy technologies, these are strongly discouraged by privacy researchers and privacy advocates. The reason is that adding backdoors or escrow possibilities to ease law enforcement, inherently weakens the security of the privacy-preserving systems as they can also be exploited by malicious actors to undermine user's rights. Therefore, we do not consider these techniques within this document.

We conclude the knowledge area by outlining the steps involved in the engineering of privacy-preserving systems (5.5). We provide guidelines for engineers to make informed choices about architectural and privacy technologies. These guidelines can help system designers to build systems in which the users' privacy does not depend on a centralised entity that may become a single point of failure.

We note that many of the privacy technologies we revise in this knowledge area rely on the cryptographic concepts introduced in the Cryptography Knowledge Area (Chapter 10). Throughout this knowledge area, we assume that the reader is familiar with these basic concepts and avoid repeating cryptographic definitions and reiterating on the explanation of common primitives.

CONTENT

5.1 PRIVACY AS CONFIDENTIALITY

[430][431][432][433][434][435][436][437]

In a technical re-interpretation of the 'right to be let alone' [422] privacy definition, a common conceptualisation of privacy is to avoid making personal information accessible to any entity, in particular to a wider public [427]. Under this definition, the objective of privacy technologies is to enable the use of services while minimising the amount of exposed information. Here, information refers to both data exchanged explicitly with the service, as well as information made implicitly available in the Metadata associated with these exchanges (e.g., identity of the users or frequency of usage).

5.1.1 Data Confidentiality

We now describe two approaches to minimise the amount of exposed information. We first present methods that provably prevent unauthorised access to information, typically based on the use of advanced cryptographic primitives to ensure that no data can be inferred. Second, we present disclosure control methods, which relax the Confidentiality definition to ensure that the information leaked to the adversary is limited to a certain amount, or is not linkable to an individual person.

5.1.1.1 Cryptography-based access control

A first flavour of Confidentiality-oriented privacy technologies focus on protecting the *data* through the use of cryptography. These technologies mainly consider two adversary models: one where the recipient is considered trusted and the data have to be protected while in transit, and one in which the recipient is not trusted and the data must be kept private even when it is processed.

Protecting data in transit. The protection of data in transit is typically known as *end-to-end encryption (E2EE)*. Here, an end refers to the origin and destination of the communication. For instance, the sender and receiver of an email, or the client and server of a service. E2EE ensures that the Confidentiality of data is ensured between both ends. That is, no third party, from the routers in the communication infrastructure, to the application (e.g., email, messaging) servers that enable the communication, can access the communication. Additionally, E2EE typically provides Integrity, impeding any intermediary from modifying the data exchanged, and Authentication, ensuring that the communication parties can be sure of each others' identity.

From a technical perspective, in E2EE the devices at the end of the communication hold the Encryption key used to protect the data. Usually, these are symmetric encryption keys and can be agreed using key transport, or can be established using any modality of the Diffie-Hellman exchange. The use of Diffie-Hellman to agree one key per session additionally provides forward secrecy, but one must be careful when implementing the exchange [438]. Typically, Digital Signatures and Message Authentication Codes are used to provide Integrity and authentication. Canonical examples of E2EE encryption are the TLS protocol [439], widely

used in client-server scenarios; or the PGP protocol, a common encryption mechanism for email communications [440].

A special type of E2EE is Off-the-Record Messaging (OTR) [431].[1] OTR seeks to provide stronger privacy properties than the above protocols. It considers an adversary that can not only observe the communication, but also eventually compromise one of the devices participating in the communication. This compromise gives the adversary the chance to get the long-term keys of the participants. In such a demanding scenario the two main goals of OTR are to provide i) perfect forward secrecy and ii) repudiable Authentication, which permits a user to deny having sent a message in the past. The protocol derives the cryptographic keys used for the conversation using an *unauthenticated* Diffie-Hellman key exchange. Then, the participants carry out a mutual authentication inside the protected channel, which guarantees the future repudiability. Encryption keys are rotated, and old keys are deleted, so as to maintain forward secrecy. Current OTR protocols also include strong protection against man-in-the-middle attacks, even if the participants do not pre-share secrets [441, 442].

Finally, we can remark that E2EE is nowadays prevalent in instant messaging applications such as Signal, WhatsApp, Facebook Messenger, or Viber. All of these applications are based on the so-called Signal Protocol (previously known as Axolotl or TextSecure) [443]. Similar to OTR, this protocol provides authenticated messaging between users with end-to-end Confidentiality, and messages are kept secret even if the messaging server is compromised, and even if the user's long-term keys are compromised. These properties rely on an authenticated key exchange protocol that mixes multiple Diffie-Hellman shared secrets, and on a protocol to refresh the keys called double ratcheting [435]. Cohn-Gordon et al. provided in [444] a detailed description of this protocol, including a formal analysis.

Note that all of the above protocols only offer strong guarantees as long as the mechanisms to authenticate the communication parties work as expected. For instance, the Confidentiality provided by TLS relies on services keeping their keys secret and the Public Key Infrastructure operating reliably, so that the communication parties can be authenticated. Similarly, WhatsApp's Confidentiality relies on the fact that phone numbers are hard to spoof and, thus, users are sure that the recipient of their message is their intended interlocutor.

Protection of data during processing. The previous protocols focus on protecting data in transit from third parties other than the communication participants. We now consider situations in which the recipient needs to perform some computation on the data, even though she is considered adversarial. We distinguish two scenarios: one in which computation is completely outsourced and one in which the sender participates in the computation.

In the first scenario, commonly known as *outsourcing*, the data belong to the sender and the recipient acts as the data processor. Typical examples are the use of cloud services to compute on big data, e.g., privacy-preserving training and classification using machine learning [445], or to hold a database in which the sender wants to perform searches [446]. The solutions to this problem are based on advanced cryptographic protocols. We now illustrate the use of these protocols in a couple of examples, and we refer the reader to the Cryptography Knowledge Area (Chapter 10) for more details on the technical details of the underlying primitives.

A common problem when outsourcing services is that accessing particular pieces of outsourced data may reveal information about the user to the entity holding the data. For instance, accessing a given entry on a patent database reveals business intentions; and accessing a

[1] **https://otr.cypherpunks.ca/**

particular entry in a messaging directory reveals relationships between users. This problem can be mitigated by using Private Information Retrieval (see the Cryptography Knowledge Area (Section 10.9.2)), which allows a database to be queried without revealing which record is being accessed. An example use case for information retrieval is the creation of privacy-preserving directories for social networks [447, 448, 449].

Another example where remote processing is needed comprises digital shops or digital banking, where a server returns information to a user depending on the inputs. The shop needs to process payments and then ship the digital item; and the bank provides money, or makes a payment, upon authentication. Users' shopping patterns, however, may reveal a lot about their profiles. In this case, Oblivious Transfer (see the Cryptography Knowledge Area (Section 10.9.1)), in which a service can transfer an item without knowing which item is being transferred, can be used to support privacy-preserving interactions [450, 451].

The previous techniques are useful for particular operations: search an item on a database, transfer that item. Ideally, we would like to be able to perform any operation on outsourced data. A very relevant technology for this is Homomorphic encryption (see the Cryptography Knowledge Area (Section 10.11)). This type of encryption allows any operation on encrypted data to be performed. Such flexibility, however, comes at a high cost in terms of computation time, and for some implementations also in terms of bandwidth, thus making it far from practical at this point. Less general versions such as somewhat homomorphic encryption or partially homomorphic encryption, which only permit limited operations (sums, multiplications or evaluating a given function) provide better trade-offs and can already be used for simple concrete tasks.

We note that in recent years, the privacy-preserving cryptographic primitives above have been combined with new secure hardware [452, 453] in order to improve performance. While this combination indeed brings the performance of privacy-preserving cryptography closer to the benchmarks needed for deployment, it is important to highlight that such an improvement comes at the expense of trusting the manufacturer of the secure hardware not to leak the information (or the key) to unintended parties.

In the case of database outsourcing, it is worth mentioning tailored solutions that combine different types of privacy-preserving cryptography in order to increase efficiency [454]. These databases rely on techniques such as homomorphic encryption, order-preserving encryption [455, 456], or deterministic encryption, among others. These schemes indeed provide great performance. However, it has been demonstrated that choosing weaker cryptographic primitives to favour efficiency may have a significant impact on privacy [457, 458, 459, 460]. Therefore, they are only recommended to support compliance, and should only be deployed in a trusted environment where attacks are unlikely. It is not recommended to use them in scenarios where data privacy is of critical importance and the entity that holds the database is not trusted.

The second scenario is collaborative computation, i.e., the entities involved in the communication collaborate to perform the computation. The result of this computation may be of interest for the sender, for the receiver, for both, or for third parties. Yet, if the participants do not trust each other, i.e., for a given entity, any of the other participants may be considered an adversary. Typical applications are comparing databases or computing statistics across datasets [461, 462]. Such applications can be supported by Multi Party Computation (see the Cryptography Knowledge Area (Section 10.9.4)), as described by Archer et al. in [463]. When the goal of the application is to find similarities between two databases (e.g., contacts [464, 465], malicious activities [466], or genetic information [467]), one can also use lighter protocols such

as Private Set Intersection [468, 469, 470]. These protocols allow two parties to compute the intersection of datasets without revealing anything except the intersection, or the cardinality of the intersection.

Verification in the encrypted domain. When data are processed in the encrypted domain, it is hard for the entities performing the computation to run any check on the adequacy of the inputs. To solve this problem, many primitives build on Zero-Knowledge Proofs (see the Cryptography Knowledge Area (Section 10.9.1)) to prove to the entity performing the computation that the inputs comply with a certain format or with certain constraints. We now describe three cases in which verification in the encrypted domain is key to enabling the use of privacy-preserving cryptographic protocols.

Private computation - input verification. Zero knowledge proofs are very well suited to ensuring that the input to a privacy-preserving protocol is of a particular form or is not malicious. For instance, they have been used, among others, to prove the adequacy of inputs in billing applications, e.g., that they belong to a set of valid inputs [471], or are within particular ranges [472], to prove that there are no malicious inputs when requesting information from a messaging system [473], or when running a private intersection protocol [474].

Private authentication. To maintain Confidentiality, entities participating in protocols may want to authenticate their communication partners. However, typical Authentication systems are based on revealing the identity of the authenticating party. Revealing one's identity, in and of itself, may result in a privacy breach (e.g., when the authentication is against a sensitive service, such as a medical service). A solution to avoid this problem is the use of Anonymous Credentials, also known as Attribute-Based Credentials (ABCs) [432, 475, 476].

Instead of authenticating an entity with respect to an identity in order to grant authorisation, ABCs enable the entity to prove possession of a combination of different attributes to obtain the same authorisation. This proof does *not* reveal any additional information about the entity authenticating, nor does it reveal the concrete values of the attributes. Furthermore, ABCs are unlinkable between contexts. In other words, credentials look different every time they are shown, such that different showings cannot be linked to each other.

While from the point of view of privacy, ABCs bring many advantages, they also introduce new challenges. Anonymity may open the door to misbehaviour. Unfortunately, the strong Anonymity and Unlinkability properties provided by original ABCs do not allow an authority to limit or revoke authorisation for misbehaving users. In response, several schemes have appeared that provide capabilities to limit the amount of times that a credential can be used before the user is identifiable [477]; capabilities to blacklist credentials so that access can be temporarily revoked [478, 479]; or capabilities to completely revoke the credentials [480].

There exist several implementations of ABCs [481, 482] available under diverse licenses. These implementations offer different subsets of the functionalities mentioned above.

Private payments. Verification of encrypted data is also key to enabling privacy-preserving payments, in which the payer may have to prove to the buyer, for instance, that he has enough funds without revealing the exact amount. Early digital cash systems relied on Blind Signatures (see the Cryptography Knowledge Area (Section 10.10)) to enable banks to sign e-coins [433]. In a nutshell, to extend an e-coin to a client, a bank would blindly sign a random value. To spend the e-coin, the client would give this number to the seller, who could redeem it at the bank. By storing the random number, banks can detect double spending, but not identify the double spender.

More recent privacy-preserving payment schemes, like the blockchain-based Zerocash [483, 484] system, include more information in the transactions to provide better guarantees. In each transaction, the user proves, in zero knowledge, that she owns the e-coins input to the transaction; that each one of the input e-coins was either recently mined (minted in Zerocash terms) or was the output of a previous transaction; and that the input and output values of the transaction are the same, i.e., no money would be lost. For the sake of efficiency, Zerocash relies on particularly efficient zero-knowledge proofs called zero-knowledge Succinct Non-interactive ARguments of Knowledge (ZK-SNARK) systems [485]. These proofs are shorter (in the order of hundreds of bytes) and relatively fast to verify.

5.1.1.2 Obfuscation-based inference control

The protocols discussed in the previous section provide strong (cryptographic) guarantees regarding the Confidentiality of data. Such strong protection, however, comes at the cost of efficiency and flexibility. On one hand, privacy-preserving cryptographic primitives require significant resources in terms of computation and/or bandwidth. On the other hand, they narrow down the type of processing that can be done on data. This is inherent to cryptographic constructions that fix inputs and outputs, and strictly define what information will be available after the protocol is executed.

In this section, we describe approaches to protect data Confidentiality based on obfuscating the data exposed to an adversary. These techniques provide a more relaxed definition of Confidentiality than cryptography, in the sense that they cannot completely conceal information. Instead, their goal is to provide a way to *control* the extent to which an adversary can make inferences about users' sensitive information. In fact, for most of these techniques, the level of protection depends on the concrete data and adversarial knowledge. Thus, it is important to run an ad-hoc analysis for the inference capability, as explained in Section 5.5. Also, we note that the privacy gained from these techniques is based on limiting the information available to one's adversary. Consequently, these techniques reduce the amount of information available for anyone and, hence, may have an impact on utility if the purpose of the application is based on sensitive information, e.g., finding matches on dating applications. However, we note that when the sensitive information is not crucial for the purpose of the application these techniques may be deployed without affecting utility, e.g., a weather application that can operate using very rudimentary location data.

Obfuscation-based inference control techniques are not suitable for protecting data in transit, but can be used to support privacy-preserving outsourcing, privacy-preserving collaborative computations, and privacy-preserving publishing. There are four main techniques to obfuscate data, as described below. We note that these techniques are mostly oriented to obfuscate numerical or categorical fields. Obfuscating more complex content, such as free text, is a much more difficult task due to correlations that are hard to remove in a systematic manner. To date, there are no known techniques that can reliably anonymise free text. However, these techniques are quite effective at reducing the information leaked by Metadata, as we discuss in Section 5.1.2.

For the sake of illustration, let us take the following microdata file as a current example. This is a very simple example, and we stress that the techniques introduced below can be applied to many types of data formats and domains.

Name	Age	Gender	ZIP	Salary
Alice	21	Female	21345	51300
Bob	32	Male	25669	67400
Carla	25	Female	18934	51500
Diana	64	Female	21223	60200
Eve	34	Female	18022	73400
Frank	37	Male	25321	55800
Gerald	19	Female	18235	68900

Table 5.1: An example database

Anonymisation. A common technique used to permit data processing without risk for individuals is **data anonymisation**. Anonymisation, as its name indicates, seeks to decouple identity from information. The idea is that removing identifying information from data points makes them unlinkable (i.e., they cannot be grouped as belonging to the same entity), thus hindering the ability of the adversary to perform inferences from the data.

However, achieving full Anonymity is extremely difficult. In fact, when a dataset can be declared anonymous remains unclear. Data in and on themselves contains enough information to correlate different attributes and/or records on a database. Given these groups, there are many techniques to re-identify individuals behind the data release. A key insight into understanding the difficulty of anonymisation is the uniqueness of individual's data patterns [486, 487]. There may be many combinations of the information released in a dataset that are unique to an individual. These are called quasi-identifiers. Finding quasi-identifiers enables the re-identified data by mapping them to identifying information in other data sources [436, 488]. Thus, anonymisation is commonly combined with the obfuscation techniques described below to limit the risk of re-identification.

At this point in the knowledge area, it is worth referring to the notion of k-anonymity, which advocates combining generalisation and suppression in order to ensure that records on a database are anonymous among (i.e., indistinguishable from) at least other k entries in the same dataset [489]. For instance, in the example above, one can generalise the ZIP code to achieve two-anonymity:

Name	Age	Gender	ZIP	Salary
*	21	Female	21*	51300
*	32	Male	25*	67400
*	25	Female	18*	51500
*	64	Female	21*	60200
*	34	Female	18*	73400
*	37	Male	25*	55800
*	19	Female	18*	68900

Table 5.2: **Anonymization**: A two-anonymous database through generalisation

While this notion is promising, there are several factors that make it unappealing and hard to use in practice. First, due to the uniqueness of the problem mentioned above, obtaining k-anonymity may require an unacceptable amount of generalisation in the database. Second, depending on the application, k-anonymity may not actually prevent inference of sensitive attributes. This is illustrated in our running example in the Gender column. Even though the

generalisation in the ZIP code ensures two-anonymity, the adversary knows with a 100% probability the gender of the users in each ZIP area, e.g., all users living in 21* are women. Similarly, the adversary learns that females in their 20s earn approximately 51000.

To address this issue, researchers argue that privacy not only requires k-anonymity, but also l-diversity, which ensures that for each k anonymous individual, there are at least l possible values for the sensitive attribute [490]. Researchers have also shown that l-diversity can be broken and so-called t-closeness, where the set of sensitive attributes is not only diverse but follows the general distribution for this attribute across a population, is needed [491].

The k-anonymity notion is very popular in health-related applications [492]. It has also been adapted to fields other than databases [493, 494].

Generalisation. This technique consists in reducing the precision with which data are shared, with the goal of reducing the accuracy of the adversary's inferences. Generalisation can be achieved via a direct precision reduction of the shared values, or a bucketisation (i.e., a mapping from values to ranges) before data are released. This technique has been applied, among others, for database anonymisation [489], reducing the precision of the values in the different cells; or in private web searches [495], where words are mapped to the closest word of a pre-defined set.

Name	Age	Gender	ZIP	Salary
Alice	10–30	Female	21***	51300
Bob	30–40	Male	25***	67400
Carla	20–30	Female	18***	51500
Diana	60–70	Female	21***	60200
Eve	30–40	Female	18***	73400
Frank	30–40	Male	25***	55800
Gerald	10–20	Female	18***	68900

Table 5.3: **Generalisation**: Reducing the precision of the ZIP code to the first two digits; reducing the precision of the Age column via bucketisation.

Suppression. This technique consists in suppressing part of the information before it is made available to the adversary. The rationale behind suppression is that the fewer the data are provided to the adversary, the more difficult is for her to make inferences. The suppression strategy, which decides which information to hide, is key for the level of privacy protection that such a scheme may provide. For instance, suppressing information at random is unlikely to destroy the patterns in the data that allow for inferences. Thus, unless most of the data are deleted, this strategy seldom provides good protection. A common strategy is small count suppression, where aggregated values below a threshold are not reported. The level of protection of this strategy depends on the type of access to the data and the knowledge of the adversary [496]. Other suppression strategies, tailored to the nature of the data under consideration and their characteristics [497, 498] provide better privacy results. This technique has been applied, among others, for database anonymisation [489], to hide some of the values in the different cells; or in location data publishing [497], to hide location samples that provide too much information.

Name	Age	Gender	ZIP	Salary
Alice	21	Female	21345	51300
Bob	32	Male	25669	67400
Carla	25	*	18934	51500
Diana	64	*	21223	60200
Eve	34	Female	18022	73400
Frank	37	*	25321	55800
Gerald	19	Female	18235	68900

Table 5.4: **Suppression**: Suppression of the Gender attribute for 50% of the records.

Dummy addition. This technique consists in adding fake data points, so-called *dummies*, to the data made available to the adversary in order to hide which are the real samples. The idea is that, as the adversary considers fake points when running the attack, her inference will have errors. For this defense to be effective, fake points have to be indistinguishable from real points. Ideally, from the point of the view of the adversary, any sample should look like a real or dummy one with equal probability. However, creating such indistinguishable points tends to be difficult [499], and the adversary can easily filter them out. Thus, this technique is useful in very few domains. Dummy addition techniques have been used to increase privacy in web searches [495, 500] or to protect databases from inferences [501].

Name	Age	Gender	ZIP	Salary
Alice	21	Female	21345	51300
Bob	32	Male	25669	67400
Carla	25	Female	18934	51500
Donald	54	Male	25669	53500
Diana	64	Female	21223	60200
Eve	34	Female	18022	73400
Frank	37	Male	25321	55800
Goofy	61	Male	21346	41500
Gerald	19	Female	18235	68900
Minnie	23	Female	18456	62900

Table 5.5: **Dummy addition**: Adding 50% of fake records (in red).

Perturbation. Perturbation techniques inject noise into the data made available to the adversary. The noise is aimed at reducing the adversary's inference performance. Similar to suppression techniques, the strategy used to introduce noise plays a crucial role in the level of privacy provided. Initial schemes drew noise from many kinds of random distributions [502, 503] and added them to the data. This approach was not really effective, as an adversary with knowledge of the noise distribution could infer the original data values with reasonable accuracy and thus risked leaking more information than intended.

Name	Age	Gender	ZIP	Salary
Alice	21	Female	21345	51345
Bob	32	Male	25669	67863
Carla	25	Female	18934	51053
Diana	64	Female	21223	60302
Eve	34	Female	18022	74558
Frank	37	Male	25321	55005
Gerald	19	Female	18235	69425

Table 5.6: **Perturbation**: Obfuscating salary with noise drawn from a normal distribution N(0,1000).

Currently, the gold standard in perturbation-based techniques is to add noise to achieve so-called ***differential privacy***. The main goal of this technique is to address the limitations of data anonymisation techniques for publishing such as the aforementioned k-anonymity.

Differential privacy, introduced by Dwork [504], is a privacy definition originally intended to enable the design of techniques that permit maximising accuracy when querying statistical information (mean, variance, median etc.) about users on a database while minimising the risk of unintended inferences. Rather than a property of a dataset (like the techniques above), differential privacy is a property of a mechanism used to output the answers to queries against a dataset. An algorithm is differentially private if, by looking at the result of the query, the adversary cannot distinguish whether an individual's data were included in the analysis or not. More formally, an algorithm $\mathcal{A}$ provides ϵ-differential privacy if, for all datasets D_1 and D_2 that differ on a single element (i.e., the data of one individual), and all possible outputs S of the algorithm:

$$\Pr[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} \times \Pr[\mathcal{A}(D_2) \in S],.$$

Differential privacy ensures that, given a perturbed data sample, the adversary gains a negligible amount of new information about the original data sample with respect to the her prior knowledge, regardless of what this prior knowledge was. There exist a number of algorithms to ensure that differential privacy is met for a variety of queries [434].

Differential privacy is an extremely useful definition because it gives a formal framework to reason about the amount of information a powerful adversary might be able to infer about individuals in the data, regardless of the adversary's prior knowledge. However, it must be noted that:

- Differential privacy provides a *relative* guarantee, as opposed to an absolute privacy protection. This means that the protection provided is regarding the prior knowledge of the adversary. If the adversary already has full knowledge, differential privacy will not improve privacy. In other words, differential privacy ensures that the release of data does not worsen the privacy loss of a user or population by more than a set threshold. However, this does not automatically ensure that a user's privacy is preserved overall. Therefore, to claim privacy, it is important to not only ensure that a scheme provides a given guarantee, but also computes the adversarial error on the inferences so as to ensure that users' sensitive information is actually protected (see Section 5.5).
- One of the current practical challenges of differential privacy is to determine what values of ϵ provide an acceptable level of privacy. The level of protection of crucially

depends on the value of this parameter. This means that merely fulfilling the differential privacy definition with arbitrary parameter values does not directly guarantee that the adversary does not learn too much new information from the data. It is important to ensure that the value of ϵ is such that the probabilities for different inferences are actually indistinguishable. For example, if $\epsilon = 3$, this only ensures that the ratio between the probability of observing a result when an individual is, or is not, in the dataset is: $\Pr[\mathcal{A}(D_1) \in S] / \Pr[\mathcal{A}(D_2) \in S] \leq e^3 = 20.08$. This probabilistic difference can typically be detected by classical statistical detectors, or any modern machine-learning classifier. In general, ϵ values greater than one deserve a closer look to verify that the algorithms provide the sought level of protection.

- The amount of noise required to hinder inferences on the data depends on the so-called *sensitivity* of the algorithm. Sensitivity measures how much a change in the input will change the output of the algorithm $\mathcal{A}$. When the input is a database and the output a statistical function, small input changes have little influence on the output and, thus, a small amount of noise is enough to make the algorithm differentially private. We note, however, that when differentially private algorithms are applied to protect the privacy of a single sample instead of to the result of a statistical query on a database, the sensitivity may be much higher. Thus, only a large amount of noise may ensure protection. For instance, when differential privacy is applied to obfuscate a user's reported position to obtain location privacy, protection may be less than expected if the parameters are not carefully chosen [505].
- Differential privacy provides a worst-case guarantee, which means that the amount of noise introduced is tailored to bound the leakage given by the data point in the dataset that provides the most information to the adversary with the best knowledge. This means that in an average case the amount of noise is larger than needed. Recent studies have been working towards tighter bounds that permit reduction in the noise required to provide a desired protection level [506].

The differential privacy notion has been extended to account for metrics other than the Hamming distance (i.e., distinguishing whether one individual is in a database or not) [507]. Perturbations with differential privacy guarantees have been used to protect, among others, privacy in collaborative learning [508], or locations when querying location-based services [509]. It has also been recently adopted by the US to protect census data [510].

Finally, it is important to remark that for many real cases, one of these inference controls cannot provide enough privacy on its own. Therefore, typically one needs to combine several of these techniques to limit the numbers of inferences that can be made.

5.1.2 Metadata Confidentiality

In the previous section, we discussed means to protect the Confidentiality of the contents of messages, databases, queries, etc. These techniques are essential to ensure privacy. Yet, they do not protect against an adversary that uses the Metadata to infer sensitive information about individuals. Concretely, there are three types of metadata that have been demonstrated to be extremely vulnerable to privacy attacks: traffic metadata, associated to the communication infrastructure; device metadata, associated with the platform generating the data, and location metadata, associated with the physical location from which data is generated.

In this section, we discuss the privacy risks associated with these types of metadata and the

relevant controls to address these risks.

Traffic Metadata. Network-layer information, such as the identities of the participants in the communication (IP addresses), the amount and timing of the data transferred, or the duration of the connection, is accessible to observers even if communications are encrypted or obfuscated. This information, commonly known as traffic data, can be exploited to deduce potentially sensitive private information about the communication. For instance, in an e-health context, messages are generally encrypted to preserve patients' privacy. However, the mere fact that a patient is seen communicating with a specialised doctor can reveal highly sensitive information even when the messages themselves cannot be decrypted. Confidential communications are not only desirable for personal reasons, but they also play an important role in corporate environments. The browsing habits of a company's employees (e.g., accessing a given patent from a patent database) can be used to infer the company's future lines of investment, thus giving an advantage to their competitors. Finally, even if the identity of the communicating parties is innocuous, encrypted flows can reveal search keywords [511] or even conversations [512].

A technique to protect traffic data is the use of ***anonymous communications networks***. These networks are typically formed by a series of *relays* such that communications do not travel directly from origin to destination, but are sent from relay to relay. These relays also change the appearance of a message through means of Encryption to provide *bitwise Unlinkability*, i.e., to ensure that packets cannot be linked just by looking at their bit content; they can also change traffic patterns by introducing delays, re-packaging messages, or introducing dummy traffic.

Anonymous communications networks follow different designs regarding how the infrastructure is built (by users or dedicated relays), how they consider communications (message-based vs. flow-based), or how they reroute messages (deciding a route at the source, or letting relays decide on routing), among others. In the following, we focus on the two most known anonymous communications network types which have real-world deployment. We refer readers to the surveys by Danezis et al. [513] for a historical overview of anonymous communications and by Shirazi et al. [437] for a comprehensive overview of more recent anonymous communication systems.

The most popular anonymous communication network is Tor[2] [514]. The core element of the Tor Network are Onion Routers (ORs), which are essentially routers that forward encrypted data. ORs encrypt, respectively, decrypt packets along the way to achieve bitwise unlinkability, as detailed below. When a user who wants to anonymously access an Internet service through the Tor network, she installs a Tor client in her device. This software builds a *circuit* of connections over three ORs, called entry, middle and exit nodes, and the client routes encrypted traffic to the destination server through this circuit.

Tor uses so-called *onion encryption*, in which the client establishes a secret key with each of the ORs in the circuit using an adapted version of authenticated Diffie-Helmann (see the Cryptography Knowledge Area (Chapter 10)). Every packet routed through the circuit gets encrypted with these three keys, first with the exit OR's key, then the middle OR's key, and finally that of the entry OR. When the message travels through the circuit, the nodes 'peel' each layer of encryption until the original packet is sent to the destination. The server sends data to the client using the same circuit, but in the inverse order; i.e., the server encrypts the message in layers that are decrypted by exit, middle, and entry ORs. In order to support low-latency

[2] **https://www.torproject.org/**

applications, Onion Routers do not impose delays on the messages they receive and resend. Thus, traffic patterns are conserved while packets travel through the network. This enables an adversary with the capability to observe both ends of the communication (i.e., the entry and exit nodes) to correlate incoming and outgoing flows in order to link the origin and destination of communications [515].

At this point it is important to highlight the difference between using Tor and using a Virtual Private Network (VPN). Both technologies give the protection against an adversary that observes only one side of the communication, and both fail to protect against an adversary that can see both extremes. However, while in Tor no single relay can on itself learn the link between sender and receiver (i.e., the trust model is *decentralized*), in a P3P the provider actually knows this correspondence and thus is a single point of failure.

In order to destroy traffic patterns and protect correlation attack relays in an anonymous communication, networks need to delay packets or add new ones. This is the principle behind the design of *mix networks* [516]. As opposed to onion routing, where all the packets from a communication are routed through a circuit, in mix-based communications routes are selected for every message. Then, when a mix relay receives a packet, instead of immediately decrypting and send it to the next hop on the path, the message is delayed. How many messages are delayed is determined by a *firing condition*, which is an event such as the arrival of a message or the expiration of a timeout that causes the mix to forward some of the messages it has stored. Which messages are fired depends on the *batching strategy*, which can select all of the messages or a fraction according to a probabilistic function. Both mixes and users can send dummy traffic, which may be absorbed by other mixes or by the recipient.

A mix network designed to provide low latency is Loopix[3] [517]. As opposed to Tor, where users' clients communicate directly with the Tor nodes, Loopix assumes that users communicate with providers that in turn send messages to each other through the Loopix anonymous communication network. Providers choose a random route composed of Loopix routers and send the message to the first node. Similar to Tor, messages get encrypted with the keys of each of these routers using the Sphinx packet format [518]. In addition to the Encryption, messages are assigned a delay for every relay they visit according to an exponential distribution. Finally, providers inject dummy traffic into the network by sending packets to themselves via a Loopix path, so as to provide cover for real messages. The combination of providers that hide mix messages from users sending (respectively receiving) messages at the same time, delays and cover traffic enable Loopix to provide provable guarantees regarding the Unlinkability of the senders and receivers of messages.

Device Metadata. In today's optimised Internet services, the concrete characteristics of users' devices are frequently sent along with their data requests in order to optimise the service providers' responses. Even if users are anonymous on the network layer, these characteristics may become a quasi-identifier that enables service providers to track users across the web [519, 520]. This is because combinations of features such as the User Agent (the browser software vendor, software revision, etc.), its Language, or the Plugins it has installed, or the platform are mostly unique.[4].

Device or browser fingerprinting is the systematic collection of this information for identification and tracking purposes. A large number of attributes, such as browser and operating system type and version, screen resolution, architecture type, and installed fonts, can be

[3] **https://katzenpost.mixnetworks.org/**
[4] **https://amiunique.org/**, **https://panopticlick.eff.org/**

collected directly, using client-side scripting and result in unique fingerprints. When this information is not directly available, other techniques can be used to learn this information, as explained below.

As an illustrative example, let us consider the list of fonts installed on a particular user's web browser as an identifier that enables tracking. There are two techniques to obtain the list of installed fonts, which is known to provide a good level of uniqueness. This is because browsers install fonts on demand depending on the sites visited. Since users have different browsing patterns, their lists of installed fonts become different as well. Font fingerprinting techniques exploit the fact that if a font is installed, browsers will render it, but if not. browsers will revert to monospace font. Thus, depending on whether a font is installed or not, sentences will be rendered differently. In the first technique, the tracking web sends a sentence to the browser to be printed with a series of fonts. Then, the client-side script checks the size of each sentence. When the size is equal to the sentence printed in monospace, the tracker learns that the font is not installed. A similar technique is called canvas fingerprinting. In this case, the tracker exploits the HTML5 Canvas feature, which renders pixels on the fly. As before, different font size result in different pixel footprints. Measuring the result of the canvas rendering the tracker can ascertain which fonts are installed in a browser.

Defending against device Metadata attacks while retaining utility is extremely difficult. On the one hand, hiding these metadata from service providers has an impact on the performance of the services, since it limits personalisation and deteriorates the rendering of information. On the other hand, it is hard to establish which combination of features would actually make users indistinguishable from other users. This is because we have no knowledge of the distribution of fingerprints in order to imitate one of them, and trying combinations at random runs the risk of being as unique as the original fingerprint [521]. Therefore, mechanisms need to be carefully crafted and evaluated [522].

We note that besides tracking based on metadata, trackers also use a series of techniques based on the use of cookies. For instance, web pages can include cookies from third parties, which allows these parties to detect when users revisit a page [523]. Third parties can also use cookie syncing, whereby, besides adding their own tracking, webs redirect cookies to other trackers to inform them of where the users are going [524]. Finally, there exist pervasive mechanisms to install cookie-like information that cannot be removed by cleaning the browser's cache [430].

Location metadata. Finally, a user's geographical location revealed to online services can be used to infer sensitive information. This information can be revealed explicitly, for example, when the user makes queries to location-based services to find nearby points of interest or friends; or implicitly, for example, when GPS coordinates are associated with photos or content published on social networks, or inferred from the access point used to access the Internet.

Clustering techniques to find groups of nearby points where the user spends not significant amounts of time can be used to infer users' points of interest such as where they live, where they work or their favourite leisure places. In many cases, points of interest can be used as quasi-identifiers for users. For instance, the three most commonly visited locations are unique to a user in a majority of cases, even when the locations are provided on a more general level (e.g., US counties) [525]. Similarly, the types and patterns of locations visited can be used to infer demographic data about users such as age or gender [526]. Furthermore, once movement patterns have been characterised, they can be used to predict individuals' future

movements [527].

There are two kinds of defence for protecting location Metadata in the literature. The first relies on cryptographic techniques to process location-based services' queries (see Section 5.1.1.1). For instance, users can privately learn whether a friend is nearby. This service can be realised by Homomorphic encryption encryption [528], private equality testing [529] or private threshold set intersection [530]. The second kind of defence is based on the obfuscation techniques described in Section 5.1.1.2 in order to control the inferences that an adversary draws from the location data. For instance, user's location can be hidden [531], i.e., not reported to the provider; perturbed [532], i.e., reporting a location different from the user's actual position; generalised [533], i.e., reported with less precision; or accompanied by dummy locations so that the user's real movement patterns cannot be identified [534].

5.2 PRIVACY AS CONTROL

[535][536][537]

In the previous section, we discussed privacy technologies that keep data confidential, by minimising the collection of data and/or minimising the amount of information that can be inferred from any released data. A wider notion of privacy, which is usually referenced in regulations, broadens privacy from the notion of concealment of personal information, to the ability to control what happens with the information that is revealed [424, 426].

The idea behind the shift from technologies that minimise disclosure to technologies that provide the means to control information use, is that in many cases, revealing data may be unavoidable or perceived as beneficial to the data subject. Thus, it is advisable to consider the use of technologies that address two major concerns: i) enable users to express how they expect that data disclosed to the service provider are used, so as to prevent undesirable processing of these data; and ii) enable organisations to define and enforce policies that prevent the misuse of information, as defined by the users.

In this section, we revise techniques that have been designed under the privacy as control paradigm. We focus on techniques for the creation and configuration of good privacy settings that help users express their preferences with respect to data disclosure and processing; and techniques that support the automated negotiation of privacy policies across services. Because much of the protection relies on trust, privacy technologies that enhance privacy in a system through improved control are less numerous and varied than those designed to achieve Confidentiality.

It is important to highlight that these techniques inherently trust the service provider that collects the data to correctly enforce the policies established by the user with respect to third parties, as well as not to abuse the collected data itself. Also, as noted by Acquisti et al. [535], providing users with tools to control information flows can reduce risk perception and increase risk-taking, effectively reducing the overall privacy of users.

5.2.1 Support for privacy settings configuration

Privacy settings are those controls in a web service that allow users to express their preferences regarding how data should be revealed to other users, shared with third parties, and processed by the service providers. Madejski et al. have shown that the complexity of these privacy settings makes them barely usable by individuals [536]. This lack of usability causes users to misconfigure their privacy settings, i.e., establish configurations that do not match their expectations. This in turn results in unintended disclosure of data. We refer readers to the Human Factors Knowledge Area (Chapter 4) for further information about the impact of usability of systems on security and privacy.

To counter this problem, researchers have proposed a number of techniques whose goal is to identify groups of individuals that share certain characteristics, and then establish the most adequate settings for each user group. One area of research suggests letting security and privacy experts define what are the best policies are. [538]. This approach, however, is difficult to generalise from targeted groups to the general population, and in many cases may result in strategies that overestimate the need for protection. This in turn limits too much the sharing and processing of data, thus rendering systems unusable. Other proposals advocate for using machine-learning techniques to infer adequate settings for a user based on the social graph of a user's friends and acquaintances [518]. This technique, however, requires users, or a centralised recommender system, to know a user's social graph in order to perform the inferences, which raises privacy concerns in itself. A third approach does not require knowledge of a user's social graph, but tries to find adequate privacy settings by looking at a larger set of users. [539]. As opposed to the previous technique, a user's suggested settings preference is derived from generic data. These techniques have been shown to be prone to produce policies that are valid for the majority of users, but often discriminate against user groups with specific privacy requirements such as activists or persons of public interest. Furthermore, ML-based techniques often augment and perpetuate biases present in the data from which the initial policies are inferred. A final research areas suggests crowdsourcing the optimum composition of these policies [540]. These techniques are more flexible in the sense that users have more leeway to influence the policies. However, they are still influenced by majority votes and may not be ideal for users who do not follow mainstream practices.

5.2.2 Support for privacy policy negotiation

The previous technologies support users at the time of configuring their privacy settings in an online service. An orthogonal line of work is dedicated to automating the communication of user preferences to the service, or between services.

Technologies such as the W3C's Platform for Privacy Preferences Project (P3P) [541], which facilitate the communication of setting preferences between user and service provider. P3P is an industry standard that allows websites to encode their privacy policies (what information is collected, how it is used etc.) in a pre-defined format. These policies can be read and interpreted by browsers equipped to do so. The browser can then compare the site's policy with a user's specified privacy preferences written in a machine readable language such as P3P Preference Exchange Language (APPEL) [542]. P3P, however, does not have any means to enforce that the service provider actually follows the practices described in the policy.

Other technologies such as purpose-based access control [543] or sticky policies [544] provide the means to specify allowed uses of collected information, and to verify that the purpose of a data access is compliant with the policy. These technologies can be supported by

cryptographic mechanisms that guarantee that the service providers must comply with the preferences established by users.

5.2.3 Support for privacy policy interpretability

In order to configure the privacy settings according to their expectations of how data should be handled, users need to understand the privacy policies that describe the meanings of these settings. These policies are often long, verbose, and contain a lot of legal terms; and they often evolve over time. Thus, users find them difficult to understand. Researchers have developed technologies that enhance users'ability to interpret privacy policies.

Currently, there exist two approaches to improve users' understanding of privacy policies. One is to trust experts to label, analyse and provide reasons for existing privacy policies [545]. Another avenue is to completely automate the interpretation process. Polisis[5] [537] is a machine-learning-based framework that enables users to ask questions about natural language privacy policies. This tool offers a visual representation of the policy specifying the types of data collected, the purpose of this collection, and the sharing practices, among others.

5.3 PRIVACY AS TRANSPARENCY

[546][547][548]

The last privacy design paradigm we consider is privacy as transparency. As opposed to technologies that limit data disclosure or the use of disclosed data, transparency mechanisms analyse users' online activities in order to either provide them with feedback about the implications of their actions, or run audits to check that there has been no violation of privacy.

As with control-oriented technologies, transparency-based privacy cannot prevent privacy violations in and of themselves. In fact, feedback or audits happen *after* the users have already disclosed data to the provider. Thus, providers are again trusted with making sure that the collected data are not processed or shared in ways not authorised by the users.

5.3.1 Feedback-based transparency

We first describe mechanisms that make transparent the way in which information is collected, aggregated, analysed and used for decision making. The common factor between these technologies is that they provide users with feedback about how their information is processed or perceived by others.

An early effort in this direction is the concept of privacy mirrors [546], which show users their 'digital selves'; i.e., how others see their data online. This concept was adopted by popular online social networks such as Facebook, which allows users to check how different audiences (e.g., friends, friends of friends, others), or even individual users, see their profiles whenever they make changes to their privacy controls. A similar line of work provides other means of visualising how privacy settings affect data sharing in order to improve users' understanding of the set of permissions they have selected. This solution provides visual cues to users that indicate the access permissions associated with the data they shared [549]. For instance, it

[5] **https://pribot.org/polisis**

can highlight fields in a social network profile with a different colour depending on who has access to that particular information. Both solutions help users understand their practices and modify their actions. However, they can only do so after the information has been revealed to the provider (and possibly to other users).

A different type of user feedback comprises so-called privacy nudges [547]. Nudges assist users in making choices about their privacy and security settings. They give users *immediate* feedback whenever the user performs an online action in a way that the action could be cancelled or modified. For instance, the nudge can inform the user that the post she is currently writing is public so that the user is careful about the words she chooses to use. Nudging tools can be even more sophisticated and use modern machine learning algorithms to analyse photos or text as they are being uploaded, and provide users with more concrete feedback such as 'the post can be perceived as negative' or 'the photo is very explicit'. While immediate feedback presents evident benefits compared to mirrors, since actions can be modified before information is sent to the provider, it also has drawbacks. Experiments with users have shown that immediate feedback results in an uncomfortable feeling for users as they feel monitored, and users sometimes perceive the advice as paternalistic and out of place [548].

5.3.2 Audit-based transparency

As mentioned before, even with privacy policies and access control in place, there is no guarantee that user preferences will be respected. Additional measures can be put in place to enable users to verify that no abuse has taken place. To realise these audits, the system is required to log all data access and processing operations. This logging may reveal when users log into the system, and when and how their data are transmitted to others. Thus, depending on the amount and granularity of the information, logging may introduce additional privacy risks.

Therefore, logging policies must be carefully crafted. One approach to do this is to derive the auditing specifications from the policies using formal methods [550]. This guarantees that the generated logs, while being minimal, still contain enough information to audit whether the policies are being respected. The solutions, however, are limited in their expressiveness and cannot handle privacy policies in modern systems where the amount of data collected and the number of entities involved make a formal analysis extremely cumbersome.

The use of formal methods assumes that data sharing is managed by a centralised authority that must be trusted. This is problematic because the centralised authority becomes a single point of failure. Recent advances in cryptography and distributed ledgers permit the design of solutions that provide the means to create highly secure logs, while ensuring that no private information is shared with unauthorised parties. When logging is made in such a distributed manner, no individual party can modify the log on its own, reducing the need for trust and eliminating any single point of failure. For instance, systems like UnLynx [551] permit entities to share sensitive data, and perform computations on them, without entrusting any entity with protecting the data. All actions are logged in a distributed ledger for auditing, and the correctness of the operations is ensured by using verifiable cryptographic primitives and zero-knowledge proofs. Therefore, it is not necessary to publish or log the sensitive data or the operations done on them.

5.4 PRIVACY TECHNOLOGIES AND DEMOCRATIC VALUES

[552][553][554]

Privacy technologies are of paramount importance to ensure that our fundamental right to privacy is respected in the digital world. Privacy protection is crucial for underpinning the values that support our democratic societies. Citing Daniel Solove: 'Part of what makes a society a good place in which to live is the extent to which it allows people freedom from the intrusiveness of others. A society without privacy protection would be suffocation' [554]. While such a society seemed science fiction not so long ago, episodes such as the Facebook Cambridge Analytica case highlight the importance of securing data from being accessed by unintended parties (e.g., using Confidentiality or control techniques) to protect citizens from interference and manipulation.

In this section, we provide two examples that highlight the importance of privacy technologies in supporting democracy. On one hand, we consider electronic voting systems that enable fair elections to take place using electronic infrastructure in adversarial conditions. On the other hand, we give an overview of censorship resistance technologies. These systems ensure that in a digital world, where communication infrastructure is dominated by a small number of companies and state actors can observe all communications, individuals have the means to communicate freely.

5.4.1 Privacy technologies as support for democratic political systems

The growing use of electronic applications to interact with governmental bodies brings great advantages to society. Providing citizens with easy means to express their opinions, comment on government initiatives, or vote in elections, increases their involvement in public decision processes. This in turn improves the power balance between those who can execute decisions and those who are affected by the outcome of the decision process.

For these improvements to be effective, citizens must be able to freely express their opinions and must be sure that their inputs cannot be modified or lost during the process. The use of common infrastructures (e.g., cloud services or unprotected communication networks) to implement these democracy-oriented applications, however, raises concerns about surveillance and manipulation. Therefore, it is important that these applications are supported by strong privacy technologies that can protect users' identities, as well as their sensitive data and inputs to the system. We describe two example applications, electronic voting and electronic petitions, whereby the technologies introduced in the previous sections are combined to enable citizens and governments to enjoy technological progress without compromising our democratic values.

Electronic voting (eVoting). Electronic voting systems have the goal of enabling fair elections to be conducted via electronic infrastructure in adversarial conditions. In particular, eVoting schemes provide:

- *Ballot secrecy*: an adversary cannot determine which candidate a user voted for.
- *Universal verifiability*: an external observer can verify that all the votes cast are counted and that the tally is correct. Some protocols provide a weaker property, *individual verifiability*, where each voter can verify that his/her vote has been correctly tallied. Benaloh et

al. provide a comprehensive overview of the aspects to assess to obtain end-to-end verifiability [555].

- *Eligibility verifiability*: an external observer can verify that all the votes cast were made by a unique eligible voter.

In order to guarantee the first aspect, it is key to break the links between the votes putting their ballots into the system, and the ballots that come out. In traditional pen-and-paper physical elections, this is done by mixing the ballots all of which have exactly the same appearance in an urn. In eVoting, Unlinkability is typically achieved using mix networks [556, 557]. the votes are passed through a series of mixes, which must not belong to the same authority. Otherwise, this authority could trace the votes and link the voters to their voting choices. The results are published on a public bulletin board which anybody can read and verify that the election was carried out in a honest manner.

Voting mix networks are designed in a slightly different way than those mentioned in Section. 5.1.2. In the case of eVoting, the mixes fire when all votes are in, and the batching strategy is to take all the votes. In simple terms, it ensures that all votes are mixed together, obtaining the maximum Anonymity set. This fulfills the ballot secrecy criterion as any vote could have been cast by any voter. Furthermore, to ensure universal verifiability, in eVoting mix networks every node does *verifiable shuffles* [552]. This means that the mixes prove, in zero knowledge, that they mix all the votes (all the votes at the input appear at the output) and the mixing is random. Eligibility verifiability can be obtained by requiring voters to prove in zero-knowledge that they are eligible to vote.

Other voting protocols provide ballot secrecy through the use of blind signatures: an authorised entity verifies the eligibility of a user and blindly signs her vote (i.e., without seeing the vote content) [558]. The user provides a zero-knowledge proof along with the vote that the vote has been correctly constructed. Then, users submit the signed votes to the tally server using an anonymous communication channel. This way no entity in the system can link voter to votes.

A third strategy is based on Homomorphic encryption encryption. In these schemes, the tally server creates a bulletin board with encrypted zero entries for every candidate [559, 560]. Then, every user adds his vote to the desired candidate, and randomises the rest of the encryptions (so that encryptions of the same number never look the same). As before, zero-knowledge proofs can be used to ensure that sums and randomisation have been performed in the correct way.

Besides the above three properties, some voting protocols additionally aim to provide *coercion resistance*, whereby a user cannot be forced to vote for a particular candidate against her will. One strategy to implement such a system is to provide users with fake credentials [561]. Then, when users are under coercion they follow the instructions of the coercer, but provide their fake credentials to the system. This enables the tally server to ignore any votes produced under coercion. Related approaches prevent coercion via re-voting, i.e., the schemes permit users to recast a vote so as to cancel their coerced choice [562]. These schemes define policies to establish how to count votes whenever a given credential has cast more than one vote. (e.g., count the last one, or add a pointer to the cancelled vote).

Anonymous petitions. We define a petition as a formal request to a higher authority, e.g., parliament or another authority, signed by one or more citizens. Signing a petition publicly, however, might raise concerns or conflicts in terms of relationships between friends, colleagues, and neighbours, discouraging citizens from participation [563]. Privacy technologies, in particular,

anonymous credentials, can help in creating secure and privacy-preserving petition systems.

In petition systems based on anonymous credentials, citizens can register with the authority managing the petition system to obtain an anonymous signing key associated with some attributes relevant for the petitions. Then, at the time of signing a particular petition, they can prove they are eligible (e.g., they are inhabitants of the municipality referred to) but do not need to reveal their identity. Advanced credential properties such as double signing detection enable the creation of this system while avoiding abuse from misbehaving citizens [564].

More modern approaches rely on advanced cryptographic primitives to remove the need for a central trusted party that registers users. For instance, Sonnino et al. [565] enable threshold issuance and verification of credentials to sign the petition, i.e., several authorities participate in the issuance. This scheme improves Confidentiality, authenticity, and availability through the use of distributed ledgers. This approach increases the level of privacy in the system, while at the same time reducing the need to trust one single party.

5.4.2 Censorship resistance and freedom of speech

Censorship systems attempt to impose a particular distribution of content across a system. They may prevent users from publishing particular content that is considered controversial or dangerous for the censorship regime; or they may prevent users from accessing content that may undermine the societal equilibrium that the censor wishes to impose on their society.

In this section, we show how privacy-preserving technologies can act as a cornerstone to support freedom of speech and freedom of access to information. We will elaborate on some examples for each of these goals in order to illustrate the fundamental principles that make censorship resistance possible. We refer the interested reader to the surveys by Khattak et al. [553] and Tschantz et al. [566] for a comprehensive revision of censorship resistance systems.

Data publishing censorship resistance. Motivated by the 'Church of Scientology' court order, which caused the closure of the Penet remailer at the end of the 1990s [567], Anderson proposed the Eternity Service. This was the first system to use privacy technologies to protect the publishing of content on the Internet [568]. Anderson's scheme proposed to distribute copies of files across servers in different jurisdictions, so that those servers cannot be subpoenaed at the same time. In this scheme, privacy technologies have fundamental roles for resistance: Encryption not only provides privacy for users, but also prevents selective denial of service at retrieval time; and anonymous Authentication not only protects users from the service, it also protects the service from being coerced into revealing the identities of users, e.g., by law enforcement, since it cannot know these users' identities.

Anderson's proposal inspired later designs such as Freenet, a peer-to-peer system to publish, replicate, and retrieve data while protecting the Anonymity of both the authors and readers [569]. Additionally, the system provides deniability for the entities storing the information; i.e., the servers cannot know the content of the files they store and thus, can always claim to be unaware of what they are serving. In Freenet, files are located according to a key that is typically the hash of the file, but can also include a file description, or be a simple string. To retrieve a file, a user obtains or computes the keys and asks Freenet nodes to find it. If a node does not contain the file, it asks a neighbour. When the file is found, it is sent back along the same path that the request followed in the network. This ensures that the node holding the data does not know the recipient. To store a file, if the key does not already exist,

the publisher sends the file along a path and every node on the path stores the file. To protect the anonymity of the publisher, nodes that store the file also decide at random whether to also claim ownership. Such random claims also provide nodes with deniability as to which of the files they are storing are actually theirs.

The design of Freenet is based on strong cryptography, which protects the content of messages. However, in the early days, the routes and timing of messages allowed attacks to break the system's anonymity. Tian et al. [570] show that a passive attacker deploying a number of nodes in the network that can monitor requests can re-identify the requester by recursively asking other nodes if they have seen the request. Freenet also allows for the collection of privacy-preserving statistics. However, the statistic obfuscation method is vulnerable to inference attacks where the adversarial node combines several queries in order to learn information about other Freenet nodes' properties (e.g., bandwidth) [571]. These issues are now addressed by Freenet, but others remain such as the attack by Levine et al. [572], which enables a single node to distinguish whether a neighbouring peer is the actual requester of a file or just forwarding the requests for other peers. The attack only requires passive observation of traffic, and exploits the fact that the Freenet protocol determines the average number of requests for a file observable by a node depending on how far this node is from the requester. Thus, a simple Bayesian inference suffices to detect whether a neighbour is the request initiator.

A different approach to censorship is followed in Tangler [573]. The system also provides publisher and reader anonymity, but achieves censorship resistance in a different way. Instead of simply storing the file replicated in many nodes in an anonymous way, Tangler files are split into small blocks that are stored on different servers. In order to recover a file, one must thus contact a number of servers to retrieve enough of these blocks. In order to avoid a server being compelled to delete a block belonging to a file, Tangler builds blocks in such a way that blocks contain parts of many documents. To 'tangle' files into blocks, Tangler uses secret sharing (See the Cryptography Knowledge Area (Chapter 10)). Tangling improves availability in two ways. First, a censor can only delete a target file by causing collateral damage to other files that may be allowed. Second, whenever one wants to replicate a file, the files entangled in the replicated file blocks are also replicated.

Data access censorship resistance. To enable censorship-free access to data, systems must be able to conceal that users are accessing these data. This can be done in a number of ways [553]. A first approach is *mimicking*, where censorship resistance is obtained by attempting to make accessing to censored data look like accessing allowed data (e.g., as a Skype call [574] or as a visit to an innocuous web page [575]). These approaches are effective, but have been shown to be vulnerable to active attacks in which the adversary probes the suspicious connection to find out if any of the expected functions of the application being mimicked are missing [576].

A second approach is tunnelling. In this case, the censored communication is directly tunnelled through an uncensored service, instead of pretending to be that service. In particular, these systems use widely used services as tunnels, e.g., cloud services [577, 578], so that blocking communications imposes a high cost for the censor. A third approach is to embed the communication inside some content (e.g., hidden in a photo or video [579]). This approach not only makes communications unobservable, but also deniable for all senders, recipients and applications hosting the content.

Finally, some censorship resistance systems rely on hiding the destination of the communi-

cation to prevent censors from blocking connections. This is achieved by relaying censored traffic through one or more intermediate nodes. These nodes can be proxies, such as *bridges* in the Tor network [514]. These bridges are Tor relays whose IPs are not public so that they cannot be identified as members of a censorship resistance system. To avoid the censor identifying connections to bridges due to their appearance, these are disguised using so-called pluggable transports [580], which transform the traffic flow following one of the approaches referenced in this section.

Another option to hide the destination is the use of *decoy routing*, also known as *refraction networking* [581, 582]. In decoy routing, clients direct their censored traffic to a benign destination. This traffic includes an undetectable signal that can only be interpreted by a cooperating Internet router. This router deflects the client's traffic to the censored site and returns the responses to the client. Obviously, the cooperating router must be outside of the censor's domain, but, depending on the scheme, it can be on the forward path from the client to the uncensored destination [581, 582], or on the downstream path [583, 584].

5.5 PRIVACY ENGINEERING

[585][586]

The growing privacy concerns in society have made the concept of 'privacy by design' very popular among policy makers. This concept advocates for the design and development of systems that integrate privacy values to address users' concerns. However, the literature around this concept rarely addresses the actual processes behind the design, implementation, and integration of privacy protections into products and services.

In this knowledge area, we first gave an overview of the landscape of privacy technologies, and subsequently provided a series of examples in which these technologies are combined to support the use of electronic systems while maintaining core democratic values. In this section, we elaborated on the design principles behind these, and other, privacy-preserving systems. We briefly discussed how these principles can be used to generally approach the engineering of systems that embed strong privacy protections. We refered the reader to the work by Gürses et al. [585] for a more comprehensive explanation of these principles and their role in the design of privacy-preserving systems. A relevant paper to help the reader understanding these principles is the work Hoepman on privacy strategies [587]

The two primary goals when designing privacy-preserving systems are to:

- **Minimise trust**: limit the need to rely on other entities to behave as expected with respect to sensitive data. For instance, in mix-based eVoting, trust is not only distributed across the entities managing the mixes, but verifiable shuffles are put in place to limit to a maximum the amount of reliance on the good behaviour of each mix. Similarly, the cryptographic primitives used to implement privacy-preserving electronic petitions do not require trust on the registration authority to protect the identities of the signers.
- **Minimise risk**: limit the likelihood and impact of a privacy breach. For instance, in Tor, compromising one relay does not provide any sensitive information about users' browsing habits. If one compromises the entry node, one cannot learn the destinations of the communications, only the middle nodes of the circuits; and if one compromises the exit node, one cannot learn the origin of the communication.

To minimise both trust and risk, privacy experts typically design systems in accordance with the following strategies:

- *Minimise Collection*: whenever possible, limit the capture and storage of data in the system.
- *Minimise Disclosure*: whenever possible, constrain the flow of information to parties other than the entity to whom the data relates. This refers both to direct flows between senders and receivers, and to indirect flows, e.g., use of control techniques to limit the information available when publishing or querying a dataset.
- *Minimise Replication*: whenever possible, limit the number of entities where data are stored or processed in the clear.
- *Minimise Centralization*: whenever possible, avoid a single point of failure regarding the privacy properties in the system.
- *Minimise Linkability*: whenever possible, limit the capability of the adversary to link data.
- *Minimise Retention*: whenever possible, limit the amount of time information is stored.

Implementing these strategies at first may seem incompatible with maintaining the Integrity of the system. For instance, if no information is disclosed or collected, how can one make sure that no entity is abusing the system? If there is no central authority, how can one make sure that Authentication and authorisation work as expected? This is where privacy technologies come into play. They enable the design of systems where as little information as possible is revealed to parties other than the ones to which the information relates, and in which there is a minimum need to trust providers or other users in order to preserve the privacy of sensitive information while still pertaining Integrity and allowing information exchange.

In order to decide which privacy technology is most adequate to build a system, a first step is to identify the data flows that should be minimised; i.e., those that move data to entities to whom the data do not relate. The second step is to identify the minimal set of data that needs to be transferred to those entities. To identify the minimum required information that needs to be transferred, the designer should attempt to keep as much data as possible out of reach of those entities without harming the functionality of the system. Strategies to minimise unnecessary information flow (based mainly on the technologies introduced throughout Section. 5.1) are:

- *Keep the data local*: perform any computation on sensitive data on the user side, and only transmit the result of the operation. Additional information, such as zero-knowledge proofs or commitments, may be needed to guarantee the correctness of the operations.
- *Encrypt the data*: encrypt the data locally and send only the encrypted version to other entities. If any operations on the data are needed, see the next point.
- *Use privacy-preserving cryptographic protocols*: process data locally to obtain inputs to a protocol in which, by interacting with the untrusted entities using one of the protocols introduced in the previous sections, the user can obtain or prove information while limiting the information made available to those entities. For instance, using anonymous credentials for Authentication without revealing the identity or even the value of an attribute, or using private information retrieval to perform a search on a database without revealing the query to the database holder.

- *Obfuscate the data*: use techniques to control inference to process the data locally and only send the perturbed version to the untrusted entity.
- *Anonymise the data*: process the data locally to remove identifiable information and send it to the untrusted party via an anonymous channel.

By seeking minimisation of trust and using the above techniques, system designers are bound to collect, process and retain fewer data than with other strategies based on compliance with regulation. We recognise that many systems and applications cannot be built without collecting some user-related data. For these cases, designers must take into account the privacy technologies outlined in Section. 5.2 and Section. 5.3. These techniques, while requiring trust, help minimise the risk of a breach and, if the breach happens, minimise the impact that the disclosure of data may have for the users.

Privacy evaluation. Once privacy technologies or end-to-end systems have been designed, it is important to conduct a privacy evaluation. This evaluation has the goal of quantifying the level of privacy that the technology, and respectively the system, can provide.

For privacy technologies based on cryptographic primitives, the privacy evaluation is typically covers the cryptographic proofs that ensure that only the intended information is leaked by the operations. On the contrary, for privacy techniques based on obfuscation, it is necessary to carry out an analysis to validate that the combination of techniques provides the desired level of privacy.

A systematic privacy evaluation typically consists of the following steps. First, one needs to model the privacy-preserving mechanism as a probabilistic transformation. This establishes the probability that, given an input, the privacy mechanism returns a given output. Second, one needs to establish the threat model, i.e., what the adversary can see and what is her prior knowledge. Third, assuming that the adversary knows the mechanism, consider how he would annul the effect of the privacy mechanism. This usually entails either doing an analysis of the probability distributions, or using inference techniques such as machine learning to compute what the adversary can learn.

At the end of the process, one usually has a distribution describing the probability that the adversary infers each of the possible inputs. This probability distribution is then used as input to a privacy metric that captures the inference capability of the adversary. We refer the reader to the survey by Wagner and Eckhoff for a comprehensive description of privacy metrics in the literature [586].

5.6 CONCLUSIONS

Protecting privacy, as we have described in this knowledge area, is not limited to guaranteeing the Confidentiality of information. It also requires means to help users understand the extent to which their information is available online, and mechanisms to enable users to exercise control over this information. We have described techniques to realise these three privacy conceptions, emphasising the adversarial model in which they operate, as well as providing guidelines to combine these techniques in order to build end-to-end privacy-preserving systems.

Preserving privacy and online rights is not only important for individuals, it is essential to support democratic societies. The deployment of privacy technologies is key to allow users free access to content, and freedom of speech. Of equal importance is to avoid that any entity

gaining a disproportionate amount of information about individuals or groups, in order to prevent manipulation and abuse that could damage democratic values.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Topic	Cite
5.1 Privacy as Confidentiality	
5.1.1 Data Confidentiality	[431, 432, 433, 434, 435]
5.1.2 Metadata Confidentiality	[430, 436, 437]
5.2 Privacy as Control	
5.2.1 Support for privacy settings configuration	[535]
5.2.2 Support for privacy policy negotiation	[536]
5.2.3 Support for privacy policy interpretability	[537]
5.3 Privacy as Transparency	
5.3.1 Feedback-based transparency	[546, 547, 548]
5.3.2 Audit-based transparency	[550]
5.4 Privacy Technologies and Democratic Values	
5.4.1 Privacy technologies as support for democratic political systems	[552, 554]
5.4.2 Censorship resistance and freedom of speech	[553]
5.5 Privacy Engineering	[585, 586]

II Attacks & Defences

Chapter 6
Malware & Attack Technologies

Wenke Lee | Georgia Institute of Technology

INTRODUCTION

Malware is short for 'malicious software', that is, any program that performs malicious activities. We use the terms malware and malicious code interchangeably. Malware comes with a wide range of shapes and forms, and with different classifications accordingly, e.g., viruses, Trojans, worms, spyware, botnet malware, ransomware, etc.

Malware carries out many of the cyberattacks on the Internet, including nation-state cyberwar, cybercrime, fraud and scams. For example, Trojans can introduce a backdoor access to a government network to allow nation-state attackers to steal classified information. Ransomware can encrypt data on a user's computer and thus making it unaccessible to the user, and only decrypt the data after the user pays a sum of money. Botnet malware is responsible for many of the Distributed Denial-of-Service (DDoS) attacks as well as spam and phishing activities. We need to study the techniques behind malware development and deployment in order to better understand cyberattacks and develop the appropriate countermeasures.

As the political and financial stakes become higher, the sophistication and robustness of both the cyber defence mechanisms and the malware technologies and operation models have also increased. For example, attackers now use various obfuscation techniques such as packing and polymorphism as well as metamorphism to evade malware detection systems [588], and they set up adaptive network infrastructures on the Internet to support malware updates, command-and-control, and other logistics such as transits of stolen data. In short, it is becoming more important but also more challenging to study malware.

The rest of this chapter is organised as follows. We will provide a taxonomy of malware and discuss their typical malicious activities as well as their eco-system and support infrastructures. We will then describe the tools and techniques to analyse malware behaviours, and network- and host- based detection methods to identify malware activities, as well as processes and techniques including forensic analysis and attribution to respond to malware attacks.

CONTENT

6.1 A TAXONOMY OF MALWARE

[589, c6]

There are many types of malware [589]. It is instructive to create a taxonomy to systematically categorise the wide spectrum of malware types. This taxonomy describes the common characteristics of each type of malware and thus can guide the development of countermeasures applicable to an entire category of malware (rather than a specific malware). Since there many facets of malware technologies and attack operations, based on which malware can be categorised and named, our taxonomy can include many dimensions. We discuss a few important ones below. It should be borne in mind that other, more specialised, attributes could also be used such as target processor architecture or operating system.

The first dimension of our taxonomy is whether malware is a standalone (or, independent) program or just a sequence of instructions to be embedded in another program. Standalone malware is a complete program that can run on its own once it is installed on a compromised machine and executed. For example, worms and botnet malware belong to this type. The

second type requires a host program to run, that is, it must infect a program on a computer by inserting its instructions into the program so that when the program is run, the malware instructions are also executed. For example, document macro viruses and malicious browser plug-ins belong to this type. In general, it is easier to detect standalone malware because it is a program or a running process in its own right and its presence can be detected by operating system or security tools.

The second dimension is whether malware is persistent or transient. Most malware is installed in persistent storage (typically, a file system) as either standalone malware or an infection of another program that already resides in persistent storage. Other malware is memory-resident such that if the computer is rebooted or the infected running program terminates, it no longer exists anywhere on the system. Memory-resident malware can evade detection by many anti-virus systems that rely on file scanning. Such transient malware also has the advantage of being easy to clean up (or, cover-up) its attack operations. The traditional way for malware to become memory-resident is to remove the malware program (that was downloaded and installed previously) from the file system as soon as it gets executed. Newer approaches exploit system administrative and security tools such as PowerShell to inject malware directly into memory [590]. For example, according to one report [591], after an initial exploit that led to the unauthorised execution of PowerShell, meterpreter code was downloaded and injected into memory using PowerShell commands and it harvested passwords on the infected computer.

The third dimension generally applies to only persistent malware and categorises malware based on the layer of the system stack the malware is installed and run on. These layers, in the ascending order, include firmware, boot-sector, operating system kernel, drivers and Application Programing Interfaces (APIs), and user applications. Typically, malware in the lower layers is harder to detect and remove, and wreaks greater havoc because it has more control of the compromised computer. On the other hand, it is also harder to write malware that can be installed at a lower layer because there are greater constraints, e.g., a more limited programming environment in terms of both the types and amount of code allowed.

The fourth dimension is whether malware is run and spread automatically vs. activated by a user action. When an auto-spreading malware runs, it looks for other vulnerable machines on the Internet, compromises these machines and installs itself on them; the copies of malware on these newly infected machines immediately do the same – run and spread. Obviously, auto-spreading malware can spread on the Internet very quickly, often being able to exponentially increase the number of compromised computers. On the other hand, user-activated malware is run on a computer only because a user accidentally downloads and executes it, e.g., by clicking on an attachment or URL in a received email. More importantly, when this malware runs, although it can 'spread', e.g., by sending email with itself as the attachment to contacts in the user's address book, this spreading is not successful unless a user who receives this email activates the malware.

The fifth dimension is whether malware is static or one-time vs. dynamically updated. Most modern malware is supported by an infrastructure such that a compromised computer can receive a software update from a malware server, that is, a new version of the malware is installed on the compromised computer. From an attacker's point-of-view, there are many benefits of updating malware. For example, updated malware can evade detection techniques that are based on the characteristics of older malware instances.

The sixth dimension is whether malware acts alone or is part of a coordinated network (i.e., a botnet). While botnets are responsible for many cyberattacks such as DDoS, spam, phishing, etc., isolated malware has become increasingly common in the forms of targeted attack. That

is, malware can be specifically designed to infect a target organisation and perform malicious activities according to those assets of the organisation valuable to the attacker.

Most modern malware uses some form of obfuscation in order to avoid detection (and hence we do not explicitly include obfuscation in this taxonomy). There is a range of obfuscation techniques and there are tools freely available on the Internet for a malware author to use. For example, polymorphism can be used to defeat detection methods that are based on 'signatures' or patterns of malware code. That is, the identifiable malware features are changed to be unique to each instance of the malware. Therefore, malware instances look different from each other, but they all maintain the same malware functionality. Some common polymorphic malware techniques include packing, which involves compressing and encrypting part of the malware, and rewriting identifiable malicious instructions into other equivalent instructions.

	standalone or host-program	persistent or transient	layers of system stack	auto-spreading?	dynamically updatable?	coordinated?
viruses	host-program	persistent	firmware and up	Y	Y	N
malicious browser extensions	host-program	persistent	application	N	Y	Y
botnet malware	both	persistent	kernel and up	Y	Y	Y
memory-resident malware	standalone	transient	kernel and up	Y	Y	Y

Table 6.1: Use of the Taxonomy to Classify Representative Malware

As an illustration, we can apply this taxonomy to several types (or names) of malware. See Table 6.1. In particular, a virus needs a host-program to run because it infects the host-program by inserting a malicious code sequence into the program. When the host-program runs, the malicious code executes and, in addition to performing the intended malicious activities, it can look for other programs to infect. A virus is typically persistent and can reside in all layers of the system stack except hardware. It can spread on its own because it can inject itself into programs automatically. A virus can also be dynamically updated provided that it can connect to a malware update server. A polymorphic malware virus can mutate itself so that new copies look different, although the algorithm of this mutation is embedded into its own code. A virus is typically not part of a coordinated network because while the infection can affect many computers, the virus code typically does not perform coordinated activities.

Other malware that requires a host-program includes malicious browser plug-ins and extensions, scripts (e.g., JavaScript on a web page), and document macros (e.g., macro viruses and PDF malware). These types of malware can be updated dynamically, form a coordinated network, and can be obfuscated.

Botnet malware refers to any malware that is part of a coordinated network with a botnet infrastructure that provides command-and-control. A botnet infrastructure typically also provides malware update, and other logistic support. Botnet malware is persistent and typically obfuscated, and usually resides in the kernel, driver, or application layers. Some botnet malware requires a host-program, e.g., malicious browser plug-ins and extensions, and needs user activation to spread (e.g., malicious JavaScript). Other botnet malware is standalone,

and can spread automatically by exploiting vulnerable computers or users on the Internet. These include trojans, key-loggers, ransomware, click bots, spam bots, mobile malware, etc.

6.1.1 Potentially unwanted programs (PUPs)

A potentially unwanted program (PUP) is typically a piece of code that is part of a useful program downloaded by a user. For example, when a user downloads the free version of a mobile game app, it may include adware, a form of PUP that displays ad banners on the game window. Often, the adware also collects user data (such as geo-location, time spent on the game, friends, etc.) without the user's knowledge and consent, in order to serve more targeted ads to the user to improve the effectiveness of the advertising. In this case, the adware is also considered spyware, which is defined as unwanted program that steals information about a computer and its users. PUPs are in a grey area because, while the download agreement often contains information on these questionable behaviours, most users tend not to read the finer details and thus fail to understand exactly what they are downloading.

From the point of view of cybersecurity, it is prudent to classify PUPs towards malware, and this is the approach taken by many security products. The simple reason is that a PUP has all the potential to become full-fledged malware; once it is installed, the user is at the mercy of the PUP operator. For example, a spyware that is part of a spellchecker browser extension can gather information on which websites the user tends to visit. But it can also harvest user account information including logins and passwords. In this case, the spyware has become a malware from just a PUP.

6.2 MALICIOUS ACTIVITIES BY MALWARE

[589, c6][588, c11-12]

Malware essentially codifies the malicious activities intended by an attacker. Cyberattacks can be analysed using the Cyber Kill Chain Model [592], which, as shown in Table 6.2, represents (iterations of) steps typically involved in a cyberattack. The first step is Reconnaissance where an attacker identifies or attracts the potential targets. This can be accomplished, for example, by scanning the Internet for vulnerable computers (i.e., computers that run network services, such as sendmail, that have known vulnerabilities), or sending phishing emails to a group of users. The next phase is to gain access to the targets, for example, by sending crafted input to trigger a vulnerability such as a buffer overflow in the vulnerable network service program or embedding malware in a web page that will compromise a user's browser and gain control of his computer. This corresponds to the Weaponization and Delivery (of exploits) steps in the Cyber Kill Chain Model. Once the target is compromised, typically another piece of malware is downloaded and installed; this corresponds to the Installation (of malware) step in the Cyber Kill Chain Model. This latter malware is the real workhorse for the attacker and can carry out a wide range of activities, which amount to attacks on:

- confidentiality – it can steal valuable data, e.g., user's authentication information, and financial and health data;
- integrity – it can inject falsified information (e.g., send spam and phish emails, create fraudulent clicks, etc.) or modify data;

- availability – it can send traffic as part of a distributed denial-of-service (DDoS) attack, use up a large amount of compute-resources (e.g., to mine cryptocurrencies), or encrypt valuable data and demand a ransom payment.

	Step	Activities
1	Reconnaissance	Harvesting email addresses, identifying vulnerable computers and accounts, etc.
2	Weaponization	Designing exploits into a deliverable payload.
3	Delivery	Delivering the exploit payload to a victim via email, Web download, etc.
4	Exploitation	Exploiting a vulnerability and executing malicious code on the victim's system.
5	Installation	Installing (additional) malware on the victim's system.
6	Command & Control	Establishing a command and control channel for attackers to remotely commandeer the victim's system.
7	Actions on Objectives	Carrying out malicious activities on the victim's system and network.

Table 6.2: The Cyber Kill Chain Model

Most modern malware performs a combination of these attack actions because there are toolkits (e.g., a key-logger) freely available for carrying out many 'standard' activities (e.g., recording user passwords) [588], and malware can be dynamically updated to include or activate new activities and take part in a longer or larger 'campaign' rather than just performing isolated, one-off actions. These are the Actions on Objectives in the Cyber Kill Chain Model.

Botnets exemplify long-running and coordinated malware. A botnet is a network of bots (or, compromised computers) under the control of an attacker. Botnet malware runs on each bot and communicates with the botnet command-and-control (C&C) server regularly to receive instructions on specific malicious activities or updates to the malware. For example, every day the C&C server of a spamming botnet sends each bot a spam template and a list of email addresses so that collectively the botnet sends a very large number of spam messages. If the botnet is disrupted because of detection and response actions, e.g., the current C&C server is taken down, the botnet malware is already programmed to contact an alternative server and can receive updates to change to a botnet that uses peer-to-peer for C&C. In general, botnets are quite noisy, i.e., relatively easy to detect, because there are many bots in many networks. Botnet C&C is an example of the Command & Control step in the Cyber Kill Chain Model.

In contrast to botnets, malware behind the so-called advanced persistent threats (APTs) typically targets a specific organisation rather than aiming to launch large-scale attacks. For example, it may look for a particular type of controller in the organisation to infect and cause it to send the wrong control signals that lead to eventual failures in machineries. APT malware is typically designed to be long-lived (hence the term 'persistent'). This means it not only receives regular updates. but also evades detection by limiting its activity volume and intensity (i.e., 'low and slow'), moving around the organisation (i.e., 'lateral movements') and covering its tracks. For example, rather than sending the stolen data out to a 'drop site' all at once, it can send a small piece at a time and only when the server is already sending legitimate traffic; after it has finished stealing from a server it moves to another (e.g., by exploiting the trust relations between the two) and removes logs and even patches the vulnerabilities in the first server.

When we use the Cyber Kill Chain Model to analyze a cyberattack, we need to examine its activities in each step. This requires knowledge of the attack techniques involved. The ATT&CK

Knowledge Base [593] documents the up-to-date attack tactics and techniques based on real-world observations, and is a valuable reference for analysts.

6.2.1 The Underground Eco-System

The early-day malware activities were largely nuisance attacks (such as defacing or putting graffiti on an organisation's web page). Present-day malware attacks are becoming full-blown cyberwars (e.g., attacks on critical infrastructures) and sophisticated crimes (e.g., ransomware, fake-AntiVirus tools, etc.). An underground eco-system has also emerged to support the full malware lifecycle that includes development, deployment, operations and monetisation. In this eco-system, there are actors specialising in key parts of the malware lifecycle, and by providing their services to others they also get a share of the (financial) gains and rewards. Such specialisation improves the quality of malware. For example, an attacker can hire the best exploit researcher to write the part of the malware responsible for remotely compromising a vulnerable computer. Specialisation can also provide plausible deniability or at the least limit liability. For example, a spammer only 'rents' a botnet to send spam and is not guilty of compromising computers and turning them into bots; likewise, the exploit 'researcher' is just experimenting and not responsible for creating the botnet as long as he did not release the malware himself. That is, while they are all liable for the damage by malware, they each bear only a portion of the full responsibility.

6.3 MALWARE ANALYSIS

[588, c1-10] [594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605]

There are many benefits in analysing malware. First, we can understand the intended malicious activities to be carried out by the malware. This will allow us to update our network and endpoint sensors to detect and block such activities, and identify which machines have the malware and take corrective actions such as removing it or even completely wiping the computer clean and reinstalling everything. Second, by analysing the malware structure (e.g., the libraries and toolkits that it includes) and coding styles, we may be able to gain information that is potentially useful to attribution, which means being able to identify the likely author and operator. Third, by comparing it with historical as well as geo-location data, we can better understand and predict the scope and trend of malware attacks, e.g., what kinds of activities (e.g., mining cryptocurrencies) are on the rise and if a cybercrime is moving from one region to another. In short, malware analysis is the basis for detecting and responding to cyberattacks.

Malware analysis typically involves running a malware instance in an analysis environment. There are ways to 'capture' malware instances on the infection sites. A network sensor can examine traffic (e.g., web traffic, email attachment) to identify possible malware (e.g., payload that contains binary or program-like data from a website with a low reputation) and run it in a sandbox to confirm. If a network sensor is able to detect outgoing malicious traffic from an internal host, a host-based sensor can further identify the program, i.e., the malware, responsible for such traffic. There are also malware collection and sharing efforts where trusted organisations can upload malware samples found in their networks and also receive samples contributed by other organisations. Academic researchers can typically just obtain malware samples without needing to contribute. When acquiring and sharing malware samples, we must consider our legal and ethical responsibilities carefully [606]. For example, we must protect the identities of the infection sites from which the malware samples

were captured, and we must not share the malware samples with any organisation that is an unknown entity or that does not have the commitment or technical capabilities to analyse malware safely.

The malware analysis pipeline typically includes the following steps: 1) identifying the format of a malware sample (e.g., binary or source code, Windows or Linux, etc.), 2) static analysis using disassembly (if the malware is in binary format), program analysis, statistical analysis of the file contents, etc., and 3) dynamic analysis using an analysis environment. Steps 2 and 3 can be combined and iterated.

6.3.1 Analysis Techniques

Malware analysis is the process of learning malware behaviours. Due to the large volume and increasing complexity of malware, we need to be able to rapidly analyse samples in a complete, reliable and scalable way. To achieve this, we need to employ techniques such as static analysis, dynamic analysis, symbolic execution and concolic execution [588]. These program analysis techniques have been developed to support the software development cycle, and they often need to be customized or extended for malware analysis because malicious programs typically include code constructed specifically to resist analysis. That is, the main challenge in malware analysis is to detect and bypass anti-analysis mechanisms.

6.3.1.1 Static Analysis

Static analysis involves examining the code (source, intermediate, or binary) to assess the behaviours of a program without actually executing it [588]. A wide range of malware analysis techniques fall into the category of static analysis. One limitation is that the analysis output may not be consistent with the actual malware behaviours (at runtime). This is because in many cases it is not possible to precisely determine a program's behaviours statically (i.e., without the actual run-time input data). A more serious problem is that malware authors are well aware of the limitations of static analysis and they leverage code obfuscation and packing to thwart static-analysis altogether. For example, the packed code cannot be statically analysed because it is encrypted and compressed data until unpacked into executable code at run-time.

6.3.1.2 Dynamic analysis

Dynamic analysis monitors the behaviours of malware execution in order to identify malicious behaviours [588]. Static analysis can provide more comprehensive coverage of program behaviours but may include unfeasible ones. Dynamic analysis identifies the precise program behaviours per the test input cases but misses behaviours that are not triggered by the input. Additionally, dynamical analysis can defeat code obfuscation techniques designed to evade static analysis. For example, when malware at run-time unpacks and executes its packed code, dynamic analysis is able to identify the (run-time) malicious behaviours in the originally packed code. When performing dynamic analysis, the main questions to consider are: what types of malicious behaviours need to be identified and correspondingly, what run-time features need to be collected and when to collect (or sample), and how to isolate the effects on the malware from those of benign system components. Typically, the run-time features to be collected need to be from a layer lower than the malware itself in the system stack so that the malware cannot change the collected information. For example, instruction traces certainly cover all

the details of malicious behaviours but the data volume is too large for efficient analysis [607]. On the other hand, system call (or API call) traces are coarser but summarise how malware interacts with the run-time system, including file I/O and networking activities [608]. Another advantage of dynamic analysis is that it is independent of the malware format, e.g., binary, script, macro, or exploit, because all malware is executed and analysed in a similar fashion.

6.3.1.3 Fuzzing

Fuzzing is a method for discovering vulnerabilities, bugs and crashes in software by feeding randomised inputs to programs. Fuzzing tools [609] can also be used to trigger malware behaviours. Fuzzing can explore the input space, but it is limited due to code-coverage issues [594], especially for inputs that drive the program down complex branch conditions. In contrast, concolic execution (see 6.3.1.5 Concolic Execution) is good at finding complex inputs by formulating constraints, but is also expensive and slow. To take advantage of both approaches, a hybrid approach [610] called *hybrid fuzzing* can be used.

6.3.1.4 Symbolic Execution

Symbolic execution [594, 597, 611, 612, 613] has been used for vulnerability analysis of legitimate programs as well as malware analysis [595]. It treats variables and equations as symbols and formulas that can potentially express all possible program paths. A limitation of concrete execution (i.e., testing on particular inputs), including fuzzing, for malware analysis is that the program has to be executed end-to-end, one run at a time. Unlike concrete execution, symbolic execution can explore multiple branches simultaneously. To explore unseen code sections and unfold behaviours, symbolic execution generalises the input space to represent all possible inputs that could lead to points of interest.

6.3.1.5 Concolic Execution

While symbolic execution can traverse all paths in theory, it has major limitations [611], e.g., it may not converge quickly (if at all) when dealing with large symbol space and complex formulas and predicates. Concolic execution, which combines *CONC*rete and symb*OLIC* execution, can reduce the symbolic space but keep the general input space.

Offline Concolic Execution is a technique that uses concrete traces to drive symbolic execution; it is also known as a *Trace Based Executor* [596]. The execution trace obtained by concrete execution is used to generate the path formulas and constraints. The path formulas for the corresponding branch is negated and Satisfiability Modulo Theories (SMT) solvers are used to find a valid input that can satisfy the not-taken branches. Generated inputs are fed into the program and re-run from the beginning. This technique iteratively explores the feasible not-taken branches encountered during executions. It requires the repetitive execution of all the instructions from the beginning and knowledge of the input format.

Online Concolic Execution is a technique that generates constraints along with the concrete execution [597]. Whenever the concrete execution hits a branch, if both directions are feasible, execution is forked to work on both branches. Unlike the offline executor, this approach can explore multiple paths.

Hybrid Execution: This approach switches automatically between online and offline modes to avoid the drawbacks of non-hybrid approaches [598].

Concolic Execution can use whole-system emulators [597, 614] or dynamic binary instrumentation tools [598, 612]. Another approach is to interpret Intermediate Representation (IR) to imitate the effects of execution [595, 599]. This technique allows context-free concolic execution, which analyses any part of the binary at function and basic block levels.

Path Exploration is a systematical approach to examine program paths. Path explosion is also inevitable in concolic execution due to the nature of symbolic space. There are a variety of algorithms used to prioritise the directions of concolic execution, e.g., Depth-First Search (DFS) or distance computation [615]. Another approach is to prioritise the directions favouring newly explored code blocks or symbolic memory dependence [598]. Other popular techniques include path pruning, state merging [597, 616, 617], under-constrained symbolic execution [599] and fuzzing support [594, 596].

6.3.2 Analysis Environments

Malware analysis typically requires a dedicated environment to run the dynamic analysis tools [588]. The design choice of the environment determines the analysis methods that can be utilised and, therefore, the results and limitations of analysis. Creating an environment requires balancing the cost it takes to analyse a malware sample against the richness of the resulting report. In this context, cost is commonly measured in terms of time and manual human effort. For example, having an expert human analyst study a sample manually can produce a very in-depth and thorough report, but at great cost. Safety is a critical design consideration because of the concern that malware being executed and analysed in the environment can break out of its containment and cause damage to the analysis system and its connected network including the Internet (see 6.3.2.1 Safety and Live-Environment Requirements). An example is running a sample of a botnet malware that performs a DDoS attack, and thus if the analysis environment is not safe, it will contribute to that attack.

	Machine Emulator	Type 2 Hypervisor	Type 1 Hypervisor	Bare-metal machine
Architecture	Code-based architecture emulation	Runs in host OS, provides virtualisation service for hardware	Runs directly on system hardware	No virtualisation
Advantages	Easy to use, Fine-grained introspection, Powerful control over the system state	Easy to use, Fine-grained introspection, Powerful control over the system state	Medium transparency, Fine-grained introspection, Low overhead for hardware interaction	High transparency, No virtual environment artifacts
Disadvantages	Low transparency, Unreliability support of architecture semantics	Low transparency, Artifacts from para-virtualisation	Less control over the system state	Lack of fine-grained introspection, Scalability and cost issues, Slower to restore to clean state
Examples	Unicorn [618], QEMU [619], Bochs [620]	VirtualBox [621], KVM [622], VMware [623]	VMwareESX [624], Hyper-V [625], Xen [626]	NVMTrace [627], BareCloud [603]

Table 6.3: Comparison of Malware Analysis Environments

Table 6.3 highlights the advantages and disadvantages of common environments used for run-time (i.e., dynamic) analysis of malware. We can see that some architectures are easier to set up and give finer control over the malware's execution, but come at the cost of transparency (that is, they are easier for the malware to detect) compared to the others. For example, bare-metal systems are very hard for malware to detect, but because they have no instrumentation, the data that can be extracted are typically limited to network and disk I/O. By contrast, emulators like QEMU can record every executed instruction and freely inspect memory. However, QEMU also has errors that do not exist in real hardware, which can be exploited to detect its presence [628]. A very large percentage of modern malware detect emulated and virtualised environments and if they do, then they do not perform their malicious actions in order to avoid analysis.

6.3.2.1 Safety and Live-Environment Requirements

Clearly, *safety* is very important when designing a malware analysis environment because we cannot allow malware to cause unintended damage to the Internet (e.g., via mounting a denial-of-service attack from inside the analysis environment) and the analysis system and its connected network. Unfortunately, although pure static techniques, i.e., code analysis without program execution, are the safest, they also have severe limitations. In particular, malware authors know their code may be captured and analysed, and they employ code obfuscation techniques so that code analysis alone (i.e., without actually running the malware) will yield as little information as possible.

Malware typically requires communication with one or more C&C servers on the Internet, e.g., to receive commands and decrypt and execute its 'payload' (or the code that performs the intended malicious activities). This is just one example that highlights how the design of a live-environment is important for the malware to be *alive* and thus exhibit its intended functionality. Other examples of live-environment requirements include specific run-time libraries [629], real user activities on the infected machine [630], and network connectivity to malware update servers [631].

6.3.2.2 Virtualised Network Environments

Given the safety and live-environment requirements, most malware analysis environments are constructed using virtualisation technologies. Virtualisation enables operating systems to automatically and efficiently manage entire networks of nodes (e.g., hosts, switches), even within a single physical machine. In addition, containment policies can be applied on top of the virtual environments to balance the live-environment and safety requirements to 1) allow malware to interact with the Internet to provide the necessary realism, and 2) contain any malicious activities that would cause undesired harm or side-effects.

Example architectures [600] include: 1) the GQ system, which is designed based on multiple containment servers and a central gateway that connects them with the Internet allowing for filtering or redirection of the network traffic on a per-flow basis, and 2) the Potemkin system, which is a prototype *honeyfarm* that uses aggressive memory sharing and dynamically binds physical resources to external requests. Such architectures are used to not only monitor, but also replay network-level behaviours. Towards this end, we first need to reverse-engineer the C&C protocol used by malware. There are several approaches based on network level data (e.g., Roleplay [632], which uses bytestream alignment algorithms), or dynamic analysis of malware execution (e.g., Polyglot and dispatcher [633]), or a combination of the two.

6.3.3 Anti-Analysis and Evasion Techniques

Malware authors are well aware that security analysts use program analysis to identify malware behaviours. As a result, malware authors employ several techniques to make malware hard to analyse [588].

6.3.3.1 Evading the Analysis Methods

The source code of malware is often not available and, therefore, the first step of static analysis is to disassemble malware binary into assembly code. Malware authors can apply a range of anti-disassembly techniques (e.g., reusing a byte) to cause disassembly analysis tools to produce an incorrect code listing [588].

The most general and commonly used code obfuscation technique is *packing*, that is, compressing and encrypting part of the malware. Some trivially packed binaries can be unpacked with simple tools and analysed statically [634], but for most modern malware the packed code is unpacked only when it is needed during malware execution. Therefore, an unpacking tool needs to analyse malware execution and consider the trade-offs of robustness, performance, and transparency. For example, unpackers based on virtual machine introspection (VMI) [601] are more transparent and robust but also slower. By contrast, unpackers built on dynamic binary instrumentation (DBI) [605] are faster, but also easier to detect because the DBI code runs at the same privilege level as the malware.

Many techniques aim at obfuscating the intended control-flows of a malware, e.g., by adding more basic blocks and edges to its control-flow graph [588, 635, 636]. A countermeasure is to analyze malware samples by their dynamic features (i.e., what a malware does). The reason is that static analysis can be made impossible via advanced obfuscation using opaque constants [637], which allows the attacker to hide what values will be loaded into registers during runtime. This in turn makes it very hard for static malware analysis to extract the control-flow graph and variables from the binary. A more effective approach is to combine static and dynamic analysis. For example, such an approach has been shown to be able to disassemble the highly obfuscated binary code [638].

A less common but much more potent obfuscation technique is code emulation. Borrowing techniques originally designed to provide software copyright protection [639], malware authors convert native malware binaries into bytecode programs using a randomly generated instruction set, paired with a native binary emulator that interprets the instruction set. That is, with this approach, the malware 'binary' is the emulator, and the original malware code becomes 'data' used by the emulator program. Note that, for the same original malware, the malware author can turn it into many instances of emulated malware instances, each with its own random bytecode instruction set and a corresponding emulator binary. It is extremely hard to analyse emulated malware. Firstly, static analysis of the emulator code yields no information about the specific malware behaviours because the emulator processes all possible programs in the bytecode instruction set. Static analysis of the malware bytecode entails first understanding the instruction set format (e.g., by static analysing the emulator first), and developing tools for the instruction set; but this process needs to be repeated for every instance of emulated malware. Secondly, standard dynamic analysis is not directly useful because it observes the run-time instructions and behaviours of an emulator and not of the malware.

A specialised dynamic analysis approach is needed to analyse emulated malware [604].

The main idea is to execute the malware emulator and record the entire instruction traces. Applying dynamic dataflow and taint analysis techniques to these traces, we then identify data regions containing the bytecode, syntactic information showing how bytecodes are parsed into opcodes and operands, and semantic information about control transfer instructions. The output of this approach is data structures, such as a control-flow graph (CFG) of the malware, which provides the foundation for subsequent malware analysis.

Malware often uses fingerprinting techniques to detect the presence of an analysis environment and evade dynamic analysis (e.g., it stops executing the intended malware code). More generally, malware behaviours can be 'trigger-based' where a trigger is a run-time condition that must be true. Examples of conditions include the correct date and time, the presence of certain files or directories, an established connection to the Internet, the absence of a specific mutex object etc. If a condition is not true, the malware does not execute the intended malicious logic. When using standard dynamic analysis, the test inputs are not guaranteed to trigger some of these conditions and, as a result, the corresponding malware behaviours may be missed. To uncover trigger-based behaviours a multi-path analysis approach [602] explores multiple execution paths of a malware. The analyser monitors how the malware code uses condition-like inputs to make control-flow decisions. For each decision point, the analyser makes a snapshot of the current malware execution state and allows the malware to execute the correct malware path for the given input value; for example, the input value suggests that the triggering condition is not met and the malware path does not include the intended malicious logic. The analyser then comes back to the snapshot and rewrites the input value so that the other branch is taken; for example, now the triggering condition is rewritten to be true, and the malware branch is the intended malicious logic.

6.3.3.2 Identifying the Analysis Environments

Malware often uses system and network artifacts that suggest that it is running in an analysis environment rather than a real, infected system [588]. These artifacts are primarily categorised into four classes: virtualisation, environment, process introspection, and user. In virtualisation fingerprinting, evasive malware tries to detect that it is running in a virtualised environment. For example, it can use red pill testing [640], which entails executing specific CPU instruction sequences that cause overhead, unique timing skews, and discrepancies when compared with executions on a bare-metal (i.e., non-virtualised) system. Regarding environment artifacts, virtual machines and emulators have unique hardware and software parameters including device models, registry values, and processes. In process introspection, malware can check for the presence of specific programs on operating systems, including monitoring tools provided by anti-virus companies and virtual machine vendors. Lastly, user artifacts include specific applications such a web browser (or lack thereof), web browsing history, recently used files, interactive user prompts, mouse and keyboard activities etc. These are signals for whether a real human uses the environment for meaningful tasks.

An analysis environment is not transparent if it can be detected by malware. There are mitigation techniques, some address specific types of evasion while others more broadly increase transparency. Binary modifications can be performed by dynamically removing or rewriting instructions to prevent detection [641], and environmental artifacts can be hidden from malware by hooking operating system functions [642]. Path-exploration approaches [602, 643] force malware execution down multiple conditional branches to bypass evasion. Hypervisor-based approaches [601, 644] use introspection tools with greater privilege than malware so that they can be hidden from malware and provide the expected answers to the malware when it

checks the system and network artifacts. In order to provide the greatest level of transparency, several approaches [603, 627] perform malware analysis on real machines to avoid introducing artifacts.

6.4 MALWARE DETECTION

[588, c11, c14-16, c18] [645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655]

6.4.1 Identifying the Presence of Malware

The process of locating a malicious program residing within a host involves identifying clues that are indicative of the malware's presence on a computer system. We call these clues 'indicator of compromise', and they are the 'features' or 'artifacts' of malware.

6.4.1.1 Finding Malware in a Haystack

In order to identify malware, we must first have an understanding of how malware is distributed to their victims' hosts. Malware is commonly distributed via an Internet download [656]. A vulnerable Internet-facing program running on a computer can be exploited to download malware onto the computer. A user on the computer can be socially engineered to open an email attachment or visit a web page, both may lead to an exploit and malware download.

Whilst being downloaded onto a host, the malware's contents can be seen in the payload section of the network traffic (i.e., network packet) [588]. As a defense, an Antivirus (AV) solution, or Intrusion Detection System (IDS), can analyse each network packet transported to an end-host for known malicious content, and block (prevent) the download. On the other hand, traffic content encrypted as HTTPS is widely and increasingly adopted by websites. Using domain reputation systems [657], network traffic coming from domains and IP addresses known to be associated with malicious activities can be automatically blocked without analysing the traffic's payload.

After being installed on a computer, malware can reside within the host's filesystem or memory (or both). At this point, the malware can sleep (where the executable does nothing to the system) until a later point in time [658] as specified by the malware author. An AV or IDS can periodically scan the host's filesystem and memory for known malicious programs [588]. As a first layer of defence, malware detectors can analyse static features that suggest malicious executable contents. These include characteristics of instructions, control-flow graphs, call graphs, byte-value patterns [659] etc.

If malware is not detected during its distribution state, i.e., a detection system misses its presence in the payloads of network traffic or the filesystem and memory of the end-host, it can still be detected when it executes and, for example, begins contacting its command-and-control (C&C) server and performing malicious actions over the Internet or on the victim computer system. An AV or IDS on the network perimeter continuously monitors network packets travelling out of an end-host. If the AV or IDS sees that the host is contacting known malicious domain names or IP addresses it can surmise that the host has been infected by malware. In addition, an AV or IDS on the end-host can look for behaviour patterns that are associated with known malware activities, such as system or API calls that reveal the specific files read or written.

Evasion and Countermeasures Since Antivirus and IDS solutions can generate signatures for malware executables, malware authors often morph the contents of their malware. They can change the contents of the executables while generating identically functional copies of their malware (i.e., the malware will perform the same dynamic behaviours when executed). Since its static contents have been changed, the malware can evade an AV or IDS that uses these static features. On the other hand, the malware can still be detected by an AV or IDS that uses the dynamic features (i.e., what the malware does).

Heuristics, e.g., signatures of a packing tool, or high entropy due to encryption, can be used to detect and block contents that suggest the presence of packed malware, but this may lead to false alarms because packing can also be used by benign software and services, such as video games, to protect proprietary information. The most reliable way to detect packed malware is to simply monitor its run-time behaviours because the packed code will be unpacked and executed, and the corresponding malicious behaviours can then be identified [645].

In addition to changing the malware executable, an attacker can also change the contents of its malicious network traffic by using *polymorphism* to modify payloads so that the same attacks look different across multiple traffic captures. However, classic polymorphic malware techniques [660] make the payloads look so different that even a naive IDS can easily differentiate them from benign payloads. On the other hand, with polymorphic malware blending attacks [646] malicious payloads can be made to look statistically similar to benign payloads.

Malware authors often implement updating routines, similar to updates for operating systems and applications such as web browsers and office tools. This allows malware authors the flexibility to make changes to the malware to not only include new malicious activities but also evade detection by AVs and IDS that have started using patterns of the old malware and its old behaviours.

6.4.2 Detection of Malware Attacks

We have discussed ways to identify static and behaviour patterns of malware, which can then be used to detect instances of the same, or similar malware. Although many popular variants of malware families have existed at one time or another (e.g., Zeus [661, 662], Spyeye [663, 664], Mirai [665]), there will always be new malware families that cannot be detected by malware detection models (such as AV signatures). Therefore, we need to go beyond identifying specific malware instances: we need to detect malicious activities in general.

6.4.2.1 Host-based and Network-Based Monitoring

The most general approach to detect malicious activities is anomaly detection [647, 648, 666]. An anomaly in system or network behaviour is an activity that deviates from normal (or seen) behaviour. Anomaly detection can identify both old and new attacks. It is important to note that an *anomalous* behaviour is not the same as a *malicious* behaviour. Anomalous behaviours describe behaviours that deviate from the norm, and of course it is possible to have abnormal benign activities occurring on a system or network.

On the other hand, a more efficient and arguably more accurate approach to detect an old attack is to find the patterns or signatures of the known attack activities [588]. This is often called the misuse detection approach. Examples of signatures include: unauthorised write to system files (e.g., Windows Registry), connection to known botnet C&C servers, etc.

Two different, but complementary approaches to deploy attack detection systems are: 1) host-based monitoring of system activities, and 2) network-based monitoring of traffic. Host-based monitoring systems monitor activities that take place in a host, to determine if the host is compromised. These systems typically collect and monitor activities related to the file system, processes, and system calls [588, 649]. Network-based monitoring systems analyse activities that are network-wide, e.g., temporal characteristics of access patterns of network traffic flows, the domain names the network hosts reach out to, the characteristics of the network packet payloads that cross the network perimeter, etc. [588, 650].

Let us look at several examples of malicious activities and the corresponding detection approaches. The first-generation spam detection systems focused on analysing the email contents to distinguish legitimate messages from spam. Latter systems included network-level behaviours indicative of spam traffic [667], e.g., spikes in email traffic volumes due to large amount of spam messages being sent.

For DDoS detection, the main idea is to analyse the statistical properties of traffic, e.g., the number of requests within a short time window sent to a network server. Once a host is identified to be sending such traffic, it is considered to be participating in a DDoS attack and its traffic is blocked. Attackers have evolved their techniques to DDoS attacks, in particular, by employing multiple compromised hosts, or bots, to send traffic in a synchronised manner, e.g., by using DDoS-as-a-service malware kits [668]. That is, each bot no longer needs to send a large amount of traffic. Correspondingly, DDoS detection involves correlating hosts that send very similar traffic to the victim at the same time.

For ransomware detection, the main approaches include monitoring host activities involved in encryption. If there is a process making a large number of *significant* modifications to a large number of files, this is indicative of a ransomware attack [669]. The '*significant*' modifications reflect the fact that encrypting a file will result in its contents changing drastically from its original contents.

Host-based and network-based monitoring approaches can be beneficially combined. For example, if we see contents from various sensitive files on our system (e.g., financial records, password-related files, etc.) being transmitted in network traffic, it is indicative that data are being exfiltrated (without the knowledge and consent of the user) to an attacker's server. We can then apply host-based analysis tools to further determine the attack provenance and effects on a victim host [670].

Since many malicious activities are carried out by botnets, it is important to include botnet detection methods. By definition, bots of the same botnet are controlled by the same attacker and perform coordinated malicious activities [651, 671]. Therefore, a general approach to botnet detection is to look for synchronised activities both in C&C like traffic and malicious traffic (e.g., scan, spam, DDoS, etc.) across the hosts of a network.

6.4.2.2 Machine Learning-Based Security Analytics

Since the late 1990s, machine learning (ML) has been applied to automate the process of building models for detecting malware and attacks. The benefit of machine learning is its ability to generalise over a population of samples, given various features (descriptions) of those samples. For example, after providing an ML algorithm samples of different malware families for 'training', the resultant model is able to classify new, unseen malware as belonging to one of those families [652].

Both static and dynamic features of malware and attacks can be employed by ML-based detection models. Examples of static features include: instructions, control-flow graphs, call graphs, etc. Examples of dynamic features include: system call sequences and other statistics (e.g., frequency and existence of system calls), system call parameters, data-flow graphs [672], network payload features, etc.

An example of success stories in applying machine learning to detect malware and attacks is botnet detection [673]. ML techniques were developed to efficiently classify domain names as ones produced by domain generation algorithm (DGA), C&C domains, or legitimate domains using features extracted from DNS traffic. ML techniques have also been developed to identify C&C servers as well as bots in an enterprise network based on features derived from network traffic data [651].

A major obstacle in applying (classical) machine learning to security is that we must select or even engineer features that are useful in classifying benign and malicious activities. Feature engineering is very knowledge- and labour- intensive and is the bottleneck in applying ML to any problem domain. Deep learning has shown some promise in learning from a large amount of data without much feature engineering, and already has great success in applications such as image classification [674]. However, unlike many classical ML models (such as decision trees and inductive rules) that are human-readable, and hence reviewable by security analysts before making deployment decisions, deep learning outputs blackbox models that are not readable and not easily explainable. It is often not possible to understand what features are being used (and how) to arrive at a classification decision. That is, with deep learning, security analysts can no longer check if the output even makes sense from the point-of-view of domain or expert knowledge.

6.4.2.3 Evasion, Countermeasures, and Limitations

Attackers are well aware of the detection methods that have been developed, and they are employing evasion techniques to make their attacks hard to detect. For example, they can limit the volume and intensity of attack activities to stay below the detection threshold, and they can mimic legitimate user behaviours such as sending stolen data (a small amount at a time) to a 'drop site' only when a user is also browsing the Internet. Every misuse or anomaly detection model is potentially evadable.

It should also come as no surprise that no sooner had researchers begun using ML than attackers started to find ways to defeat the ML-based detection models.

One of the most famous attacks is the Mimicry attack on detection models based on system call data [653]. The idea is simple: the goal is to morph malicious features to look exactly the same as the benign features, so that the detection models will mistakenly classify the attack as benign. The Mimicry attack inserts system calls that are inconsequential to the intended malicious actions so that the resultant sequences, while containing system calls for

malicious activities, are still legitimate because such sequences exist in benign programs. A related attack is polymorphic blending [646] that can be used to evade ML models based on network payload statistics (e.g., the frequency distribution of n-grams in payload data to a network service). An attack payload can be encoded and padded with additional n-grams so that it matches the statistics of benign payloads. Targeted noise injection [654] is an attack designed to trick a machine-learning algorithm, while training a detection model, to focus on features not belonging to malicious activities at all. This attack exploits a fundamental weakness of machine learning: garbage in, garbage out. That is, if you give a machine-learning algorithm bad data, then it will learn to classify data 'badly'. For example, an attacker can insert various no-op features into the attack payload data, which will statistically produce a strong signal for the ML algorithm to select them as 'the important, distinguishing features'. As long as such features exist, and as they are under the attacker's control, any ML algorithm can be misled to learn an incorrect detection model. Noise injection is also known as 'data poisoning' in the machine learning community.

We can make attacks on ML harder to succeed. For example, one approach is to squeeze features [675] so that the feature set is not as obvious to an attacker, and the attacker has a smaller target to hit when creating adversarial samples. Another approach is to train separating classes, which distance the decision boundary between classes [676]. This makes it more difficult for an attacker to simply make small changes to features to 'jump' across decision boundaries and cause the model to misclassify the sample. Another interesting approach is to have an ML model forget samples it has learned over time, so that an attacker has to continuously poison every dataset [677].

A more general approach is to employ a combination of different ML-based detection models so that defeating all of them simultaneously is very challenging. For example, we can model multiple feature sets simultaneously through ensemble learning, i.e., using multiple classifiers trained on different feature sets to classify a sample rather than relying on singular classifier and feature set. This would force an attacker to have to create attacks that can evade each and every classifier and feature set [655].

As discussed earlier, deep learning algorithms produce models that cannot be easily examined. But if we do not understand how a detection model really works, we cannot foresee how attackers can attempt to defeat it and how we can improve its robustness. That is, a model that seemingly performs very well on data seen thus far can, in fact, be very easily defeated in the future - we just have no way of knowing. For example, in image recognition it turned out that some deep learning models focused on high-frequency image signals (that are not visible to the human eye) rather than the structural and contextual information of an image (which is more relevant for identifying an object) and, as a result, a small change in the high-frequency data is sufficient to cause a mis-classification by these models, while to the human eye the image has not changed at all [678].

There are promising approaches to improve the 'explainability' of deep learning models. For example, an attention model [679] can highlight locations within an image to show which portions it is focusing on when classifying the image. Another example is LEMNA [680], which generates a small set of interpretable features from an input sample to explain how the sample is classified, essentially approximating a local area of the complex deep learning decision boundary using a simpler interpretable model.

In both the machine learning and security communities, adversarial machine learning [681] is and will continue to be a very important and active research area. In general, attacks on machine learning can be categorised as data poisoning (i.e., injecting malicious noise into

training data) and evasion (i.e., morphing the input to cause mis-classification). What we have discussed above are just examples of evasion and poisoning attacks on ML models for security analytics. These attacks have motivated the development of new machine-learning paradigms that are more robust against adversarial manipulations, and we have discussed here examples of promising approaches.

In general, attack detection is a very challenging problem. A misuse detection method which is based on patterns of known attacks is usually not effective against new attacks or even new variants of old attacks. An anomaly detection method which is based on a normal profile can produce many false alarms because it is often impossible to include all legitimate behaviours in a normal profile. While machine learning can be used to automatically produce detection models, potential 'concept drift' can render the detection models less effective over time [682]. That is, most machine-learning algorithms assume that the training data and the testing data have the same statistical properties, whereas in reality, user behaviours and network and system configurations can change after a detection model is deployed.

6.5 MALWARE RESPONSE

[20, 683, 684, 685, 686, 687]

If we have an infected host in front of us, we can remove the malware, and recover the data and services from secure backups. At the local network access point, we can update corresponding Firewall and Network intrusion detection system rules, to prevent and detect future attacks. It is unfeasible to execute these remediation strategies if the infected machines cannot be accessed directly (e.g., they are in private residences), and if the scale of infection is large. In these cases, we can attempt to take down malware command-and-control (C&C) infrastructure instead [20, 683], typically at the internet service provider (ISP) or the top-level domain (TLD) level. Takedowns aim to disrupt the malware communication channel, even if the hosts remain infected. Last but not least, we can perform attack attribution using multiple sources of data to identify the actors behind the attack.

6.5.1 Disruption of Malware Operations

There are several types of takedowns to disrupt malware operations. If the malware uses domain names to look up and to communicate with centralised C&C servers, we perform take-down of C&C domains by 'sinkholing' the domains, i.e., making the C&C domains resolve to the defender's servers so that botnet traffic is 'trapped' (that is, redirected) to these servers [683]. If the malware uses peer-to-peer (P2P) protocol as a decentralised C&C mechanism, we can partition the P2P botnet into isolated sub-networks, create a sinkholing node, or poison the communication channel by issuing commands to stop the malicious activities [20]. However, it should be borne in mind that, in most territories active defence or intelligence gathering, such as hack-backs, access to or modification of servers, DNS, or networks, is unlawful without appropriate legal authority.

6.5.1.1 Evasion and Countermeasures

Malware often utilises agility provided by DNS fast-flux network and Domain-name Generation Algorithms (DGAs) to evade the takedown. A DNS fast-flux network points the C&C domain names to a large pool of compromised machines, and the resolution changes rapidly [688]. DGAs make use of an algorithm to automatically generate candidate C&C domains, usually based on some random seed. Among the algorithm-generated domains, the botmaster can pick a few to register (e.g., on a daily basis) and make them resolve to the C&C servers. What makes the matter worse are the so-called bullet-proof hosting (BPH) services, which are resilient against takedowns because they ignore abuse complaints and takedown requests [684].

We can detect the agile usage of C&C mechanisms. As the botmaster has little control of the IP address diversity and down-time for compromised machines in a fast-flux network, we can use these features to detect fast-flux [689]. We can also identify DGA domains by mining NX-Domains traffic using infected hosts features and domain name characteristic features [673], or reverse-engineering the malware to recover the algorithm. To counter bullet-proof hosting, we need to put legal, political and economic pressures on hosting providers. For example, the FBI's operation ghost click issued a court order for the takedown of DNSChanger [690, 691].

Malware has also become increasingly resilient by including contingency plans. A centralised botnet can have P2P as a fallback mechanism in case the DNS C&C fails. Likewise, a P2P botnet can use DNS C&C as a contingency plan. A takedown is effective only if all the C&C channels are removed from the malware. Otherwise, the malware can bootstrap the C&C communication again using the remaining channels. If we hastily conduct botnet takedowns without thoroughly enumerating and verifying all the possible C&C channels, we can fail to actually disrupt the malware operations and risk collateral damage to benign machines. For example, the Kelihos takedown [692] did not account for the backup P2P channel, and the `3322.org` takedown disabled the dynamic DNS service for many benign users.

We need to have a complete view of the C&C domains and other channels that are likely to be used by a botnet, by using multiple sources of intelligence including domain reputation, malware query association and malware interrogation [683]. We start from a seed set of C&C domains used by a botnet. Then, we use passive DNS data to retrieve related historical IP addresses associated with the seed set. We remove sinkholing, parking, and cloud hosting provider IP addresses from them to mitigate the collateral damage from the takedowns. The resulting IPs can also give us related historical domains that have resolved to them. After following these steps, we have an extended set of domains that are likely to be used by the botnet. This set captures agile and evasive C&C behaviours such as fast-flux networks. Within the extended set, we combine 1) low reputation domains, 2) domains related to malware, and 3) other domains obtained by interrogating the related malware. Malware interrogation simulates situations where the default C&C communication mechanism fails through blocking DNS resolution and TCP connection [687]. By doing so, we can force the malware to reveal the backup C&C plans, e.g., DGA or P2P. After enumerating the C&C infrastructure, we can disable the complete list of domains to take the botnet down.

6.5.2 Attribution

Ideally, law enforcement wants to identify the actual criminal behind the attacks. Identifying the virtual attacker is an important first step toward this goal. An attacker may have consistent coding styles, reuse the same resources or infrastructures, or use similar C&C practices.

From the malware data, we can compare its 'characteristics' with those of known historical adversaries, e.g., coding styles, server configurations, etc. [685]. At the source code level, we can use features that reflect programming styles and code quality. For instance, linguistic features, formatting style, bugs and vulnerabilities, structured features such as execution path, abstract syntax tree (AST), Control Flow Graph (CFG), and program dependence graph (PDG) can be used. Other features extracted from the binary file can also indicate authorship, e.g., the sequence of instructions and register flow graph.

From the enumerated attack infrastructure, we can associate the expanded domain name set with previously known adversaries. For instance, unknown TDSS/TDL4 botnet ad-fraud C&C domains share the same IP infrastructure with known domains, and they are registered by the same set of email addresses and name servers. This allows us to attribute unknown domains to known TDSS/TDL4 actors [686].

6.5.2.1 Evasion and Countermeasures

Many malware authors reuse different kits for the convenience offered by the business model of the underground economy. Common for-sale kits allow malware authors to easily customise their own malware. They can also evade attribution by intentionally planting 'false flags' in malware.

Domain registration information, WHOIS, is a strong signal for attack attribution. The same attacker often uses a fake name, address and company information following a pattern. However, WHOIS privacy protection has become ubiquitous and is even offered for free for the first year when a user purchases a domain name. This removes the registration information that could be used for attack attribution.

We need to combine multiple, different streams of data for the analysis. For instance, malware interrogation helps recover more C&C domains used by the fallback mechanism, which offers more opportunity for attribution [687, 693].

CONCLUSION

Attackers use malware to carry out malicious activities on their behalf. Malware can reside in any layer of the system stack, and can be a program by itself or embedded in another application or document. Modern malware comes with a support infrastructure for coordinated attacks and automated updates, and can operate low-and-slow and cover its tracks to avoid detection and attribution. While malware can cause wide-spread infection and harm on the Internet, it can also be customised for attacks targeting a specific organisation. Malware analysis is an important step in understanding malicious behaviours and properly updating our attack prevention and detection systems. Malware employs a wide range of evasion techniques, which include detecting the analysis environment, obfuscating malicious code, using trigger-conditions to execute, and applying polymorphism to attack payloads, etc. Accordingly, we need to make analysis environments transparent to malware, continue to develop

specialised program analysis algorithms and machine-learning based detection techniques, and apply a combination of these approaches. Response to malware attacks goes beyond detection and mitigation, and can include take-down and attribution, but the challenge is enumerating the entire malware infrastructure, and correlating multiple pieces of evidence to avoid false flags planted by the attackers.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Sections	Cites
6.1 A taxonomy of Malware	[589]:c6
6.2 Malicious Activities by Malware	[589]:c6, [588]:c11-12
6.3 Malware Analysis	
6.3.1 Analysis Techniques	[588]:c1-10
6.3.1.1 Static Analysis	[588]:c4-7
6.3.1.2 Dynamic analysis	[588]:c8-10
6.3.1.3 Fuzzing	[594, 595]
6.3.1.5 Concolic Execution	[596, 597, 598, 599]
6.3.2 Analysis Environments	[588]:c2
6.3.2.1 Safety and Live-Environment Requirements	
6.3.2.2 Virtualised Network Environments	[588]:c2, [600]
6.3.3.2 Identifying the Analysis Environments	[588]:c15-18, [601, 602, 603]
6.3.3 Anti-Analysis and Evasion Techniques	[588]:c15-16, [602, 604, 605]
6.4 Malware Detection	
6.4.1 Identifying the Presence of Malware	
6.4.1.1 Finding Malware in a Haystack	[588]:c11,c14
6.4.1.1 Evasion and Countermeasures	[588]:c15-16,c18, [645, 646]
6.4.2 Detection of Malware Attacks	
6.4.2.1 Host-based and Network-Based Monitoring	[588]:c11,c14, [647, 648, 649, 650, 651]
6.4.2.2 Machine Learning-Based Security Analytics	[651, 652]
6.4.2.3 Evasion, Countermeasures, and Limitations	[653, 654, 655]
6.5 Malware Response	
6.5.1 Disruption of Malware Operations	[20, 683]
6.5.1.1 Evasion and Countermeasures	[684]
6.5.2 Attribution	[685, 686]
6.5.2.1 Evasion and Countermeasures	[687]

Chapter 7
Adversarial Behaviours

Gianluca Stringhini | Boston University

INTRODUCTION

The technological advancements witnessed by our society in recent decades have brought improvements in our quality of life, but they have also created a number of opportunities for attackers to cause harm. Before the Internet revolution, most crime and malicious activity generally required a victim and a perpetrator to come into physical contact, and this limited the reach that malicious parties had. Technology has removed the need for physical contact to perform many types of crime, and now attackers can reach victims anywhere in the world, as long as they are connected to the Internet. This has revolutionised the characteristics of crime and warfare, allowing operations that would not have been possible before.

In this document, we provide an overview of the malicious operations that are happening on the Internet today. We first provide a taxonomy of malicious activities based on the attacker's motivations and capabilities, and then move on to the technological and human elements that adversaries require to run a successful operation. We then discuss a number of frameworks that have been proposed to model malicious operations. Since adversarial behaviours are not a purely technical topic, we draw from research in a number of fields (computer science, criminology, war studies). While doing this, we discuss how these frameworks can be used by researchers and practitioners to develop effective mitigations against malicious online operations.

7.1 A CHARACTERISATION OF ADVERSARIES

[694][695][696, 697][17, 698, 699][700][701, 702]

In this section, we present a characterisation of adversaries who perform malicious actions. This characterisation is based on their motivation (e.g., financial, political etc.). Although alternative characterisations and taxonomies exist (e.g., from the field of psychology [703]), we feel that the one presented here works best to illustrate known attackers' capabilities and the tools that are needed to set up a successful malicious operation, such as a financial malware enterprise. This characterisation also follows the evolution that cybercrime has followed in recent decades, from an ad-hoc operation carried out by a single offender to a commoditised ecosystem where various specialised actors operate together in an organised fashion [704, 705]. The characterisation presented in this section is driven by case studies and prominent examples covered in the research literature, and as such is not meant to be complete. For example, we do not focus on accidental offenders (e.g., inadvertent insider threats), or on criminal operations for which rigorous academic literature is lacking (e.g., attacks on financial institutions or supply chain attacks). However, we believe that the set of crimes and malicious activities presented is comprehensive enough to draw a representative picture of the adversarial behaviours that are occurring in the wild at the time of writing. We begin by defining two types of cyber offences as they have been defined in the literature, cyber-enabled and cyber-dependent crimes, and we continue by presenting different types of malicious activities that have been covered by researchers.

Cyber-enabled and cyber-dependent crimes

One of the main effects that the Internet has had on malicious activity has been to increase the reach of existing crimes, in terms of the ease of reaching victims, effectively removing the need for physical proximity between the victim and the offender. In the literature, these crimes are often referred to as *cyber-enabled* [694].

According to Clough [94], criminals have five main incentives to move their operations online:

1. Using the Internet, it is easier to find and contact victims. Email lists are sold on underground markets [706], while online social networks have search functionalities embedded in them, allowing criminals to easily identify potential victims [707, 708].
2. By using the Internet, criminal operations can be run more cheaply. Sending emails is free, while scammers previously had to pay postage to reach their victims. This also allows criminals to increase the scale of their operations to sizes that were previously unthinkable.
3. Compared to their physical counterparts, the Internet allows crimes to be performed faster. For example, emails can reach victims in a matter of seconds, without having to wait for physical letters to be delivered.
4. Using the Internet, it is easier to operate across international boundaries, reaching victims located in other countries. In this setting, often the only limitation is language, with criminals only targeting victims who speak a language that they are familiar with (e.g., people in English-speaking countries) [709].
5. By operating over the Internet, it is more difficult for criminals to get caught. This is mainly due to the transnational nature of cybercrime, and the fact that the problem of harmonising the appropriate laws of different countries is far from being solved [710]. In addition, research shows that online crime is often under reported, both because victims do not know whom to report it to (given that the offender might be located in another country), as well as the fact that they believe that they are unlikely to get their money back [711].

Cyber-dependent crimes, on the other hand, are crimes that can only be committed with the use of computers or technology devices [694]. Although the final goal of this type of crime often has parallels in the physical world (e.g., extortion, identity theft, financial fraud), the Internet and technology generally enable criminals to give a new shape to these crimes, making them large-scale organised endeavours able to reach hundreds of thousands, if not millions, of victims.

In the rest of this section we analyse a number of cyber-enabled and cyber-dependent criminal schemes in detail.

Interpersonal offenders

The first category that we are going to analyse is that of *interpersonal crimes*. These crimes include targeted violence and harassment, directed at either close connections (e.g., family members) or strangers. While these crimes have always existed, the Internet has made the reach of harassers and criminals much longer, effectively removing the need for physical contact for the offence to be committed. As such, these crimes fall into the cyber-enabled category. In the rest of this section, we provide an overview of these adversarial behaviours.

Cyberbullying. Willard [695] defines cyberbullying as 'sending or posting harmful material or engaging in other forms of social aggression using the Internet or other digital technologies'. While not always illegal[1], cyberbullying often occupies a grey area between what is considered a harmful act and a criminal offence [712]. This practice has become a serious problem for young people, who are often targeted by their peers not only in real life, but also on online platforms [713]. While the practice of bullying is nothing new, the Internet has changed the dynamics of these harassment practices significantly. What used to be a harmful practice limited to school hours now can be perpetrated at any time, effectively exposing victims to non-stop harassment [714].

One aspect that makes cyberbullying different from traditional, physical harassment is that people online can be anonymous, and do not have their name or face attached to the abusive activity that they are carrying out [715, 716]. Researchers found that interacting with people online creates a *disinhibition* effect wherby personal traits are accentuated (i.e., negative people become meaner and positive people become nicer) [717]. This disinhibition effect can have the effect of making some people more likely to engage in abusive activity than they would do in the offline world [716]. Another aspect that contributes to disinhibition is the fact that online content distributed on certain platforms (e.g., snapchat, 4chan) is ephemeral, in the sense that it is deleted after a certain period of time [718]. As such, harassers feel that their actions have no adverse consequences since there will be no hard evidence of it in the future.

Doxing. Another type of online harassment is the practice of *doxing*, an attack where the victim's private information is publicly released online [719]. This operation is usually part of a larger harassment campaign, where the release of sensitive information is used as a way of embarrassing the victim or facilitating further harassment, even in the physical world (for example, by releasing information at the workplace or the home address of the victim).

The practice of doxing has become increasingly popular in recent years as a way of polarising online discussion and silencing people. A prominent example is the `#GamerGate` controversy, where women activists were often attacked and had their personal information posted online [720]. Doxing has been a primary vehicle for coordinated hate attacks run by polarised online communities such as 4chan's Politically Incorrect board (/pol/) [718]. As part of these attacks, anonymous users post information about their targets online (e.g., social media pages, phone numbers, physical addresses), and then invite other people to carry out loosely coordinated attacks (called *raids*) against those people. These attacks usually consist of hate speech and other abusive language.

While prominent in the online harassment space, the practice of doxing is also used by other offenders. For example, it is one of the techniques used by hacktivist groups such

[1]While there is no definition of cyberbullying in UK law, some forms of it can be prosecuted under the Protection from Harassment Act 1997.

as Anonymous to put their targets on notice. We will discuss the other techniques used by hacktivists, together with their motivations, later in this section.

Cyberstalking. Another harmful activity that has been facilitated by the Internet is stalking. Cyberstalking is the practice of using electronic means to stalk another person [721, 722]. Broadly speaking, we can identify two types of cyberstalkers: those who use the information that they find online to help them stalk their victim in real life (e.g., monitoring social media to know their whereabouts), and those who use the means offered by online services to stalk their victim purely online. Further, the stalkers who operate online are divided into those who act purely passively, without any interaction with the victim, and those who perform interactions, for example, by sending their messages on a social network platform [723]. To counter cyberstalking, legislation has recently been introduced in many countries, including the 2012 Protections of Freedoms act in the UK and the 2000 Violence Against Women Act in the US.

Sextortion. An emerging crime that has risen to relevance is *sextortion*, where a criminal lures victims to perform sexual acts in front of a camera (e.g., a webcam in a chatroom), records those acts, and later asks for a monetary payment in order not to release the footage [724]. Sextortion is becoming such a relevant threat that crime prevention agencies such as the National Crime Agency (NCA) in the UK are launching dedicated awareness campaigns against it.[2]

Child predation. Another crime that is facilitated by the Internet is child predation [725]. Online services are a fertile ground for criminals to find victims, whether on chats, online social networks, or online gaming platforms. The offender will then groom their victims to either perform physical or online abuse [725]. Compared to the corresponding offline offence, online sexual predation has two main differences: first, the victim and the perpetrator almost never know each other in real life. Second, the victim demographics are more skewed towards adolescents than young children, because the age at which kids start going online is slightly higher [726]. Offenders use a range of tactics, including pretending to be young people and children in order to groom their victims [727] and research has shown the potential vulnerability of children of all ages to such online identity deception [728].

Other offenders do not interact with children directly, but download and share child pornography on the Internet. In such cases hands-on abusers often know their victims and disseminate child abuse material to these “users” of such material. This has been facilitated by peer-to-peer sharing platforms [729, 730] as well as anonymising technologies such as Tor [731]. The challenges of identifying originators of new child abuse material (and the deceptive tactics used by offenders, e.g., specialised vocabulary for filenames to thwart investigations) in such peer-to-peer networks have also been studied [730].

[2] http://www.nationalcrimeagency.gov.uk/crime-threats/kidnap-and-extortion/sextortion

Cyber-enabled organized criminals

In this section, we focus on cyber-enabled crimes that are carried out by career criminals. In particular, we provide two prominent examples of cyber-enabled crimes that have received significant attention by the research community: *advance fee fraud* and *drug dealing*. These crimes are not usually carried out by single offenders, but rather by multiple criminals who act together in small organisations [732] or in actual structured criminal organisations [733]. We acknowledge that other crimes exist that have seen increased reach because of technology. However, these crimes have yet to be studied in depth by the research community and, therefore, we decided to focus on the one which the research community has a better understanding of.

Advance fee fraud. In this type of scam, the victim is promised a reward (financial or otherwise), but in order to obtain it has to first pay a small fee to the fraudster. After the payment takes place, the victim often does not hear from the scammer again, while sometimes the relationship lasts for long periods of time and the victim is repeatedly defrauded of large sums of money [734].

The archetypal example of advance fee fraud comprises so-called 419 scams [696]. Named after the section of the Nigerian Criminal Code dealing with fraud, these scams became popular in the 1980s, when victims would receive physical letters from an alleged Nigerian prince, looking to transfer large amounts of money outside of the country. To initiate the process, the victim is required to transfer a small amount of money to the fraudster (e.g., to cover legal fees). As it can be imagined, the Internet allowed this type of fraud to flourish, by enabling criminals to instantly reach a large number of potential victims.

Another example of advanced fee fraud is consumer fraud perpetrated on classifieds websites such as Craigslist [735]. As part of this fraud, victims respond to a classified advertisement for a desirable item (e.g., a used car or a rental property) which has much better conditions (such as a lower price) than similar posts. The fraudster responds that they will need a small upfront payment to deliver the goods. After receiving it, the victim will not hear from the fraudster again.

A final example of advanced fee fraud is the online romance fraud. Taking place on online dating sites, this type of fraud usually consists in criminals posing as attractive individuals looking to start a relationship with the victim. Unlike the 419 scam, these online relationships often last for months before the fraudster demands money, for example, to help their family or to open a business [732]. By that time, the victim, who is likely emotionally involved with the persona impersonated by the criminal, is likely to comply. Previous research reported that victims of this crime can lose between £50 and £240,000 [734]. Unlike other types of advanced fee fraud, however, the psychological damage of losing the fictional relation can be much greater than the financial one.

A common element of every type of advanced fee fraud is the need for criminals to build an enticing narrative that will lure victims into paying the fraudulent fee. To this end, criminals often target specific demographics and impersonate specific personas. For example, previous research showed that romance fraudsters often pretend to be members of the military stationed abroad [736]. By doing so, the fraudsters can build a credible narrative as to why they cannot meet the victim in person, and they can build an emotional connection with the victim, which will increase the chances of their falling for the scam. Often, fraudsters pretend to be widowed middle-aged men who target widowed women in the same demographic, in an attempt to establish an emotional connection with their victim [732]. In other cases, fraudsters employ

psychological tricks to win their victims over, such as applying time pressure or remarking that they specifically selected the victim because of their high moral characters [696].

More cynically, Herley argues that fraudsters are incentivised to build the most absurd narratives possible, to make sure that only those who are gullible enough to believe them will reply, and that these people will be the most likely to fall for the scam [737]. His argument is that responding to the first boilerplate message is expensive for the fraudster, while sending the first copy to as many victims as they wish is free. For this reason, it is in their interest to rule out those who are not likely to fall for the scam as soon as possible.

Drug dealing. Another category of crimes for which the Internet has offered opportunities is the drug trade. Thanks to anonymising technologies such as Tor [738] and cryptocurrencies [739], online marketplaces have emerged where drug users can purchase illicit substances and have them delivered directly to their home. Research has shown that this business is thriving, despite the instability of these marketplaces, which are often taken down by law enforcement [697, 740]. Online drug markets provide an interesting paradigm switch for drug users, because they remove the need for the buyer to interact with criminals in a physical and potentially unsafe setting. Recent work has shown, however, that the inception of the online drug market has not changed the worldwide drug trade ecosystem: the big players who produce and dispatch drugs remain broadly unchanged, while what has changed is the 'last mile' in the delivery (i.e., how local dealers and drug users get in touch and do business) [733].

Cyber-dependent organized criminals

In this section, we describe crimes that have a financial goal and are carried out using complex technical infrastructures (e.g., botnets [741]). Unlike the cyber-enabled crimes described in the previous section, where the criminal is essentially replicating a physical criminal operation and using the Internet to enhance his/her reach, in the case of cyber-dependent crimes criminals have to set up complex technological infrastructures to achieve their goals. The complexity of these operations has prompted a compartmentalisation in the criminal ecosystem, where each malicious actor specialises in a specific part of a cybercriminal operation (e.g., infecting computers with malware or performing money laundering) and works together towards achieving a common goal. In this section, we provide some examples of cyber-dependent crimes that have been studied by the research literature in recent years. Then, in Section 7.2, we cover in detail the various elements that criminals need to put in place to make their operations successful.

Email spam. Email spam has been a major nuisance for Internet users for the past two decades, but it has also been at the forefront of very successful criminal operations, who have managed to monetise the sale of counterfeit goods and pharmaceuticals by reaching billions of potential customers through malicious messages [742]. Email spam is defined as *unsolicited bulk email*; this definition highlights the two main elements of the problem: the fact that the messages received by victims are unsolicited (i.e., they were not requested in the first place), and that they are sent in bulk to reach as many victims as possible.

Although the very first spam email was recorded in 1978 [743], email spam rose to prominence in the 1990s, when criminals set up small operations, not dissimilar from the advance-fee fraud ones described in the previous section [744]. The goal of these operations was to sell goods online, which could span from diet supplements to Nazi memorabilia [744]. At this stage, relying on their own expertise and on the help of a small number of associates, criminals would carry out all the activities required to set up a successful spam operation: (i) harvesting

email addresses to send the malicious messages to, (ii) authoring the email content, (iii) sending the spam emails in bulk, (iv) processing the orders from people who wanted to purchase the advertised items, (v) reacting to raids by law enforcement (e.g., the seizure of an email server). Although they were still rudimentary compared to the spam operations that came during the next decade, these criminal endeavours prompted the development of legislation to regulate unsolicited bulk emails, such as the Directive on Privacy and Electronic Communications in the EU,[3] the Privacy and Electronic Communications Regulations in the UK[4] and the CAN-SPAM Act in the US.[5] These pieces of legislation helped prosecute some of those early-day spammers. In 2004, America Online (AOL) won a court case against Davis Wolfgang Hawke, who was selling Nazi gadgets through spam emails. Hawke was sentenced to pay a 12.8M USD fine.

The technical advancements of the early 2000s, and in particular the development of botnets, networks of compromised computers controlled by the same cybercriminal [741], gave unprecedented opportunities to criminals who want to engage in email spam today. Email spam is not a one-person operation anymore, rather it is supported by thriving criminal ecosystems. Spammers can rent botnets from criminals who are specialised in infecting computers with malware [706], purchase lists of target email addresses from specialised actors [745] and sign up to an affiliate programme [746, 747], which will provide the spammer with a way of advertising, as well as taking care of shipments and payments.

The arms race connected to spam mitigation has been going on since the 1990s, with a number of mitigations being proposed [748]. Currently, anti-spam techniques ensure that the vast majority of malicious emails will never reach their victims' mailboxes. To solve this issue, criminals have to send tens of billions of emails [706] to keep their operations profitable. Another issue is that, out of the victims reached by those spam emails that make it through, only a small fraction will purchase the advertised goods and turn a profit for the criminals. Researchers performed a case study for the Storm botnet [17], showing that out of 469 million spam emails sent by the botnet, only 0.01% reach their targets. Of these, only 0.005% of the users click on the links contained in the emails, while an even lower number ends up purchasing items - only 28 users in total out of the 469 million reached, or 0.0004% of the total. Despite this steep drop, McCoy et al. showed that popular spam affiliate programmes were able to make up to 85 million USD of revenue over a three-year period [746]. They also showed that key to this success are returning customers. In fact, spam emails need to reach an interested customer only once, and this person can later keep purchasing on the site without having to worry about spam filters.

Phishing. A particular type of spam is phishing, where criminals send emails that pretend to be from genuine services (e.g., online banking, social network websites) [698]. These emails typically lure users into handing out their usernames and passwords to these services by presenting them with a believable email asking them to visit the website (e.g., to retrieve their latest account statement). By clicking on the link in the email, users are directed to a website displaying fake but realistic login pages. Once they have input their credentials, the criminals gain access to them and they will be able to later log in to those services on behalf of the users, potentially making money directly or selling the credentials on the black market.

For the criminal, a key component to the success of phishing pages is setting up web pages that resemble the original ones as much as possible. To facilitate this task, specialised

[3] https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32002L0058
[4] https://en.wikipedia.org/wiki/Privacy_and_Electronic_Communications_(EC_Directive)_Regulations_2003
[5] https://en.wikipedia.org/wiki/CAN-SPAM_Act_of_2003

cybercriminals develop and sell so-called *phishing kits* [749], programmes that can be installed on a server and will produce an appropriately-looking web page for many popular services. These kits typically also provide functionalities to make it easier for the criminal to collect and keep track of the stolen credentials [749]. Another element needed by criminals to host these pages is servers under their control. Similar to spam, criminals, researchers, and practitioners are involved in an arms race to identify and blacklist phishing Web pages [750], therefore it does not make economic sense for criminals to set up their own servers. Rather, criminals often host these websites on compromised servers, for which they do not have to pay [751].

After stealing a large number of credentials, criminals can either exploit these accounts themselves or sell the usernames and passwords on the black market. Previous research has shown that often these criminals log into the accounts themselves and spend time evaluating their value, for example, by looking for financial information in webmail accounts.[709, 752].

Financial malware. Another popular criminal operation is financial malware. In this setting, criminals aim to install malware on their victims' computers and steal financial credentials such as credit card numbers and online banking usernames and passwords. This trend started with the Zeus malware, which criminals could purchase on the black market and use to set up their operations [753]. Once installed on a victim computer, Zeus would wait for the user to visit a website on a pre-configured list of interesting ones that the criminal could specify. It would then record usernames and passwords as the user typed them in, and send them to the command and control server set up by the criminal.

A more sophisticated information stealing botnet was Torpig [699]. Unlike Zeus, Torpig used a botnet-as-a-service model, where a single specialised criminal was responsible for hosting the botnet infrastructure, while other criminals could run their campaigns to infect victim computers, pay a fee to use the torpig infrastructure and later retrieve the stolen credentials. Researchers showed that, in 2009, the Torpig botnet was able to steal 8,310 unique bank account credentials and 1,660 unique credit card numbers over a ten-day period. [699].

To monetise their operations, cybercriminals can sell the stolen financial information on dedicated underground forums [754]. The price that criminals can ask for these credentials varies based on the type of records that they were able to steal. For example, on the underground market there are two types of credit card records that are traded: *dumpz*, which contain the information that allows a criminal to clone a credit card (i.e., card number, expiration date, security code), and *fullz*, which also contain the billing address associated with the card. Fullz are worth more money on the black market, because they allow miscreants to purchase items online.

A related type of crime that is becoming more popular is card skimming [755]. In this cyber-enabled crime, criminals install devices on ATM machines which collect details of the cards inserted into the machines by unwitting users. The criminal can then collect the devices to retrieve the stolen financial credentials. While this type of crime is serious, it is also a good example of the limitations of physical crime compared to their online counterparts: the need for physical action by the criminal limits the scale of the operation, while financial malware operations can affect much higher numbers of victims. For example, the Torpig malware was installed on over 100,000 computers [699].

Note that malware is not always needed to perform financial fraud. In some cases, *insider threats* within financial organisations could act maliciously and defraud both their institutions and their customers [756, 757]. In other cases, financial information such as credit card numbers could be stolen by exploiting a vulnerability in an online system (e.g., by dumping

the database of an online store) [758]. In other cases, stolen SWIFT credential of banks can be used to perform large fraudulent money transfers [759]

Click fraud. Web advertisements are the main way the Web is monetised. A Web administrator can decide to host advertisements on his/her website, and whenever visitors view them or click on them they receive a small fee from an advertiser. To mediate this interaction, specialised services known as *ad exchanges* have emerged. Because of their easy monetisation, Web advertisements are ripe for fraud. In particular, criminals can host advertisements on their own websites and then generate 'fake' clicks (e.g., by using bots). This results in an ad exchange paying criminals for ad impressions that were not 'genuine,' eventually defrauding the advertiser.

Once again, criminals are involved in an arms race with ad exchanges, who are interested in keeping fraud on their services minimal. To help criminals generate large numbers of clicks and remain under the radar by gaining access from large numbers of IP addresses, so-called *click fraud* botnets have emerged. An example is Zeroaccess [16], which was active in 2013. On an infected machine, this malware would act like a regular user, browsing websites and clicking on advertisements that its owner chose. Researchers showed that this botnet was responsible for losses to the advertising industry of approximately 100,000 USD per day [16].

Unauthorised cryptocurrency mining. With the increasing popularity of cryptocurrencies, a new opportunity has opened up for criminals: using infected computers to mine currency. In 2014, Huang et al.revealed this threat, showing that botnets were used to mine Bitcoin [15]. While revealing this new monetisation for malware, the authors also concluded that these operations did not appear to be making much money, totaling at most 900 USD a day.

A more recent study, however, showed that cryptocurrency mining by botnets could be much more rewarding than previously thought. Pastrana and Suarez-Tangil showed that by mining Monero and using a number of techniques to increase their chances of mining currency (e.g., using mining pools) criminals could make up to 18 million USD over a two-year period. [760].

Another emerging trend in cybercrime comprises leveraging Web browsers to mine cryptocurrencies. Instead of installing malware on victim computers and using them for mining, miscreants add scripts to webpages and have their visitors mine cryptocurrencies. This type of malicious activity is called *cryptojacking*. Although using these scripts is not necessarily illegal (i.e., Web administrators can legitimately install them on their webpages in a similar way to advertisements), criminals have been caught adding them to compromised websites on multiple occasions. Konoth et al. showed that a malicious campaign can make GBP 31,000 over a week [761], while Rüth et al. [762] showed that 1.18% of the mined blocks in the Monero blockchain can be attributed to `Coinhive`, the most popular cryptojacking library.

Ransomware. The newest trend in malware is *Ransomware*. As part of this operation, criminals infect their victim systems with malware which encrypts the user's personal files (e.g., documents) and sends the encryption key to the criminal, who then asks for a ransom in exchange for giving the user access to their data again [763]. The idea of malicious software that uses public key cryptography to hold the victim's data hostage is not new, and it was theorised by Yung in 1996 already [764]. In 20 years, however, the technological advancements on the malware delivery end have made it possible to reach large numbers of victims, and the introduction of anonymous payment methods such as Bitcoin has made it safer for criminals to collect these payments.

Ransomware is, at the time of writing, the gold standard for cybercriminals. This type of malware operation has solved the monetisation problems that were so important in other

types of cybercriminal schemes: the criminal does not have to convince the victim to purchase a good, like in the case of email spam, or to fall for a fraud, like in the case of phishing. In addition, the victim is highly incentivised to pay the ransom, because the probability that the criminals have encrypted files that the user will need (and for which they have no backup copy) is high. In fact, recent research was able to trace 16 million USD in payments on the Bitcoin blockchain that can be attributed to ransomware campaigns [765].

Although the most sophisticated ransomware campaigns involve encrypting the victim's files, Kharraz et al. showed that it is not uncommon for malware authors to use other techniques to lock the victim out of his/her computer [766]. These techniques include setting up a password-protected bootloader and not giving the password to the user unless he/she pays. While these techniques are likely to yield a profit for the criminal, they are also easier to mitigate, as the victim's files are safe on the computer and a simple clean up of the malware (and restoring the original master boot record) can fix the problem.

Denial of service. A feature that all Internet-connected devices have is network connectivity. A criminal can leverage the bandwidth of an infected device to perform a Distributed Denial of Service (DDoS) attack against a target. Criminals can simply use the bandwidth generated by the botnet, or leverage *amplification attacks* (i.e., network traffic generated by misconfigured network devices, or devices with poor default settings) to enhance the power of their DDoS attacks [668].

The criminals can then set up services where they offer DDoS for hire. These services are appealing for example to unscrupulous actors who want their business competitors to go offline or to online gamersonline gaming who want to knock their opponents off the Internet to win the game [767]. To hide the illicit nature of their business, these services often advertise themselves as 'stress testers', services that a Web administrator can use to test how their Web applications perform under stress [767]. In reality, however, these services do not check whether the customer purchasing a DDoS attack is actually the same person who owns the target domain.

Hacktivists

While criminals driven by profit are a big threat, not all adversaries are driven by money. In particular, we define the act of computer crime motivated by a political goal as *hacktivism* [700]. These crimes can take various forms, from denial of service attacks [700] to compromising computer systems with the goal of releasing sensitive information to the public [768]. There is an ongoing debate among scholars on whether actions by hacktivists fall under political activism (e.g., civil disobedience) or cyber terrorism [769]. Holt et al. studied cyber attacks carried out by far left groups in the US and found that there was an increase in online attacks during periods that observed a decrease in physical violence from those same groups [770].

Denial of service. The practice of hacktivism started in the 1990s with *netstrikes* [771]. As part of this practice, Internet users would connect to target the websites simultaneously to deplete their resources and make them unresponsive. This was often done to protest against actions and policies by government agencies and corporations. Twenty years later, with the increased sophistication offered by technology, hacktivist groups such as Anonymous [772] took the idea of netstrikes and made it bigger in size. This collective became popular for launching denial of service attacks against organisations that were guilty of performing actions that did not match their moral stance, such as governments linked to the repression of the Arab Spring, credit card companies who would not make donations to entities such as Wikileaks or

radical religious organisations.

To perform their attacks, Anonymous would ask its sympathisers to install a computer program, called Low Orbit Ion Cannon (LOIC), which would act as a bot in a botnet: their controller would use the computer's bandwidth to carry out a denial of service attack against a chosen target. The difference with traditional botnets (and the ones used to carry out DDoS attacks in particular) is that the user is accepted to be part of it by installing the LOIC program, and suffered law enforcement action as a consequence.

Data leaks. Another trend that we have been observing in recent years in the area of hacktivism is the release of stolen documents into the public domain, for example, to raise awareness about secret surveillance programs by governments [773]. A prominent example of an organisation that performs these data leaks is Wikileaks [768]. Similar techniques have also been used by Anonymous (e.g., about the identity of 1,000 Ku Klux Klan members).

Web Defacements. The last trend that is typical of politically-motivated actors is *Web defacement* [774]. As part of this activity, miscreants exploit vulnerabilities (ranging from weak passwords to software vulnerabilities) in the websites of organisations they disagree with, and use them to change the home page of the website to a politically-charged one. An example of an organisation that is prominently using Web defacements to spread their message is the Syrian Electronic Army [775], a group of hackers close to the Assad regime. Although popular with criminals with a political agenda, Web defacement is not just their prerogative. In fact, Maimon et al. showed that this is a popular way for early career cybercriminals to prove their worth [776].

State actors

Another type of malicious actor involved in adversarial behaviours online comprises nation states. In the past few years, we have observed an escalation in the use of computer attacks by state actors to achieve their goals. Broadly speaking, this type of attack differs from those performed by financially motivated cybercriminals for two reasons:

1. Commodity cybercrime needs to gather as many victims as possible to maximise their profits. For instance, criminals setting up a botnet to steal financial information from their victims will want to reach the highest possible number of victims to improve their revenue. This means that the cybercriminal's attacks need to be either generic or diversified enough to cover a large population of devices (e.g., by using exploit kits, as explained in Section 7.2). In a state-sponsored attack, on the other hand, there is no need to make money, and usually the victim is well defined (e.g., a specific organisation or a person of interest). In this setting, the attack can be tailored to the victim; this increases the chances of success, because of the time that can be spent designing the attack and the fact that the attack will be unique (e.g., by using a zero day attack [777]), and it will be unlikely that existing protection software will catch it.
2. Because of the need to make money, traditional cybercriminals need their attacks to be fast. This is not the case for state-sponsored attacks, where the reward for achieving its goal (e.g., stealing sensitive information from a government) makes it acceptable to wait for long periods of time before finalising the attack.

State-sponsored attacks fall broadly into three categories, depending on the purpose of the attack: sabotage, espionage, and disinformation. In the following, we describe these three types of attacks in more detail.

Sabotage. Modern critical infrastructure can be disrupted by electronic means. Research has shown that it is not uncommon for critical facilities such as power plants to have some sort of network connectivity between the computers controlling the machinery and the ones connected to the Internet [778]. In the case of a state adversary, even having network security appliances to guard the boundary between the two networks is not enough, since, as we said, attacks can be so sophisticated and tailored that off-the shelf solutions fail to detect them [701]. Once a piece of malware manages to get into the control network, it could make the machinery malfunction and potentially destroy it. Even when there is a physical separation between the control network and the wider Internet, attacks are still possible when we are faced with adversaries with virtually unlimited resources [701].

A prominent example is the Stuxnet worm [701, 779], a sophisticated attack performed against the Nathanz nuclear enrichment facility in Iran in 2010. Allegedly, the malware was introduced into the facility by first infecting the laptop of one of the consultants who was maintaining the machinery. Once the malware was in the right environment, it identified the pieces of equipment that it was designed to target and sabotaged the enrichment experiments, making the centrifuges spin out of control. To date, Stuxnet is a textbook example of the lengths to which state-sponsored attackers can go to achieve their objectives, and of the sophistication that their attacks can achieve.

Sabotage is not always linked to state actors. Major incidents have been caused by disgruntled employees of companies who acted as insider threats, like in the case of the Maroochy Water Services [780]. In this incident an insider whose employment had not been confirmed decided to get revenge on the company by spilling sewage, causing major environmental damage [780].

Espionage. Another goal that state-sponsored actors have for their attacks is spying on opponents and prominent adversaries. Research has shown that state actors make prominent use of spearphishing (i.e., targeted phishing) to lure activists and companies into installing malware that is later used to spy on them [709, 781]. In other cases, state actors infect sensitive systems (e.g., servers in large corporations), with the goal of stealing sensitive information [782]. The security industry has dubbed these long-standing, sophisticated attacks *Advanced Persistent Threats*.

Disinformation. In the past two years evidence has emerged that state-sponsored actors have been involved in spreading disinformation on social media [702, 783, 784, 785]. This has been done through *troll* accounts that acted to polarise online discussion on sensitive topics [786]. While social networks such as Twitter have made data about accounts related to state-sponsored disinformation publicly available [702, 783], rigorous evidence is still missing on how these operations are carried out on the backend. For example, the extent in which the accounts involved in disinformation are controlled by human operators as opposed to bots is not clear.

7.2 THE ELEMENTS OF A MALICIOUS OPERATION

[787][788][789][790, 791][705][792, 793]

As we showed in Section 7.1, malicious operations can use rather complex infrastructures, particularly in the case of organised crime, which is mostly motivated by two facts. First, the criminal needs these operations to be as cost effective as possible (and consequently make the highest possible profit). Second, multiple actors (law enforcement, security companies, the users themselves) are constantly attempting to take down these malicious operations, and the criminal has, therefore, a need to make them resilient to these takedown attempts.

To ensure that the criminals' needs are met in this scenario, in recent years we have witnessed a *specialisation* in the cybercriminal ecosystem, where different actors specialise in a specific element required for the operation to succeed; the miscreants then trade these services with each other on the black market. In this section, we provide an overview of the elements required for a cyber-dependent organised criminal operation to succeed, as described in Section 7.1. Many of the elements discussed, however, also apply to the other types of adversarial behaviours described in that section.

Affiliate Programmes

The main goal of organised crime is to make money from their operations. This requires not only a well-oiled technical infrastructure to make sure that their botnets operate properly but, perhaps more importantly, a working method to collect payments from victims (or from sponsors, in the case of DoS), while making sure that all the actors involved in the operation get paid.

In the cybercriminal world, this is typically done through *affiliate programmes*. An affiliate programme is a scheme where main organisation provides a 'brand' and all the means required to carry out orders, shipments and payments. Affiliates can join the program, direct traffic to the platform, and get a cut of the sales that they are responsible for. Although this scheme exists for legitimate businesses (e.g., Amazon has an affiliate programme), it has been particularly successful for cybercriminal operations. The main difference between legitimate and criminal affiliate programmes is that the second category of operations typically deals with products that are considered illegal in most jurisdictions (e.g., counterfeit pharmaceuticals, gambling, counterfeit designer products) and they typically endorse criminal promotion techniques (e.g., the use of malware or black hat search engine optimisation).

Affiliate programmes are popular in the cybercriminal world because they mean affiliates do not have to set up their operations from start to finish, but rather focus on attracting traffic, for example by setting up botnets and sending email spam advertising the affiliate marketplace. The first successful examples of affiliate programmes for cybercrime were centred around email spam, and were advertising counterfeit pharmaceuticals [742, 746, 747]. However, affiliate programmes are present in most types of cyber-dependent crime, an example being the Cryptowall ransomware operation.[6]

In addition to providing the monetisation necessary for cybercriminal operations, affiliate programmes also act as facilitators for criminals to get in contact and trade the services that are needed for the operation to succeed. This is typically done by setting up a forum where

[6] https://www.secureworks.com/research/cryptowall-ransomware

affiliates can trade their services [706, 746]. Gaining access to these forums typically requires vetting by the affiliate programme administrators.

Infection vectors

As discussed earlier, the first step required by criminals to perform a malicious activity is often infecting their victims with malware. To this end, the criminals need to first expose their potential victims to the malicious content, and then have them install it on their machines (through either deception or by exploiting a software vulnerability in their system). In the following, we survey three popular methods on delivering malware to victim computers. Note that, while other infection vectors are possible, such as physical access to a network or hijacking a wireless network, to date we are not aware of any large-scale compromise involving these infection vectors, and therefore we do not focus on them.

Malicious attachments. Possibly the oldest method of delivering malware is attaching malicious software to spam emails, disguising it as useful content that the user might want to open. This spreading technique was made popular by email worms in the early 2000s, such as the 'I love you' worm [794], but it is still a popular way of delivering malware to victims [706]. In the commoditised economy described previously, it is often the case that a criminal who wants to spread a malware infection pays another criminal who already has control of a botnet to deliver the payloads [699]. To be successful, the content used for this infection vector needs to convince the user to click on the attachment and install it. To this end, criminals often use deception techniques to make the content look interesting and appealing, similar to the techniques discussed for phishing [698]. This deception falls into the area of social engineering [795].

Black hat search engine optimisation. Search Engine Optimization (SEO) is a popular practice whereby webmasters optimise their content so that it is better indexed by search engines and appears among the first hits for relevant searches. Cybercriminals are also interested in having their malicious Web pages appear high in search results, because this increases the chances that potential victims will find them and click on them. To accomplish this, specialised criminals offer black hat SEO services. As a result of these services, malicious websites are pushed high up in search engine rankings for keywords that are unrelated to the website [796]. This happens particularly often in proximity with popular events (e.g., sports and political events), because people will be more likely to search for keywords related to the event. To achieve effective black hat SEO, cybercriminals compromise vulnerable websites and use them to promote their customers' webpages (e.g., by adding invisible links and text pointing to the target webpage).

Drive-by download attacks. Although deceptively luring users into installing malware works, having an automated method that does not require human interaction is more advantageous for cybercriminals. To this end, cybercriminals have perfected so-called *drive-by download* attacks [788]. As part of one of these attacks, the victim visits a webpage under the control of the criminal (e.g., encountered through black hat SEO). The webpage contains malicious JavaScript code that will attempt to exploit a vulnerability in the user's Web browser or in one of its plugins. If successful, the Web browser will be instructed to automatically download and install the malware.

To host their malicious scripts, cybercriminals often compromise legitimate websites [797]. An alternative trend is purchasing Web advertisement space and serving the malicious content as part of the ad, in a practice known as *malvertisement* [798].

Compromising of Internet-connected devices. As more devices get connected to the Internet (e.g., Internet of Things (IoT) devices), an additional opportunity provided to attackers is scanning the Internet for devices that present known vulnerabilities and exploit them to build large botnets. A prominent example of this was the Mirai botnet [665].

Infrastructure

Another important element that criminals need for their operations to succeed is where to host their infrastructure. This is important for both affiliate programmes (e.g., where to host fraudulent shopping websites) as well as for botnet operations. Law enforcement and Internet Service Providers (ISPs) are continuously monitoring servers for evidence of malicious activity [789], and will take them down if this activity can be confirmed, which would put the criminal operation in jeopardy.

Bulletproof hosting service providers. To maximise the chances of their operations being long-lived, cybercriminals resort to using so-called *bulletproof hosting service providers* [742, 799]. These providers are well known not to comply with law enforcement takedown requests. This is made possible by either being located in countries with lax cybercrime legislation, or by the service provider operators actively bribing local law enforcement [742]. Bulletproof hosting service providers typically charge their customers more money than a regular ISP would. As such, they become a hotspot of illicit activity, since malicious users congregate there because of their guarantees, but legitimate users have no incentive to use them. Despite providing higher guarantees for cybercriminals, bulletproof hosting service providers are not invincible to takedown efforts. In particular, ISPs need to be connected to each other to be able to route traffic, and an ISP that is uniquely hosting malicious content could be disconnected by the other providers without many consequences for legitimate Internet traffic [742].

Command and control infrastructure. A botnet requires a command and control (C&C) infrastructure that infected computers can be instructed to connect to, receive orders and report on progress in the malicious operation. Originally, botnets would use a single command and control server, although this would be a single point of failure. Even assuming that the server was hosted by a bulletproof hosting provider, and could not therefore be taken down, the fact that the server had a unique IP address meant that it could easily be blacklisted by security companies.

To mitigate this problem, cybercriminals came up with C&C infrastructures that are redundant and more difficult to take down. An example is the *multi-tier* botnet infrastructure, where bots are instructed to connect to an intermediary C&C server, which is then responsible for relaying the information to and from a central control server [800]. This infrastructure makes the botnet more resilient, because even if some of the relays are taken down, the central C&C is still operational and additional relays can be added. In addition, the infected computers never see the IP address of the central C&C server, making it more difficult to locate and take down. A variation of this model is *peer-to-peer* botnets, where infected computers with particularly good connectivity and public IP addresses are 'elected' to act as relays [801]. This infrastructure increases the flexibility that the criminal has and reduces the cost of the operation, because the criminal does not have to spend money to install relays. However, the botnet infrastructure becomes vulnerable to infiltration, whereby researchers can create fake bots, be elected as relays and are thus suddenly able to monitor and modify the traffic coming from the central C&C [17].

Additional techniques used by cybercriminals to make their control infrastructure more resilient

are *Fast Flux* [689], where criminals use multiple servers associated with the C&C infrastructure and rotate them quickly to make takedowns more difficult, and *Domain Flux* [802], in which the domain name associated to the C&C server is also rotated quickly. Both methods are effective in making the operation more resilient, but they also make the operation more expensive for the criminal to run (i.e., they have to purchase more servers and domain names).

Specialised services

In this section, we describe specialised services that help criminals to set up their operations. In addition to these dedicated malicious services, others that have a legitimate use (e.g., VPNs, Tor) are also misused by criminals, for example hosting drug market websites on the Dark Net [740, 803].

Exploit kits. In the previous section, we saw that drive-by download attacks are a powerful weapon that a cybercriminal can use to infect computers with malware without any human interaction. The problem with effectively performing these attacks, however, is that they require an exploit to a software vulnerability in the victim's system. Since cybercriminals want to infect as many victims as possible, it is challenging to find an exploit that can work on the systems of the majority of potential victims. In addition to this issue, exploits do not age well, since software vendors routinely patch the vulnerabilities that they know about. A cybercriminal performing a sustained drive-by download operation, therefore, would need to continuously collate exploits to multiple vulnerabilities, a task that is unfeasible, especially when the criminal also has to run other parts of the business (e.g., the monetisation part). Once a victim visits the exploit kit's webpage, this tool first fingerprints the victim's system, looking for a potential vulnerability to be exploited. It then delivers the exploit to the victim. If successful, the victim's computer is instructed to download the malware of the customer's choice.

These issues have created an opportunity for specialised criminals to provide services for the rest of the community. This has led to the creation of *exploit kits* [790], which are tools that collect a large number of vulnerabilities and are sold on the black market for other criminals to use. An exploit kit is typically accessible as a Web application. Customers can point their victims towards it by compromising websites or using malicious advertisements.

Pay-per-install services. Infecting victim computers and maintaining a botnet is a complex task, and research has shown that malware operators who attempt to do so without the proper expertise struggle to make profits [804]. To solve this issue and satisfy the demand for stable botnets, a new criminal service has emerged called Pay Per Install service (PPI) services [791]. PPI operators are proficient in setting up a botnet and having it run properly. Other criminals can then pay the PPI operator to install malware on the infected computers on their behalf. PPI services typically offer a good level of choice granularity to their customers, who not only choose how many infections they want to install, but also their geographical location (with bots in developed countries costing more than infections in developing ones [791]).

An advantage of using PPI services is that they make their customers' cybercriminal operations more resilient: if their malware stops working, for example, because law enforcement has taken down the C&C servers that it uses, the criminal can resume operations by asking the PPI operator to install an updated version of their malware on the victim machines. For this reason, this *malware symbiosis* between PPI services and other botnets is very common in the criminal ecosystem (see, for example, the symbiosis between Pushdo and Cutwail [706], and between Mebroot and Torpig [699]).

Human services

In this section, we discuss the auxiliary services that are needed for an end-to-end cybercriminal operation to succeed. Although these elements are not usually thought to be part of cybercrime, they are as important to the success of a cybercriminal operation as the more technical elements.

CAPTCHA solving services. In some cases, cybercriminals need to set up accounts on online services to initiate their operations (e.g., a spam operation running on social networks [707, 708]). To protect themselves against large-scale automated account creation, however, online services widely use CAPTCHAs, which are notoriously difficult for automated programs to solve. To solve this problem faced by cybercriminals, new CAPTCHA solving services have been established [805]. These services take advantage of crowdsourced workers. Once the CAPTCHA solving customer encounters a CAPTCHA, this is forwarded by the service to one of these workers, who will solve it. This way, the customer can proceed and create the account on the online service.

In other cases, online services require whoever has created an online account to receive a code texted to a phone number and issue that code back to the service. To overcome this issue, cybercriminals can use services that automate this type of interaction [705].

Fake accounts. Since creating fake accounts is time consuming and requires the use of auxiliary services such as CAPTCHA solvers, cybercriminals have started specialising in the creation of fake accounts on multiple online services, and selling them on the black market [806]. Accounts on different services can have different prices, depending on the ease of creating new accounts on the platform and on how aggressively the service suspends suspected fake accounts.

A problem with newly purchased fake accounts is that they do not have an established 'repuation' on the social network, thus reducing their credibility to potential victims and their reach in spreading malicious messages. This can be mitigated by using 'reputation boosting' services, which help to build a network of contacts for accounts that otherwise would not have any. Examples of these are services offering fake likes on Facebook [807] and luring compromised accounts into following the service's customers on Twitter [808].

Content generation. In some cases, cybercriminals need to set up fake content to send to their victims, whether this is for spam emails, fake websites used for black hat SEO or online social network sites. To generate this content, the criminals can recruit workers on underground forums [809].

Money mules. The main goal of many cybercriminal operations is to make money from their victims. However, extracting money from an operation is not easy. In the case of bank fraud, for example, even if the criminals obtain access to the victim's bank account, they still need to transfer money to accounts under their control without being detected and apprehended.

To facilitate these monetisation operations, criminals take advantage of *money mules* [810]. These are people who are recruited by criminals to perform money laundering operations and make it more difficult for law enforcement to track the money obtained from an illicit operation. In a money mule scheme, the criminal recruits a person to act as a mule and sends them money by using traceable means (e.g., a check or a wire transfer). The mule is then instructed to transfer the money to an account under the criminal's control by using untraceable means (e.g., Western Union). The mule is also told that they can keep a percentage of the amount as a payment. Since these untraceable transactions need to be carried out in person by the mule,

they constitute a weak point in the monetisation operation, meaning that law enforcement could identify and arrest the mule before the money is transferred. In fact, even if stolen money is never mentioned, the mule is participating in money laundering when he/she accepts this job.

An alternative way of monetising malicious operations, which is used in the case of stolen credit cards, is *reshipping mules* [758]. In these operations, criminal agencies recruit unsuspecting users advertising a 'shipping agent' job. Then other criminals can recruit the services of these agencies, and purchase expensive items using stolen credit cards (e.g., electronics, designer goods), while sending them to the mule's home address. The mule is then instructed to open the packages and reship the goods to a foreign address, where they will be sold on the black market.

Payment methods

As criminals need to have money transferred to them, they can use a number of different payment methods, each carrying a different level of risk and being more or less familiar to the victims.

Credit card processors. Most transactions online are performed by credit cards. To collect as many customers as possible, cybercriminals tend to accept credit card payments too. McCoy et al. showed that 95% spam affiliate programmes between 2007 and 2012 accepted credit card payments [746], and that DDoS services that did not accept credit cards suffered with regard to the numbers of customers that they were able to attract [767]. Credit card processors keep track of the chargebacks that a company has on its accounts, and too many complaints from customers usually result in the company's accounts being terminated. For this reason, many cybercriminal operations offer 'customer support' to their victims, offering refunds if they are not satisfied with their purchases [792].

A challenge that cybercriminals face is finding banks that are willing to process their payments. Typically, these banks would charge them higher transaction fees (10-20%) to cover the risk of dealing with criminal operations [746]. Despite these increased fees, it is not guaranteed that the criminal operation will be safe: similar to what happens with bulletproof hosting ISPs, banks need to maintain good relations with their peers, otherwise they will be disconnected from the financial network [787].

Paypal. Another payment method that is familiar to users is Paypal. For this reason, Paypal is often accepted by criminals offering illicit services. While user friendly, criminals face the issue that the platform is centralised, and Paypal can keep track of fraudulent payments and terminate the accounts that are found to be in breach of the terms of service [796].

Western Union and other 'untraceable' payments. Other forms of payment offer more anonymity for cybercriminals, and are less risky as well as being not as well regulated. Examples are money exchanges (e.g., Western Union, Money Gram) or pre-paid vouchers (Money Park). These are often used by criminals to transfer funds [811]. To cash the money, these services only require a unique code and an identification document. Depending on the country where the criminal is located, however, the ID requirement might not be very rigorous.

Historically other 'anonymous' payment methods have existed such as Liberty Reserve, Web Money and eGold [705]. These virtual currencies allowed criminals to easily make payments as they took advantage of the loose regulations in their country of origin (e.g., Liberty Reserve

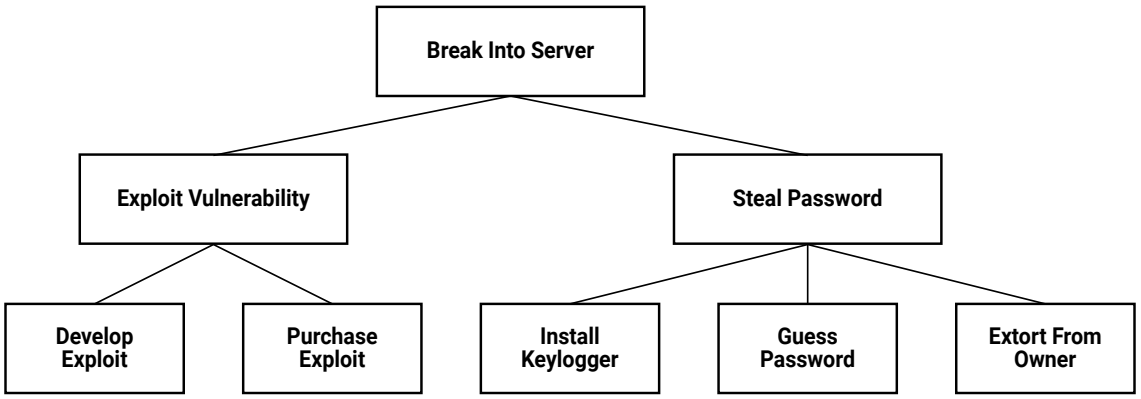


Figure 7.1: Example of an attack tree describing the action of breaking into a server.

was based in Costa Rica). After crackdowns on these payment methods by law enforcement, criminals moved to other payment methods.

Cryptocurrencies. At the time of writing, probably the safest form of payment for cybercriminals is cryptocurrencies. These payments have become popular for multiple types of cybercriminal operations, from ransomware [766] to drug market payments [697]. While research has shown that customers are reluctant to use services that only accept cryptocurrencies [767], this type of payment still works when victims have no choice (e.g., in the case of ransomware) or are very motivated (e.g., in the case of drug markets).

While more anonymous than other payment methods, research has shown that payments made in Bitcoin can be traced [793]. In addition, often cryptocurrencies need to be converted into real money by criminals, and the money ceases to be anonymous at that point. Additional concerns arise from the risks involved in making payments on cryptocurrency exchanges. Moore et al. showed that it is not uncommon for Bitcoin exchanges to suffer breaches that result in losses of currency [812]. Exit scams, where an exchange vanishes with all the currency stored in it, are also a problem [813].

7.3 MODELS TO UNDERSTAND MALICIOUS OPERATIONS

[62][814][815, 816, 817][705][818]

As shown in the previous sections, malicious operations can be quite complex and entail multiple technical elements and multiple actors. It is, therefore, necessary for defenders to have the appropriate means to understand these operations, so that they can develop the best countermeasures. In the following, we survey a number of models that have been proposed to model malicious operations. These models come from a number of research areas, including computer security, criminology and war studies. Note that for space reasons we cannot discuss all the techniques that have been proposed in the literature to model attacks. For a more comprehensive list, we point the reader to [819].

Attack trees

The first way to model attacks against computer systems involve *attack trees* [62]. Attack trees provide a formalised way of visualising a system's security during an attack. In an attack

tree, the root node is the goal of the attack, and its child nodes are the ways an attacker can achieve that goal. Going down the tree, each node becomes a sub-goal that is needed for the attack to succeed, and its children are possible ways to achieve it.

Figure 7.1 represents an example of an attack tree. In this example, the attackers aim to compromise a server. To do this, they have two choices: they can either exploit a vulnerability or they can obtain the password to the root account and log in using normal means. To exploit a vulnerability, they can either develop the exploit themselves or purchase an already existing one, perhaps through an exploit kit. If the attackers decide to use the account's password to log into the server, they first need to obtain it. To do this, they can either install malware on the server administrator's computer to log the password as they input it (i.e., a keylogger), guess the password using a list of commonly used ones or perform a bruteforce attack, and finally extort the password from the owner. The attack graph could then be further refined with the possible ways the attacker could perform these actions (e.g., extorting the password by blackmailing the owner, by kidnapping them etc.).

Attack trees allow two types of nodes, '*or*' nodes and '*and*' nodes. 'Or' nodes represent the different ways attackers can achieve a goal (i.e., the children of any node in Figure 7.1). 'And' nodes, on the other hand, represent the different steps that all need to be completed to achieve the goal. Once the tree has been created, security analysts can annotate it to assess the system's risk to the attack, for example, by marking the various attack strategies as feasible or unfeasible, by assigning likelihood scores to them or by estimating the cost for an attacker to perform a certain action. The scores can then be propagated along the tree following specific rules [62] to assess the overall feasibility and likelihood of the attack.

Another model that is related to attack trees is *attack graphs* [820]. While attack trees are limited to single targets, attack graphs allow to model attack actors, vectors, vulnerabilities, and assets. Another useful model to understand network attacks are *attack nets* [821].

Kill chains

Another useful tool that can be used to model and understand attacks is *kill chains*. In the military context, a kill chain is a model that identifies the various phases involved in an attack.[7] In the computer world, Hutchins et al. developed a *Cyber Kill Chain* [814] that models the different steps involved in a malicious operation conducted against computer systems. In their model, Hutchins et al. identify seven phases. The model is designed for operations where the attacker identifies, compromises and later exploits a computer system, and, therefore, not all the phases apply to all the adversarial behaviours discussed in this document. The seven phases are the following:

1. **Reconnaissance**, when attackers identify possible targets. This phase could comprise an attacker scanning the network looking for vulnerable servers or a spammer purchasing a list of victim email addresses on the black market.
2. **Weaponisation**, when an attacker prepares the attack payload for use. This could consist in developing a software exploit against a newly identified vulnerability or crafting an advance-fee-fraud email.
3. **Delivery**, when the attacker transmits the payload to its victim. This could consist in setting up a malicious webserver, purchasing advertisement space to perform a malvertising attack or sending an email containing a malicious attachment.

[7] https://en.wikipedia.org/wiki/Kill_chain

4. **Exploitation**, when the target's vulnerability is exploited. This phase could entail a drive-by download attack, or the victim being lured into clicking on a malicious attachment through deception.
5. **Installation**, when malicious software is downloaded, thus allowing the attacker to benefit from the victim machine. In their paper, Hutchins et al. considered an attacker wanting to maintain constant access to the victim computer, using a type of malware known as a Remote Access Trojan (RAT) [822].
6. **Command and control**, when the attacker establishes a C&C infrastructure and a communication protocol to control the infected computer.
7. **Actions on objectives**, when the infection is monetised. This could entail stealing sensitive information from the victim computer, encrypting the victim's data with ransomware, mining cryptocurrencies, etc.

For each of the seven steps, Hutchins et al. identified strategies to disrupt the malicious operations, following five possible goals (Detect, Deny, Disrupt, Degrade, Deceive). Examples of these techniques include patching vulnerabilities, setting up intrusion detection systems on the network or deceiving the attacker by setting up honeypots [823].

Similar kill chains have been proposed by other researchers over the years. An example is the one proposed by Gu et al. to model botnet infections [671]. In this model, the authors identify five phases where an infection is separated: an inbound scan (similar to phase one in the previously described model), an inbound infection (similar to phase four from the previous model), an 'egg' download (analogous to phase five), a C&C phase (the same as phase six), and an outbound scan. At the time of developing this model, botnets were mostly acting as computer worms [824], scanning for vulnerable computers, infecting them, and using them to propagate further. While this model correctly depicted early botnets, it ceased to map reality when botmasters started using other methods to install their malware and monetise their infections. Nowadays, worms are almost extinct, with the exception of the infamous WannaCry malware [825]. This example shows that it is difficult to develop models of attacker behaviour that are resilient to changes in the modus operandi of attackers.

Environmental criminology

While cybercrime is a relatively new threat, physical crime has been studied by scholars for decades. It is, therefore, interesting to investigate whether this established body of knowledge can be applied to better understand and mitigate the emerging threat of online crime. Environmental criminology, in particular, is a branch of criminology that focuses on criminal patterns in relation to the space where they are committed and to the activities of the actors involved (victims, perpetrators, and guardians) [815]. A particular challenge that arises when we attempt to apply environmental criminology theory to cybercrime is that the concept of 'place' on the Internet is not as well defined as in the real world. In the following, we briefly review the key concepts of environmental criminology, and provide some examples of how they could be applied to mitigating Internet crime.

Routine activity theory. *Routine activity theory* is a commonly used concept in environmental criminology, postulating that the occurrence of crime is mostly influenced by an immediate opportunity for one to commit a crime [826]. In particular, routine activity theory states that for a crime to happen, three components need to converge: (i) a motivated offender, (ii) a suitable target and (iii) the absence of a capable guardian.

These concepts could be useful for better modelling malicious activity online. For example, research has shown that botnet activity reaches a peak during daytime, when most vulnerable computers are switched on and the victims are using them, while it drops significantly overnight [699]. In routine activity theory terms, this can be translated to the fact that when more potential victims are online, the opportunity for criminals to infect them increases and this results in an increase in botnet activity.

Rational choice theory. *Rational choice theory* aims to provide a model as to why offenders make rational choices to commit crime [827]. In the case of cybercrime, this model could be useful for understanding the reaction of criminals to mitigation as a rational choice, and help to model the implementation issues introduced by situational crime prevention such as displacement. For example, when a bulletproof hosting provider is taken down by law enforcement, what factors play a part in the criminal's choice of the next provider?

Pattern theory of crime. Another theory, called the *pattern theory of crime*, allows researchers to identify various places that are related to crime. These places are likely to attract offenders (crime *attractors*), they generate crime by the availability of crime opportunities (crime *generators*) and they enable crime by the absence of place managers (crime *enablers*).

Although defining places in cyberspace is not as straightforward as in physical space, thinking in terms of pattern theory can help identify locations that are hotspots for cybercrime, whether they are particularly appealing targets, such as corporations storing sensitive data (attractors), poorly configured systems that are easier to compromise (generators) or online services with poor hygiene that do not react promptly to spam/malware posted on their platforms (enablers). Identifying these hotspots can then be used to design appropriate countermeasures against the malicious activity (e.g., to whom to direct education campaigns).

Situational crime prevention. Situational crime prevention comprises a set of theories and techniques that aim to reduce crime by reducing the opportunities for crime [828]. The ideas behind situational crime prevention are based on three main concepts, which also apply to cybercrime:

- Crime is much more likely to happen in certain places (*hotspots*). This idea applies to the context of cybercrime. As we have seen, criminals tend to concentrate their malicious servers in bulletproof hosting service providers, which provide them with guarantees that their operations can continue for long periods of time. At the opposite end of the spectrum, regarding victims, criminals tend to target computers with vulnerable software configurations, which also constitute hotspots in this acception.
- Crime is concentrated in particular 'hot products'. This also applies to cybercrime, with miscreants focusing on whichever operations yield the highest profits (i.e., at the time of writing, ransomware).
- Repeat victims are more likely to experience crime compared to other people. In the context of cybercrime, the same concept applies. A vulnerable computer that is not patched is likely to be compromised again [824]. Similarly, in the case of advance fee fraud, victims are likely to repeatedly fall for the fraud, because the narrative used by the criminals particularly resonates with them [734]. In addition to the natural predisposition of victims to fall for similar scams again, criminals actively seek to contact past victims of fraud, by compiling so-called *suckers lists* and sharing them with each other [829].

To reduce the opportunities for crime, situational crime prevention proposes five categories of mitigations. In the following, we list them along with some examples of mitigations against

cybercrime that have been proposed in the computer science literature and that can be grouped into these categories:

- **Increase the effort of crime.** Mitigations here include deploying firewalls and setting up automated updates for software installed on computers.
- **Increase the risk of crime.** Mitigations here include reducing payment anonymity (e.g., requesting an ID when someone cashes money from Western Union).
- **Reduce rewards.** Mitigations here include blocking supicious payments or parcels, or penalising malicious search results.
- **Reduce provocations.** Examples here include applying peer pressure to rogue ISPs and banks.
- **Remove excuses.** Typical mitigations in this category include running education campaigns or setting up automated redirects to divert victims who would have viewed malicious content, explain to them what happened and urge them to secure their systems.

An interesting aspect of the situational crime prevention framework is that it identifies, for each mitigation, the *implementation issues* that arise when putting the mitigation in place [828]. In the case of cybercrime, the two implementation issues that are most relevant are *adaptation* and *displacement*.

Adaptation embodies the fact that criminals will actively attempt to circumvent any mitigation by making their operation stealthier or more sophisticated. This is a typical arms race that can be observed in computer security research. When researchers started compiling blacklists of IP addresses known to belong to C&C servers, criminals reacted by developing Fast Flux. When making payments through traditional means became more difficult due to increased vetting, criminals moved on to cryptocurrencies. Considering adaptation is important when designing mitigations against cybercrime. In particular, effective mitigations are those which the criminal cannot easily react to, or where adaptation comes at a financial price (e.g., a reduction in revenue).

Displacement represents the fact that once mitigations are put in place, criminals can simply move their operations elsewhere. While in the physical world how far criminals can travel is dictated by practical constraints, on the Internet moving from one 'place' to another is virtually free. Examples of displacement include criminals starting to register DNS domains with another registrar after their preferred one increased the domain price to curb misuse [830], or a multitude of drug markets opening to fill the gap left by Silk Road's takedown [697]. Displacement effects are important when planning action against cybercrime. Generally speaking, a mitigation should make it difficult for criminals to move elsewhere. Conversely, a mitigating action that simply displaces a cybercriminal operation without affecting its effectiveness is probably not worth pursuing.

Researchers have applied Situational Crime Prevention to a number of computer crimes, including organisational data breaches [816] and the mitigation of software vulnerabilities [817]. Following the discussion in this section, however, this framework could be applied to any criminal activity that happens online.

Crime scripting. Another useful technique that can aid the analysis of malicious activities on the Internet from the field of criminology is crime scripting [831]. As part of this technique, researchers extrapolate the sequence of steps performed by an adversary to commit their

offences. For example, in a romance scam, fraudsters create a fake account on a dating profile, the identify a suitable victim, go through a grooming phase, followed by the actual fraud when the scammer asks their victim for money. Dissecting the various steps of an offence can be useful to better understand it and to identify potential interventions. Crime scripting is somewhat related to kill chains, although the two techniques were developed in completely independent areas.

Modelling the underground economy as a flow of capital

As discussed in Section 7.1, many malicious operations are performed by criminals with the goal of making money from their victims. For this reason, following the flow of money is a useful way to better understand malicious operations, and in particular identify bottlenecks that could be leveraged to develop mitigations against them and stop criminals [787, 832].

Thomas et al. presented a model that is designed to keep track of a money flow within a cybercriminal operation [833]. As part of this model, they introduced two elements that are needed for a cybercrime operation to run: *profit centres*, through which victims transfer new capital into the criminal operation, and *support centres*, which can facilitate the criminal operation by providing several services for a fee. Money is introduced into the ecosystem through profit centres, and is then consumed by the various actors involved in it, who provide tools and services for each other. As an example, in an email spam operation, the profit centre would be victims purchasing counterfeit pharmaceuticals from an affiliate programme, while all the services needed by the spammers to operate (e.g., bulletproof hosting providers to host the C&C servers, pay-per-install services to deliver the malware, content generation services to create the spam content) are support centres. This model provides an interesting conceptualisation of how money flows into the cybercriminal ecosystem and how wealth is divided between the different actors there. By cross-referencing it with real world data, it can also help to form an idea of the profit that each criminal is making, and of the revenue of the operation.

Another interesting aspect of tracing the cash flow of cybercriminal operations is that at some point the criminals will want to cash out, which will be done using traditional payment methods (see Section 7.2). Since these interactions happen in the physical world, it is easier for law enforcement to trace them and potentially apprehend the criminals [832].

Attack attribution

When talking about malicious activities, attribution is important. Law enforcement is interested in understanding what criminals are behind a certain operation, and in particular attributing apparently unrelated cybercriminal operations to the same actors could help build a legal case against them. In similar fashion, governments are interested in identifying the culprits behind the attacks that they receive. In particular, they are interested in finding which nation states (i.e., countries) are behind these attacks.

Attribution, however, is a controversial topic in cyberspace. As we discussed previously, the concept of 'place' is relative for computer attacks, and attackers can easily route their network connections through proxies or compromised machines in third countries, thus hiding their actual location. It is reasonable to assume that the same actors will follow a similar *modus operandi* in their attacks, and in particular will use the same software exploits to break into their victims' systems. These exploits and code artefacts could be used to identify state-

sponsored groups or other attackers (See the Malware & Attack Technologies Knowledge Area (Section 6.5.2), for more details). Unfortunately, this approach has two main drawbacks. The first is that the commodisation of cybercrime services has enabled attackers to use exploit kits, which contain a large number of exploits and, therefore, increase the likelihood of an attack happening. While advantageous for attackers, this trend means that the exploits used become a less significant signal for identifying attackers, especially those who do not have the sophistication to exploit vulnerabilities in house (e.g., cyber-enabled cybercriminals). The exception to this trend is state-sponsored actors, who unlike traditional criminals usually have very specific targets. For this reason, they can tailor their attacks more carefully, and even develop new exploits to hit a specific victim. Most importantly, they often develop exploits for vulnerabilities that are not publicly known, also known as *zero days attacks* [777]. Being unique to an actor, they could be used to identify who is behind a specific attack. An issue here is that, once an exploit is used, it could be intercepted by the victim (or anyone on the network) and later used against another target affected by the same vulnerability. This would actively mislead attribution. Recent leaks have shown that the CIA has been actively collecting exploits used by other nation states and adding them to their arsenal, thus allowing them to make it look like another country was behind any given computer attack.[8]

Rid et al. proposed a framework to systematise the attribution efforts of cyberattacks [818]. Within this framework, they identified three layers of analysis that are needed to correctly perform attribution: tactical, operational and strategic. The tactical component consists of understanding the technical aspects that composed the attack (the *how*), the operational component consists of understanding the attack's high-level characteristics architecture and the type of attacker that we are dealing with (the *what*), while the strategic component deals with understanding the motivation behind the attack (the *why*).

While this framework was developed with state-sponsored attacks in mind, it could be used to attribute other types of malicious activity. For example, to attribute an online hate attack orchestrated by 4chan's Politically Incorrect Board, [718] one could trace the hate messages reaching the victim (*how*), observe the personal information of the victim on the board (*what*) and analyse the discussion about the victim to understand the motivation behind the attack (*why*).

CONCLUSION

In this document, we presented an overview of the adversarial behaviours that exist on the Internet at the time of writing. We surveyed various types of malicious operations, depending on the attacker's motivations and capabilities, and analysed the components that are required to set up successful malicious operations. Finally, we described a number of modelling techniques from a variety of fields (computer science, criminology, war studies) that can help researchers and practitioners to better model these operations. We argued that having good models is of fundamental importance to developing effective mitigations that are difficult to circumvent.

[8] https://en.wikipedia.org/wiki/Vault_7

CROSS REFERENCE OF TOPICS VS REFERENCE MATERIAL

Sections	Cites
7.1 A Characterisation of Adversaries	
Cyber-enabled and cyber-dependent crimes	[694]
Interpersonal offenders	[695, 719, 722, 724, 725]
Cyber-enabled organised criminals	[696, 697, 734]
Cyber-dependent organised criminals	[15, 16, 17, 698, 699, 766, 767]
Hacktivists	[700, 709, 774]
State actors	[701, 702, 781, 784]
7.2 The Elements of a Malicious Operation	
Affiliate programmes	[747, 787]
Infection vectors	[706, 788]
Infrastructure	[689, 789, 801]
Specialised services	[790, 791]
Human services	[758, 805, 806, 810]
Payment methods	[746, 792, 793]
7.3 Models to Understand Malicious Operations	
Attack trees	[62]
Environmental criminology	[815, 816, 817]
Modelling the underground economy as a flow of capital	[705]
Attack attribution	[818]

Chapter 8
Security Operations & Incident Management

Hervé Debar | Telecom SudParis

INTRODUCTION

The roots of Security Operations and Incident Management (SOIM) can be traced to the original report by James Anderson [834] in 1981. This report theorises that full protection of the information and communication infrastructure is impossible. From a technical perspective, it would require complete and ubiquitous control and certification, which would block or limit usefulness and usability. From an economic perspective, the cost of protection measures and the loss related to limited use effectively require an equilibrium between openness and protection, generally in favour of openness. From there on, the report promotes the use of detection techniques to complement protection. The next ten years saw the development of the original theory of intrusion detection by Denning [835], which still forms the theoretical basis of most of the work detailed in this KA.

Security Operations and Incident Management can be seen as an application and automation of the *Monitor Analyze Plan Execute-Knowledge* (MAPE-K) autonomic computing loop to cybersecurity [836], even if this loop was defined later than the initial developments of SOIM. Autonomic computing aims to adapt ICT systems to changing operating conditions. The loop, described in figure 8.1, is driven by events that provide information about the current behaviour of the system. The various sequential steps of the loop analyse the event stream (trace) to provide *feedback* to the system, changing its behaviour according to observations and policies, enabling automatic adaptation to best provide service for users. The developments of SOIM have increased in automation and complexity over the years, as a result of our increasing reliance on the proper service delivery of the ICT infrastructure. These developments have slowly covered most of the spectrum of the MAPE-K loop.

After nearly 40 years of research and development, the Security Operations and Incident Management domain has reached a sufficient maturity to be deployed in many environments. While early adopters were mainly located in ICT-intensive sectors such as telecoms and banking, it is finding its place in sectors that are increasingly embracing or converting to digital technologies. Yet, research is still very active in addressing the many remaining challenges. With respect to detection, new emerging environments driven by new technologies and services are requiring the acquisition and analysis of new data streams. The tools, techniques and processes available today for detecting and mitigating threats also regularly fail to prevent successful attackers from penetrating and compromising ICT infrastructures, without regular users noticing. Extremely large-scale events also occur at regular intervals, and there is a definite need for progress in terms of reaction to attacks.

The Security Operations and Incident Management knowledge area description starts by introducing some of the vocabulary, processes and architecture in section 8.1. It then follows the loop concepts, discussing detection at the sensor level, both looking at data sources (*Monitor*, section 8.2) and detection algorithms (*Analyze*, section 8.3). It then discusses Security Information and Event Management, instantiating *Analyze* from a more global perspective than sensors, *Plan* in section 8.4 and examples of *Execute*. Using the *Security Orchestration, Automation and Response* (SOAR) concept, it further develops the modern aspects of the *Plan* and *Execute* activities in section 8.5. Of course, all these activities are built upon a *Knowledge* base. Several knowledge components are described in section 8.6. The KA concludes with human factors in section 8.7.

CONTENT

8.1 FUNDAMENTAL CONCEPTS

[835, 837]

The SOIM domain assumes that the workflow of the MAPE-K loop is implemented in technical components, deployed in an ICT infrastructure. Section 8.1.1 establishes a few fundamental vocabulary references in the SOIM domain, and section 8.1.2 describes the deployment of these concepts in a generic ICT infrastructure.

8.1.1 Workflows and vocabulary

Figure 8.1 adapts the generic MAPE-K loop to SOIM. In addition to the ICT system being protected and monitored to detect attacks, two major actors influence the evolution of the loop; the Internet as a whole and the regulatory context in which the ICT system provides services. The Internet is the source of both service requests and threats, but also of intelligence about these threats. Regulatory bodies such as national agencies, and industry bodies provide additional threat and detection information and request information sharing.

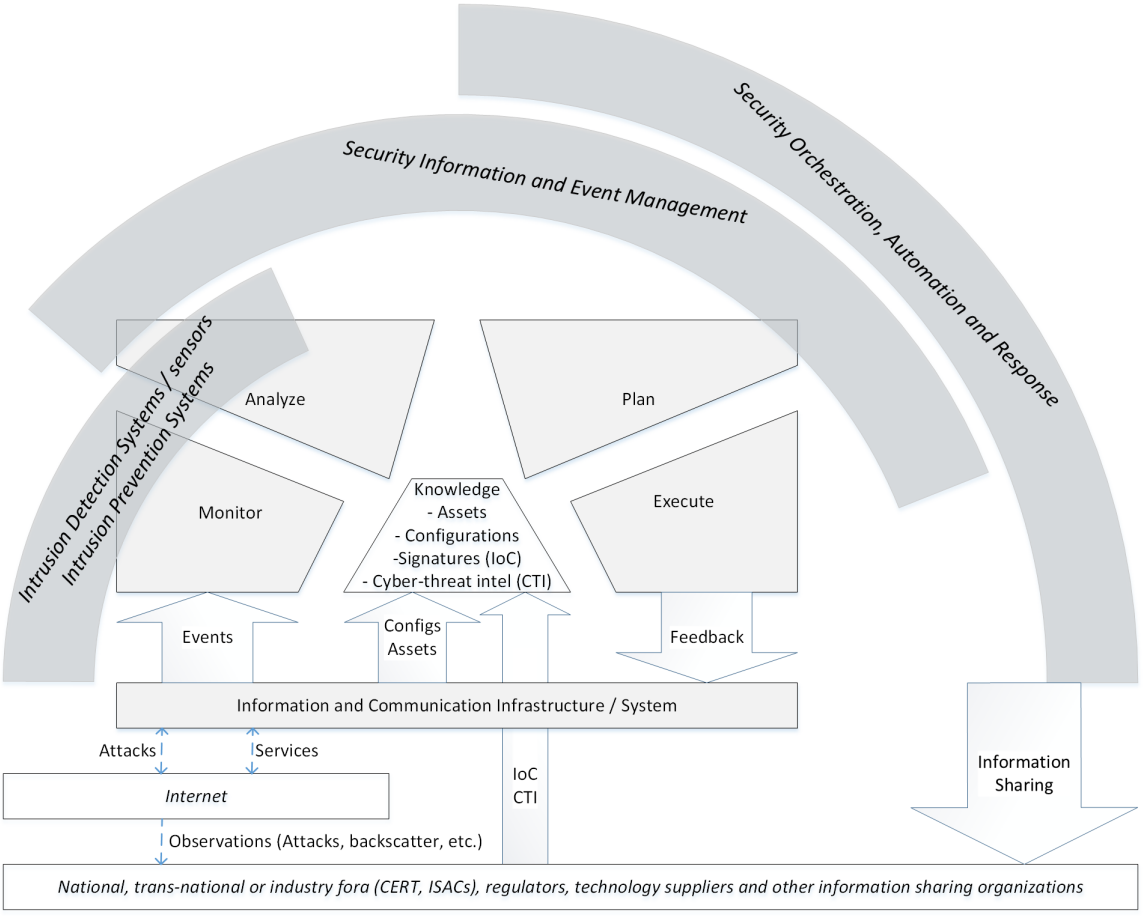


Figure 8.1: MAPE-K Autonomic computing loop instantiated to SOIM

Figure 8.1 illustrates the positions of the components that carry out the SOIM workflows, using three partial loops. The innermost one, *Intrusion Detection Systems* (IDS), was the subject of

the earliest work, covering monitoring and detection. The second one, *Security Information and Event Management* (SIEM) platforms, extended detection and started covering response planning and execution. More recently, *Security Orchestration, Automation and Response* (SOAR) platforms have driven further analytics and responses, enabling more advanced and global responses to cyberthreats. The knowledge base used in SOIM has gradually expanded over the years, as more intelligence has become necessary to detect and mitigate attacks. The key difference between knowledge and events is time. Events are produced and consumed, while knowledge is more stable.

The *Monitor* activity is essentially covered by IDSes. The various data sources included within the scope of monitoring are described in section 8.2.

The *Analyse* activity, also covered by IDSes, aims to determine whether some of the information acquired constitutes evidence of a potential attack. From 1990 to 2000, many research projects developed advanced Intrusion Detection System prototypes. As a result, the first network-based IDS was commercialised in 1996, automating the first part of the MAPE-K loop. However, section 8.3 illustrates that the constraints associated with real-time event processing and limited coverage require additional tools. This is the objective of the second loop, SIEM platforms.

Technology has evolved to a point where IDSes have been transformed into Intrusion Prevention Systems (IDPS) [838]. This is elaborated further in section 8.5.1. The text of the KA will use IDPS from now on, except when the concept is focusing on detection, where IDS will remain.

Plan activity is essentially the realm of SIEM platforms. The deployment of these IDS sensors created the need to manage operationally large volumes of alerts, which led to the development of these SIEM platforms. They provide both additional analysis and initial planning to respond to attacks. These large-scale, complex and expensive platforms are now consolidated in the *Security Operating Center* (SOC), providing both technological and human resources. We are now deploying the second generation of these SIEM platforms to accommodate increasingly large volumes of diverse data, and to provide additional processing capabilities.

Execute activity started being implemented in SIEM platforms mostly through manual processes. Security orchestrators or dedicated components are now enabling partial automation of feedback to the ICT infrastructure, although this activity is less mature than the others.

The first three (*Monitor*, *Analyse*, *Plan*) activities are now fully or partially automated. Automation is absolutely necessary to handle the huge amounts of event data generated by modern ICT systems, and to describe the huge body of knowledge related to cyberattacks. They all rely on a large body of knowledge, covering, for example, the configuration of a monitored system, or detection signatures of many types and forms. New trends are also emerging, for example, *Cyber-Threat Intelligence* (CTI) (section 8.6.3), to better understand and defend against cyberattacks. This is the topic of *Security Orchestration, Automation and Response* (SOAR), which aims to support better responses to threat, as well as more global information exchange. The SOAR acronym describes an increasingly required set of functionalities extending SOIM coverage for risk and incident management.

8.1.2 Architectural principles

Cybersecurity does not function in a vacuum. The Security Operations and Incident Management domain assumes that there is an ICT system to be protected. Thus, an SOIM deployment assumes a few general architectural principles on which tools and processes can be deployed. These concepts are described in figure 8.2.

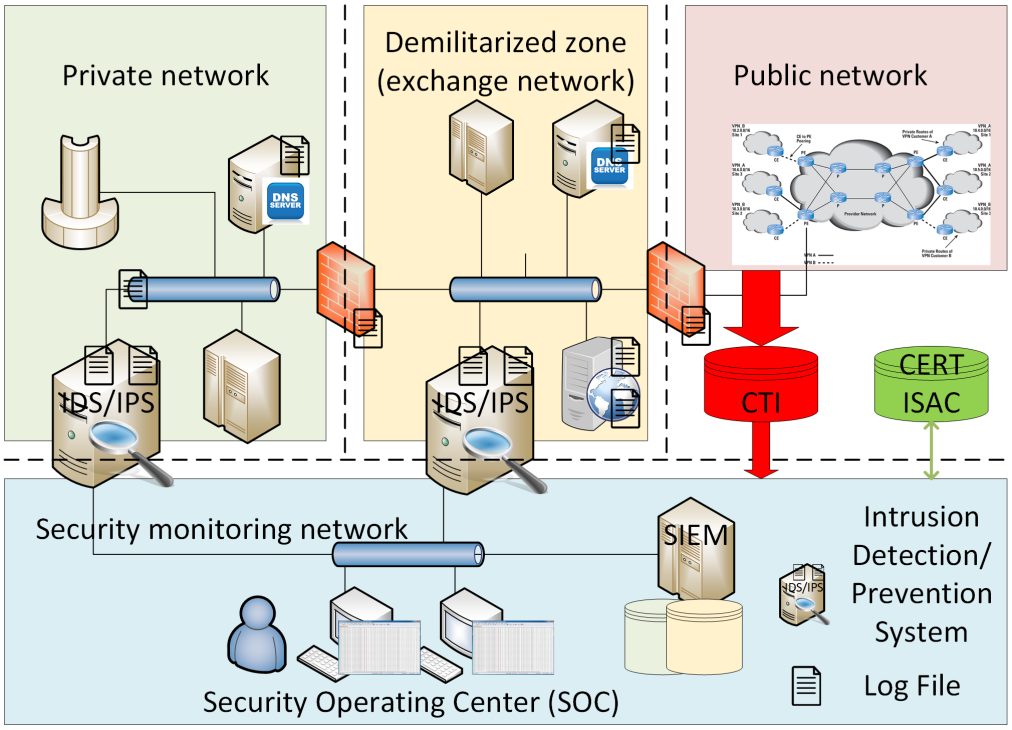


Figure 8.2: Simplified deployment of SOIM technologies in an ICT infrastructure

An Information System, connected (or not) to the Internet, is subject to attacks. Not all these attacks can be blocked by protection mechanisms such as firewalls. Best practices recommend defining zones of different sensitivities, to control the data exchange. This frequently and minimally takes the form of a Demilitarised Zone (DMZ) located between the inside private network and the outside Internet, to serve as communication termination, exchange and increased scrutiny through monitoring. To detect threats that are not blocked by protection mechanisms, operators deploy Intrusion Prevention Systems (IDPS). IDPS sensors can use system (section 8.2.5) or application log files (section 8.2.4), depicted as pages in figure 2. They can also be deployed at the network level (section 8.2.1), depicted as the two larger pieces of equipment with magnifiers.

The SOIM infrastructure is shown at the bottom of figure 8.2. The sensors often have at least two network attachments, an invisible one in the monitored Information System network for collecting and analysing data, and a regular one in a protected specific SOIM network infrastructure, where the SIEM is installed and receives the alerts. Analysts monitor consoles to receive alerts, assess their impact and deploy the appropriate mitigation actions. Sensor management might either use this secondary network attachement as a maintenance channel for software and signature updates, or use yet another mechanism such as a virtual private network to carry out the sensor maintenance.

The SOIM domain also implies processes, which are defined by the Chief Information Security Officer and followed by analysts. The first process is related to alert processing, where the op-

erator, with the help of decision support techniques provided by the SIEM, will decide to ignore the alert, react to it following procedures, or escalate the alert to skilled analysts for further analysis, diagnosis and decision. The second process is the deployment and maintenance of sensors, deciding on where to locate them, what to capture and how to maintain continuous monitoring. The third process is reporting, particularly crucial for managed services, where the functioning of the SIEM and SOC are analysed for improvement.

The Security Orchestration, Automation and Response components are included through the *Cyber-Threat Intelligence* (CTI, red) and *Information Sharing and Analysis Center* (ISAC, green) disks, representing the added benefit for the management platform to obtain information from external, relevant sources and to leverage this information to increase their detection efficiency (section 8.3) and impact assessment (section 8.5). While both interfaces provide information to a SOC, this information is of a fundamentally different nature. CERT and ISAC entities are trusted organisations, sometimes enabling sectoral information exchange, often established and governed by regulations. CTI is a much more fuzzy area, including open source intelligence as well as dedicated information feeds provided by commercial companies.

8.2 MONITOR: DATA SOURCES

[835, 839]

The detection issue is relatively simple; from a continuous stream of data, the objective is to detect localised attempts to compromise ICT infrastructures in real time. This is achieved first by collecting information about the operation of these ICT infrastructures from traces with many different origins.

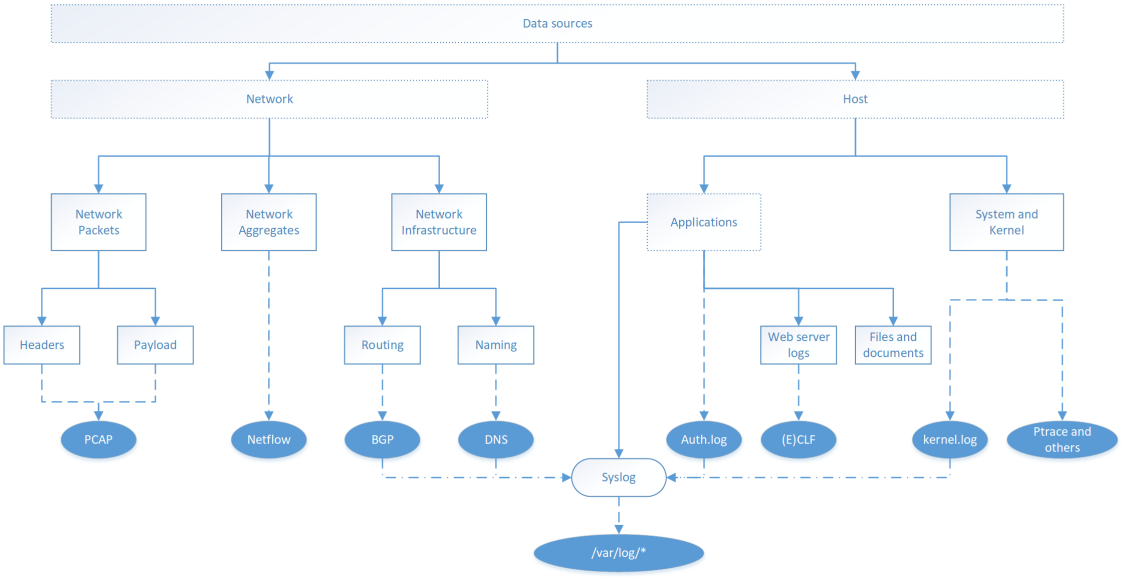


Figure 8.3: Data sources landscape

Figure 8.3 provides a simplified conceptual view of possible data sources. The rectangles describe concepts. The ovals describe concrete implementations of these data sources. The rounded rectangles describe an actual format, syslog, documented and standardised, which plays a specific role. Since it is a standardised protocol and format, it also supports log feeds provided by networking equipment, operating systems and applications.

Figure 8.3 is by no means complete. Many data sources have been considered over the years, depending on the requirements of the use case and the detection algorithms.

Data sources broadly describe either host behaviours reporting on operating systems or applications, or network behaviours reporting communication patterns.

Data sources are event streams, traces of activity that represent the services accessed by the users of an Information System. Data sources are inputs to sensors, which produce alerts as outputs. Alerts represent information of interest from a security perspective. In the general case, an event or a stream of events, acquired by a sensor, generates an alert that synthesises the security issue found by the sensor. Alerts are covered in section 8.4.1.

The move to external resources such as cloud providers or Internet Service Providers may limit the availability of some of the data sources, for practical reasons such as volume, or due to privacy constraints and entanglement of multiple customer data in the same trace. It is also possible that traces from hosted environments might be compromised or be made available without the knowledge or authorisation of the customer.

8.2.1 Network traffic

Network data have become the de-facto standard for collecting input data for intrusion detection purposes, because of the overall reliance on networks and the ease of use of standard formats. While the capture of packets is the most prevalent format, the scientific literature has also used other information sources for security. Network information is sometimes not available internally and it may be necessary to rely on Internet Service Providers, for example, to identify attackers' addresses and routes.

The most prevalent type of network traffic data is the full packet capture, exemplified by the libpcap library and the tcpdump and wireshark applications. The pcap library has been ported to many environments, and is widely available as open source, hence its success. Numerous datasets have been made available or exchanged privately as pcaps, for almost as long as intrusion detection research has existed and needs to be evaluated. While packet capture is widely used, it does not mean that this information is stored in sensors. Storing pcaps require an enormous amount of storage, hence pcap files are often reserved for research datasets or forensics purposes. Network-based sensors may offer the capability to store a few packets along with an alert when a detection occurs, generally the packet that triggered the detection and a few ones belonging to the same context (TCP, etc.) that appeared quickly afterwards. This capability is generally limited to misuse detection.

The pcap library requires the availability of a network interface that can be placed in so-called *promiscuous* mode, meaning that the interface will retrieve all packets from the network, even the ones that are not addressed to it. Also, there is no need to bind an IP address to the network interface to capture traffic. In fact, this is a recommended practice, to avoid interference. This means that, in general, packet capture can occur silently and is undetectable. Despite its popularity, there are a few issues with the pcap format that need to be considered when manipulating it.

Volume Pcap files tend to be extremely large for any practical operational use. This often limits capture to the investigation. Sensors generally analyse network traffic on the fly but do not record actual packets.

Packet size The default configuration of the library acquires only the beginning (headers) of

an IP packet. This means that a packet trace might be limited to only header information. An incomplete or missing packet payload strongly limits detection.

Segmentation and fragmentation Information circulated on the network is recorded on a per-packet basis. This implies that the receiving software must reconstruct the application-level data stream. Beginnings or ends of communications might be missing.

Timestamps Network packet headers do not include any timestamp. This is added by the capturing software and relies on an external clock.

MAC layer interpretation Capturing the MAC layer is possible, but requires a specific configuration. Interpreting of MAC layer information requires knowledge of the configuration of the network segment to which the collection network interface is attached. Capturing the MAC layer is required in order to detect attacks such as ARP poisoning. For certain types of industrial control networks which run directly on top of the Ethernet layer, capturing traffic requires adding a node and may break real-time assumptions.

Application layer interpretation The most crucial aspect of pcap analysis for cybersecurity is analysing the application layer. IP packets are relatively autonomous bits of data. Reliable transports, such as TCP, have inherent dynamics that need to be taken into account when analysing the data, such as the existence of a connection or not. At the application layer, inside the TCP/IP payload, information might be inconsistent with the headers, or require an understanding of application logic, which is often hard to acquire, understand and reproduce.

Encryption Encrypted traffic, and particularly TLS, is widespread. TLS ensures both the authentication of the server to the client, and the confidentiality of the exchange over the network. For monitoring, the issue is the second aspect, the impossibility to analyse the payload of packets. The classic approach to this problem is to put an additional dedicated box close to the application server (web, mail, etc.), often named the *Hardware Security Module* (HSM). The HSM is responsible for establishing the TLS session before the application server provides any content. This moves the load of establishing the TLS session outside of the application server. TLS-protected traffic is encrypted and decrypted at the HSM, and flows in clear to the server. This enables network-based IDPSes and WAFs to analyse the traffic.

Due to changing requirements, new network protocols have been introduced to support the Internet of Things (IoT). Low-power communication protocols such as LoRa have limitations in both packet size and the number of the packets that can be transmitted per day. These communication protocols are used mostly today as data harvesting on a large scale. Thus, IDPSes will need information about the context of the communication to provide useful detection. Isosynchronous protocols in use such as PROFINET IRT have stringent requirements in terms of communication cycle time and determinism. These protocols are typically used in manufacturing environments. As they mostly rely on hubs for communication, inserting a network-based sensor may seem easy. However, the strict timing requirements of such protocols require careful validation that the IDPS does not alter these requirements. Also, this necessitates the deployment of a second communication channel for the IDPS to send alerts to a SIEM, which may be costly, technically difficult and may introduce additional vulnerabilities to the system.

8.2.2 Network aggregates: Netflow

The sheer size of packet captures has created the need to obtain a synthetic view of network activity. This has created the need for a synthetic aggregated view of traffic at a relatively low layer. Network aggregates are mechanisms for counting packets sharing certain characteristics, such as source, destination, protocol or interface. These counts are performed by network equipment as packets cross their interfaces.

Netflow [840, 841] is a widely used network monitoring tool used for detecting and visualising security incidents in networks [842, 843]. In brief, this protocol records counters of packet headers flowing through router network interfaces. Initially developed by Cisco, it has been standardised as IPFix, RFC 7011.

As Netflow was developed by network equipment providers, it is extremely well integrated in networks, and widely used for network management tasks. It is standardised, and even though the commercial names differ, similar information is collected by the manufacturers supporting the technology. Its strongest uses are certainly visualising network communications and relationships, [842] and highlighting communication patterns. Visual analytics provide a user-friendly way of understanding anomalies and their impact. Hence, Netflow is also widely used for cybersecurity tasks.

Netflow, however, may suffer from performance degradation, both in terms of computation and storage. Handling packets to compute Netflow counters requires access to routers CPU (central or on interface boards). This significantly reduces the performance of network equipment. Newer routers are now able to generate netflow records at the hardware layer, thus limiting the performance impact. Another alternative is to span or tap a network interface and to generate the netflow records independentely of the routing equipment.

Originally, to limit the CPU performance impact, operators often deploy Netflow in sampling mode, where only one in every several thousand packets is analysed. Thus, the view recorded by Netflow might be extremely limited and may completely miss events that do not reach the scale of the sampling. Except for large-scale Denial of Service events, it is thus difficult to rely on sampled Netflow alone for security.

8.2.3 Network infrastructure information

The networking infrastructure relies on many protocols for proper communication. Two of its main components, the naming and the routing infrastructure, are also of significant interest for both attacks and detection. Reporting on routing or naming operations requires direct access to a view of the infrastructure. Operators who participate in routing and naming usually rely on syslog to collect information.

8.2.3.1 Naming

The *Domain Name System* (DNS) is one of the most crucial services on the Internet. It resolves domain names, meaningful bits of text, to IP addresses required for network communications but which are difficult to remember. In addition, naming is required for the *Transport Layer Security* (TLS, RFC 8446) protocol and certain HTTP mechanisms such as virtual hosting.

Despite its importance, DNS has been the subject of many vulnerabilities and attacks. The main problem with DNS is its lack of authentication in its basic form. An attacker can thus steal a domain through fake DNS messages or responses. The deployment of DNSSEC offers an authenticated response to DNS queries that will provide users with evidence of domain name ownership.

The DNS protocol is also a natural DDoS amplifier, as it is possible for an attacker to mimic the IP address of a victim in a DNS request, thus causing the DNS server to send unsolicited traffic to the victim [844, 845]. Unfortunately, the current move to DNSSEC is unlikely to be able to help [846, 847].

Another issue related to DNS is the detection of botnet activity. Once a malware has infected a computer, it needs to communicate with the C&C server to receive orders and carry out the requested activity. While it is not the only C&C communication channel used by bot herders, DNS is attractive as a communication channel for attackers because it is one of the few protocols that is highly likely to go through firewalls, and whose payload will be unaltered. In order for this to work, attackers need to set up, and defenders need to detect malicious domains [848]. The most common defence mechanism is DNS domain name blacklists, but its efficiency is hard to evaluate [849]. This blacklist defence mechanism can also be extended to other C&C channels.

Note that DNS is not the only protocol to be prone to DDoS amplification attacks. NTP is also a frequent culprit [850]. More information about DDoS attacks can be found in [851].

8.2.3.2 Routing

Another related source of information for attacks is routing information. Incidents in the border gateway protocol routing infrastructure have been studied for some time [852, 853], but many of the recorded incidents are due to human error. There are recorded instances of malicious BGP hijacks [854, 855], but the effort required by attackers to carry out these attacks seems, at this point in time, not be worth the gain.

8.2.4 Application logs: web server logs and files

Higher up the computing stack, application logs provide an event stream that documents the activity of a specific application. The main advantage of application logs over system logs is their similarity to reality and the precision and accuracy of the information proposed. These logs were initially created for debugging and system management purposes, so they are textual and intelligible.

Applications can share log files through the syslog infrastructure (section 8.2.6). For example, the *auth.log* log file will store user connection information regardless of the mechanism used (pam, ssh, etc.).

8.2.4.1 Web server logs

A frequent source of information is provided by web server and proxy logs, known as the *Common Log Format* (CLF) and *Extended Common Log Format* (ECLF). This format is a de-facto standard provided by the Apache web server and others. While it is very similar to Syslog, there are no standards documents normalising the format. At this stage, the W3C standard for logging remains a draft document. This format is extremely simple and easy to read. It provides information about the request (the resource that the client is trying to obtain) and the response of the server, as a code. Thus, it has been widely used in Intrusion Detection Systems over the years. The main issue with the format is the lack of information about the server, since the log file is local to the machine generating the log.

As server logs are written once the request has been served by the server, the attack has already occurred when the sensor receives the log information. Thus, this information source does not satisfy the requirements of *Intrusion Detection and Prevention Systems* (IDPS), which need to be hooked as interceptors to act on the data stream (packet stream, instruction stream), to block the request or modify its content.

8.2.4.2 Files and documents

Another source of application-level information that is particularly interesting and can be found both in transit (in networks) or at rest (in systems) comprises the documents produced by some of these applications. The introduction of rich document formats such as PDF, Flash or office suites, not to mention the rich HTML format used in mail exchanges today, has created a wealth of opportunity for attackers to include malware. Exchanged over the web or via email, they constitute another trace of exchange that can reveal malicious code embedded in these documents, such as macros or javascript.

Parsing information in documents, both simple ones such as TLS certificates or complex ones such as PDF, is complex and provides attackers with a wealth of opportunity to create different interpretations of the same document, leading to vulnerabilities and malware. At the same time, it should be acknowledged that the rich document formats are here to stay and that rich (and thus complex) specifications such as HTML5 need to be well written so that they can be unambiguously interpreted, thus leaving less room for attackers in the specification itself.

Using documents as a data source is increasingly required for malware detection.

8.2.5 System and kernel logs

The earliest ‘intrusion detection’ paper by Denning [835] already included in the model the generation of an audit trail by the system being monitored. Operating systems generally provide logs for debugging and accounting purposes. These logs were exploited in early designs such as Haystack. However, Denning has already stated that most system logs are insufficient for intrusion detection, as they lack the required precision. For example, the Unix accounting system records only the first eight characters, without a path, of any command launched by a user. This makes it impossible to differentiate commands with identical names at different locations, or long command names.

Another trend pursued by intrusion detection researchers and operating system designers was the creation of a specific audit trail to generate a trace of privileged user activity, as required

by the *Orange Book*. This led to the development of more precise host-based IDS such as STIDE and eXpert-BSM. These specific system traces are acquired through the interception of system calls, which represent the transition between regular program execution and request to protected kernel resources. This is typically implemented using a dedicated audit trail, as specified in the *Orange book*, or kernel/processor debugging accesses such as ptrace for Linux. However, the complexity of the specification led to divergences in the implementation of the audit trail by the different operating system vendors. It also imposed such a performance penalty to program execution that it became impossible to operate ICT systems with the audit trail being activated. It therefore became of little use and was quietly removed from most operating systems. This factor has prevented the emergence of a standard system audit trail, even in certified operating systems.

Kernel logs now focus on monitoring the internal operations of an operating system, close to the hardware. They have also greatly diversified, targeting a broad range of devices. They have been integrated in the commercial world under the term 'endpoint protection', which has become a generalised term for antivirus engines. This addresses the general problem of protecting not only the system but also the applications, such as the browser or the mail client, which not only exchange data but also execute untrusted code provided by external sources. They rely on dedicated interceptors that capture only the activity that they are interested in analysing. This solves the main issue of this data source, a very fine granularity that ensures everything is captured, but makes analysis and detection very difficult, as it is hard to link the assembly code being executed on the processor with programs and information that a user or analyst can easily understand and react to. Malware is the subject of the Malware & Attack Technologies Knowledge Area (Chapter 6), and in the context of SOIM malware detection engines and endpoint protection tools are considered sensors.

Other logs provide higher level information, such as a report of the boot process on Unix machines, or on the main kernel activity. These logs often rely on a Syslog infrastructure, as described in section 8.2.6.

8.2.6 Syslog

As already mentioned in this section several times, Syslog provides a generic logging infrastructure that constitutes an extremely efficient data source for many uses.

The initial source for these logs is the Syslog protocol, introduced in BSD Unix, retro-specified from existing implementations by RFC 3164.The current specification of Syslog is provided by RFC 5424.This new specification introduces several improvements over the original implementation.

A Syslog entry is a timestamped text message coming from an identified source. It contains the following information in this order:

Timestamp The date and time of the event creation, usually in text format with a resolution up to the second.

Hostname The name of the equipment generating the log. It might be a fully qualified name or an IP address, or the *localhost* for the local machine. Using IP addresses in private ranges or localhost may induce errors when consolidating logs.

Process The name of the process (program) generating the log.

Priority The priority (category and severity, computed according to a standard formula) of the log. In practice, it is often summed up according to severity on a scale of 0 (system panic and crash) to 7 (debugging information).

PID The process ID of the process generating the log.

Message An ASCII 7-bit message qualifying the information, provided by the developer of the application.

Syslog also uses the notion of facility to categorise and orient logs. This information is aggregated in different files, usually in the `/var/log/` directory in Unix systems.

Syslog is also a protocol, running on UDP/513. This facilitates transmission, as UDP can be resilient to difficult network conditions and lose a few messages without losing the capability. However, using UDP often requires the segmentation to be limited, thus a suggested limit of a Syslog message's size is often around a thousand bytes. Many systems include a standard programming interface to implement calls to Syslog in applications.

As a text message, Syslog is extremely useful. Many, if not most, heavy SOC implementations rely on Syslog to centralise both events and alerts. This use of Syslog is covered in section 8.4.1.

8.3 ANALYSE: ANALYSIS METHODS

[835, 837, 839, 856]

Collected traces are analysed according to different strategies that aim to separate 'good' events from those that indicate attacks. The fundamental work of Denning [835] already defined the two families of data analysis techniques that have been researched, developed and commercialised over the years. Misuse detection, detailed first, aims to characterise malicious behaviours present in the traces in order to send an alert when the set of malicious behaviour events is recognised in the traces. Conversely, anomaly detection aims to characterise 'normal' behaviour, and sends an alert when events in traces are not associated with normal behaviours. In both cases, a large number of algorithms have been described in the scientific literature. A few of these algorithms have been applied both to misuse and anomaly detection.

In SOIM processes, and as shown in figure 8.1, analysis is performed by two components, the sensors and the SIEM platform. Figure 8.4 refines this process. The monitored Information System generates traces representative of activity, as log files or through dedicated IDPS appliances or software (shown as looking-glass-boxes and files in figure 8.2). One or several events in each trace may trigger the generation of an alert by a sensor. Several of these alerts, possibly coming from several sensors, may be assembled by the SIEM in incidents that need to be handled by operators.

In this section, the KA addresses the transformation of events in alerts, that may characterise malicious activity. In section 8.4, the KA addresses the transformation of alerts in incidents.

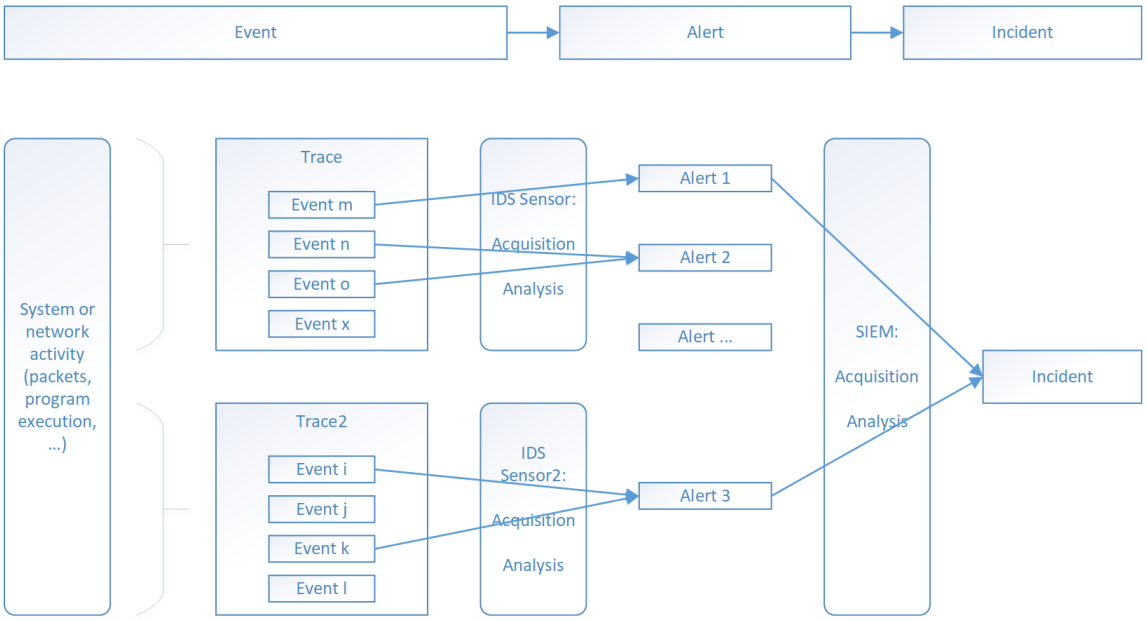


Figure 8.4: Analysis: from event to alert to incident

8.3.1 Misuse detection

Misuse detection leverages the vast body of knowledge characterising malicious code and the vulnerabilities that this malicious code exploits. Software vulnerabilities, particularly in the Common Vulnerabilities and Exposures (CVE) nomenclature, are particularly relevant for this approach, but misuse detection has a broader reach. A misuse Intrusion Detection System seeks evidence of known malicious events in the trace, and alerts when they are found, informing the analyst about the specifics of the vulnerability exploited and its impact.

The earliest Intrusion Prevention Systems in this area are antivirus engines, which capture execution traces such as system calls, library calls or assembly, identify known malicious patterns using so-called *signatures* that describe these malicious codes, and quarantine the associated container. The IDPS thus seeks exploits, very specific instances of malicious codes represented as bitstrings.

Modern malicious code has evolved complex mechanisms to avoid detection, and modern anti-malware tools have become extremely complex in response, in order to create more efficient representations of exploits and vulnerabilities. More recently, researchers have proposed more generic signatures, to attempt to capture malicious behaviour more generally [839]. Also, the emergence of sandboxes and tainting [857, 858] has enabled newer detection and protection methods that can detect malware despite obfuscation and polymorphism. The risks of generic signatures are, of course increased false positives and increased difficulty in understanding the precise attack.

Another branch of system analysis is UNIX system analysis, exemplified by the Haystack and NIDES prototypes. These prototypes aimed to create high-level audit trails for analysis. The canonisation aspect of the data had a significant impact on detection performance, and the current state of the art is focusing on assembly and binary language analysis for detection.

From a network perspective, an IDPS seeks evidence of malicious activity in multiple forms. The malicious code can be found in the packets' payloads. Malicious code can also exhibit specific network activity related to command and control, access to known addresses or

to known services. The best known network-based misuse Intrusion Detection System is probably Snort [859]. Snort's signature language is simple and was a de-facto standard for describing attack patterns, before being superseded by YARA. The initial version relied only on string matching, which made it sensitive to false positives [860]. The Suricata IDS, using the same signature language but newer implementation technologies [861], is also being used in research and operations.

The key advantage of misuse detection is the ability to document the cause of the alert, from a security perspective. This helps the analyst decide how to further process the alert, particularly its relevance and its impact on the monitored system. The key difficulty of misuse detection is the process of creating signatures, which requires time, expertise and access to the proper vulnerability information. Frequent signature updates are required, mostly to take into account a rapidly evolving threat environment, but also to take into account errors in the initial signature, or new Indicators of Compromise which were not initially detected.

8.3.2 Anomaly detection

Anomaly detection is a fundamental tool for detecting of cyber attacks, due to the fact that any knowledge about the attacks cannot be comprehensive enough to offer coverage. Anomaly detection is a domain where not only research has been extremely active, but there are several thousand patents that have been granted on the topic.

The key advantage of anomaly detection is its independence from the knowledge of specific vulnerabilities. This theoretically enables the detection of 0-day attacks, provided that these attacks effectively show up as deviations in the traces. Also, these methods are often computationally fast, which enables them to keep pace with the increasing volume of traces to be processed.

However, pure statistical methods highlight anomalies that are hard to understand and qualify for analysts. The lack of precise diagnosis, and of a clear link to security (instead of an anomaly related to another cause) requires an in-depth understanding of both the monitored system and the detection process, which is hard to combine. Thus, anomaly detection, while heavily marketed, must be operated with caution as a first line of detection, because it requires strong domain knowledge to transform a diagnosed anomaly into actionable defence. Applied to alert streams, which are richer in information, anomaly detection is often more successful in SIEMs and is implemented in newer SIEM platforms such as the Elasticsearch-Kibana-Logstash stack or commercial tools such as Splunk.

Anomaly detection was included in Denning's model [835] from the start, and has consistently been developed over the years [856, 862, 863]. As the difficulty of creating attack signatures became more significant, IDPS vendors also included these models in their products.

8.3.2.1 Models

Anomaly detection relies on the definition of a model against which the current observations of the trace are evaluated. Very early researchers proposed behaviour models to detect deviations from the norm. However, the statistical models developed in early IDS prototypes such as Haystack and NIDES were not accurate enough to detect skilled attackers. Therefore, more complex models have been developed over the years.

In network anomaly detection, the model must first define whether it will look at multiple data points or compare a single data point to the model. A data point could be a packet, or a complete connection. Models can also correlate between connections to detect more complex attacks spanning multiple packets. An example of this kind of behaviour is the correlation between web traffic and DNS traffic. When using regular browsers, the user is likely to perform a DNS request before accessing a website; an anomaly could manifest itself if the web request directly addresses the website through its IP address. Of course, in this case caching phenomena must be taken into account.

Another interesting aspect of network anomaly detection is the definition of the technique. Unsupervised techniques look at outliers, creating clusters out of the data and using a distance to determine outliers that cannot be covered in clusters. In this technique, the selection of features, that become the coordinates for each data point, is critical. Feature combinations must effectively differentiate between normal behaviours and attacks. Frequent methods for creating clusters and measuring distance include k-nearest neighbors or the Mahalanobis distance. Supervised anomaly detection techniques use labelled features to create optimal clusters. Support Vector Machines or C4.5 are frequently used for this task.

Graph-based models represent the structure of the network and of the communication paths. They enable a representation of network behaviour that highlights changes in communication patterns. These techniques also offer attractive vizualisation capabilities, that enable operators to weight the exchanges between various parts of their networks, to identify anomalous communication patterns, and then to dig further in to qualify the anomaly as security relevant or not.

The choice of an anomaly model is extremely important. In fact, many publications related to anomaly detection are made in thematic venues such as statistics, signal processing or information fusion, outside of the cybersecurity domain. This field of study is thus extremely rich and fundamentally multi-disciplinary.

8.3.2.2 Specification versus learning

A prevalent form of anomaly detection is specification-based detection. An attack is considered to be a breach of the specification of a system. The key issue in this approach is to obtain a specification that can be reliably recognised in the traces. This approach was initially developed for network-based IDPS, such as Bro [650], which was developed at around the same time as Snort, but follows a radically different approach. Bro is built up as a stack of protocol analysers, checking at each layer the coherence of the captured information with the standards, in this case the RFCs.

Further development of specification-based detection is expected in industrial control networks [864], where specifications are much more precise and enable the detection of perturbations. In these networks, the behaviour is much better specified, because of the underlying control loop of the physical process that is piloted by networked controllers. This also creates

additional regularity that can be picked up by anomaly detection algorithms. Also, the system specifications are more accurate.

Alternatively, supervised learning is used to create models when ground truth is available, or unsupervised learning to let models self organise. In both cases, it is frequently necessary to select a threshold that will separate data points considered normal from those considered outside of the model. The application of machine learning techniques for detection is further developed in section 8.3.4.

8.3.2.3 Adherence to use cases

An important point on anomaly detection is its adherence to a use case, possibly even a specific deployment. Network anomaly detection has been broadly applied to TCP/IP networks initially, and has over the years focused on new applications, covering ad-hoc networks, sensor networks and, more recently, industrial control systems [864]. Malware anomaly detection has also evolved from personal computers to web malware to Android malware today [865].

This adherence to a use case is important for creating the model, validation and testing. It requires that from the start, operators understand the behaviour of their systems and have sufficient business domain knowledge to understand why and how anomalies manifest themselves, and what their significance is with respect to cybersecurity. Specific care must be taken to associate the detection of anomalies with as much domain knowledge as is possible to diagnose and qualify the anomaly. Equipment roles imply different behaviour models, and thus different qualifications for anomalies.

This adherence to use cases also prevents the definition and qualification of generic behaviour models. Therefore, operators deploying anomaly detection systems must prepare for a period of testing and qualification. It is also likely that new systems, new services, or upgrades to existing systems or services, will perturb existing models and require re-qualification.

8.3.3 Blended misuse and anomaly detection

In practice, it is very hard to separate anomaly detection and misuse detection, as they are often intertwined in current sensors. For example, it is extremely useful to pre-filter input data before applying misuse detection. The pre-filtering performed on a packet stream follows the TCP/IP specification, for example. When a network-based misuse-detection sensor such as Snort [859], Suricata [861] or Bro [650] processes a packet stream, it verifies that the packet headers are correct before applying more complex detection processes such as signatures. This not only increases efficiency but also prevents false positives when a signature pattern is found in the wrong traffic context [860], for example, when a packet circulates over the network but the TCP session has not been established.

A similar approach can be applied to IDSes using application logs [866, 867]. This approach organises both misuse and anomaly detection in order to leverage the strengths of both approaches and limit their drawbacks. It also leverages the specifications of the application protocol to understand not only the syntax of the trace but also its semantic, in order to propose a better diagnosis.

8.3.4 Machine learning

Another, more subtle, way of mixing anomaly and misuse detection is using machine learning techniques, and particularly supervised learning, which requires ground truth. Machine learning basically associates an output class with a characteristics vector presented at the input. If the machine learning algorithm requires a definition of the different classes to which it assigns the input, then the definition of the output classes (for example, normal and attack) in itself enables mixing anomaly and misuse detection.

Machine learning, in many forms, has been applied to anomaly detection, and particularly in the network domain to the infamous Lincoln Lab/KDD dataset [868]. There are so many research papers presenting the use of support vector machines, C4.5, random forest, that one can only reference the best survey published so far by Chandola et al. [856]. There has also been a lot of work looking at Internet traffic classification [869]. Another study looks at the aspect of pre-processing network traces for anomaly detection [862]. This is a crucial operation, as shown by the failure of the KDD dataset, as it may either remove artefacts that are necessary for detection, or introduce new ones that create false positives, as discussed in section 8.3.5.

On the system and application side, there has been a lot of work on using machine learning for malware detection, both at the system call level [870], at file system [871] or for PDF files [872, 873]. Gandotra [874] lists many relevant approaches of applying machine-learning techniques to malware analysis, principally looking at whether they rely on static analysis (the file) or on dynamic analysis (the behaviour). Also, the recent development of the smartphone ecosystem [875], Android and its rich ecosystem of applications, with the associated malicious code, has created significant interest in Android malware detection.

Looking further afield, there is increasing interest in using machine learning and artificial intelligence for cybersecurity, as shown by the DARPA Cyber Grand Challenge. One can expect equal interest from attackers and thus the emergence of adversarial machine learning where, as shown for the specifics of Neural Networks, attackers can introduce irrelevant information to escape detection or to make it harder.

8.3.5 Testing and validating intrusion detection systems

One of the key issues for Intrusion Detection System designers is testing and validating their tools. This issue has been around for a long time in the research community, as exposed by an early paper on the topic by McHugh [868].

The detection problem is a classification task. The evaluation of an IDS therefore compares the output of the detector with the ground truth known to the evaluator, but not to the detector. *True Negatives* (TN) are normal events that exist in the trace and should not be reported in alerts by the detector. *True Positives* (TP) are attack events that should be reported in alerts by the detector. As detectors are not perfect, there are two undesirable measures that quantify the performance of a detector. *False positives* (FP), also known as false alerts or type I errors, are defined as an attack that does not exist in the trace, but is reported by the IDS. *False negatives* (FN), also known as miss or type II errors, are defined as an attack that exists in the trace, but has not been detected by the IDS.

The first issue is to define the criteria for detection. In misuse detection (section 8.3.1), the IDS developer must define a set of attacks that he wants to detect and create the set of signatures that will detect them. The issue with testing is then to create traces that will trigger

signatures on behaviours that are considered normal (FP), or to launch attacks in a way that compromises the system but is not recognised by the IDS (FN).

In anomaly detection (section 8.3.2), the IDS developer must define normal behaviours. As most anomaly detectors use machine learning approaches, this means that the developer must obtain one or several datasets of significant size, possibly labelled. These datasets should, for some or all of them, include attack data. The detector is then trained on part of the datasets, and its performance evaluated on the others. For parametric and learning algorithms, several trials should be performed to obtain an average performance. Determining FP and FN also relies on the availability of reliable ground truths associated with the datasets.

Generating datasets, as already mentioned, is very difficult. The most commonly used one, the Lincoln Lab/KDD dataset, suffers from several of such issues which are good examples [876]. For example, the process by which the attack and normal traffic were generated (manual versus simulations) created obvious differences in the packet's *Time To Live* (TTL) and session duration. These features, which are not normally distinguishable in operations, tend to be picked up by learning algorithms, inducing a significant bias in the process with respect to TP. Another example is the lack of distinguishing features in the SNMP traffic, which leads to large FN rates.

The second issue is how to determine and present the actual success criteria of an IDS. From the raw TP, FP, TN, FN values, detectors are often evaluated on two metrics, *Precision* and *Recall*. Precision (equation 8.1) measures the fraction of real alerts in all alerts. This, in short, measures the usefulness of the alerts.

$$Precision = TP/(TP + FP) \tag{8.1}$$

Recall (equation 8.2) measures the fraction of real alerts over all the relevant information present in the ground truth. Thus, recall evaluates the completeness of the detection. An unavailable or incomplete ground truth may limit its usefulness.

$$Recall = TP/(TP + FN) \tag{8.2}$$

Several other metrics are reported in the literature, but these two must be considered the minimum information provided for evaluation. Another relevant aspect of evaluation is the fact that detection algorithms require the operator to select the parameter, such as thresholds or numbers of clusters. Setting these parameters strongly influences the performance of the sensors. Thus, it is a good practice to evaluate the performance of a detection algorithm using *Receiver Operating Characteristic* (ROC) curves to explicitly present the relationship and trade-off between FP and FN. A gain in one direction often decreases the performance of the other.

Depending on the detector and definition, the actual values computed during the evaluation of the detector may vary. For example, it might be sufficient for a detector to find and report one attack event in the trace to consider it a TP, even if the attack consists of many events. Conversely, another evaluator may require the IDS to highlight all the malicious events in a given attack to consider it a TP. Again, the experimental validation process should be extremely detailed and peer-reviewed to ensure that it does not contain any obvious errors.

Another issue is the operational qualification of the IDS. Albin [861] compares Snort and Suricata, both on synthetic and on real traffic. Synthetic traffic provides the evaluator with

access to the ground truth, thus enabling him to effectively compute FN and FP. When testing on real traffic, the evaluator may be able to approximate the FP better because real traffic artefacts are always likely to trigger cases that the IDS has not encountered during validation. This process, however, does not support evaluating FN. As evaluation is the basis for certification, it is no surprise that Intrusion Detection Systems are generally not certified at any security level.

8.3.6 The base-rate fallacy

One of the fundamental problems of intrusion detection is the *base-rate fallacy* formalised by Axelsson [837]. The problem stems from the fact that there is a large asymmetry between the number of malicious events and the number of benign events in the trace.

The general hypothesis followed by Axelsson is that there are few attacks per day. This may not be true anymore, but an ICT system flooded with attacks is also unrealistic, unless we are concerned with Denial of Service. Also, in the case of DDoS, malicious packets far outnumber normal traffic, so the asymmetry is reversed, but still exists. In Axelsson's case, it comes from Bayes' theorem that the probability of detecting an actual attack is proportional to the false alarm rate FP.

In essence, the base-rate fallacy must be addressed by IDS sensors that rely on processing large amounts of data, which is typically the case for machine-learning-based anomaly detection.

While this may sound like a theoretical issue, it has crucial implications with respect to human operators in front of a SIEM console, who have to deal with thousands of alerts, most of which are 'false'. There is thus a significant risk of missing an important alert and thus an incident. This risk is even higher in MSSP settings, where operators have a limited amount of time to process alerts. The usual process for solving this is to limit the detection to the most relevant elements. For example, it is not necessary to look for attacks against a windows server when the monitored server is running the Linux operating system. This tuning of the detection range can happen either before detection, by removing irrelevant signatures in the IDS, or after the fact in the SIEM by entering the proper correlation rules. The detection tuning approach has, however, encountered limitations in recent years, because cloud platforms are more dynamic and likely to host a variety of operating systems and applications at any given point in time. It then becomes harder to ensure proper coverage of the detection.

8.3.7 Contribution of SIEM to analysis and detection

From the Analyse perspective, a SIEM aims to provide further information about malicious activity reported by sensors.

Due to the event volume and real-time nature of the detection performed by IDS sensors, these sensors usually look at a single information source in a specific location of the ICT infrastructure. Therefore, it is difficult for them to detect large-scale or distributed attacks. Therefore, the centralisation of alerts, which is the initial central characteristic of SIEM platforms, as described in section 8.4.1, enables additional detection algorithms that may indicate attacks or anomalies that have not been significantly indicated by sensors, but whose properties when aggregated are significant.

8.4 PLAN: SECURITY INFORMATION AND EVENT MANAGEMENT

[877]

security information and event management form the core of the contribution to the *Plan* activity of the MAPE-K loop, the bottom (blue) part of figure 8.2, and the left-hand part of figure 8.4 (transforming alerts in incidents). It should be considered a decision support system and, as such, covers the Analyse and Plan activities. From a Plan perspective, the SIEM platform aims to define the set of actions that can be performed to block an attack or mitigate its effects.

The fundamentals of Security Information and Event Management can be traced back to December 1998, at a meeting organised by DARPA. The original goal was to enable a comparison of the performances of the various intrusion detection research projects that DARPA was funding, and this delivered several works, the Lincoln Labs/KDD dataset [878], the critique by McHugh [868] and, much later on, the three requests for comment that formalised the SIEM domain, the requirements (RFC 4766 [879]), the alert message format *Intrusion Detection Message Exchange Format* (IDMEF) (RFC 4765 [880]) and the *Intrusion Detection eXchange Protocol* (IDXP) (RFC 4767 [881]).

8.4.1 Data collection

The first objective of a SIEM platform is to collect and centralise information coming from multiple sensors into a single environment. Several issues need to be addressed to make this happen.

First of all, there must be a communication channel between the sensors providing the alerts and the SIEM platform. This communication channel must be strongly protected, because sensitive information may be included in the alerts. It must also be properly sized so that there is sufficient bandwidth to carry the required information. As sensors often have limited storage capabilities, the availability of the link is essential.

Secondly, the SIEM must be able to interpret the information provided by the sensors in a coherent manner. Given the wide range of available data sources and detection methods, this requires a lot of work to match the information from the alerts with the SIEM internal data formats. The general approach of a SIEM platform is to define a single data structure for the alerts, often a single database table. This means that the database contains many columns, but that inserting an alert often results in sparse filling of the columns.

Data collection is generally handled by the SIEM platform, benefiting from hooks from the sensors. SIEM platform vendors generally define their own connectors and formats, handling both the issue of transport security and of data import at the same time.

Classically, communicating an alert message requires the definition of three layers:

Schema The schema defines the structure of messages and the type and semantic of the attributes. It also includes the definition or use of dictionaries. Many alert schemas, for example, rely on CVE to document attacks.

Encoding The encoding defines how the messages and attributes are encoded to form a bitstring. Examples of textual format include Syslog, JSON XML or YAML. Examples of

binary formats include BER, CER or BSON. Textual formats are usually easier to process because they can be read directly by humans. Binary formats are more compact, which eases storage and transport.

Transport protocol The transport protocol describes how the alert bitstring is moved from one place to another. Examples of transport protocols include Syslog, IDXP, HTTP or AMQP. Transport protocols typically take care of the access control, confidentiality, compression and reliability of the communication.

Table 8.1 provides a factual analysis of frequently used alert message formats. The first two, CEF and LEEF, are proprietary formats of commercial SIEM vendors, but whose specification is at least partially open for analysis. The next two formats (CIM and CADF) have been specified by the DMTF, but not specifically for cybersecurity purposes. Nevertheless, they have been used to convey alerts. The last two have been specifically designed with the purpose of standardising the transmission of events or alerts. The text in *italics* indicates that the specification does not force a specific technology. However, when the specification, although generic, includes a proposal, this text is in *(brackets)*.

Format	Owner	Transport	Encoding	Structure	Number of attributes (keys)
CEF	HP/Arcsight	Syslog	Key/value	Flat	117
LEEF	IBM/QRadar	Syslog	Key/value	Flat	50
CIM	DMTF	*Any*	*(XML)*	UML	58
CADF	The Open Group, DMTF, (NetIQ)	*Any*	*(JSON)*	Classes with common attributes	48
CEE	MITRE	*(Syslog)*	JSON, XML	Structured: CEE event model, CEE profile	56
IDMEF	IETF	IDXP	XML	UML	166

Table 8.1: Formats characteristics summary

The flexibility of textual encodings enables large-scale deployment, and as such is the only one presented in table 8.1.

Syslog (RFC 5424) is the de-facto standard for SIEM platforms alert acquisition, as it is widely available, easy to understand and parse, and quite reliable. When using UDP, there is no transport-layer security. There is no guarantee of message integrity or delivery. Yet, in practice, it is very successful and scalable. Its drawback is the limitation of its schema (timestamp, origin and ASCII text string) and the size of the message (practically limited to 1000 bytes). Syslog is widely used by network operators or for large systems such as the Olympic Games.

CEF The *Common Event Format* is the proprietary exchange format of the Arcsight SIEM platform. It is oriented towards the expression of security relevant events and includes the essential information required to describe them. This format is representative of the flat structures used in SIEM platform databases. While it has a large number of attributes, some are not sufficiently documented for use.

LEEF The *Log Event Enhanced Format* is the proprietary exchange format of the QRadar SIEM platform. It focuses on network security events, and as such is not as rich as CEF.

CIM The *Common Information Model* is a standard of the *Distributed Management Task Force* (DMTF). It is widely used for managing distributed systems. As it is very generic, its expressiveness for cybersecurity events is limited.

XDAS/CADF The *Cloud Auditing Data Federation* is still being developed, initially as XDAS, and discussions are ongoing with DMTF to include it in CADF. It focuses on system events and cloud environments.

CEE The *Common Event Expression* was initiated by the MITRE corporation as a standard format for log files in computer systems. It was developed in collaboration between US governmental entities and SIEM vendors. It clearly separates the message format (CEE event Model or Profile), encoding (CEE Log Syntax) and transport (CEE Log Transport). Unfortunately, the work on CEE has stopped.

IDMEF The *Intrusion Detection Message Exchange Format* [880] is an informational document from the IETF. It does not specify a standard, and as such its adoption by the industry has been very limited. It is seen as complex, and in fact the specification is large in size. The IDMEF specification attempts to be very precise and unambiguous, which is shown in the number of attributes, the largest of all the considered formats. This difference in expressiveness is probably even greater, as the use of dictionaries (enumerated types) in the IDMEF UML design further increases its ability to represent information. Its attempt to be exhaustive has also made some of the data structures obsolete over time. The choice of XML messages also creates a significant burden in transport, particularly as the IDXP transport protocol, based on BEEP, has not been widely deployed.

The broad scope of the available specifications demonstrates that at this stage, there is no consensus between SIEM vendors and sensor vendors to agree on what an alert should contain. While many of the specifications are accessible to sensor vendors, SIEM platform vendors provide the connectors and take charge of translating the sensor information into their own formats, at the risk of missing information or misinterpreting the content. The issue of conveying alerts remains an issue in the lower layers, while the standards related to incident information exchange, such as MILE IODEF (RFC 7970), have been much more successful [882].

8.4.2 Alert correlation

Alert correlation [883, 884], aims to make sense of the alert stream received by the SIEM platform. The correlation has several objectives;

1. to reduce the number of alerts that the analyst has to process by grouping alerts together,
2. to add contextual elements to enable more accurate and faster analysis of the group of alerts,
3. to add alerts to ongoing higher-level planning and mitigation elements so that they are handled properly, and
4. to discard alerts that are considered false positives and do not require further processing.

To meet these objectives, alert correlation can take several forms:

Correlation between alerts The first kind of alert correlation aims to group together alerts from one or several sensors that correspond to the same threat. IDPS sensors tend to have a narrow view of the data stream. If events occur repeatedly in the trace, for example,

when a malware propagates, multiple alerts will be reported to the SIEM. Grouping alerts that correspond to the same phenomenon helps the analyst to recognise it and to judge its importance.

Correlation between alerts and the environment Another important source of knowledge is related to the context of the detection, the environment in which the sensors are located. Information about the environment comes from many sources, the two most interesting ones being network inventory and vulnerability scans. These two sources identify active assets and the risks they are potentially subject to. This type of correlation is particularly interesting as it provides the analyst with information about the impact the alerts are having.

Correlation between alerts and external sources Recently, situational awareness has started to provide information about attackers and their motivations [885]. This again provides additional information about the paths that an attacker might follow, and helps the analyst proactively to decide to block the attacker's progress, instead of reacting after the event.

Incident and information exchange Another relevant trend is information exchange. Through regulatory pressure, critical infrastructure operators are required to inform authorities when they are the victims of cybersecurity breaches. This has been the case for banks and credit unions for a long time. Sharing information about breaches helps others in the same domain, or using similar technologies, to protect themselves proactively.

The initial approach to alert correlation was based on rules. Rule-based correlation explicitly describes logical relationships between alerts, or rules to infer such relationships [884, 886, 887, 888]. A variety of languages and techniques have been used over the years by the research community, leading to exhaustive and formal models. This led to the development of the first generation of SIEM platforms, which combined strongly structured, high-performance SQL databases with logic engines interpreting rules. This first generation encountered two issues, performance as the volume of alerts increased, and the difficulty of creating and maintaining the rule base. SQL databases incur a significant performance penalty for indexing. This is good for querying, whereas SIEM platforms are insert-intensive tools.

Despite performance increase and database tuning, a second generation of SIEM platforms has been developed, leveraging less-structured database technologies such as NoSQL. This big data, or data-intensive approach started quite early on using counters [883], statistical models [889] or other techniques [842, 890]. Technologically, this approach is implemented through log aggregation and summarising queries, as can be done with the well-known ElasticSearch-Kibana-Logstash (ELK) stack. This data-oriented approach has become very common today, as it is able to cope with large volumes of incoming unstructured information. It remains to be seen whether the lack of relational structure does not introduce inconsistencies and naming confusion, impacting analysts' ability to diagnose and mitigate threats, and whether the focus on volume does not prevent handling rare attack phenomena such as APTs.

8.4.3 Security operations and benchmarking

The activity of a SOC needs to be measured, for several reasons. First, a SOC is the combination of technology platforms, information, processes and skilled personnel. Thus, it is difficult to identify where a specific SOC is performing well, and which areas should be improved. As SOCs are sometimes outsourced to MSSPs, the security service level agreement must be negotiated between the customer and the service provider, and verified by the customer. The customer may also be subject to regulations, which must be satisfied by the service provider as part of its contract. It is thus necessary to measure the activity of a SOC in a way that enables measurement, comparison between industries and to the state of the art, and to decide which areas of activity should be improved.

The *Information Security Indicators* (ISI) Industry Specification Group at ETSI develops indicators to this effect. These indicators are the product of a consensus approach, where several industry leaders (Thales, Airbus), users (banks, telcos) and technology providers (ESI Group, Bertin) have defined and tested these indicators jointly. The approach is Europe-wide, as the ETSI ISI group is supported by members from France, Germany and Italy, as well as the network of R2GS chapters in Europe (in addition to the countries in ETSI ISI, the UK, Luxembourg, Belgium, the Netherlands). In the end, these indicators should enable a comparative measurement of SOC performance, and a general measurement of the resistance of any given organisation to threats, cyber, physical or organisational.

The ISI specification is freely available from ETSI, and reference information charts are available from several sources. The main difficulty of this approach is the ability to automatically produce the indicators, or at least a subset of them, as some indicators are of a very high level.

8.5 EXECUTE: MITIGATION AND COUNTERMEASURES

[891]

For a long time, the SOIM community has focused on detection and analysis, both from a research and operational deployment aspect. There is a clear reluctance to automate the last part of the loop of figure 8.1, as system and network operators fear losing control over complex environments, although there are many reasons why it has become important to include automated mitigation in scope. This is an extremely important area, as exemplified by the *Respond* and *Recover* topics of the NIST cybersecurity framework.

8.5.1 Intrusion prevention systems

IDPS sensors have been rapidly extended to include *Execute* capabilities to respond to attacks. IDPS has the additional capability to act on the monitored stream upon detection. This requires the ability to act as a gateway or proxy through which all exchanges will be analysed, in order to reach a benign or malicious decision. Once a malicious decision has been reached, additional actions can be applied to the data stream, such as blocking, terminating or altering a data stream. Of course, the additional action relies heavily on the reliability of the detection, which is why common practice limits actions to a subset of the signatures of a misuse-based sensor.

Actions executed by the sensors are linked directly to the result of detection. As such, the *Plan* phase is performed through static configuration, and the response to an attack is thus

independent of the context during which the attack occurs.

The initial deployment of network-based IDS sensors was based on passive devices, unable to act on the network. The response was thus carried out by sending reconfiguration actions to a firewall located upstream or downstream from the sensor, through out-of-band dedicated communications. This mechanism induced significant delays in responding, as the first few packets of the attack were accepted before the rule was put in place. There were also undesirable side effects to dynamically changing the configuration of a firewall, such as losing connexion tracking. Also, system operators are extremely attentive about maintaining stable firewall configurations, as an essential part of SRE.

Given the need to respond in real time to well-identified attacks, modern network-based IDPSes are positioned inline in the network, to couple detection and firewalling. If malicious activity is detected by the sensor, the packet is immediately dropped or rejected, or the connection is terminated. The advantage of this solution is that attacks are handled at line rate, as soon as they occur. Of course, FP and FN of the detection mechanism will have a direct impact on the efficiency of the IDPS, denying service to legitimate users or letting attacks go through undetected. The main drawback of the IDPS is the action in the packet layer. This creates side effects that may leak information to an attacker. It also requires a device to be put into the network that has the ability to break the connection, injecting another point of failure into the ICT infrastructure.

Specialised examples of IDPS technology include *session border controllers* (SBC) or *Web Application Firewalls* (WAF). In the example of a WAF, the implementation could take the form of an external device acting as a proxy (and/or reverse proxy) or be implemented as an intercepting module in a web server.

More recently, inline IDPSes have been given the ability to modify the payloads of packets, under the term of 'virtual patching'. The result is that the server receives innocuous content instead of the content, and that the response sent back to the attacker indicates that the attack has failed. The main advantage of this approach is that it does not require breaking the flow, as do application-layer sensors such as WAF or SBC.

8.5.2 Denial-of-service

The most obvious area where automated network-based mitigation is required is *Denial of Service* (DoS), and particularly large-scale *Distributed Denial of Service* (DDoS) attacks. DDoS attacks have grown continuously in terms of volume and the number of sources involved, from 300 Gbps in 2013 to 680 Gbps (the Krebs-on-security incident) and 1 Tbps (the Mirai/OVH incident). The Arbor Networks survey of 2016 stated that half of the responding cloud infrastructure providers suffered from a loss of connectivity, which had a fundamental impact on their businesses. The emergence of attacks compromising Internet of Things (IoT) infrastructures and using them for DDoS, such as Mirai, helped reach new attack volume records, although the average DDoS attacks remain relatively small at 500 Mbps. [892] and [893] provide surveys and taxonomies of DDoS attacks and defences. There has also been more recent work, particularly on amplification attacks [851], which abuse protocols such as DNS [847] and NTP [850] to create large volumes of traffic with low bandwidth requirements.

DDoS attacks are large-scale phenomena which affect many components and operators in Internet infrastructures, from Autonomous System (AS) operators to cloud providers to service providers. Attacks on certain services also have a large-scale impact. For example, the DDoS

attack on DynDNS impacted the availability of well-known services such as Netflix, Spotify, Twitter etc. The move to cloud infrastructures obviously means that these cascading effects will continue.

Given their scale and impact, DDoS attacks are prime targets for automated remediation. This has led to the emergence of dedicated DDoS mitigation service operators in cloud mode. These service operators offer load management services, such as adding new servers to face the flow, redirecting traffic to other services, or selectively decreasing traffic.

Classic techniques for decreasing traffic include blacklisting, for example, with IP ingress filtering, or at the application level using TCP Syn cookies to ensure legitimate TCP session establishment. This helps resist DDoS attacks, although one has to acknowledge that these services will be unable to prevent or fight very large-scale attacks.

At the core network, MPLS provides an interesting option to mitigate DDoS attacks [894], as it enables bandwidth reservation and bandwidth usage control, to ensure that the legitimate traffic receives sufficient bandwidth and that potentially malicious traffic is got rid of. At the edge, the deployment of *Software Defined Networking* (SDN) as the fundamental network control technique for cloud centres permits flexibility of the network configuration and control, and enables collaboration between Internet service providers and cloud infrastructure operators to mitigate DDoS attacks [895].

Beyond networking access (which is at this time the biggest threat), DoS attacks may also target computing resources, storage, or power. The emergence of the Internet of Things, and the increasing requirement of connecting low-cost, battery-operated objects to the Internet might increase the DoS attack surface in the future.

8.5.3 SIEM platforms and countermeasures

The contribution of SIEM platforms to the MAPE-K Execute activity today is limited; once plans have been defined and validated by analysts, other functions such as change-control ticketing systems take over to ensure that the deployed actions are appropriate and do not adversely impact business activity.

Internally in SOCs, analysts use ticketing systems to follow up on the progress of incident resolution and escalate issues to more skilled or specialised analysts when needed. Ticketing systems can also serve for incident post-mortem analysis, to evaluate and improve SOC processes.

SOC analysts also interact with ticketing platforms to push change requests to other teams, in charge of network or system management. This can even extend to security functions, for example, if the organisation has a dedicated firewall management platform. The fact that this remains mostly a manual activity introduces a significant delay in threat mitigation. It also relies on system or network operators on the other side of the ticketing system to understand the requested change and effectively implement it. However, this delay is often seen as necessary to deal with potential false positives, and to assess the effective impact on business activities, as elaborated in the following section.

8.5.4 SOAR: Impact and risk assessment

Risk assessment in cybersecurity mainly focused in the past on protecting ICT assets, machines, network equipment and links. Risk assessment methodologies focus on determining assets, analysing their vulnerabilities, and modelling cascading effects. Attack trees, informally described by Schneier [62] and formally defined by Mauw [896], are now implemented as attack graphs in software tools [897]. They enable a network or system security officer to model the ICT environment and the associated vulnerabilities, to determine the paths an attacker might follow to compromise interesting targets. These more complex attack graphs enable a quantification of the likelihood that an attacker will propagate in an Information System, of the damage, and of the possible protection measures that could block the attack.

From a business perspective, attack graphs and vulnerability management technologies enable risk management and compliance with regulations. As the impact of cyber-attacks increases, and potentially becomes a threat to human life or business continuity, regulators impose protection and detection measures to ensure that cyber-risk is properly managed in organisations. While there are many possible protection techniques available, from identification and authentication to filtering and firewalling, the complexity and interconnectivity of complex ICT infrastructures makes it unfeasible, either technically or economically, to protect them against all possible threats. As such, cybersecurity becomes an economic trade-off between deploying protection measures, assuming the risk, and insuring it. Cyber-insurance has been difficult but there is an increasing interest in the economics of cybersecurity, which might support the development of cyber-insurance models [898].

Another aspect of attack graphs is their use for countermeasures. Work on countermeasures has focused on technical assets, as they can be activated to block threats. This means adding or modifying firewall rules to block unwanted traffic, disabling or removing privileges of user accounts, preventing unauthorised or suspected machines of connecting to the network or the Information System, or shutting down a service or machine. However, the deployment of countermeasures requires an impact assessment, not only at the asset level but also at the business level. The heavy reliance of business missions on technical ICT assets means that these firewall rules or blocked accounts may have a detrimental effect on an organisation's business. This detrimental effect might even be worse than suffering an attack, at least for some time. New models for impact assessment must take into account not only the ICT asset fabric but also the business services that they support to determine their criticality and the cost of altering their behaviour [899].

One cannot emphasise enough, as in section 8.5.3, the importance of the processes and workflows associated with the set of tools implemented for SOAR. This, for example, implies that there is a clear understanding of responsibilities in the SOC, a chain of validation when countermeasures are deployed, and an effective verification that the mitigation is efficient and has stopped the attack or its effects.

8.5.5 Site reliability engineering

Another relevant aspect of threat protection and mitigation is that ICT environments have to prepare for incident management and mitigation. As is required for safety engineering, operators have to define and deploy procedures such as activity continuity planning to ensure that they will continue to operate even when faced with certain threats [900]. This means that operators must deploy and operate sensors up to a certain level of efficiency. They must also deploy and operate protection tools such as firewall or authentication systems that might impact the performance and usual behaviour of their systems. Also, all of this new equipment will require manpower for monitoring and maintenance.

A recent significant change to SRE is an extension of scope. Much, if not all, of the equipment used in any organisation will include digital technology and will require maintenance. Many devices powering physical access control or building management will be interconnected with and accessible through the ICT infrastructure. As such, they will be subject to similar, if not identical, attacks as the ICT infrastructure. New maintenance models should be developed and adapted to include these IoT devices in the reliability engineering process. The *Network and Information Systems* (NIS) European Union directive requires that all devices should be patched to remove vulnerabilities. Remote maintenance will become a requirement for many objects, large and small. Depending on their computing abilities, storing and communicating security elements, these maintenance processes will be difficult to develop and put into place [901]. However, there are many systems, for example, in the transportation or health domains, where the move to digital technology must include software maintenance that is timely and secure.

This is driving increased convergence between reliability, safety and cybersecurity. SRE teams in cyber-physical environments thus need to operate systems, monitoring them for failures and attacks, in order to ensure continuous operation. SRE is thus also increasingly applied in pure IT environments such as cloud computing platforms, which must be robust against accidental failures such as power.

8.6 KNOWLEDGE: INTELLIGENCE AND ANALYTICS

[877, 891]

Intelligence and analytics focus on two specific components, as shown in figure8.2, CTI and CERTs. The CTI platform (section 8.6.3) replaces and includes honeypots to provide a comprehensive view of malicious activity that may impact an organisation. CERTs and ISACs are regulatory bodies with which an organisation can obtain additional information, such as the industry-specific indicator of compromise, or best practice for incident detection and handling.

8.6.1 Cybersecurity knowledge managment

As described in section 8.4, SIEM platforms are the main technical tool supporting analysts to defend Information Systems and networks. The earliest attempt at managing cybersecurity-related knowledge is vulnerability information sharing, formalised as CERT advisories first and now managed through the *Common Vulnerabilities and Exposures* (CVE) dictionary, the *Common Vulnerability Scoring System* (CVSS) and databases such as the NIST National Vulnerability Database. However, the performance of these platforms relies heavily on the information made available to the analysts manning them. Understanding attackers has been a long-standing area of research, but there have been many recent advances in the state of the art on understanding attack processes and motivations, and on providing analysts with better information to make appropriate decisions.

CVE provides a way to reference specific vulnerabilities attached to specific versions of products. This information is very useful for IDS signatures, because they clearly identify the targeted product. However, they are insufficient for more global processing, hence higher level classifications have been defined.

The *Common Vulnerability Scoring System* (CVSS) provides a standard way to rate the impact of vulnerabilities by providing a synthetic numerical score that is easy to comprehend. Each vulnerability is assigned a score according to six base metrics that reflect the intrinsic characteristics of a vulnerability, and in particular the ease with which the vulnerability can be leveraged by an attacker to impact confidentiality, integrity and availability. This base metric is modulated by three temporal metrics that indicate whether exploits (increasing the risk) or patches (decreasing the risks) are available; these three temporal metrics evolve over time, as more information becomes available. Finally, four temporal metrics measure the specific exposure of an organisation. Each CVE entry is usually qualified by a CVSS score.

The *Common Weakness Enumeration* (CWE) dictionary provides a higher level structure on top of the CVE dictionary, to qualify further the kind of software weaknesses involved in the vulnerability. It serves as an additional description of the CVE entry, to identify weaknesses that appear in multiple software tools, and to identify common mitigation and prevention strategies. The structure of the CWE is relatively complex, and identifying commonalities accross vulnerabilities is sometimes difficult. CWE references are frequently found in CERT advisories.

The *Common Attack Pattern Enumeration and Classification* (CAPEC) and *Adversarial Tactics, Techniques & Common Knowledge* (ATT&CK) frameworks provide two additional views focusing on attacker activities. CAPEC references multiple CWE entries focusing on common attributes and techniques used by attackers. Examples include SQL injection or cross-site request forgery. More recently, ATT&CK has been formalising operational information about attackers, to develop threat models and operational procedures for defending networks.

It is important to note that the performance of SIEM and SOAR relies on accurate and complete information being present in the knowledge base. As such, this information must be maintained, and the appropriate links to other system or network management functions should be established to this effect.

8.6.2 Honeypots and honeynets

Honeypots are a relatively old technology, as exemplified in Stoll's book [313]. They were popularised by the *Honeynet Project* and Spitzner's book [902]. The community commonly defines a honeypot as *an Information System resource whose value lies in unauthorised or illicit use of that resource.* More concretely, a honeypot is a machine (a honeynet is a set of machines) which is offered as bait to attackers. As such, honeypots use 'free' resources in an Information System or network to provide realistic-looking services for the outside world. In normal use, these machines should never be accessed by legitimate users, thus any interaction is deemed to be related to malicious use. By monitoring the attackers' use of the honeypot, researchers hope to obtain relevant information about attack processes and new malicious code, and to leverage this information for attack detection and mitigation.

There are several categories of honeypots. Initially, honeypots were very simple tools, alerting on the connection to a given port with a given IP address. However, as attackers and malware evolved, they became able to detect interactions that are different from the service that should be offered by the platform to which they are connected. Honeypot and honeynet technologies have thus developed in a fairly sophisticated manner in large-scale, complex infrastructures. They have given rise to attacker analytics, from observations to statistical analysis, to what is now identified as the *Indicator Of Compromise* (IoC), organised pieces of evidence that an attacker is trying to compromise an Information System or network.

The main hypothesis behind honeypots is that attackers will actively seek victims, while regular users will only use resources that are publicly and officially advertised through configuration, routing and naming. This was probably true during the main period of Internet-scanning worms such as Slammer. However, attackers have other means of silently gathering information about their targets, for example, through search engines. The scanning is thus done by legitimate, or at least known actors, but it provides no information about the attackers. Also, there is a significant amount of background noise activity on the Internet [903]. Thus, the main premise of honeypots, that there are no false positives because all activity is malicious, cannot be guaranteed.

The information collected by honeypots is provided entirely by the attackers, and they are also developing techniques to understand whether they are running in controlled environments or not. If they detect a controlled environment such as a virtual machine, they will stop interactions. While cloud computing has generalised the use of virtualisation, there are other tell-tale signs that indicate control and monitoring. Today's best use of honeypots is probably within sensitive data, in the form of fake email addresses and fake rows or columns in databases.

8.6.3 Cyber-threat intelligence

Honeypots have shown that it is useful to observe malicious activity, to capture malware and to detect new threats before they can spread widely. Since the peak of the honeypot period, researchers have started looking at attack mechanisms and trends from a wider perspective [904], but maintaining the objective of both looking at Internet-wide malicious activity [905, 906] and at malware analysis [907, 908].

In addition to honeypots, cyber-threat intelligence has included the dimension of information sharing, as increasingly required by national authorities. Information sharing is both the outcome of data analytics [909] and is extremely useful for defenders to better understand the risks and possibilities for protection and mitigation. As such, it is as much a human

process [910] as platforms and tools, such as the open source Malware Information Sharing Platform (MISP) [911], also included in TheHive project.

Another important topic is the definition of IoCs [891], which is a more general term than signatures. Signatures, as is generally understood, are pieces of evidence of an ongoing attack. IoCs generalise the concept in two ways. First, they indicate evidence of an attack being prepared or of the evidence that remains after a system has been compromised by an attacker. IoCs are defined for sharing, hence their inclusion in standards such as RFC 7970, the *Incident Object Description Exchange Format* (IODEF) version 2 and the Structured Thread Information eXchange (STIX).

While early signature sharing attempts used the Snort signature language, the YARA language has been quite widely adopted and is, for example, the support of the *YARA Signature Exchange Group*, a non-commercial indicator of compromise exchange platform.

In order to support and regulate information sharing, the authorities have also promoted the creation of *Information Sharing and Analysis Centers* (ISAC). These ISACs are both regional (in the U.S., in Europe, etc.) and sectoral (for energy, transportation, banking, etc.). The objective is to facilitate information sharing between persons with similar organisations and objectives. It also brings the economic dimension to cybersecurity, analysing the benefits of information sharing for organisations for better efficiency.

8.6.4 Situational awareness

Situational Awareness is a complex subject, which has been the subject of research both from a technical and a social sciences standpoint. Early work focused on users operating complex systems, for example, pilots in aircrafts [912], defining situational awareness as a cognitive process, *the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future*. This work was considered foundational for a lot of the later work in CyberSA and a 2014 survey paper by Franke and Brynielsson [877] promoted this definition by Endsley [912]. In the context of cyberattacks and the digital society, this definition implies that CyberSA implies the *awareness of any kind of suspicious or interesting activity taking place in cyberspace* [877].

Beyond technology [913], cyber-situational awareness has seen broad contributions from the social sciences. It has also been widely studied in military circles [914]. Several of the aforementioned contributions also use machine-learning techniques. When analysing the performance of cyber-responders (SOC operators and analysts) Tadda [913] already uses existing SIEMs and Intrusion Detection Systems as the technical platform for implementing cyber-situational awareness.

The SIEM world is undergoing profound changes through regulation and the impact of cyber-attacks. From a regulation perspective, critical infrastructure operators are required to embed detection and mitigation capabilities. This represents the instantiation of the European NIS directive in national law. ENISA regularly provides information about cyber-incidents, particularly procedures for detection and management. The most recent report on a cyber-incident simulation in June 2017 indicated that progress is still required in CyberSA, but that cooperation is increasing and that information sharing is of the utmost importance for appropriate decision-making.

8.7 HUMAN FACTORS: INCIDENT MANAGEMENT

[891]

In the current state of affairs, it remains clear that complete protection is both technically unfeasible and economically undesirable. Hence, systems will be compromised, and it is likely that attacks will bring them down, having a significant impact. There have been, for example, several instances of businesses shutting down for days due to ransomware attacks, such as Wannacry. Beyond ensuring business continuity, technical and regulatory obligations require that investigations are undertaken, following a cybersecurity compromise. This is a mandatory step in restoring an ICT system to a reliable state. This step is where, beyond tools and processes, the human aspects are key, particularly education, training and exercising.

Figure 8.5 presents a simplified incident management process inspired from NIST SP800-61 [915], a definition of challenges by Ahmad et al. [916] and a survey by Tondel et al. [917]. It defines three broad activities that an organisation must carry out, *prepare* itself for incidents, *handle* incidents when they occur, and *follow up* on incidents when they are closed.

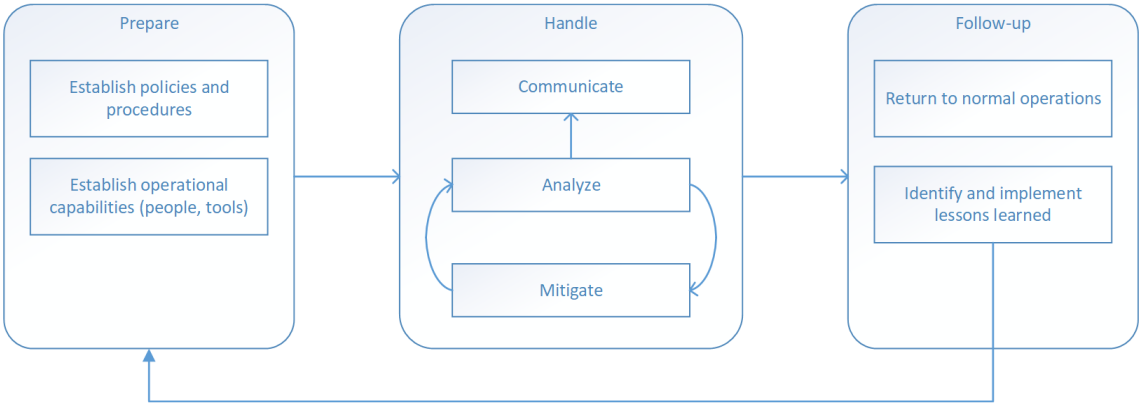


Figure 8.5: incident management lifecycle

While the incident management topic comes at the end of the KA, it leverages all the capabilities and tools that have been described in the previous sections. It is also necessary to highlight that there is a required balance between prevention and response [918]. Full prevention has been demonstrated to be unfeasible, for ease of use and cost reasons on one hand, and because attackers have ways and imagination beyond what system designers envisage on the other hand. Therefore, devoting resources to prevention versus response is highly organisation-specific, but it is an important exercise that must be carried out carefully because of the consequences it has for an organisation. On one hand, prevention will increase the operational costs of an organisation. On the other hand, relying only on response may lead to fatal consequences where the organisation would not be able to recover from an incident. Also, responding properly to incidents incurs costs that should not be ignored. Risk assessment is thus an integral part of incident management.

8.7.1 Prepare: Incident management planning

As shown in figure 8.5, the first step in incident management is to put in place the appropriate processes and capabilities before an incident occurs. This is, in fact, a legal requirement for all critical infrastructure operators, and is established by regulations such as the EU Network and Information Systems (NIS) directive.

Establishing policies and procedures relies on the structure of the organisation and the sector to which it belongs. Policies must involve higher level management, in order to properly define the scope and the organisational structure that is devoted to incident management, as well as performance and reporting procedures. Policies must include formalised response plans that provide a roadmap for implementing the incident response capability, based on risk assessment methods. Plans should be refined in procedures, in order to define standard operating procedures that can be quickly followed by responders to concretely define the actions that need to be taken when specific situations occur. All of these policies, plans and procedures are organisation and sector-dependent, and will be affected by different regulations. As an example, financial organisations have to take into account the *Basel II* and *Sarbanes-Oxley* regulations in their incident management procedures, to properly implement reporting to regulators.

An important part of this preparation activity is related to communication in many forms. First of all, regulations now generally require that incidents are reported to the authorities, either a national CERT hosted by a national cybersecurity agency, law enforcement agencies, or a sectoral organisation such as an ISAC. It is also important beforehand to establish trusted communication channels with technology and service providers such as software vendors and Internet service providers. Similar channels should be set up between peers such as CISOs, to facilitate sharing of early warnings and best practices. Transnational organisers and facilitators of exchanges include the Computer Security Incident Response Teams (TF-CSIRT), the Forum of Incident Response and Security Teams (FIRST) and the European Union Agency for Cybersecurity (ENISA).

Another constituency comprises customers, media and the general public. Organisations should be ready to communicate when cyber-security incidents affect customers, or when they become largely visible. For example, the European *General Data Protection Regulation [105]* (GDPR) establishes the need to report to users in case of information leakage. Therefore, we expect that the requirements of GDPR compliance will have an impact on cyber-security, as organisations realise that they have to protect and monitor their systems to comply with this regulation.

Finally, preparation also includes the establishment of a team, usually a CSIRT. This includes practical considerations such as the decision to handle incidents internally or to subcontract, fully or partially, the tasks to qualified MSSPs, the choice of a centralised or distributed organisation for incident response, and the reporting chain. Choosing between a centralised or a distributed structure is guided by the structure of the organisation. A distributed structure enables better proximity (geographical as well as functional) between the incident responders and the business units. However, it may increase the required coordination efforts and cost.

Incident response is very much a person-intensive task, which is related to crisis management. It requires the ability to work under pressure, both internally (to prevent the incident from propagating or blocking the organisation) and externally (to deal with management, regulatory or media pressure). There is thus a need for qualified personnel to practise incident response exercises, as is done in the military, for example. It also requires continuous training in order

to keep up with the most recent threats. The integration of key people with the relevant communities such as ISACs or CERTs also helps information sharing and ensures that best practices are exchanged within the right community.

8.7.2 Handle: Actual incident response

As shown in figure 8.5, handling incidents requires three different activities, analysis, mitigation and communication.

Analysis is related to incident investigation, to understand the extent of the compromise and of the damage to the systems, particularly data. If data have been lost or altered, the damage might be extremely significant. Therefore, the investigation must assess what exactly was compromised, and what was not, as well as the time the compromise occurred. This is extremely difficult, due to the duration of certain attacks (months), the stealthy techniques attackers deploy to remain hidden (erasing logs or systems, encrypting communications), and the difficulty of freezing and interacting with systems (attackers detecting interaction may take very destructive action) and gathering evidence.

Mitigation is related to the deployment of emergency measures that can contain the incident and limit its impact. Mitigation must first limit the damage that is brought to systems, such as information erasure or disclosure, that an attacker could trigger if he is discovered. It must also ensure that attackers do not propagate to other systems. Containment may include blocking network accesses in certain perimeters, or shutting down services, systems or communications. Containment may unfortunately have an adverse impact on desirable functions. For example, cutting network access prevents attackers from communicating with compromised systems, but also makes patching them or backing them up more difficult.

It is common that mitigation measures reveal additional information about the attacker, its methods and targets. Hence, figure 8.5 includes a closed loop between analysis and mitigation, to emphasise the fact that analysis and mitigation should be understood as feeding each other.

As already mentioned in section 8.7.1, communication is an integral part of incident handling. Once the extent of the damage has been established, it is necessary to alert the authorities and comply with regulations as needed.

8.7.3 Follow-up: post-incident activities

The final step in an incident response is to verify that the full extent of the compromise has been realised and to clean up the system. Restoring a system is also connected to reliability engineering, as system integrators must plan and system operators must maintain for restoration in the case of compromise.

Another important aspect of post-incident activities is to measure the performance of the team and the procedures, in order to improve them. This is often difficult, and Ahmad et al. [916] pointed out several factors related to this difficulty. First, this means sacrificing short-term goals (handling current incidents and returning to normal operations) to improve long-term behaviour (e.g., faster and/or more accurate mitigation). Another aspect of follow-up that should be taken into account is the impact of the incident. While major incidents generally lead to post-mortem analysis and changes in policy, low-impact incidents may be left out of the follow-up procedure. However, it is often the case that these low-impact incidents take up

a major part of the resources devoted to incident management and they should be explored as well.

Communication is also an important aspect of follow-up. Lessons learned from incidents should impact incident training, to ensure that responders are up to date with attacker methods. It should also enable information sharing with peers, so that best practices are propagated to the community as a whole, to learn from incidents beyond the ones affecting each organisation.

Another related subject is attack attribution [818]. The objective is to understand where and why the attack came from, and in particular the motivations of the attacker. This will help restore the system to a working state and prevent later compromise.

Some of the work on attribution has focused on malware analysis, to provide technical evidence of the source of the attack. The objective is to find in the malware code evidence of its roots, such as code reuse or comments that may explain the motivations of the author. This enables the definition of malware families, which then may help define more generic IoCs to detect the propagation of malicious code even if the exact variant is not known. Malware authors do use many techniques to make this difficult, as explained in section 8.3.1.

Other works on attribution observe network activity to extract commonalities. Groups of attackers may share Command and Control (C&C) infrastructures, thus attacks may come from the same IP addresses or use the same domain names. They might reuse services, thus using similar-looking URLs or commands.

However, attribution is very expensive, particularly if the objective is to use forensics techniques to support legal action. At this point in time, forensics and attribution remain an extremely specific field and are not included in Security Operations and Incident Management, because they require expertise, tools and time beyond what SIEM analysts manning consoles can provide.

Legal action using the information gathered through forensics techniques is discussed in the Forensics key area description.

8.8 CONCLUSION

The Security Operations and Incident Management domain includes many topics. From a technical standpoint, SOIM requires the ability to observe the activity of an Information System or network, by collecting traces that are representative of this activity. It then requires the ability to analyse these traces in real time, or almost real time, to detect malicious events included in these traces, and to send out alerts related to these events. The definition of a malicious event depends on the analysis technique and on the data source used to perform the detection. Once an attack is detected, it must be reported and analysed on a SIEM platform, to assess the impact of the attack and to determine the potential remedial actions that can be applied to block the attack or mitigate its effects.

From an operational standpoint, SOIM is very much a process, and the definition of this process requires strong management. It relies on people to perform many of the tasks, from configuring the detectors to analysing the alerts to deciding on remediations. Therefore, skilled analysts are one of the cornerstones of Security Operations and Incident Management. Another key aspect is planning, as all the tools and personnel must be in place before anything can happen. Finally, SOIM is expensive, requiring both complex tools and skilled, round-the-clock personnel to man them. However, the heavy reliance of our society on digital tools,

as well as the regulatory context, require that these tools and processes are put in place everywhere.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[837]	[856]	[835]	[839]	[877]	[891]
8.1 Fundamental concepts	X		X			
8.2 Monitor: data sources			X	X		
8.3 Analyse: analysis methods	X	X	X	X		
8.4 Plan: security information and event management					X	
8.5 Execute: Mitigation and countermeasures						X
8.6 Knowledge: Intelligence and analytics					X	X
8.7 Human factors: Incident management						X

Chapter 9
Forensics

Vassil Roussev | University of New Orleans

INTRODUCTION

Digital forensic science, or *digital forensics*, is the application of scientific tools and methods to identify, collect and analyse digital (data) artifacts in support of legal proceedings. From a technical perspective, it is the process of identifying and reconstructing the *relevant* sequence of events that has led to the currently observable state of a target IT system or (digital) artifacts. The importance of digital evidence has grown in lockstep with the fast societal adoption of information technology, which has resulted in the continuous accumulation of data at an exponential rate. Simultaneously, there has been rapid growth in network connectivity and the complexity of IT systems, leading to more complex behaviour that may need investigation.

The primary purpose of this Knowledge Area is to provide a technical overview of digital forensic techniques and capabilities, and to put them into a broader perspective with regard to other related areas in the cybersecurity domain. The discussion on legal aspects of digital forensics is limited only to general principles and best practices, as the specifics of the application of these principles tend to vary across jurisdictions. For example, the Knowledge Area discusses the availability of different types of evidence, but does not work through the legal processes that have to be followed to obtain them. The Law & Regulation Knowledge Area (Chapter 3) discusses specific concerns related to jurisdiction and the legal process to obtain, process, and present digital evidence.

CONTENT

9.1 DEFINITIONS AND CONCEPTUAL MODELS

[919, 920, 921, 922, 923]

Broadly, *forensic science* is the application of scientific methods to collect, preserve and analyse evidence related to legal cases [919]. Historically, this involved the systematic analysis of (samples of) physical material in order to establish causal relationships between various events, as well as to address issues of provenance and authenticity. The rationale behind it, *Locard's exchange principle*, is that physical contact between objects inevitably results in the exchange of matter, leaving *traces* that can be analysed to (partially) reconstruct the event.

With the introduction of digital computing and communication, which we refer to as the *cyber domain*, the same general assumptions were extended largely unchallenged. Although a detailed conceptual discussion is beyond the scope of this chapter, it is important to recognise that the presence of a persistent *digital (forensic) trace* is neither inevitable, nor is it a "natural" consequence of the processing and communication of digital information.

A *digital (forensic) trace* is an explicit, or implicit, record that testifies to the execution of specific computations, or the communication and/or storage of specific data. These events can be the result of human-computer interaction, such as a user launching an application, or they can be the result of the autonomous operation of the IT system (e.g., scheduled backup). Explicit traces directly record the occurrence of certain types of events as part of the normal operation of the system; most prominently, these include a variety of timestamped system and application event logs. Implicit traces take many forms, and allow the occurrence of some events to be deduced from the observed state of the system, or artifact, and engineering knowledge of how the system operates. For example, the presence on a storage device of a

unique chunk of data that is part of a known file can demonstrate that the file was likely to have been present once, and was subsequently deleted and partially overwritten. The observed absence of normal log files can point to a security breach during which the perpetrators wiped the system logs as a means to cover their tracks.

Although they frequently exist, these traces of cyber interactions are the result of conscious engineering decisions that are *not* usually taken to specifically facilitate forensics. This has important implications with respect to the provenance and authenticity of digital evidence, given the ease with which digital information can be modified.

9.1.1 Legal Concerns and the Daubert Standard

The first published accounts of misuse and manipulation of computer systems for illegal purposes such as theft, espionage and other crimes date back to the 1960s. During the 1970s, the first empirical studies of computer crime were carried out using established criminological research methods. In the early-to-mid 1980s, targeted computer crime legislation emerged across Europe and North America [924, 925]; in recognition of the inherent cross-jurisdictional scope of many such crimes, international cooperation agreements were also put in place.

In the UK, the Computer Misuse Act 1990 [920] defines computer-specific crimes – S1 Unauthorised Access To Computer Material, S2 Unauthorised Access with Intent to Commit Other Offences, S3 Unauthorised Acts with Intent to Impair Operation, and S3A Making, Supplying or Obtaining. The Police & Criminal Evidence Act 1984 and Criminal Justice & Police Act 2001 address computer-specific concerns with respect to warrants, search and seizure.

In many jurisdictions, legal statutes related to misuse of telecommunications are separate (and older than) those related to computer crimes. We use the umbrella term *cybercrime* to collectively refer to all crimes related to computer and telecommunications misuse; broadly, these include the use of cyber systems to commit any type of crime, as well as the criminal targeting of cyber systems.

As is usually the case, legal systems require time to assimilate new laws and integrate them into routine law practice. Conversely, legislation usually requires corrections, clarification and unified interpretation in response to concerns encountered in the courtroom. One of the earliest and most influential legal precedents was set by the US supreme court, which used three specific cases – Daubert v. Merrell Dow Pharmaceuticals, 509 U.S. 579 (1993); General Electric Co. v. Joiner, 522 U.S. 136 (1997); and Kumho Tire Co. v. Carmichael, 526 U.S. 137 (1999) – to establish a new standard for the presentation of scientific evidence in legal proceedings, often referred to as the Daubert standard [926].

As per Goodstein [921], “The presentation of scientific evidence in a court of law is a kind of shotgun marriage between the two disciplines. ... The Daubert decision is an attempt (not the first, of course) to regulate that encounter.” These cases set a new standard for expert testimony, overhauling the previous Frye standard of 1923 (Frye v. United States, 293 F. 1013, D.C. Cir. 1923). In brief, the supreme court instructed trial judges to become gatekeepers of expert testimony, and gave four basic criteria to evaluate the admissibility of forensic evidence:

1. The theoretical underpinnings of the methods must yield testable predictions by means of which the theory could be falsified.
2. The methods should preferably be published in a peer-reviewed journal.

3. There should be a known rate of error that can be used in evaluating the results.
4. The methods should be generally accepted within the relevant scientific community.

The court also emphasised that these standards are flexible and that the trial judge has a lot of leeway in determining the admissibility of forensic evidence and expert witness testimony. The Daubert criteria have been broadly accepted, in principle, by other jurisdictions subject to interpretation in the context of local legislation. In the UK, the Law Commission for England and Wales proposed in consultation paper No. 190 [922] the adoption of criteria that build on Daubert.

The *ACPO Good Practice Guide for Digital Evidence* codifies four basic principles for the acquisition and handling of digital evidence:

1. No action taken by law enforcement agencies, persons employed within those agencies or their agents should change data which may subsequently be relied upon in court.
2. In circumstances where a person finds it necessary to access original data, that person must be competent to do so and be able to give evidence explaining the relevance and the implications of their actions.
3. An audit trail or other record of all processes applied to digital evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result.
4. The person in charge of the investigation has overall responsibility for ensuring that the law and these principles are adhered to.

These principles seek to provide operational guidance to digital forensic investigators on how to maintain the integrity of the evidence and the investigative process, such that the evidence can be used in a court of law.

In the UK, the *forensic science regulator* mandates that any provider of digital forensic science must be "accredited to BS EN ISO/IEC 17020:2012 for any crime scene activity and BS EN ISO/IEC 17025:2005 for any laboratory function (such as the recovery or imaging of electronic data)" [927]. ISO/IEC 17025 [928] is an international standard specifying general requirements for the competence of testing and calibration laboratories; in other words, the certification attests to the quality and rigour of the processes followed in performing the forensic examination.

In the US, there is no strict legal requirement for digital forensic science providers to be certified to particular standards. Most large federal and state forensic labs do maintain ISO 17025 certifications; as of 2019, eighty five of them have such credentials for the processing of digital evidence.

Digital forensic techniques are also applied in a much broader range of inquiries, such as internal corporate investigations, that often do not result in formal proceedings in public court. Despite the fact that investigations may not require the same standard of proof, forensic analysts should always follow sound forensic practices in collecting and analysing the artifacts. This includes adherence to any judicial requirements when working with inherently personal data, which can be a non-trivial concern when the investigation is multi-jurisdictional. In such cases, it is important to seek timely legal advice to preserve the integrity of the inquiry.

9.1.2 Definitions

In 2001, the first Digital Forensics Research Workshop (DFRWS) was organised in response to the need to replace the prevalent *ad hoc* approach to digital evidence with a systematic, multi-disciplinary effort to firmly establish digital forensics as a rigorous scientific discipline. The workshop produced an in-depth report outlining a research agenda and provided one of the most frequently cited definitions of digital forensic science in the literature:

> [DFRWS] **Digital forensics** *is the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorised actions shown to be disruptive to planned operations*. [929]

This definition, although primarily stressing the investigation of criminal actions, also includes an anticipatory element, which is typical of the notion of forensics in operational environments, and brings it closer to incident response and cyber defence activities. In these situations, the analysis is primarily to identify the vector of attack and the scope of a security incident; the identification of adversaries with any level of certainty is rare and prosecution is not the typical outcome. In contrast, the reference definition provided by NIST a few years later [923] is focused entirely on the legal aspects of forensics, and emphasises the importance of a strict chain of custody:

> [NIST] **Digital forensics** (NIST) *is considered the application of science to the identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody for the data. Data refers to distinct pieces of digital information that have been formatted in a specific way.* [923]

The above law-centric definitions provide a litmus test for determining whether specific investigative tools and techniques qualify as being forensic. From a legal perspective, a flexible, open-ended definition is normal and necessary during legal proceedings to fit the case. However, from a technical perspective, they do not provide a meaningful starting point; therefore, we can adapt a refinement of the working definition first introduced in [930]:

> [Working] **Digital forensics** *is the process of identifying and reconstructing the relevant sequence of events that have led to the currently observable state of a target IT system or (digital) artifacts.*

The notion of *relevance* is inherently case-specific, and a large part of forensic analysts' expertise is the ability to identify evidence that concerns the case at hand. Frequently, a critical component of forensic analysis is the causal attribution of event sequence to specific human actors of the system (such as users, administrators, attackers). The provenance, reliability, and integrity of the data used as evidence data is of *primary* importance.

According to this definition, we can view every effort made to perform system or artifact analysis after the fact as a form of digital forensics. This includes common activities such as incident response and internal investigations, which almost never result in any legal action. On balance, only a tiny fraction of forensic analyses make it to the courtroom as formal evidence, although this should not constrain us from exploring the full spectrum of techniques for

reconstructing the past of digital artifacts. The benefit of employing a broader view of forensic computing is that it helps us to identify closely related tools and methods that can be adapted and incorporated into forensics.

9.1.3 Conceptual Models

In general, there are two possible approaches to rebuilding the relevant sequence of events in the analysis of a cyber system from the available data sources – *state*-centric, and *history*-centric/*log*-centric. The starting point for state-centric approaches is a snapshot of the state of the system of interest; for example, the current content of a hard drive or another storage medium. Using the knowledge of how a particular system/application operates, we can deduce a prior state of interest. For example, if unique pieces of a known file are on the medium, but the file is not available via the normal file system interface, the most likely explanation is that the file was once stored in the file system but was subsequently deleted (the space was marked for reuse) and partially overwritten by newer files. The main constraint here is the dearth of historical data points, which limits our ability to deduce the state of the system at any given point in the past.

Log-centric approaches rely on an explicit, timestamped history of events (a log) that documents the updates to the system's state. For example, a packet capture contains the complete history of network communications over a period of time. Operating Systems (OSs) maintain a variety of monitoring logs that detail various aspects of the operation of the OS kernel and different applications; additional auditing and security monitoring tools can provide yet more potentially relevant events. Many applications, especially in the enterprise domain, provide application-level logs. Thus, a log-rich environment contains potentially all the relevant details to an investigation; the challenge is to sift through the log entries, which often number in the millions, to find and put together the relevant events.

Historically, storage has been a precious resource in computer systems, leading to software designs that emphasise space efficiency by updating the information in place, and keeping a minimal amount of log information. Consequently, the principal approach to forensics has been predominantly state-centric.

Over the last ten to fifteen years, technology advances have made storage and bandwidth plentiful and affordable, which has led to a massive increase in the amount of log data maintained by IT systems and applications. There is a clear trend towards increasing the amount and granularity of telemetry data being sent over the network by operating systems and individual applications as part of their normal operations. Consequently, there is a substantial need to evolve a forensic methodology such that log information takes on a correspondingly higher level of importance. In other words, the current period marks an important evolution in digital forensic methodology, one that requires substantial retooling and methodological updates.

9.1.3.1 Cognitive Task Model

Differential analysis [931] is a basic building block of the investigative process, one that is applied at varying levels of abstraction and to a wide variety of artifacts. However, it does not provide an overall view of how forensic experts actually perform an investigation. This is particularly important in order to build forensic tools that better support cognitive processes.

Unfortunately, digital forensics has not been the subject of any serious interest on the part of cognitive scientists and there has been no coherent effort to document forensic investigations. Therefore, we adopt the sense-making process originally developed by Pirolli & Card [932] to describe intelligence analysis - a cognitive task that is very similar to forensic analysis. The Pirolli & Card cognitive model is derived from an in-depth Cognitive Task Analysis (CTA), and provides a reasonably detailed view of the different aspects of an intelligence analyst's work. Although many of the tools are different, forensic and intelligence analysis are very similar in nature - in both cases analysts have to go through a mountain of raw data to identify (relatively few) relevant facts and put them together into a coherent story. The benefit of using this model is that: a) it provides a fairly accurate description of the investigative process in its own right, and allows us to map the various tools to the different phases of the investigation; b) it provides a suitable framework for explaining the relationships of the various models developed within the area of digital forensics; and c) it can seamlessly incorporate information from other lines of the investigation.

The overall process is shown in Figure 9.1. The rectangular boxes represent different stages in the information processing pipeline, starting with raw data and ending with presentable results. The arrows indicate transformational processes that move information from one box to another. The x axis approximates the overall level of effort required to move information from the raw to the specific processing stage. The y axis shows the amount of structure (with respect to the investigative process) in the processed information for every stage. Thus, the overall trend is to move the relevant information from the lower left-hand to the upper right-hand corner of the diagram. In reality, the processing can both meander through multiple iterations of local loops and jump over phases (for routine cases handled by an experienced investigator).

External data sources include all potential evidence sources for a specific investigation such as disk images, memory snapshots, network captures and reference databases such as hashes of known files. The *shoebox* is a subset of all the data that have been identified as potentially relevant, such as all the email communications between two persons of interest. At any given time, the contents of the shoebox can be viewed as the analyst's approximation of the information content that is potentially relevant to the case. The *evidence file* contains only the parts that are directly relevant to the case such as specific email exchanges on a topic of interest.

The *schema* contains a more organised version of the evidence such as a timeline of events or a graph of relationships, which allows higher-level reasoning over the evidence. A *hypothesis* is a tentative conclusion that explains the observed evidence in the schema and, by extension, could form the final conclusion. Once the analyst is satisfied that the hypothesis is supported by the evidence, the hypothesis turns into a *presentation*, which is the final product of the process. The presentation usually takes on the form of an investigator's report that both speaks to the high-level conclusions that are relevant to the legal case and also documents the low-level technical steps based on which the conclusion has been formed.

The overall analytical process is *iterative* in nature with two main activities loops: a *foraging*

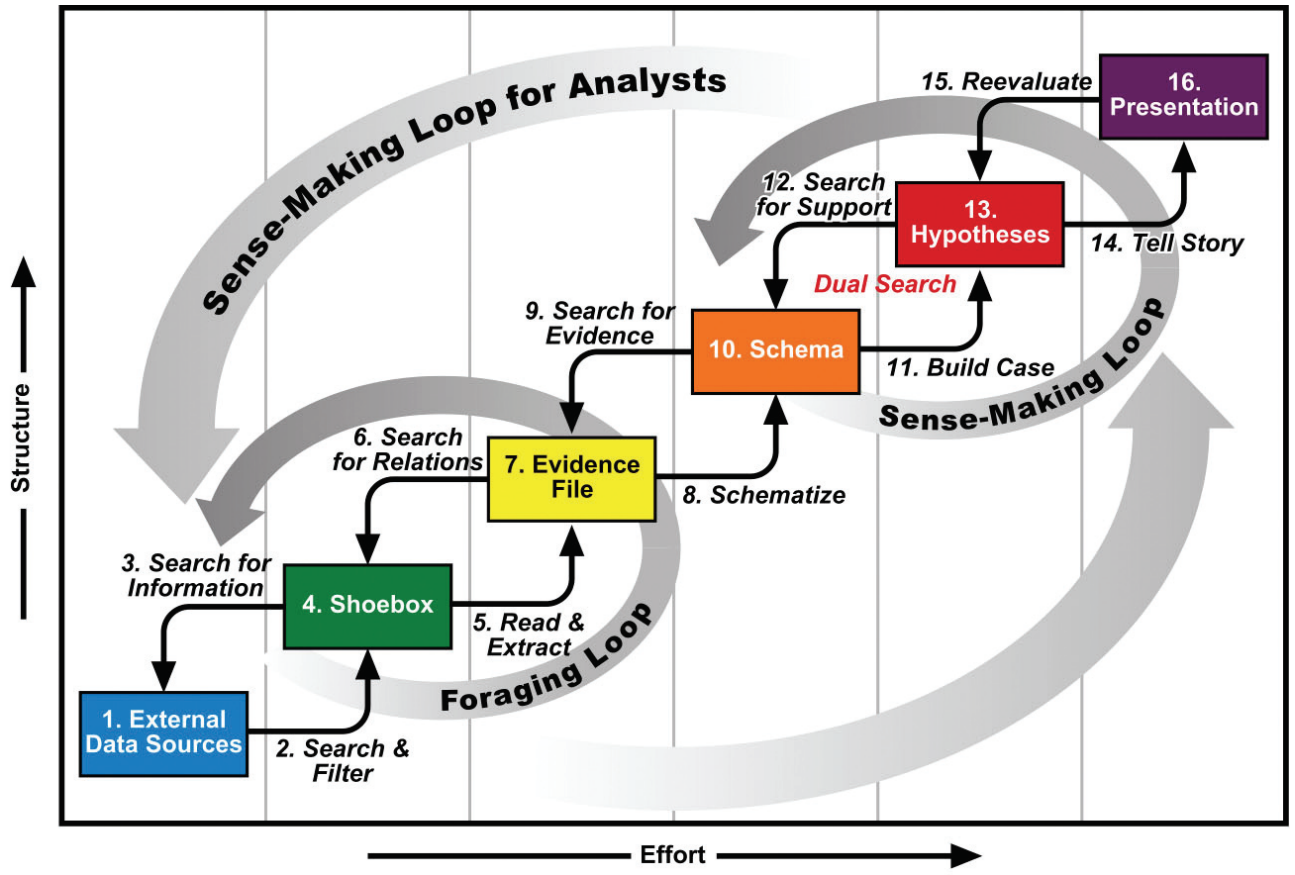


Figure 9.1: Notional model of sense-making loop for analysts derived from cognitive task analysis [933, p.44].

loop that involves the actions taken to find potential sources of information, and which then queries them and filters them for relevance; and a *sense-making loop* in which the analyst develops – in an iterative fashion – a conceptual model that is supported by the evidence. The information transformation processes in the two loops can be classified into bottom-up (organising data to build a theory) or top-down (finding data based on a theory) ones. Analysts apply these techniques in an opportunistic fashion with many iterations, in response to both newly discovered pieces of evidence, and to high-level investigative questions.

9.1.3.2 Bottom-Up Processes

Bottom-up processes are synthetic – they build higher-level (more abstract) representations of information from more specific pieces of evidence.

- *Search and filter*: External data sources, hard drives, network traffic, etc. are searched for relevant data based on keywords, time constraints and others in an effort to eliminate the vast majority of irrelevant data.
- *Read and extract*: Collections in the shoebox are analysed to extract individual facts and relationships that can support or disprove a theory. The resulting pieces of artifacts (e.g., individual email messages) are usually annotated with their relevance to the case.
- *Schematize*: At this step, individual facts and simple implications are organised into a schema that can help organise and identify the significance of and relationships between a growing number of facts and events. Timeline analysis is one of the basic tools of

the trade; however, any method of organising and visualising the facts-graphs, charts, etc.-can greatly speed up the analysis. This is not an easy process to formalise, and most forensic tools do not directly support it. Therefore, the resulting schemas may exist on a piece of paper, on a whiteboard or only in the mind of the investigator. Since the overall case could be quite complicated, individual schemas may cover specific aspects of it such as the discovered sequence of events.

- *Build case*: From the analysis of the schemas, the analyst eventually comes up with testable theories, or *working hypotheses*, that can explain the evidence. A working hypothesis is a tentative conclusion and requires more supporting evidence, as well as rigorous testing against alternative explanations. It is a central component of the investigative process and is a common point of reference that brings together the legal and technical sides in order to build a case.
- *Tell story*: The typical result of a forensic investigation is a final report and, perhaps, an oral presentation in court. The actual presentation may only contain the part of the story that is strongly supported by the digital evidence; weaker points may be established by drawing on evidence from other sources.

9.1.3.3 Top-Down Processes

Top-down processes are analytical – they provide context and direction for the analysis of less structured data search and they help organise the evidence. Partial or tentative conclusions are used to drive the search for supporting and contradictory pieces of evidence.

- *Re-evaluate*: Feedback from clients may necessitate re-evaluations, such as collecting stronger evidence or pursuing alternative theories.
- *Search for support*: A hypothesis may need more facts to be of interest and, ideally, would be tested against every (reasonably) possible alternative explanation.
- *Search for evidence*: Analysis of theories may require the re-evaluation of evidence to ascertain its significance/provenance, or it may trigger the search for more/better evidence.
- *Search for relations*: Pieces of evidence in the file can suggest new searches for facts and relations on the data.
- *Search for information*: The feedback loop from any of the higher levels can ultimately cascade into a search for additional information; this may include new sources, or the re-examination of information that was filtered out during previous passes.

9.1.3.4 The Foraging Loop

It has been observed [934] that analysts tend to start with a high-recall/low-selectivity query, which encompasses a fairly large set of documents – many more than the analyst can read. The original set is then successively modified and narrowed down before the documents are read and analysed.

The *foraging loop* is a balancing act between three kinds of processing that an analyst can perform- *explore*, *enrich* and *exploit*. Exploration effectively expands the shoebox by including larger amounts of data; enrichment shrinks it by providing more specific queries that include fewer objects for consideration; exploitation is the careful reading and analysis of an artifact to extract facts and inferences. Each of these options has varying costs and potential rewards and, according to information foraging theory [935], analysts seek to optimise their cost/benefit trade-offs.

Information foraging in this context is a highly iterative process with a large number of incremental adjustments in response to the emerging evidence. It is the responsibility of the investigator to keep the process on target and within the boundaries of any legal restrictions.

9.1.3.5 The Sense-Making Loop

Sense-making is a cognitive term and, according to Klein's [936] widely quoted definition, is the ability to make sense of an ambiguous situation. It is the process of creating situational awareness and understanding to support decision making in the face of uncertainty – an effort to understand connections between people, places and events in order to anticipate their trajectories and act effectively.

There are three main processes involved in the sense-making loop: *problem structuring*-the creation and exploration of hypotheses, *evidentiary reasoning* – the employment of evidence to support/disprove a hypothesis and *decision making*-selecting a course of action from a set of available alternatives.

It is important to recognize that the described information processing loops are closely tied together and often trigger iterations in either directions. New evidence may require a new working theory, whereas a new hypothesis may drive the search for new evidence to support or disprove it.

9.1.3.6 Data Extraction vs. Analysis vs. Legal Interpretation

Considering the overall process from Figure 9.1, we gain a better understanding of the roles and relationships among the different actors. At present, digital forensic researchers and tool developers primarily provide the means to acquire the digital evidence from the forensic targets, extract (and logically reconstruct) data objects from it, and the essential tools to search, filter, and organize it. In complex cases, such as a multi-national security incident, identifying and acquiring the relevant forensic targets can be a difficult and lengthy process. It is often predicated in securing the necessary legal rulings in multiple jurisdictions, as well as the cooperation of multiple organizations.

Forensic investigators are the primary users of these technical capabilities, employing them to analyse specific cases and to present legally-relevant conclusions. It is the responsibility of the investigator to drive the process and to perform all the information foraging and sense-making tasks. As the volume of the data being analysed continues to grow, it becomes ever

more critical for the forensic software to offer higher levels of automation and abstraction. Data analytics and natural language processing methods are starting to appear in dedicated forensic software, and – going forward – an expanding range of statistical and machine learning tools will need to be incorporated into the process.

Legal experts operate in the upper right-hand corner of the depicted process in terms of building/disproving legal theories. Thus, the investigator's task can be described as the translation of highly specific technical facts into a higher level representation and theory that explains them. The explanation is almost always connected to the sequence of the actions of the people that are part of the case, such as suspects, victims, and witnesses.

In summary, investigators need not be forensic software engineers, but they must be technically proficient enough to understand the significance of the artifacts extracted from data sources, and they must be able to read the relevant technical literature (peer-reviewed articles) in full. As the sophistication of the tools grows, investigators will need to have a working understanding of a growing list of data science methods that are employed by the tools in order to correctly interpret the results. Similarly, analysts must have a working understanding of the legal landscape, and they must be able to produce a competent report and present their findings on the witness stand, if necessary.

9.1.3.7 Forensic Process

The defining characteristic of forensic investigations is that their results must be admissible in court. This entails following established procedures for acquiring, storing, and processing of the evidence, employing scientifically established analytical tools and methods, and strict adherence to a professional code of practice and conduct.

Data Provenance and Integrity. Starting with the data acquisition process, an investigator must follow accepted standards and procedures in order to certify the provenance and maintain the integrity of the collected evidence. In brief, this entails acquiring a truthful copy of the evidence from the original source using validated tools, keeping custodial records and detailed case notes, using validated tools to perform the analysis of the evidence, cross-validating critical pieces of evidence, and correctly interpreting the results based on peer-reviewed scientific studies.

As discussed in the following section, data acquisition can be performed at different levels of abstraction and completeness. The traditional gold standard is a bit-level copy of the forensic target, which can then be analysed using knowledge of the structure and semantics of the data content. As storage devices increase in complexity and encryption becomes the default data encoding, it is increasingly infeasible to obtain a true physical copy of the media and a (partial) logical acquisition may be the only possibility. For example, the only readily available source of data content for an up-to-date smartphone (with encrypted local storage) might be a cloud backup of the user's data. Further, local data may be treated by the courts as having higher levels of privacy protection than data shared with a third party, such as a service provider.

Scientific Methodology. The notion of *reproducibility* is central to the scientific validity of forensic analysis; starting with the same data and following the same process described in the case notes should allow a third party to arrive at the same result. Processing methods should have scientifically established error rates and different forensic tools that implement the same type of data processing should yield results that are either identical, or within known statistical error boundaries.

The investigator must have a deep understanding of the results produced by various forensic computations. Some of the central concerns include: inherent uncertainties in some of the source data, the possibility for multiple interpretations, as well as the recognition that some of the data could be fake in that it was generated using anti-forensics tools in order to confuse the investigation. The latter is possible because most of the data item used in the forensic analysis is produced during the normal operation of the system, and is not tamper-proof. For example, an intruder with sufficient access privileges can arbitrarily modify any of the millions of file timestamps potentially making timeline analysis – a core analytical technique – unreliable. Experienced forensic analysts are alert to such issues and seek, whenever possible, to corroborate important pieces of information from multiple sources.

Tool Validation. Forensic tool validation is a scientific and engineering process that subjects specific tools to systematic testing in order to establish the validity of the results produced. For example, data acquisition software must reliably produce an unmodified and complete copy of the class of forensic targets it is designed to handle.

Forensic Procedure. The organizational aspect of the forensic process, which dictates how evidence is acquired, stored, and processed is critical to the issue of admissibility. Strict adherence to established standards and court-imposed restriction is the most effective means of demonstrating to the court that the results of the forensic analysis are truthful and trustworthy.

Triage. The volume of data contained by a forensic target typically far exceeds the amount of data *relevant* to an inquiry. Therefore, in the early stages of an investigation, the focus of the analysis is to (quickly) identify the relevant data and filter out the irrelevant. Such initial screening of the content, often referred to as *triage*, results in either follow up deep examination, or in deprioritisation, or removal of the target from further consideration.

Legally, there can be a number of constraints placed on the triage process based on the the case and the inherent privacy rights in the jurisdiction. From a technical perspective [937], "triage is a partial forensic examination conducted under (significant) time and resource constraints." In other words, investigators employ fast examination methods, such as looking at file names, examining web search history, and similar, to estimate (based on experience) the value of the data. Such results are inherently less reliable than a deep examination as it is easy to create a mismatch between data attribute and actual content. Therefore, courts may place constraints on the use of computers by convicted offenders to facilitate fast screening by officers in the field without impounding the device.

9.2 OPERATING SYSTEM ANALYSIS

[923, 938, 939]

Modern computer systems generally still follow the original von Neumann architecture [940], which models a computer system as consisting of three main functional units – CPU, main memory, and secondary storage – connected via data buses. To be precise, the *actual* investigative targets are not individual pieces of hardware, but the different Operating System (OS) modules controlling the hardware subsystems and their respective data structures.

Our discussion takes a high level view of OS analysis – it is beyond the scope of the Knowledge Area to delve into the engineering details of how different classes of devices are analysed. For example, smartphones present additional challenges with respect to data acquisition;

however, they are still commodity computers with the vast majority of them running on a *Linux* kernel. The same applies to other classes of embedded devices, such as UAVs and vehicle infotainment systems.

The OS functions at a higher level of privilege relative to user applications and directly manages all the computer system's resources – CPU, main memory, and I/O devices. Applications request resources and services from the OS via the system call interface and employ them to utilize them to accomplish a specific task. The (operating) system maintains a variety of accounting information that can bear witness to events relevant to an inquiry [938].

System analysis employs knowledge of how operating systems function in order to reach conclusions about events and actions of interest to the case. Average users have very little understanding of the type of information operating systems maintain about their activities, and usually do not have the knowledge and/or privilege level to tamper with system records thereby making them forensically useful, even if they do not fit a formal definition for secure and trustworthy records.

9.2.1 Storage Forensics

Persistent storage in the form of Hard Disk Drives (HDDs), Solid State Drives (SSDs), optical disks, external (USB-connected) media etc. is the primary source of evidence for most digital forensic investigations. Although the importance of (volatile) memory forensics in solving cases has grown significantly, a thorough examination of persistent data has remained a cornerstone of most digital forensic investigations.

9.2.1.1 Data Abstraction Layers

Computer systems organise raw storage in successive layers of abstraction – each software layer (some may be in firmware) builds an incrementally more abstract data representation that is only dependent on the interface provided by the layer immediately below it. Accordingly, forensic analysis of storage devices can be performed at several levels of abstraction [939]:

PHYSICAL MEDIA. At the lowest level, every storage device encodes a sequence of bits and it is possible, in principle, to use a custom mechanism to extract the data bit by bit. Depending on the underlying technology, this can be an expensive and time-consuming process, and often requires reverse engineering. One example of this process is the acquisition of mobile phone data, in some of which it is possible to physically remove (desolder) the memory chips and perform a true hardware-level acquisition of the content [941]. A similar "chip-off" approach can be applied to a flash memory devices, such as SSD, and to embedded and Internet of Things (IoT) devices with limited capabilities and interfaces. Another practical approach is to employ engineering tools that support the hardware development process and employ, for example, a standard JTAG interface [942] – designed for testing and debugging purposes – to perform the necessary data acquisition.

In practice, the lowest level at which typical examinations are performed is the Host Bus Adapter (HBA) interface. Adapters implement a standard protocol (SATA, SCSI) through which they can be made to perform low-level operations, such as accessing the drive's content. Similarly, the NVMe protocol [943] is used to perform acquisition from PCI Express-based solid-state storage devices.

All physical media eventually fail and (part of) the stored data may become unavailable.

Depending on the nature of the failure, and the sophistication of the device, it may be possible to recover at least some of the data. For example, it may be possible to replace the failed controller of a HDD and recover the content. Such hardware recovery becomes more difficult with more integrated and complex devices.

BLOCK DEVICE. The typical HBA presents a block device abstraction – the medium is presented as a sequence of fixed-size blocks, commonly consisting of 512 or 4096 bytes, and the contents of each block can be read or written using block read/write commands. The typical data acquisition process works at the block device level to obtain a working copy of the forensic target – a process known as *imaging* – on which all further processing is performed. Historically, the term *sector* is used to refer to the data transfer units of magnetic hard disks; a (logical) block is a more general term that is independent of the storage technology and physical data layout.

FILE SYSTEM. The block device has no notion of files, directories or – in most cases – which blocks are considered allocated and which ones are free; it is the filesystem's task to organise the block storage into a file-based store in which applications can create files and directories with all of their relevant metadata attributes – name, size, owner, timestamps, access permissions etc.

APPLICATION ARTIFACTS. User applications use the filesystem to store various artifacts that are of value to the end-user – documents, images, messages etc. The operating system itself also uses the file system to store its own image – executable binaries, libraries, configuration and log files, registry entries – and to install applications. Some application artifacts such as compound documents have a complex internal structure integrating multiple artifacts of different types. An analysis of application artifacts tends to yield the most immediately relevant results, as the recorded information most directly relates to actions and communications initiated by people. As the analysis goes deeper (to a lower level of abstraction), it requires greater effort and more expert knowledge to independently reconstruct the actions of the system. For example, by understanding the on-disk structures of a specific filesystem, a tool can reconstitute a file out of its constituent blocks. This knowledge is particularly costly to obtain from a closed system such as Microsoft Windows, because of the substantial amount of blackbox reverse engineering involved.

Despite the cost, independent forensic reconstruction is of critical importance for several reasons:

- It enables the recovery of evidentiary data not available through the normal data access interface.
- It forms the basis for recovering partially overwritten data.
- It allows the discovery and analysis of malware agents that have subverted the normal functioning of the system, thus making the data obtained via the regular interface untrustworthy.

9.2.2 Data Acquisition

In line with best practices [923], analysing data at rest is not carried out on a live system. The target machine is powered down, an exact bit-wise copy of the storage media is created, the original is stored in an evidence locker and all the forensic work is performed on the copy. There are exceptions to this workflow in cases where it is not practical to shut down the target system and, therefore, a media image is obtained while the system is live. Evidently, such an approach does not provide the same level of consistency guarantees, but it can still yield valuable insights. The problem of consistency, also referred to as *data smearing*, does not exist in virtualised environments, where a consistent image of the virtual disk can be trivially obtained by using the built-in snapshot mechanism.

As already discussed, obtaining data from the lowest level system interface available and independently reconstructing higher-level artifacts is considered the most reliable approach to forensic analysis. This results in a strong preference for acquiring data at lower levels of abstraction and the concepts of *physical* and *logical* acquisition.

> **Physical data acquisition** is the process of obtaining the data directly from hardware media, without the mediation of any (untrusted) third-party software.

An increasingly common example of this approach is mobile phone data acquisition that relies on removing the physical memory chip[941] and reading the data directly from it. More generally, getting physical with the evidence source is usually the most practical and necessary method for low-end embedded systems with limited hardware capabilities. Physical acquisition also affords access to additional over-provisioned raw storage set aside by the storage device in order to compensate for the expected hardware failures. As a general rule, devices offer no external means to interrogate this shadow storage area.

Chip-off techniques present their own challenges in that the process is inherently destructive to the device, the data extraction and reconstruction requires additional effort, and the overall cost can be substantial.

For general-purpose systems, tools use an HBA protocol such as SATA or SCSI to interrogate the storage device and obtain a copy of the data. The resulting image is a block-level copy of the target that is generally referred to as physical acquisition by most investigators; Casey uses the more accurate term *pseudo*-physical to account for the fact that not every area of the physical media is acquired and that the order of the acquired blocks does not necessarily reflect the physical layout of the device.

In some cases, it is necessary to perform additional recovery operations before a usable copy of the data is obtained. One common example is RAID storage devices, which contain multiple physical devices that function *together* as a single unit, providing built-in protection against certain classes of failures. In common configurations such as RAID 5 and 6 the content acquisition of individual drives is largely useless without the subsequent step of RAID data reconstruction.

Modern storage controllers are quickly evolving into autonomous storage devices, which implement complex (proprietary) wear-levelling and load-balancing algorithms. This has two major implications: a) the numbering of the data blocks is completely separated from the actual physical location; and b) it is possible for the storage controller itself to become compromised [944], thus rendering the acquisition process untrustworthy. These caveats notwithstanding, we will refer to block-level acquisition as being physical, in line with the

accepted terminology.

Logical data acquisition relies on one or more software layers as intermediaries to acquire the data from the storage device.

In other words, the tool uses an Application Programing Interface (API), or message protocol, to perform the task. The integrity of this method hinges on the correctness and integrity of the implementation of the API, or protocol. In addition to the risk, however, there is also a reward – higher level interfaces present a data view that is closer in abstraction to that of user actions and application data structures. Experienced investigators, if equipped with the proper tools, can make use of both physical and logical views to obtain and verify the evidence relevant to the case.

Block-level acquisition can be accomplished in software, hardware or a combination of both. The workhorse of forensic imaging is the dd Unix/Linux general purpose command-line utility, which can produce a binary copy of any file, device partition or entire storage device. A hardware write blocker is often installed on the target device to eliminate the possibility of operator error, which can lead to the accidental modification of the target.

Cryptographic hashes are computed for the entire image and (preferably) for every block; the latter can be used to demonstrate the integrity of the remaining evidence if the original device suffers a partial failure, which makes it impossible to read its entire contents. The National Institute of Standards and Technology (NIST) maintains the Computer Forensic Tool Testing (CFTT) project [945], which independently tests various basic tools, such as write blockers and image acquisition tools and regularly publishes reports on its findings.

Encryption Concerns

Apart from having the technical capability to safely interrogate and acquire the content of a storage device, one of the biggest concerns during data acquisition can be the presence of encrypted data. Modern encryption is pervasive and is increasingly applied by default to both stored data and data in transit over the network. By definition, a properly implemented and administered data security system, which inevitably employs encryption, will frustrate efforts to acquire the protected data and, by extension, to perform forensic analysis.

There are two possible paths to obtaining encrypted data – technical and legal. The technical approach relies on finding algorithmic, implementation, or administrative errors, which allow the data protection to be subverted. Although it is nearly impossible to create a complex IT system that has no bugs, the discovery and exploitation of such deficiencies is becoming increasingly more difficult and resource intensive.

The legal approach relies on compelling the person with knowledge of the relevant encryption keys to surrender them. This is relatively new legal territory and its treatment varies across jurisdictions. In the UK, the *Regulation of Investigatory Powers Act 2000* specifies the circumstances under which individuals are legally required to disclose the keys. Disclosure may run counter the legal right against self-incrimination and in some jurisdictions, such as in the United States, it is not yet definitively resolved.

The remainder of this discussion assumes that access to the raw data is ensured by either technical, or legal means, which are beyond the scope of this knowledge area.

9.2.3 Filesystem Analysis

A typical storage device presents a block device interface with B_{max} number of blocks of size B_{size}. All read and write I/O operations are executed at the granularity of a whole block; historically, the standard block size adopted by HDD manufacturers has been 512 bytes. With the 2011 introduction of the advanced format standard [946], storage devices can support larger blocks, with 4,096 bytes being the new preferred size.

Regardless of the base block size, many operating systems manage storage in clusters; a *cluster* is a contiguous sequence of blocks and is the smallest unit at which raw storage is allocated/reclaimed. Thus, if the device block/sector size is 4KiB but the chosen cluster size is 16KiB, the OS will allocate blocks in groups of four.

For administration purposes, the raw drive may be split into one or more contiguous areas called *partitions*, each of which has a designated use and can be independently manipulated. Partitions can further be organised into volumes – a *physical volume* maps onto a single partition, whereas a *logical volume* can integrate multiple partitions potentially from multiple devices. Volumes present a block device interface but allow for the decoupling of the physical media organisation from the logical view presented to the operating system.

With a few exceptions, volumes/partitions are formatted to accommodate a particular file system (filesystem), which organizes and manages the blocks to create the abstraction of files and directories together with their relevant metadata. The Operating System (OS), as part of its system call interface used by applications to request services, provides a filesystem API that allows applications to create, modify and delete files; it also allows files to be grouped into a hierarchical structure of directories (or folders).

> A **file** is a named (opaque) sequence of bytes that is stored persistently.

As a general rule, the format and interpretation of file content is almost always outside the purview of the operating system; it is the concern of relevant applications acting on behalf of users.

> A **file system** (filesystem) is an OS subsystem that is responsible for the persistent storage and organisation of user and system files on a partition/volume.

It provides a high-level standard API such as POSIX, that is used by applications to store and retrieve files by name without any concern for the physical storage method employed or the layout of the data (and metadata) content.

Filesystem forensics uses knowledge of the filesystem's data structures and the algorithms used to create, maintain, and delete them to: a) extract data content from devices independently of the operating system instance which created it; and b) extract leftover artifacts to which the regular filesystem API does not offer access.

The first feature is important to ensure that the data are not being modified during acquisition and that any potential security compromises do not affect the validity of the data. The second provides access to (parts of) deallocated files that have not been overwritten, purposely hidden data, and an implied history of the filesystem operation – the creation/deletion of files – that is not explicitly maintained by the OS.

9.2.4 Block Device Analysis

Before the OS can organise a filesystem on a raw device, it typically splits it into a set of one or more disjoint *partitions*.

A block device **partition**, or *physical volume*, is a contiguous allocation of blocks for a specific purpose, such as the organisation of a file system.

Partitions are the basic method used for coarse-grained storage management; they allow a single physical device to be dedicated to multiple purposes such as hosting different filesystems or separating system from user files. If a subdivision is not needed, the entire device can be trivially allocated to a single partition.

A **logical volume** is a collection of physical volumes presented and managed as a single unit.

Logical volumes allow storage capacity from different devices to be pooled transparently (to the filesystem) to simplify the use of available capacity. They also enable automated block-level replication in the form of RAIDs [947] for enhanced performance and durability.

9.2.5 Data Recovery & File Content Carving

One of the early staples of data recovery tools was the 'undelete' functionality, which can reverse the effects of users deleting data. The most common case is that of users deleting a file and needing to reverse the operation. On a HDD, this reversal is readily achievable immediately after the deletion - the storage taken up by the file's content is merely deallocated (marked as available), but no actual destruction (sanitisation) of the data takes place.

A more difficult case is a HDD that has been in use for some time and that has been subsequently formatted (e.g., by somebody attempting to destroy evidence). The often employed quick format command has the effect of overlaying a set of data structures that correspond to an empty filesystem (a full format sanitizes the content of the media but can take hours to complete so it is used less frequently). Thus, the normal filesystem interface, after querying these structures, will report that there are no files. The reality is that – at that moment – only filesystem metadata has been partially overwritten, and all the data blocks representing the file content are still present on the media in full.

Forensic computing, unlike most other types of computation, is very interested in all recoverable (partial) artifacts, including (and sometimes especially) deallocated ones. Unless a user has taken special measures to securely wipe a hard disk, at any given time the media contains recoverable application artifacts (files) that have ostensibly been deleted. The process of restoring the artifacts is commonly accomplished by *carving*.

File (content) carving is the process of recovering and reconstructing file content directly from block storage without using the filesystem metadata. More generally, **data (structure) carving** is the process of reconstructing logical objects (such as files and database records) from a bulk data capture (disk/RAM image) without using metadata that describes the location and layout of the artifacts.

File carving is the oldest and most commonly used, technique and its basic form is based on

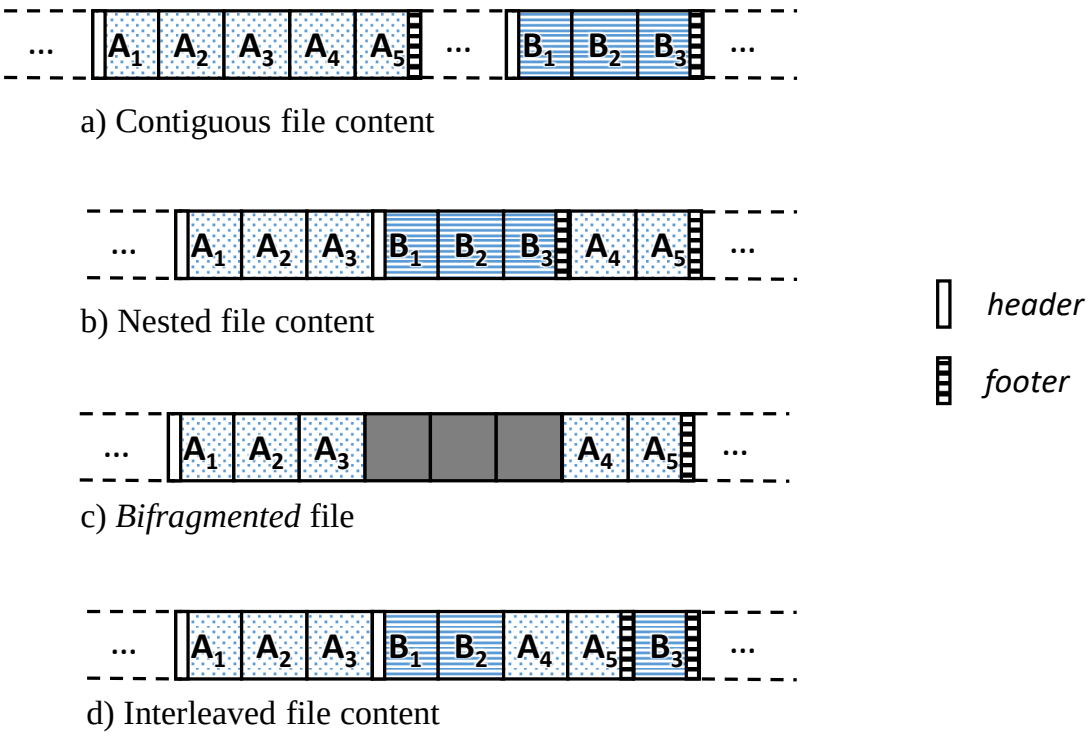


Figure 9.2: Common file content layout encountered during carving.

two simple observations: a) most file formats have specific beginning and end tags (a.k.a. *header* and *footer*); and b) file systems strongly favour a sequential file layout to maximise throughput.

Put together, these yield a basic recovery algorithm: 1) scan the capture sequentially until a known header is found; for example, JPEG images always start with the (hexadecimal) FF D8 FF header; 2) scan sequentially until a corresponding footer is found; FF D9 for JPEG; 3) copy the data in between as the recovered artifact. Figure 9.2 illustrates some of the most common cases encountered during file carving:

1. *No fragmentation* is the most typical case, as modern filesystems require extra effort to ensure sequential layout for optimal performance.
2. *Nested content* is often the result of deletion; in the example, after the initial sequential back-to-back layout of the files, the content ahead and behind file B was deleted and replaced by A. In some cases, the file format allows nesting; e.g., JPEGs commonly have a thumbnail version of the image, which is also in JPEG format. This case can be solved by making multiple passes – once B is carved out (and its blocks removed from further consideration) the content of A becomes contiguous, so a subsequent pass will readily extract it.
3. *Bi-fragmented files* are split into two contiguous pieces with the other content in between, which also determines how difficult the reconstruction is; if the content in the middle is easily distinguished from the content of the file (e.g., the pieces of text in the middle of a compressed image) then the problem is relatively easy. Otherwise, it is ambiguous and it could be quite difficult to identify the matching pieces.
4. *Interleaved content* is a more complicated version of nesting which happens when larger files are used to fill the gaps created by the deletion of smaller ones.

This simple carving approach usually yields a good number of usable artifacts; however, real data can contain a number of atypical patterns, which can lead to a large number of repetitive and/or false positive results. One major reason is that file formats are not designed with carving in mind and rarely have robust internal metadata that connect the constituent pieces together. Some do not even have a designated header and/or footer, and this can result in a large number of false positives, potentially producing results substantially larger in volume than the source data.

Slack space recovery. Both RAM and persistent storage are almost always allocated in multiples of a chosen minimum allocation units. Therefore, at the end of the allocated space, there is storage capacity – slack space – that is not used by the application, but is also *not* available for other uses. For example, if the minimum allocation is 4KiB, and a file needs 14KiB, the filesystem will allocate four 4KiB blocks. The application will fully use the first three blocks, but will only use 2KiB from the last block. This creates the potential to store data that would be inaccessible via the standard filesystem interface and can provide a simple means to hide data.

> **Slack space** is the difference between the *allocated* storage for a data object, such as file, or a volume, and the storage in actual use.

Once aware of the potential for storing hidden data in slack space, it is relatively easy to identify and examine it, and this is a standard step in most investigations.

Upcoming challenges. As solid state drives continue to grow in capacity and displace hard disks from an increasing proportion of operational data storage, file carving's utility is set to diminish over time. The reason lies in the fact that SSD blocks need to be written twice in order to be reused (the first write resets the state of the block, thereby enabling its reuse). To improve performance, the TRIM and UNMAP commands were added to the ATA and SCSI command sets, respectively; they provide a mechanism for the filesystem to indicate to the storage device which blocks need to be garbage collected and prepared for reuse.

King & Vidas [948] established experimentally that file carving would only work in a narrow set of circumstances on modern Solid State Drives (SSDs). Specifically, they show that for a TRIM-aware operating system, such as Windows 7 and after, the data recovery rates in their tests were almost universally zero. In contrast, using a pre-TRIM OS (Windows XP) allows for near-perfect recovery rates under the same experimental conditions.

9.3 MAIN MEMORY FORENSICS

[949]

The early view of best forensic practices was to literally pull the plug on a machine that was to be impounded. The rationale was that this would remove any possibility of alerting the processes running on the host and would preempt any attempts to hide information. Over time, experience has shown that these concerns were largely exaggerated and that the substantial and irreversible loss of important forensic information such as open connections and encryption keys was rarely justified. Studies have clearly demonstrated that data tend to persist for a long time in volatile memory ([950], [951]). There is a wealth of information about a system's run-time state that can be readily extracted, even from a snapshot [949]:

Process information. It is practical to identify and enumerate all the running processes, threads and loaded systems modules; we can obtain a copy of the individual processes' code, stack, heap, code, and data segments. All this is particularly useful when analysing compromised machines, as it allows the identification of suspicious services, abnormal parent/child relationships, and, more generally, to search for known symptoms of compromise, or patterns of attack.

File information. It is practical for identifying any open files, shared libraries, shared memory, and anonymously mapped memory. This is particularly useful for identifying correlated user actions and file system activities, potentially demonstrating user intent.

Network connections. It is practical for identifying open and recently closed network connections and protocol information, as well as sending and receiving queues of data not yet sent or delivered, respectively. This information could readily be used to identify related parties and communication patterns between them.

Artifacts and fragments. Just like the filesystem, the memory management system tends to be reactive and leaves a lot of artifact traces behind. This is primarily an effort to avoid any processing that is not absolutely necessary for the functioning of the system; caching disk and network data tends to leave traces in memory for a long time.

Memory analysis can be performed either in real time on a live (running) system, or it could be performed on a snapshot (memory dump) of the state of the system. In addition to using specialized memory acquisitions tools, or a build-in snapshot mechanism (in virtualized environments) memory content can also be obtained from a system hibernation file, page swap, or a crash dump.

In live forensics, a trusted agent (process) designed to allow remote access over a secure channel is pre-installed on the system. The remote operator has full control over the monitored system and can take snapshots of specific processes, or the entire system. Live investigations are an extension of regular security preventive mechanisms, which allow for maximum control and data acquisition; they are primarily used in large enterprise deployments.

The main conceptual problem of working on a live system is that, if it is compromised, the data acquisition and analysis results are not trustworthy; therefore, forensic analysis is most frequently performed on a snapshot of the target system's RAM. Analysing a snapshot is considerably more difficult than working with a live system, which provides access to the state of the running system via a variety of APIs and data structures. In contrast, a raw memory capture offers no such facilities and forensic tools need to rebuild the ability to extract semantic information from the ground up. This is a semantic gap problem, and the purpose of memory forensics is to bridge it.

9.4 APPLICATION FORENSICS

Application forensics is the process of establishing a data-centric theory of operation for a specific application. The goal of the analysis is to objectively establish causal dependencies between data input and output, as a function of the user interactions with the application. Depending on whether an application is an open or closed source and on the level of the accompanying documentation, the analytical effort required can vary from reading detailed specifications to reverse engineering code, data structures and communication protocols, to performing time-consuming black box differential analysis experiments. Alternatively, forensic tool vendors may license code from the application vendor to gain access to the proprietary data structures.

The big advantage of analysing applications is that we have a better chance of observing and documenting direct evidence of user actions, which is of primary importance to the legal process. Also, the level of abstraction of the relevant forensic traces tend to have a level of abstraction corresponding to a particular domain.

9.4.1 Case Study: the Web Browser

Although there are at least four major web browsers in common use, after more than 20 years of development, their capabilities have converged, thus allowing us to talk about them in common terms. There are six main sources of forensically interesting information:

URL/search history. At present, there are no practical barriers to maintaining a complete browsing history (a log of visited websites), and making it available to users is a major usability feature; most users rarely delete this information. Separately, service providers such as Google and Facebook, are interested in this information for commercial reasons, and make it easy to share a browsing log with multiple devices. Combined with the content of the local file cache, the browsing history allows an investigator to almost look over the shoulder of the user of interest as they were navigating the Web. In particular, analysing user queries to search engines is among the most commonly employed techniques. The search query is encoded as part of the URL, and can often provide very clear and targeted clues as to what the user was trying to accomplish.

Form data. Browsers offer the convenience of remembering auto-completing passwords and other form data (such as address information). This can be very helpful to an investigator, especially if the user is less security conscious and does not use a master password to encrypt all of this information.

Temporary files. The local file cache provides its own chronology of web activities, including a stored version of the actual web objects that were downloaded and shown to the user (these may no longer be available online). Although caching has become considerably less effective owing to the increased use of dynamic content, this is tempered by the large increase in available storage capacity, which places very few, if any, practical constraints on the amount of data cached.

Downloaded files are, by default, never deleted providing another valuable source of activity information.

HTML5 local storage provides a standard means for web applications to store information locally; for example, this could be used to support disconnected operations, or to provide a

measure of persistence for user input. Accordingly, the same interface can be interrogated to reconstruct web activities.

Cookies are opaque pieces of data used by servers to keep a variety of information on the web client in order to support transactions such as web mail sessions. In practice, most cookies are used by websites to track user behaviour, and it is well-documented that some providers go to great lengths to make sure that this information is resilient. Some cookies are time-limited access tokens that can provide access to online accounts (until they expire); others have a parsable structure and may provide additional information.

Most local information is stored in SQLite databases, which provide a secondary target for data recovery. In particular, ostensibly deleted records may persist until the database is explicitly 'vacuumed'; otherwise, they remain recoverable at a later time ([952, 953]).

9.5 CLOUD FORENSICS

Cloud computing is fast emerging as the primary model for delivering information technology (IT) services to Internet-connected devices. It brings both disruptive challenges for current forensic tools, methods and processes, as well as qualitatively new forensic opportunities. It is not difficult to foresee that, after an intermediate period of adjustment, digital forensics will enter a new period marked by substantially higher levels of automation and will employ much more sophisticated data analytics. Cloud computing environments will greatly facilitate this process, but not before bringing about substantial changes to currently established tools and practices.

9.5.1 Cloud Basics

Conceptually, cloud-based IT abstracts away the physical compute and communication infrastructure, and allows customers to rent as much compute capacity as needed. Cloud systems have five essential characteristics: on-demand self service, broad network access, resource pooling, rapid elasticity, and measured service. [124]

The cloud is enabled by a number of technological developments, but its adoption is driven primarily by business considerations, which drive changes to how organisations and individuals use IT services. Accordingly, it also changes how software is developed, maintained and delivered to its customers. Cloud computing services are commonly classified into one of three canonical models – Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). In actual deployments, the distinctions can be blurred and many cloud deployments (and potential investigative targets) incorporate elements of all of these.

The differences between the models are best understood when we consider the virtualised computing environments as a stack of layers: hardware such as storage, and networking; virtualisation, consisting of a hypervisor allowing the installation and lifecycle management of virtual machines; operating system, installed on each virtual machine; middleware and runtime environment; and application and data.

Each of the cloud models splits the responsibility between the client and the Cloud Service Provider (CSP) at different levels in the stack (Figure 9.3). In a private (cloud) deployment, the entire stack is hosted by the owner and the overall forensic picture is very similar to the

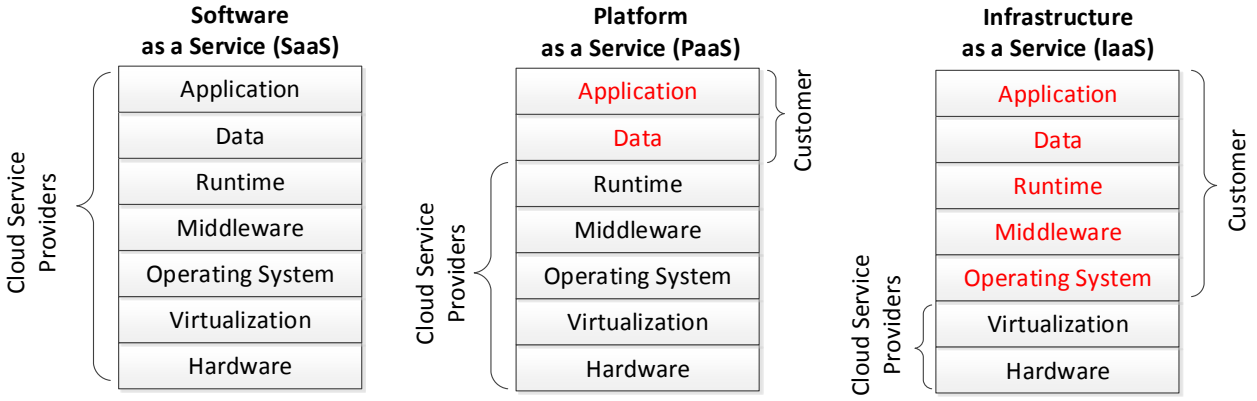


Figure 9.3: Layers of cloud computing environment owned by customer and cloud service provider on three service models: IaaS, PaaS, and SaaS (public cloud).

problem of investigating a non-cloud IT target. Data ownership is clear, as is the legal and procedural path to obtain it; indeed, the very use of the term 'cloud' in this situation is not particularly significant to a forensic inquiry.

In a public deployment, the SaaS/PaaS/IaaS classification becomes important, as it indicates the ownership of data and service responsibilities. Figure 9.3 shows the typical ownership of layers by customer and service providers under different service models. In hybrid deployments, layer ownership can be split between the customer and the provider, and/or across multiple providers. Further, it can change over time, as, for example, the customer may handle the base load on private infrastructure, but burst into the public cloud to handle peak demand, or system failures.

9.5.2 Forensic Challenges

The main technical challenges to established forensic practices can be summarised as follows.

Logical acquisition is the norm. The existing forensic toolset is almost exclusively built to work with the leftover artifacts of prior computations. It relies on algorithmic knowledge of different OS subsystems such as the filesystem in order to interpret the physical layout of the data as acquired from the device.

Physical acquisition is almost completely inapplicable to the cloud, where data moves, resources are shared and ownership and jurisdictional issues can be complicated. Cloud service APIs are emerging as the primary new interface through which data acquisition is being performed.

The cloud is the authoritative data source. Another important reason to query cloud services for relevant information is that they store the primary historical record of computations and interactions with users. Most residual information on the client, such as a cloud drive is transient and often of uncertain provenance.

Logging is pervasive. Cloud-based software is developed and organised differently. Instead of one monolithic piece of code, the application logic is decomposed into several layers and modules that interact with each other over well-defined service interfaces. Once the software components and their communication are formalised, it becomes easy to organise extensive logging of every aspect of the system. Indeed, it becomes critical to have this information

just to be able to debug, test and monitor cloud applications and services.

These developments point to logs (of user and system activities) becoming the primary source of forensic information. The immediate implication is that much more will be explicitly known – as opposed to deduced – about the historical state of applications and artifacts. This will require a new set of data analytics tools and will completely transform the way forensic investigations are performed. It will also bring new challenges in terms of long-term case data preservation.

Distributed computations are the norm. The key attribute of the client/standalone model is that practically all computations take place on the device itself. Applications are monolithic, self-contained pieces of code that have immediate access to user input and consume it instantly with (almost) no traces left behind. Since a large part of forensics comprises attributing the observed state of a system to user-triggered events, forensic research and development has relentlessly focused on two driving problems – discovering every last piece of log/timestamp information, and extracting every last bit of discarded data left behind by applications or the operating system.

The cloud model, particularly SaaS, completely breaks with this approach – the computation is split between the client and the server, with the latter performing the heavy computational lifting and the former performing predominantly user interaction functions. Code and data are downloaded on demand and have no persistent place with regard to the client. The direct consequence is that the vast majority of the established forensic tool chain becomes irrelevant, which points to a clear need for a different approach.

9.5.3 SaaS Forensics

The software industry's traditional delivery model is Software as a Product (SaaP); that is, software acquired like any physical product and is installed by the owner on a specific machine, where all the computations are performed. As a result, the traditional analytical model of digital forensics is physical device-centric – the investigator works with physical evidence carriers such as storage media or integrated compute devices (e.g., smartphones). On the client (or standalone) device, it is easy to identify where the computations are performed and where the results/traces are stored. The new software delivery model – Software as a Service (SaaS) – is subscription-based and did not start becoming practical until the widespread adoption of fast broadband access some ten to fifteen years ago.

The cloud renders many device-centric methods — especially those focused on low-level physical acquisition and analysis — irrelevent. It also requires the development of new tools that can work in the new deployment environment, where the code execution is split between the server and the client devices, the primary storage interface is a service API and the application artifacts are not persistently stored on the device (although local storage may be used as a cache).

Case Study: Cloud Drive Acquisition. Cloud drive services, such as *Dropbox*, *Google Drive* and *Microsoft OneDrive* are the SaaS version of the local storage device, which is central to modern digital forensics. The problem of cloud drive acquisition, a clear first investigative step, is a good illustration of the challenges and opportunities offered by SaaS with respect to forensics.

At first, it may appear that simply copying a local replica of the drive's content is a simple and effective solution. However, this approach offers no guarantees with respect to the accuracy and completeness of the acquisition. Specifically, there are three major concerns:

Partial replication. The most obvious problem is that there is no guarantee that any of the clients attached to an account will have a complete copy of the (cloud) drive's content. As data accumulates online, it quickly becomes impractical to keep full replicas on every device; indeed, it is likely that most users will have *no* device with a complete copy of the data. Furthermore, the acquisition tool needs direct access to the cloud drive's metadata to ascertain its contents; without this information, the acquisition is of an unknown quality, subject to potentially stale and omitted data.

Revision acquisition. Most drive services provide some form of revision history; the look-back period varies, but this is a standard feature that users expect, especially in paid services. Although there are some analogous data sources in traditional forensics, such as archival versions of important OS data structures, the volume and granularity of the revision information in cloud application are qualitatively and quantitatively different. Revisions reside in the cloud and clients rarely have anything but the most recent version in their cache; a client-side acquisition will clearly miss prior revisions, and does not even have the means to identify these omissions.

Cloud-native artifacts. The mass movement towards web-based applications means that forensics needs to learn how to deal with a new problem – digital artifacts that have no serialised representation in the local filesystem. For example, Google Docs documents are stored locally as a link to the document which can only be edited via a web app. Acquiring an opaque link without the actual content of the document has minimal forensic utility. Most services provide the means to export the web app artifact in a standard format such as PDF; however, this can only be accomplished by requesting directly from the service (manually or via an API).

In summary, bringing the traditional client-side approach to drive acquisition to bear on SaaS acquisition has major *conceptual* flaws that are beyond remediation; a new approach is needed, one that obtains the data directly from the cloud service.

9.6 ARTIFACT ANALYSIS

[954, 955, 956, 957, 958]

Once the external (serialised) representation of a digital artifact such as a text document or an image is standardised, it provides a convenient level of abstraction, thus allowing the development of artifact-centric forensic techniques.

9.6.1 Finding a Known Data Object: Cryptographic Hashing

The lowest common denominator for all digital artifacts is to consider them as a sequence of bits/bytes without trying to parse, or assign any semantics to them. Despite this low level of abstraction, some crucial problems can be addressed, the most important one being to identify known content, usually files.

Cryptographic hashing is the first tool of choice when investigating any case; it provides a basic means of validating data integrity and identifying known artifacts. Recall that a hash function takes an arbitrary string of binary data and produces a number, often referred to as a digest, within a predefined range. Ideally, given a set of different inputs, the hash function will map them onto different outputs.

Hash functions are *collision-resistant* if it is computationally infeasible to find two different inputs for which the output is the same. Cryptographic hash functions such as MD5, RIPEMD-160, SHA-1, SHA-2 and the current NIST standard SHA-3[954], are designed to be collision-resistant and produce large 128- to 512-bit results.[1] Since the probability that hashing two different data objects will produce the same digest by chance is astronomically small, we can safely assume that, *if two objects have the same crypto digest, then the objects themselves are identical*.

Current practice is to apply a crypto hash function either to an entire target (drive, partition etc.) or to individual files. The former is used to validate the integrity of the forensic target by comparing before-and-after results at important points in the investigation (e.g., to demonstrate that the integrity of the evidence throughout the chain of custody) whereas the latter are used to work with known files. This involves either removing from consideration common files such as OS and application installations or pinpointing known files of interest such as malware and contraband. The US National Institute of Standards and Technology (NIST) maintains the National Software Reference Library (NSRL) [955], which covers the most common operating system installation and application packages. Other organisations and commercial vendors of digital forensic tools provide additional hash sets of other known data.

From a performance and efficiency perspective, hash-based file filtering is very attractive – using a 20-byte SHA-1 hash, the representation of 50 million files takes only 1 GB. This makes it possible to load a reference set of that size in the main memory and filter out, on the fly, any known files in the set as data is read from a forensic target.

9.6.2 Block-Level Analysis

In addition to whole files, investigators are often interested in discovering known file remnants, such as those produced when a file is marked as deleted and subsequently partially overwritten. One routinely used method to address this problem is to increase the granularity of the hashes by splitting the files into fixed-size blocks and storing the hash for each individual block. The block size is commonly set to 4 KiB to match the minimum allocation unit used by most operating systems' installations. Given a block-based reference set, a forensic target (RAM capture or disk image) can be treated as a sequence of blocks that can be read block by block, hashed and compared to the reference set.

In this context, we say that a block is *distinct*, if the probability that its exact content arises by chance more than once is vanishingly small. If we knew for a fact that a specific block was unique and specific to a particular file, then (in terms of evidentiary value) finding it on a forensic target would be almost the same as finding the entire file from which it was derived. In practice, we cannot definitely know the distinctiveness of every possible data block; therefore, we use an approximating assumption based on empirical data:

"If a file is known to have been manufactured using some high-entropy process, and if the blocks of that file are shown to be distinct throughout a large and representative corpus, then those blocks can be treated as if they are distinct." [956] Perhaps the most common transformation that yields high-entropy data is data compression, which is routinely employed in many common file formats, such as audio/video and office documents.

Apart from the direct use of blocks as trace evidence for the (past or current) presence of known files, block hashes can be used to improve file carving results by excluding every known

[1] A discussion on the known vulnerabilities of cryptographic hash functions is outside the scope of this text.

blocks *before* performing the carving process. This can improve results by reducing gaps and eliminating certain classes of false positive results.

9.6.3 approximate matching

A natural generalisation of the problem of finding identical data objects is to find *similar* ones. In the context of digital forensics, the accepted umbrella term for similarity-based techniques is Approximate Matching (AM). As per NIST's definition, 'approximate matching is a generic term describing any technique designed to identify similarities between two digital artifacts'. [957]

This broad term encompasses methods that can work at different levels of abstraction. At the lowest level, artifacts can be treated as bit strings; at the highest levels, similarity techniques could employ, for example, natural language processing and image recognition methods to provide a level of reasoning that is much closer to that of a human analyst. With regard to the whole spectrum of similarity methods, lower-level ones are more generic and computationally affordable, whereas higher-level ones tend to be more specialised and require considerably more computational resources. Therefore, we would expect a forensic investigation to customise its use of AM techniques based on the goals of the analysis and the target data.

Use Cases. Using a common information retrieval terminology, it is useful to consider two variations of the similarity detection problem: *resemblance* and *containment* [959]. Resemblance queries compare two comparably-sized data objects (peers) and seek to infer how closely related they are. Two common forensic applications include: (a) object similarity detection – correlating artifacts that a person would classify as versions of each other; and (b) cross correlation – correlating objects that share the same components, such as an embedded image.

In the case of containment, we compare artifacts that have a large disparity in terms of size and seek to establish whether a larger one contains (pieces of) a smaller one. Two common variations are *embedded object detection* – establishing whether a smaller object (such as an image) is part of a larger one (such as a PDF document), and *fragment detection* - establishing whether a smaller object is a fragment (such as a network packet or disk block) of a bigger one, such as a file.

The difference between resemblance and containment is case-specific and the same tool may work in both cases. However, it is important for analysts to put the tool results into the correct context and to understand the performance envelope of the tools they are using in order to correctly interpret the results.

Definitions. The notion of similarity is specific to the particular context in which it is used. An approximate matching algorithm works by defining two essential elements – features and a similarity function. Features are the atomic components derived from the artifacts through which the artifacts are compared. Comparing two features yields a binary outcome – zero or one – indicating whether the feature match was successful or not. The set of all the features computed by an algorithm for a given artifact constitutes a feature set. It can be viewed as an approximate representation of the original object for the purposes of matching it with other objects.

The similarity function maps a pair of feature sets to a similarity range; it is increasingly monotonic with respect to the number of matching features. That is, all else being equal,more

feature matches yield a higher similarity score.

Classes. It is useful to consider three general classes of approximate matching algorithms. *Bytewise* matching considers the objects it compares to a sequence of bytes, and makes no effort to parse or interpret them. Consequently, the features extracted from the artifact are also byte sequences, and these methods can be applied to any data blob. The utility of the result depends heavily on the encoding of the data. If small changes to the content of the artifact result in small changes to the serialised format (e.g., plain text), then the bytewise similarity tends to correlate well with a person's perception of similarity. Conversely, if a small change can trigger large changes in the output (e.g., compressed data), then the correlation would be substantially weaker.

Syntactic matching relies on parsing the format of an object, potentially using this knowledge to split it into a logical set of features. For example, a zip archive or a PDF document could easily be split into constituent parts without understanding the underlying semantics. The benefit is that this results in a more accurate solution with more precisely interpretable results; the downside is that it is a more specialised solution, requiring additional information to parse different data formats.

Semantic matching (partially) interprets the data content in order to derive semantic features for comparison. Examples include perceptual hashes that can detect visually similar images, and methods of information retrieval and natural language processing that can find similarities in the subject and content of text documents.

Researchers use a variety of terms to name the different approximate matching methods they have developed: *fuzzy hashing* and *similarity hashing* refer to bytewise approximate matching; *perceptual hashing* and *robust hashing* refer to semantic approximate matching techniques.

Bytewise approximate matching algorithms are the most frequently used AM algorithms in forensics; they follow an overall pattern of extracting a feature set and generating a similarity digest, followed by a comparison of the digests. A similarity digest (a.k.a., fingerprint or signature) is a (compressed) representation of the feature set of the target artifact. It often employs hashing and other techniques to minimise the footprint of the set and to facilitate fast comparison.

9.6.4 Cloud-Native Artifacts

Forensic analysis of cloud systems is still in its early stages of development, but it will quickly grow in importance. One new and promising area is the analysis of cloud(-native) artifacts—data objects that maintain the persistent state of web/SaaS applications. [958] Unlike traditional applications, in which the persistent state takes the form of files in a local file system, web apps download the necessary state on the fly and do not rely on local storage. Recall that a web app's functionality is split between server and client components, and the two communicate via web APIs. From a forensic perspective, the most interesting API calls involve (complete) state transfer; for example, opening a document or loading a prior version, triggers the transfer of its full content. Conceptually, this is analogous to the process of opening and reading the content of a local file by an application installed on a device. The main difference is that cloud artifacts are internal data structures that, unlike a file, are not readily available for analysis.

Cloud artifacts often have a completely different structure from traditional snapshot-centric encoding. For example, internally, Google Docs' documents are represented as the complete

history (log) of every editing action performed on it; given valid credentials, this history is available via Google Docs' internal API. It is also possible to obtain a snapshot of the artifact of interest in a standard format such as a PDF, via the public API. However, this is inherently forensically deficient in that it ignores potentially critical information on the evolution of a document over time.

9.7 CONCLUSION

Digital forensics identifies and reconstructs the relevant sequence of events that has led to a currently observable state of a target IT system or (digital) artifacts. The provenance and integrity of the data source and the scientific grounding of the investigative tools and methods employed are of *primary* importance in determining their admissibility to a court of law's proceedings. Digital forensic analysis is applied to both individual digital artifacts such as files and to complex IT systems comprising multiple components and networked processes.

Following the rapid cloud-based transition from Software as a Product (SaaP) to Software as a Service (SaaS), forensic methods and tools are also in a respective process of transition. One aspect is a change of emphasis from state-centric analysis, which seeks to deduce events and actions by looking at different snapshots and applying knowledge about the system's operations, to log-centric analysis, which employs explicitly collected log entries to infer the sequence of relevant (to the inquiry) events. Another aspect is the transition from the low-level physical acquisition of storage device images to the high-level logical acquisition of (primarily) application artifacts via well-defined cloud service APIs. Some of the most important emerging questions in digital forensics are the analysis of the large variety of IoT devices, which are forecast to increase in number to as many as 125 billion by 2030, and the employment of machine learning/AI in order to automate and scale up forensic processing.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Topics	Cites
9.1 Definitions and Conceptual Models	[919, 920, 921, 922, 923]
9.2 Operating System Analysis	[923, 938, 939]
9.3 Main Memory Forensics	[949]
9.4 Application Forensics	
9.5 Cloud Forensics	
9.6 Artifact Analysis	[954, 955, 956, 957, 958]

III Systems Security

Chapter 10
Cryptography

Nigel Smart | KU Leuven

INTRODUCTION

The purpose of this chapter is to explain the various aspects of cryptography which we feel should be known to an expert in cyber-security. The presentation is at a level needed for an instructor in a module in cryptography; so they can select the depth needed in each topic. Whilst not all experts in cyber-security need be aware of all the technical aspects mentioned below, we feel they should be aware of all the overall topics and have an intuitive grasp as to what they mean, and what services they can provide. Our focus is mainly on primitives, schemes and protocols which are widely used, or which are suitably well studied that they could be used (or are currently being used) in specific application domains.

Cryptography by its very nature is one of the more mathematical aspects of cyber-security; thus this chapter contains a lot more mathematics than one has in some of the other chapters. The overall presentation assumes a basic knowledge of either first-year undergraduate mathematics, or that found in a discrete mathematics course of an undergraduate Computer Science degree.

The chapter is structured as follows: After a quick recap on some basic mathematical notation (Section 10.1), we then give an introduction to how security is defined in modern cryptography. This section (Section 10.2) forms the basis of our discussions in the other sections. Section 10.3 discusses information theoretic constructions, in particular the one-time pad, and secret sharing. Sections 10.4 and 10.5 then detail modern symmetric cryptography; by discussing primitives (such as block cipher constructions) and then specific schemes (such as modes-of-operation). Then in Sections 10.6 and 10.7 we discuss the standard methodologies for performing public key encryption and public key signatures, respectively. Then in Section 10.8 we discuss how these basic schemes are used in various standard protocols; such as for authentication and key agreement. All of the sections, up to and including Section 10.8, focus exclusively on constructions which have widespread deployment.

Section 10.9 begins our treatment of constructions and protocols which are less widely used; but which do have a number of niche applications. These sections are included to enable the instructor to prepare students for the wider applications of the cryptography that they may encounter as niche applications become more mainstream. In particular, Section 10.9 covers Oblivious Transfer, Zero-Knowledge, and Multi-Party Computation. Section 10.10 covers public key schemes with special properties, such as group signatures, identity-based encryption and homomorphic encryption.

The chapter assumes the reader wants to **use** cryptographic constructs in order to build secure systems, it is not meant to introduce the reader to attack techniques on cryptographic primitives. Indeed, all primitives here can be assumed to have been selected to avoid specific attack vectors, or key lengths chosen to avoid them. Further details on this can be found in the regular European Key Size and Algorithms report, of which the most up to date version is [960].

For a similar reason we do not include a discussion of historical aspects of cryptography, or historical ciphers such as Caesar, Vigenère or Enigma. These are at best toy examples, and so have no place in a such a body of knowledge. They are best left to puzzle books. However the interested reader is referred to [961].

CONTENT

10.1 MATHEMATICS

[962, c8–c9,App B][963, c1–c5]

Cryptography is inherently mathematical in nature, the reader is therefore going to be assumed to be familiar with a number of concepts. A good textbook to cover the basics needed, and more, is that of Galbraith [964].

Before proceeding we will set up some notation: The ring of integers is denoted by $\mathbb{Z}$, whilst the fields of rational, real and complex numbers are denoted by $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$. The ring of integers modulo N will be denoted by $\mathbb{Z}/N\mathbb{Z}$, when N is a prime p this is a finite field often denoted by $\mathbb{F}_p$. The set of invertible elements will be written $(\mathbb{Z}/N\mathbb{Z})^*$ or $\mathbb{F}_p^*$. An RSA modulus N will denote an integer N, which is the product of two (large) prime factors $N = p \cdot q$.

Finite abelian groups of prime order q are also a basic construct. These are either written multiplicatively, in which case an element is written as g^x for some $x \in \mathbb{Z}/q\mathbb{Z}$; when written additively an element can be written as $[x] \cdot P$. The element g (in the multiplicative case) and P (in the additive case) is called the generator.

The standard example of finite abelian groups of prime order used in cryptography are elliptic curves. An elliptic curve over a finite field $\mathbb{F}_p$ is the set of solutions (X, Y) to an equation of the form

$$E : Y^2 = X^3 + A \cdot X + B$$

where A and B are fixed constants. Such a set of solutions, plus a special point at infinity denoted by $\mathcal{O}$, form a finite abelian group denoted by $E(\mathbb{F}_p)$. The group law is a classic law dating back to Newton and Fermat called the chord-tangent process. When A and B are selected carefully one can ensure that the size of $E(\mathbb{F}_p)$ is a prime q. This will be important later in Section 10.2.3 to ensure the discrete logarithm problem in the elliptic curve is hard.

Some cryptographic schemes make use of lattices which are discrete subgroups of the subgroups of $\mathbb{R}^n$. A lattice can be defined by a generating matrix $B \in \mathbb{R}^{n \cdot m}$, where each column of B forms a basis element. The lattice is then the set of elements of the form $\mathbf{y} = B \cdot \mathbf{x}$ where $\mathbf{x}$ ranges over all elements in $\mathbb{Z}^m$. Since a lattice is discrete it has a well-defined length of the shortest non-zero vector. In Section 10.2.3 we note that finding this shortest non-zero vector is a hard computational problem.

Sampling a uniformly random element from a set A will be denoted by $x \leftarrow A$. If the set A consists of a single element a we will write this as the assignment $x \leftarrow a$; with the equality symbol $=$ being reserved for equalities as opposed to assignments. If A is a randomized algorithm, then we write $x \leftarrow A(y; r)$ for the assignment to x of the output of running A on input y with random coins r.

10.2 CRYPTOGRAPHIC SECURITY MODELS

[962, c1–c4][963, c11]

Modern cryptography has adopted a methodology of 'Provable Security' to define and understand the security of cryptographic constructions. The basic design procedure is to define the **syntax** for a cryptographic scheme. This gives the input and output behaviours of the algorithms making up the scheme and defines correctness. Then a **security model** is presented which defines what security goals are expected of the given scheme. Then, given a **specific instantiation** which meets the given syntax, a formal **security proof** for the instantiation is given relative to some known **hard problems**.

The security proof is not an absolute guarantee of security. It is a proof that the given instantiation, when implemented correctly, satisfies the given security model assuming some hard problems are indeed hard. Thus, if an attacker can perform operations which are outside the model, or manages to break the underlying hard problem, then the proof is worthless. However, a security proof, with respect to well studied models and hard problems, can give strong guarantees that the given construction has no fundamental weaknesses.

In the next subsections we shall go into these ideas in more detail, and then give some examples of security statements; further details of the syntax and security definitions can be found in [965, 966]. At a high level the reason for these definitions is that the intuitive notion of a cryptographic construction being secure is not sufficient enough. For example the natural definition for encryption security is that an attacker should be unable to recover the decryption key, or the attacker should be unable to recover a message encrypted under one ciphertext. Whilst these ideas are necessary for any secure scheme they are not sufficient. We need to protect against an attacker aims for find *some* information about an encrypted message, when the attacker is able to mount chosen plaintext and chosen ciphertext attacks on a legitimate user.

10.2.1 Syntax of Basic Schemes

The syntax of a cryptographic scheme is defined by the algorithms which make up the scheme, as well as a correctness definition. The correctness definition gives what behaviour one can expect when there is no adversarial behaviour. For example, a symmetric encryption scheme is defined by three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The KeyGen algorithm is a probabilistic algorithm which outputs a symmetric key $\mathfrak{k} \leftarrow \mathsf{KeyGen}()$; Enc is a probabilistic algorithm which takes a message $m \in \mathcal{M}$, some randomness $r \in \mathcal{R}$ and a key and returns a $c \leftarrow \mathsf{Enc}(m, \mathfrak{k}; r) \in \mathcal{C}$; whilst Dec is (usually) a deterministic algorithm which takes a ciphertext and a key and returns the underlying plaintext. The correctness definition is:

$$\forall \mathfrak{k} \leftarrow \mathsf{KeyGen}(), r \leftarrow \mathcal{R}, m \leftarrow \mathcal{M}, \;\; \mathsf{Dec}(\mathsf{Enc}(m, \mathfrak{k}; r), \mathfrak{k}) = m.$$

For public key encryption schemes the definitions are similar, but now $\mathsf{KeyGen}()$ outputs key pairs and the correctness definition becomes:

$$\forall (\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(), r \leftarrow \mathcal{R}, m \leftarrow \mathcal{M}, \;\; \mathsf{Dec}(\mathsf{Enc}(m, \mathfrak{pk}; r), \mathfrak{sk}) = m.$$

The equivalent constructions for authentication mechanisms are Message Authentication Codes (or MACs) in the symmetric key setting, and digital signatures schemes in the public key setting. A MAC scheme is given by a triple of algorithms $(\mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify})$, where the MAC

function outputs a tag given a message and a key (and possibly some random coins), and the Verify function checks the message, tag and key are consistent. A signature scheme is given by a similar triple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where now the tag produced is called a 'signature'. Thus the correctness definitions for these constructions are as follows

$$\mathfrak{k} \leftarrow \mathsf{KeyGen}(), r \leftarrow \mathcal{R}, m \leftarrow \mathcal{M}, \ \ \mathsf{Verify}(m, \mathsf{MAC}(m, \mathfrak{k}; r), \mathfrak{k}) = \mathsf{true}.$$

and

$$(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(), r \leftarrow \mathcal{R}, m \leftarrow \mathcal{M}, \ \ \mathsf{Verify}(m, \mathsf{Sign}(m, \mathfrak{sk}; r), \mathfrak{pk}) = \mathsf{true}.$$

Note, that for deterministic MACs the verification algorithm is usually just to recompute the MAC tag $\mathsf{MAC}(m, \mathfrak{k})$, and then check it was what was received.

10.2.2 Basic Security Definitions

A security definition is usually given in the context of an attacker's **security goal**, followed by their **capabilities**. So, for example, a naive security goal for encryption could be to recover the underlying plaintext, so-called One-Way (or OW) security. This process of an attacker trying to obtain a specific goal is called a *security game*, with the attacker winning the *game*, if they can break this security goal with greater probability than random guessing. This advantage in probability over random guessing is called the adversary's *advantage*. The capabilities are expressed in terms of what oracles, or functions, we give the adversary access to. So, for example, in a naive security game for encryption we may give the adversary no oracles at all, producing a so-called Passive Attack (or PASS) capability.

The attacker is modelled as an arbitrary algorithm, or Turing machine, A, and if we give the adversary access to oracles then we write these as subscripts $A^{\mathcal{O}}$. In our naive security game (called OW-PASS) the adversary has no oracles and its goal is simply to recover the message underlying a given ciphertext. The precise definition is given in Figure 10.1, where $\mathsf{Adv}^{\mathsf{OW-PASS}}(A, t)$ denote the advantage over a random guess that a given adversary has after running for time t. We say that a given construction is secure in the given model (which our naive example would be named OW-PASS), if the above advantage is *negligible* for all probabilistic polynomial time adversaries A. Here, negligible and polynomial time are measured in terms of a *security parameter* (which one can think of as the key size). Note, for OW-PASS this assumes that the message space is not bigger than the space of all possible keys. Also note, that this is an asymptotic definition, which in the context of schemes with fixed key size, makes no sense. In such situations we require that (t/Adv) is greater than some given concrete bound such as 2^{128}, since it is believed that performing an algorithm requiring 2^{128} steps is infeasible even for a nation-state adversary.

In the context of encryption (both symmetric and public key) the above naive security goal is not seen as being suitable for real applications. Instead, the security goal of Indistinguishable encryptions (or IND) is usually used. This asks the adversary to first come up with two plaintexts, of equal length, and then the challenger (or environment) encrypts one of them and gives the resulting challenge to the adversary. The adversary's goal is then to determine which plaintext was encrypted. In the context of a passive attack this gives an advantage statement as given in the second part of Figure 10.1, where the two stages of the adversary are given by A_1 and A_2.

In terms of encryption, the above passive attack is almost always not sufficient in terms of capturing real-world adversarial capabilities, since real systems almost always give the attacker additional attack vectors. Thus two other (increasingly strong) attack capabilities

are usually considered. These are a Chosen Plaintext Attack (or CPA capability), in which the adversary is given access to an encryption oracle to encrypt arbitrary messages of his choice, and a Chosen Ciphertext Attack (or CCA capability), in which the adversary has both an encryption and decryption oracle. In the case of a public key scheme the adversary always has access to an encryption oracle because it can encrypt plaintexts for itself using the public key, so in this case PASS and CPA are equivalent. In the case of a CCA capability we restrict the decryption oracle so that the adversary may not ask of it the challenge ciphertext c^*; otherwise it can trivially win the security game. Thus the advantage of an IND-CCA adversary against a public key encryption scheme would be defined as the third definition in Figure 10.1. Other security definitions are possible for encryption (such as Real-or-Random) but the above are the main ones.

OW-PASS Definition:

$$\mathsf{Adv}^{\mathsf{OW-PASS}}(A,t) = \Pr\Big[\quad \mathfrak{k} \leftarrow \mathsf{KeyGen}(),\ \ m^* \leftarrow \mathcal{M},\ \ r \leftarrow \mathcal{R},\\ c^* \leftarrow \mathsf{Enc}(m, \mathfrak{k}; r),\ \ m \leftarrow A(c^*) :\ \ m = m^* \Big] - \frac{1}{|\mathcal{M}|}.$$

One reads the probability statement as the being the probability that $m = m^*$, given that m and m^* are produced by first sampling $\mathfrak{k}$ from algorithm $\mathsf{KeyGen}()$, then sampling m^* and r from the spaces $\mathcal{M}$ and $\mathcal{R}$ at random, then determining c^* by calling $\mathsf{Enc}(m, \mathfrak{k}; r)$ and finally passing c^* to the Adversary A, and getting m in return.

IND-PASS Symmetric Key Encryption Definition:

$$\mathsf{Adv}^{\mathsf{IND-PASS}}(A,t) = \Pr\Big[\quad \mathfrak{k} \leftarrow \mathsf{KeyGen}(),\ \ b \leftarrow \{0,1\},\ \ m_0,\ m_1,\ \mathsf{state} \leftarrow A_1(),\\ r \leftarrow \mathcal{R},\ \ c^* \leftarrow \mathsf{Enc}(m_b, \mathfrak{k}; r),\ \ b' \leftarrow A_2(c^*, \mathsf{state}) :\ \ b = b' \Big] - \frac{1}{2}.$$

IND-CCA Public Key Encryption Definition:

$$\mathsf{Adv}^{\mathsf{IND-CCA}}(A,t) = \Pr\Big[\quad (\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(),\ \ b \leftarrow \{0,1\},\ \ m_0,\ m_1, \mathsf{state} \leftarrow A_1^{\mathsf{Dec}(\cdot,\mathfrak{sk})}(\mathfrak{pk}),\\ r \leftarrow \mathcal{R},\ \ c^* \leftarrow \mathsf{Enc}(m_b, \mathfrak{pk}; r),\ \ b' \leftarrow A_2^{\mathsf{Dec}(\cdot,\mathfrak{sk})}(c^*, \mathsf{state}) :\ \ b = b' \Big] - \frac{1}{2}.$$

UF-CMA Signature Security Definition:

$$\mathsf{Adv}^{\mathsf{UF-CMA}}(A,t) = \Pr\Big[(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(),\ \ (m, \sigma) \leftarrow A^{\mathsf{Sign}(\cdot,\mathfrak{sk})}(\mathfrak{pk}) :\ \ \mathsf{Verify}(m, \sigma, \mathfrak{pk}) = \mathsf{true}\Big].$$

IND-CCA KEM Security Definition:

$$\mathsf{Adv}^{\mathsf{IND-CCA}}(A,t) = \Pr\Big[\quad (\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KEMKeyGen}(),\ \ b \leftarrow \{0,1\},\ \ \mathfrak{k}_0 \leftarrow \mathcal{K},\ \ r \leftarrow \mathcal{R},\\ \mathfrak{k}_1, c^* \leftarrow \mathsf{KEMEnc}(\mathfrak{pk}; r),\ \ b' \leftarrow A_2^{\mathsf{KEMDec}(\cdot,\mathfrak{sk})}(c^*, \mathfrak{k}_b) :\ \ b = b' \Big] - \tfrac{1}{2}.$$

Figure 10.1: Technical Security Definitions

For MACs (resp. digital signature schemes) the standard security goal is to come up with a

message/tag (resp. message/signature) pair which passes the verification algorithm, a so-called Universal Forgery (or UF) attack. We make no assumption about whether the message has any meaning, indeed, the attacker wins if he is able to create a signature on *any* bit-string. If the adversary is given no oracles then he is said to be mounting a passive attack, whilst if the adversary is given a tag generation (resp. signing oracle) he is said to be executing a Chosen Message Attack (CMA). In the latter case the final forgery must not be one of the outputs of the given oracle. In the case of MAC security, one may also give the adversary access to a tag verification oracle. However, for deterministic MACs this is implied by the CMA capability and is hence usually dropped, since verification only involves re-computing the MAC.

Again we define an advantage and require this to be negligible in the security parameter. For digital signatures the advantage for the UF-CMA game is given by the fourth equation in Figure 10.1.

10.2.3 Hard Problems

As explained above, security proofs are always relative to some hard problems. These hard problems are often called *cryptographic primitives*, since they are the smallest atomic object from which cryptographic schemes and protocols can be built. Such cryptographic primitives come in two flavours: Either they are keyed complexity theoretic definitions of functions, or they are mathematical hard problems.

In the former case one could consider a function $F_k(\cdot) : D \longrightarrow C$ selected from a function family $\{F_k\}$ and indexed by some index k (thought of as a key of varying length). One can then ask whether the function selected is indistinguishable (by a probabilistic polynomial time algorithm A which has oracle access to the function) from a uniform random function from D to C. If such an assumption holds, then we say the function family defines a (keyed) Pseudo-Random Function (PRF). In the case when the domain D is equal to the co-domain C we can ask whether the function is indistinguishable from a randomly chosen permutation, in which case we say the family defines a (keyed) Pseudo-Random Permutation (PRP).

In the case of a block cipher, such as AES (see later), where one has $C = D = \{0,1\}^{128}$, it is a basic assumption that the AES function family (indexed by the key k) is a Pseudo-Random Permutation.

In the case of mathematical hard problems we have a similar formulation, but the definitions are often more intuitive. For example, one can ask the question whether a given RSA modulus $N = p \cdot q$ can be factored into its prime components p and q, the so-called factoring problem. The RSA group $\mathbb{Z}/N\mathbb{Z}$ defines a finite abelian group of unknown order (the order is known to the person who created N), finding the order of this group is equivalent to factoring N. The RSA function $x \longrightarrow x^e \pmod N$ is believed to be hard to invert, leading to the so-called RSA-inversion problem of, given $y \in (\mathbb{Z}/N\mathbb{Z})^*$, finding x such that $x^e = y \pmod N$. It is known that the function can easily be inverted if the modulus N can be factored, but it is unknown if inverting the function implies N can be factored. Thus we have a situation where one problem (factoring) seems to be harder to solve than another problem (the RSA problem). However, in practice, we assume that both problems are hard, given appropriately chosen parameters. Details on the best method to factor large numbers, the so-called Number Field Sieve, can be found in [967].

In finite abelian groups of known order (usually assumed to be prime), one can define other

problems. The problem of inverting the function $x \longrightarrow g^x$, is known as the Discrete Logarithm Problem (DLP). The problem of, given g^x and g^y, determining $g^{x \cdot y}$ is known as the Diffie–Hellman Problem (DHP). The problem of distinguishing between triples of the form (g^x, g^y, g^z) and $(g^x, g^y, g^{x \cdot y})$ for random x, y, z is known as the Decision Diffie–Hellman (DDH) problem. When written additively in an elliptic curve group, a DDH triple has the form $([x] \cdot P, [y] \cdot P, [z] \cdot P)$.

Generally speaking, the mathematical hard problems are used to establish the security of public key primitives. A major issue is that the above problems (Factoring, RSA-problem, DLP, DHP, DDH), on which we base all of our main existing public key algorithms, are easily solved by large-scale quantum computers. This has led designers to try to build cryptographic schemes on top of mathematical primitives which do not appear to be able to be broken by a quantum computer. Examples of such problems are the problem of determining the shortest vector in a high dimensional lattice, the so-called Shortest Vector Problem (SVP), and the problem of determining the closest lattice vector to a non-lattice vector, the so-called Closest Vector Problem (CVP). The best algorithms to solve these hard problems are lattice reduction algorithms, a nice survey of these algorithms and applications can be found in [968]. The SVP and CVP problems, and others, give rise to a whole new area called Post-Quantum Cryptography (PQC).

Example: Putting the above ideas together, one may encounter statements such as: *The public key encryption XYZ is IND-CCA secure assuming the RSA-problem is hard and AES is a PRP.* This statement tells us that *any* attack against the XYZ scheme must either be against some weakness in the implementation, or must come from some attack not captured in the IND-CCA model, or must come from solving the RSA-problem, or must come from showing that AES is not a PRP.

10.2.4 Setup Assumptions

Some cryptographic protocols require some setup assumptions. These are assumptions about the environment, or some data, which need to be satisfied before the protocol can be considered secure. These assumptions come in a variety of flavours. For example, one common setup assumption is that there exists a so-called Public-Key Infrastructure (PKI), meaning that we have a trusted binding between entities' public keys and their identities.

Another setup assumption is the existence of a string (called the Common Reference String or CRS) available to all parties, and which has been set up in a trusted manner, i.e. such that no party has control of this string.

Other setup assumptions could be physical, for example, that the algorithms have access to good sources of random numbers, or that their internal workings are not susceptible to an invasive attacker, i.e. they are immune to side-channel attacks.

10.2.5 Simulation and UC Security

The above definitions of security make extensive use of the notion of indistinguishability between two executions. Indeed, many of the proof techniques used in the security proofs construct *simulations* of cryptographic operations. A simulation is an execution which is indistinguishable from the real execution, but does not involve (typically) the use of any key material. Another method to produce security models is the so-called *simulation paradigm*, where we ask that an adversary cannot tell the simulation from a real execution (unless they can solve some hard problem). This paradigm is often used to establish security results for more complex cryptographic protocols.

A problem with both the game/advantage-based definitions defined earlier and the simulation definitions is that they only apply to stand-alone executions, i.e. executions of one instance of the protocol in one environment. To cope with arbitrarily complex executions and composition of cryptographic protocols an extension to the simulation paradigm exists called the Universal Composability (UC) framework.

10.3 INFORMATION-THEORETICALLY SECURE CONSTRUCTIONS

[962, c2][963, c19]

Whilst much of cryptography is focused on securing against adversaries that are modelled as probabilistic polynomial time Turing machines, some constructions are known to provide security against unbounded adversaries. These are called information-theoretically secure constructions. A nice introduction to the information theoretic side of cryptography can be found in [969].

10.3.1 One-Time Pad

The most famous primitive which provides information-theoretic security is the one-time pad. Here, a binary message $m \in \{0,1\}^t$ is encrypted by taking a key $k \in \{0,1\}^t$ uniformly at random, and then producing the ciphertext $c = m \oplus k$. In terms of our earlier security models, this is an IND-PASS scheme even in the presence of a computationally unbounded adversary. However, the fact that it does not provide IND-CPA security is obvious, as the encryption scheme is determinisitic. The scheme is unsuitable in almost all modern environments as one requires a key as long as the message and the key may only be used once; hence the name one-time pad.

10.3.2 Secret Sharing

Secret sharing schemes allow a secret to be shared among a set of parties so that only a given subset can reconstruct the secret by bringing their shares together. The person who constructs the sharing of the secret is called the dealer. The set of parties who can reconstruct the secret are called qualified sets, with the set of all qualified sets being called an *access structure*.

Any set which is not qualified is said to be an unqualified set, and the set of all unqualified sets is called an *adversary structure*. The access structure is usually assumed to be monotone, in that if the parties in A can reconstruct the secret, then so can any super-set of A.

Many secret sharing schemes provided information-theoretic security, in that any set of parties which is unqualified can obtain no information about the shared secret even if they have unbounded computing power.

A special form of access structure is a so-called *threshold* structure. Here we allow any subset of $t+1$ parties to reconstruct the secret, whereas any subset of t parties is unable to learn anything. The value t is being called the threshold. One example construction of a threshold secret sharing scheme for a secret s in a field $\mathbb{F}_p$, with $n > p$ is via Shamir's secret sharing scheme.

In Shamir secret sharing, one selects a polynomial $f(X) \in \mathbb{F}_p[X]$ of degree t with constant coefficients s, the value one wishes to share. The share values are then given by $s_i = f(i) \pmod p$, for $i = 1, \ldots, n$, with party i being given s_i. Reconstruction of the value s from a subset of more than t values s_i can be done using Lagrange interpolation.

Due to an equivalence with Reed-Solomon error correcting codes, if $t < n/2$, then on receipt of n share values s_i, a reconstructing party can detect if any party has given it an invalid share. Additionally, if $t < n/3$ then the reconstructing party can correct for any invalid shares.

Replicated secret sharing is a second popular scheme which supports any monotone access structure. Given a boolean formula defining who should have access to the secret, one can define a secret sharing scheme from this formula by replacing all occurrences of AND with $+$ and all occurrences of OR with a new secret. For example, given the formulae

$$(P_1 \text{ AND } P_2) \text{ OR } (P_2 \text{ AND } P_3),$$

one can share a secret s by writing it as $s = s_1 + s_2 = s_2' + s_3$ and then, giving party P_1 the value s_1, party P_2 the pair of values s_2 and s_2', and party P_3 the value s_3. Replicated secret sharing is the scheme obtained in this way when putting the boolean formulae into Conjunctive Normal Form.

Of importance in applications of secret sharing, especially to Secure Multi-Party Computation (see Section 10.9.4) is whether the adversary structure is Q_2 or Q_3. An adversary structure is said to be Q_i if no set of i unqualified sets have union the full set of players. Shamir's secret sharing scheme is Q_2 if $t < n/2$ and Q_3 when $t < n/3$. The error detection (resp. correction) properties of Shamir's secret sharing scheme mentioned above follow through to any Q_2 (resp. Q_3) adversary structure.

10.4 SYMMETRIC PRIMITIVES

[962, c3–c6][963, c11–c14]

Symmetric primitives are a key component of many cryptographic constructions. There are three such basic primitives: block ciphers, stream ciphers, and hash functions. Theoretically, all are keyed functions, i.e. they take as input a secret key, whilst in practice one often considers hash functions which are unkeyed. At a basic level, all are functions $f : \mathcal{K} \times D \longrightarrow C$ where $\mathcal{K}$ is the key space, D is the domain (which is of a fixed finite size for block ciphers and stream ciphers).

As explained in the introduction we will not be discussing in this report cryptanalysis of symmetric primitives, we will only be examining secure constructions. However, the main two techniques for attacks in this space are so-called differential and linear cryptanalysis. The interested reader is referred to the excellent tutorial by Howard Heys [970] on these topics, or the book [971].

10.4.1 Block Ciphers

A block cipher is a function $f : \mathcal{K} \times \{0,1\}^b \longrightarrow \{0,1\}^b$, where b is the block size. Despite their names such functions should not be thought of as an encryption algorithm. It is, however, a building block in many encryption algorithms. The design of block ciphers is a deep area of subject in cryptography, analogous to the design of number theoretic one-way functions. Much like number-theoretic one-way functions, cryptographic constructions are proved secure relative to an associated hard problem which a given block cipher is assumed to satisfy.

For a fixed key, a block cipher is assumed to act as a permutation on the set $\{0,1\}^b$, i.e. for a fixed key k, the map $f_k : \{0,1\}^b \longrightarrow \{0,1\}^b$ is a bijection. It is also assumed that inverting this permutation is also easy (if the key is known). A block cipher is considered *secure* if no polynomial time adversary, given oracle access to a permutation on $\{0,1\}^b$, can tell the difference between being given a uniformly random permutation or the function f_k for some fixed hidden key k, i.e. the block cipher is a PRP. In some applications, we only require that the block cipher is a function, i.e. not a bijection. In which case we require the block cipher is a PRF.

One can never prove that a block cipher is a PRP, so the design criteria is usually a task of building a mathematical construction which resists all known attacks. The main such attacks which one resists are so-called *linear cryptanalysis*, where one approximates non-linear components within the block cipher by linear functions, and *differential cryptanalysis*, where one looks at how two outputs vary on related input messages, e.g. one applies f_k to various inputs m_0 and m_1 where $m_0 \oplus m_1 = \Delta$ a fixed value.

The design of a block cipher is made up of a number of simpler components. There are usually layers of simple fixed permutations, and layers of table lookups. These table lookups are called S-boxes, where the S stands for substitutions. There are two main techniques to design block ciphers. Both repeat a simple operation (called a round) a number of times. Each round consists of a combination of permutations and substitutions, and a key addition. The main key is first expanded into round-keys, with each round having a different round-key.

In the first methodology, called a *Feistel Network*, the S-Boxes allowed in each round can be non-injective, i.e. non-invertible. Despite this, the Feistel constructions still maintain the

overall invertibility of the block cipher construction. The second method is a *Substitution-Permutation Network* design in which each round consists of a round-key addition, followed by a fixed permutation, followed by the application of bijective S-boxes. In general, the Feistel construction requires more rounds than the Substitution-Permutation network construction.

The DES (Data Encryption Standard) block cipher (with an original key of 56-bits and block size of $b = 64$) is a Feistel construction. The DES algorithm dates from the 1970s, and the key size is now considered far too small for any application. However, one can extend DES into a 112- or 168-bit key block cipher to construct an algorithm called 2DES or 3DES. The use of 2DES or 3DES is still considered secure, although in some applications, the block size of 64-bits is considered insecure for real-world use.

The AES (Advanced Encryption Standard) block cipher is the modern replacement for DES, and it is a block cipher with a 128-, 192- or 256-bit key, and with a block size of $b = 128$ bits. The AES algorithm has hardware support on many microprocessors, making operations using AES much faster than using other cryptographic primitives. Readers who wish to understand more about the design of the AES block cipher referred to [972].

10.4.2 Stream Ciphers

A stream cipher is one which produces an arbitrary length string of output bits, i.e. the co-domain of the function is essentially unbounded. Stream ciphers can be constructed from block ciphers, by using a block cipher in Counter Mode (see Section 10.5.1). However, the stream cipher is usually reserved for constructions which are special-purpose and for which the hardware complexity is much reduced.

Clearly, a stream cipher cannot be a permutation, but we require that no polynomial time adversary can distinguish oracle access to the stream cipher from oracle access to a uniformly random function with infinite co-domain. The design of stream ciphers is more ad-hoc than that of the design of block ciphers. In addition, there is less widespread adoption outside specific application areas. The interested reader is referred to the outcome of the eStream competition for details of specific ad-hoc stream cipher designs [973].

10.4.3 Hash Functions

Hash functions are much like block ciphers in that they should act as PRFs. However, the input domain can be unbounded. Since a PRF needs to be keyed to make any sense in theoretical tracts, a hash function is usually a keyed object. In practice, we often require an unkeyed object, in which case one considers the actual hash function used to have an implicit inbuilt fixed key, and have been chosen from a function family already.

When considering a fixed hash function, one is usually interested in the intractability of inverting the hash function (the one-way property), the intractability of finding two inputs with the same output (the collision resistance property), or the intractability of finding, given an input/output pair, a new input which gives the same output (the second-preimage resistance property).

10.4.3.1 Merkle-Damgård Construction

Early hash functions were based on the Merkle-Damgård construction. The family of such functions (MD4, MD5, SHA-1, SHA-2) have a number of issues, with only SHA-2 now being considered secure. The Merkle-Damgård construction takes a compression function $f(x, y)$ taking two inputs x, y of fixed length with the output length of x. This is used to derive a function which allows arbitrary length inputs by first dividing a message m into t blocks $m_1, \ldots, m_t$ each of length $|y|$, and then applying

$$h_i = f(h_{i-1}, m_i) \text{ for } i = 1, \ldots, t,$$

where the output is h_t and h_0 is some initial value, which can be thought of as a fixed key for the hash function.

The above methodology requires a method to pad the initial input block to encode the length, and it suffers from a number of practical issues. For example, there are obvious length extension attacks (namely a hash on a message m can be extended to a hash on $m\|m'$ without knowing the whole of m) which render the use of such hash functions problematic in some applications. For example, in HMAC (see Section 10.5.2), one requires two applications of the hash function to prevent such attacks.

10.4.3.2 Sponge Constructions

A more modern approach to hash function design is to create a so-called sponge construction. This is the design philosophy behind SHA-3 (a.k.a.). A sponge is a function which operates in two phases. In the first phase, one enters data into the sponge state and, in the second phase, one squeezes data out from the sponge.

The sponge state is a pair $(r, c) \in \{0, 1\}^{|r|+|c|}$, where the length of r denotes the input/output rate and c is a variable holding an internal state hidden from any attacker. The size of c is directly related to the security of the construction. At each round, a public permutation p is applied to the state (r, c).

In the input phase of the sponge, the input data m, after suitable padding, is divided into t blocks $m_1, \ldots, m_t$ of size $|r|$. Then the state is updated via the mapping

$$(r_i, c_i) = p(r_{i-1} \oplus m_i, c_{i-1}) \text{ for } i = 1, \ldots, t,$$

where r_0 and c_0 are initialized to be fixed all-zero bit strings. After data is entered into the sponge, one can obtain s blocks of $|r|$-bit outputs $o_1, \ldots, o_s$ by computing

$$(o_i, c'_i) = p(o_{i-1}, c'_{i-1}) \text{ for } i = 1, \ldots, s,$$

where $o_0 = r_t$ and $c'_0 = c_t$. Thus the whole function is given by

$$H(m_1, \ldots, m_t) = o_1, \ldots, o_s.$$

Further details on sponge constructions, and the further objects one can construct from them, and the SHA-3 design in particular can be found at the Keccak web page [974].

10.4.3.3 Random Oracle Model

Many cryptographic constructions are only secure if one assumes that the hash function used in the construction behaves 'like a random oracle'. Such constructions are believed to be secure in the real world, but theoretically, they are less pleasing. One can think of a proof of security in the random oracle model as a proof in which we allow the attacker to have their usual powers; however, when they (or any of the partners they are attacking) call the underlying hash function the call is made to an external party via an oracle call. This external party then simply plays back a random value, i.e. it does not use any algorithm to generate the random values. All that is required is that if the input is given to the oracle twice, then the same output is always returned.

This clearly does not capture attacks in which the adversary makes clever use of exactly how the hash function is defined etc, and how this definition interacts with other aspects of the scheme/protocol under analysis. However, this modelling methodology has proved remarkably good in enabling cryptographers to design schemes which are secure in the real world.

10.5 SYMMETRIC ENCRYPTION AND AUTHENTICATION

[962, c3–c4][963, c13–c14]

A block cipher, such as AES or DES, does not provide an effective form of data encryption or data/entity authentication on its own. To provide such symmetric cryptographic constructions, one needs a scheme, which takes the primitive and then utilizes this in a more complex construction to provide the required cryptographic service. In the context of symmetric encryption, these are provided by modes of operation. In the case of authentication, it is provided by a MAC construction. Additionally, block ciphers are often used to take some entropy and then expand, or collapse, this into a pseudo-random stream or key; a so-called XOF (or Extendable Output Function) or KDF (or Key Derivation Function). Further details on block cipher based constructions can be found at [975], whereas further details on Sponger/Keccak based constructions can be found at [974].

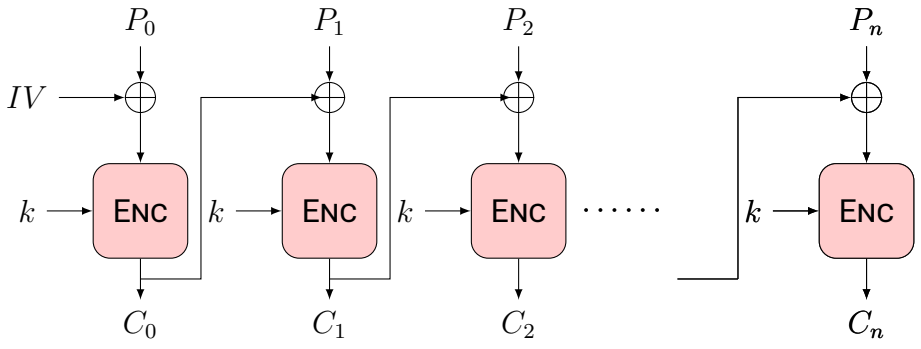


Figure 10.2: CBC Mode Encryption (All Figures are produced using *TikZ for Cryptographers* ***https://www.iacr.org/authors/tikz/***).

10.5.1 Modes of Operation

Historically, there have been four traditional modes of operation to turn a block cipher into an encryption algorithm. These were ECB, CBC, OFB and CFB modes. In recent years, the CTR mode has also been added to this list. Among these, only CBC mode (given in Figure 10.2) and CTR mode (given in Figure 10.3) are used widely within current systems. In these Figures, the block cipher is represented by the function Enc

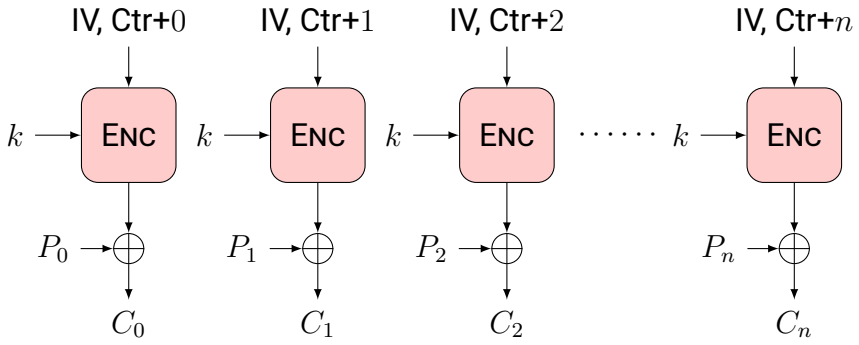


Figure 10.3: CTR Mode Encryption

On their own, however, CBC and CTR modes only provide IND-CPA security. This is far weaker than the 'gold standard' of security, namely IND-CCA (discussed earlier). Thus, modern systems use modes which provide this level of security, also enabling additional data (such as session identifiers) to be tagged into the encryption algorithm. Such algorithms are called AEAD methods (or Authenticated Encryption with Associated Data). In such algorithms, the encryption primitive takes as input a message to be encrypted, plus some associated data. To decrypt, the ciphertext is given, along with the associated data. Decryption will only work if both the key is correct and the associated data is what was input during the encryption process.

The simplest method to obtain an AEAD algorithm is to take an IND-CPA mode of operation such as CBC or CTR, and then to apply a MAC to the ciphertext and the data to be authenticated, giving us the so-called Encrypt-then-MAC paradigm. Thus, to encrypt m with authenticated data a, one applies the transform

$$c_1 \leftarrow \mathsf{Enc}(m, \mathfrak{k}_1; r), \qquad c_2 \leftarrow \mathsf{MAC}(c_1 \| a, \mathfrak{k}_2; r),$$

with the ciphertext being (c_1, c_2). In such a construction, it is important that the MAC is applied to the ciphertext as opposed to the message.

A major issue with the Encrypt-then-MAC construction is that one needs to pass the data to the underlying block cipher twice, with two different keys. Thus, new constructions of AEAD schemes have been given which are more efficient. The most widely deployed of these is GCM (or Galois Counter Mode), see Figure 10.4, which is widely deployed due to the support for this in modern processors.

One time AEAD constructions, otherwise known as DEMs, can be obtained by simply making the randomized AEAD deterministic by fixing the IV to zero.

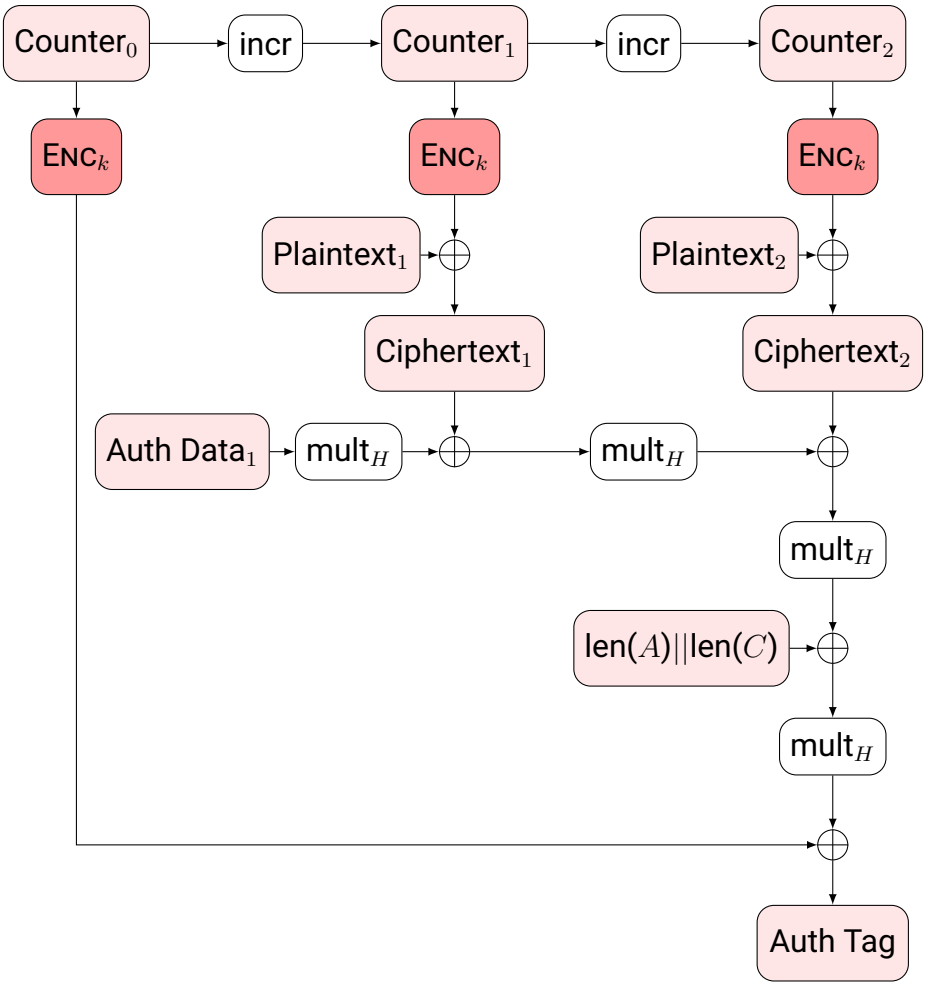


Figure 10.4: GCM Mode Encryption

10.5.2 Message Authentication Codes

Message authentication codes can be produced in roughly the same manner as modes of operation. In particular, the standard MAC function is to utilize CBC mode with a zero-IV, and then to output the final ciphertext block as the MAC tag, thus producing a deterministic MAC function. On its own, even with suitable padding of the message, this is only secure when used with fixed length messages. Thus, often a form of post-processing of the MAC output tag is performed. For example, the final CBC ciphertext block is then passed through another application of the underlying block cipher, but using a different key.

The GCM AEAD method of the previous section can be thought of as an Encrypt-then-MAC construction, with the IND-CPA encryption being CTR mode, and the MAC function being a function called GMAC. Although this is rarely used on its own as a MAC function.

Hash functions can also be used to construct MAC functions. The most famous of these is HMAC which is a construction designed for use with Merkle–Damgård-based hash functions. Since Merkle–Damgård-based hash functions suffer from length extension attacks, the HMAC function requires two applications of the underlying hash function. The construction produces a deterministic MAC function given by

$$\text{HMAC}(m, \mathfrak{k}) = H\left((\mathfrak{k} \oplus \text{opad}) \| H((\mathfrak{k} \oplus \text{ipad}) \| m)\right),$$

where opad is the string 0x5c5c...5c5c and ipad is the string 0x3636...3636.

As HMAC is designed specifically for use with Merkle–Damgård-based hash functions, it makes no-sense to use this construction when using a sponge based hash function such as SHA-3. The standardized MAC function derived from SHA-3 is called KMAC (or Keccak MAC). In this function, the sponge construction is used to input a suitably padded message, then the required MAC output is taken as the squeezed output of the sponge; whereas as many bits as squeezed are as needed for the MAC output.

10.5.3 Key Derivation and Extendable Output Functions

The security definition of a deterministic MAC is essentially equivalent to the definition that the output of the MAC function is indistinguishable from a random string, if one does not know the underlying secret key. As such, MAC functions can be used for other cryptographic operations. For example, in many situations, one must derive a long (or short) string of random bits, given some random input bits. Such functions are called KDFs or XOFs (for Key Derivation Function and Extendable Output Function). Usually, one uses the term KDF when the output is of a fixed length, and XOF when the output could be of an arbitrary length. But the constructions are, usually, essentially the same in both cases.

Such functions can take an arbitrary length input string, and produce another arbitrary length output string which is pseudo-random. There are three main constructions for such functions; one based on block ciphers, one on the Merkle–Damgård hash functions, and one based on sponge-based hash functions.

The constructions based on a block cipher are, at their heart, using CBC-MAC, with a zero key to compress the input string into a cryptographic key and then use the CTR mode of operation under this key to produce the output string. Hence, the construction is essentially given by

$$\mathfrak{k} \leftarrow \text{CBC-MAC}(m, \mathbf{0}), \qquad o_1 \leftarrow \text{Enc}(\mathbf{1}, \mathfrak{k}), \qquad o_2 \leftarrow \text{Enc}(\mathbf{2}, \mathfrak{k}), \quad \ldots$$

where Enc is the underlying block cipher.

The constructions based on the Merkle–Damgård hash function use a similar structure, but using one hash function application per output block, in a method similar to the following

$$o_1 \leftarrow H(m\|1), \qquad o_2 \leftarrow H(m\|2), \quad \ldots$$

Due to the way Merkle–Damgård hash functions are constructed, the above construction (for large enough m) can be done more efficiently than simply applying H as many times as the number of output blocks will dictate.

As one can imagine, the functions based on Keccak are simpler—one simply inputs the suitably padded message into the sponge and then squeezes as many output bits out as required.

Special KDFs can also be defined which take as input a low entropy input, such as a password or PIN, and produce a key for use in a symmetric algorithm. These *password based key derivation functions* are designed to be computationally expensive, so as to mitigate problems associated to brute force attacking of the underlying low entropy input.

10.5.4 Merkle-Trees and Blockchains

An application of cryptographic hash functions which has recently come to prominance is that of using Merkle-Trees and by extension blockchains. A Merkle-Tree, or hash-tree, is a tree in which each leaf node contains data, and each internal node is the hash of its child nodes. The root node is then publicly published. Merkle-Trees enable efficient demonstration that a leaf node is contained in the tree, in that one simply presents the path of hashes from the leaf up to the root node. Thus verification is logarithmic in the number of leaf nodes. Merkle-Trees can verify any form of stored data and have been used in various protocols such as version control systems, such as Git, and backup systems.

A blockchain is a similar structure, but now the data items are aligned in a chain, and each node hashes both the data item and a link to the previous item in the chain. Blockchains are used in cryptocurrencies such as *Bitcoin*, but they have wider application. The key property a blockchain provides is that (assuming the current head of the chain is authenticated and trusted) the data provides an open distributed ledger in which previous data items are immutable, and the ordering of data items is preserved.

10.6 PUBLIC KEY ENCRYPTION

[962, c11][963, c15–c17]

As explained above, public key encryption involves two keys, a public one $\mathfrak{pk}$ and a private one $\mathfrak{sk}$. The encryption algorithm uses the public key, whilst the decryption algorithm uses the secret key. Much of public key cryptography is based on number theoretic constructions, thus [964] provides a good coverage of much in this section. The standard security requirement for public key encryption is that the scheme should be IND-CCA. Note that since the encryption key is public we have that IND-PASS is the same as IND-CPA for a public key encryption scheme.

10.6.1 KEM-DEM Philosophy

In general, public key encryption schemes are orders of magnitude less efficient than symmetric key encryption schemes. Thus, the usual method in utilizing a public key scheme, when large messages need to be encrypted, is via a hybrid method. This hybrid methodology is called the KEM-DEM philosophy A KEM, which stands for Key Encapsulation Mechanism, a public key method to transmit a short key, selected at random from a set $\mathcal{K}$, to a designated recipient. Whereas, a DEM, or Data Encryption Mechanism, is essentially the same as an IND-CCA symmetric encryption scheme, which has key space $\mathcal{K}$. Since a DEM is only ever used once with the same key, we can actually use a weaker notion of IND-CCA encryption for the DEM, in which the adversary is not given access to an encryption oracle; which means the DEM can be *deterministic*.

For a KEM, we call the encryption and decryption mechanisms encapsulation and decapsulation, respectively. It is usual for the syntax of the encapsulation algorithm to not take any input, bar the randomness, and then to return both the ciphertext and the key which it encapsulates. Thus, the syntax, and correctness, of a KEM becomes

$$(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KEMKeyGen}(), r \leftarrow \mathcal{R}, (\mathfrak{k}, c) \leftarrow \mathsf{KEMEnc}(\mathfrak{pk}; r), \;\; \mathsf{KEMDec}(c, \mathfrak{sk}) = \mathfrak{k}.$$

The security definition for a KEM is described in the last equation of Figure 10.1. To construct the hybrid public key encryption scheme we define $\mathsf{KeyGen}()$ to be equal to $\mathsf{KEMKeyGen}$, then $\mathsf{Enc}(m, \mathfrak{pk}; r)$ outputs (c_0, c_1) where

$$\mathfrak{k}, c_0 \leftarrow \mathsf{KEMEnc}(\mathfrak{pk}; r), \quad c_1 \leftarrow \mathsf{DEM}(m, \mathfrak{k}),$$

with $\mathsf{Dec}((c_0, c_1), \mathfrak{sk})$ being given by

$$\mathfrak{k} \leftarrow \mathsf{KEMDec}(c_0, \mathfrak{sk}), \quad m \leftarrow \mathsf{DEM}^{-1}(c_1, \mathfrak{k}).$$

10.6.2 Constructions based on RSA

The simplest public key encryption scheme is the RSA scheme, which is based on the difficulty of factoring integers. In the key generation algorithm, two large primes p and q are selected and multiplied together to form $N = p \cdot q$. Then a (usually small) integer e is selected which is co-prime to $\phi(N) = (p-1) \cdot (q-1)$. Finally, the integer d is found, using the extended Euclidean algorithm, such that $d = 1/e \pmod{\phi(N)}$. The public key is set to be $\mathfrak{pk} = (N, e)$, whereas the secret key is set to be $\mathfrak{sk} = (N, d)$. Note that given the $\mathfrak{pk}$ only, finding the secret key $\mathfrak{sk}$ is provably equivalent to factoring N.

The public/private keys are used via the RSA function $x \longrightarrow x^e \pmod{N}$, which has the inverse map $x \longrightarrow x^d \pmod{N}$. Thus, the RSA function is a trapdoor permutation on the group $(\mathbb{Z}/N\mathbb{Z})^*$. It is not believed that inverting the RSA map is equivalent to factoring, so inversion of this map is identified as a separate problem (the RSA problem). At the time of writing, one should select p and q to be primes of at least 1536 bits in length to obtain suitable security.

There are many historic ways of using the RSA function as a public key encryption scheme, many of which are now considered obsolete and/or insecure. The two recommended methodologies in using RSA for encryption are RSA-OAEP and RSA-KEM; which are both IND-CCA secure in the random oracle model.

OAEP, or Optimized Asymmetric Encryption Padding, is a method to use the RSA function directly as an IND-CCA public key encryption algorithm. OAEP is parametrized by two integers $k_0, k_1 \geq 128$ such that $n = \log_2 N - k_0 - k_1 \geq 0$. We then encrypt messages of at most n bits in length as follows, using hash functions (which are assumed to be random oracles for the security proof)

$$G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{n+k_1}$$
$$H : \{0,1\}^{n+k_1} \longrightarrow \{0,1\}^{k_0}.$$

We then encrypt using the function

$$c \leftarrow \left(\{(m \parallel 0^{k_1}) \oplus G(R)\} \parallel \{R \oplus H\left((m \| 0^{k_1}) \oplus G(R)\right)\}\right)^e \pmod{N}$$

where

- $m \parallel 0^{k_1}$ means m followed by k_1 zero bits,
- R is a random bit string of length k_0,
- $\parallel$ denotes concatenation.

RSA-KEM, on the other hand, is a KEM which is much simpler to execute. To produce the encapsulated key and the ciphertext, one takes the random input r (which one thinks of as a uniformly random element in $(\mathbb{Z}/N\mathbb{Z})^*$). Then the KEM is defined by

$$c \leftarrow r^e \pmod{N}, \quad \mathfrak{k} \leftarrow H(r),$$

where $H : (\mathbb{Z}/N\mathbb{Z}) \longrightarrow \mathcal{K}$ is a hash function, which we model as a random oracle.

10.6.3 Constructions based on Elliptic Curves

Elliptic Curve Cryptography, or ECC, uses the fact that elliptic curves form a finite abelian group. In terms of encryption schemes, the standard method is to use ECIES (Elliptic Curve Integrated Encryption Scheme) to define a public key, KEM which is IND-CCA in the random oracle model, assuming the DDH problem in the subgroup of the elliptic curve being used. In practice, this means that one selects a curve $E(\mathbb{F}_p)$ for which there is a point $P \in E(\mathbb{F}_p)$ whose order is a prime $q > 2^{256}$.

For ECIES, the KeyGen algorithm is defined as follows. A secret key $\mathfrak{sk} \leftarrow \mathbb{F}_q^*$ is selected uniformly at random, and then the public key is set to be $Q \leftarrow [\mathfrak{sk}]P$. Key encapsulation is very similar to RSA-KEM in that it is defined by

$$r \leftarrow \mathbb{F}_q^*, \quad C \leftarrow [r] \cdot P, \quad \mathfrak{k} \leftarrow H([r] \cdot Q),$$

where $H : E(\mathbb{F}_p) \longrightarrow \mathcal{K}$ is a hash function (modelled as a random oracle). To decapsulate the key is recovered via

$$\mathfrak{k} \leftarrow H([\mathfrak{sk}]C).$$

Compared to RSA-based primitives, ECC-based primitives are relatively fast and use less bandwidth. This is because, at the time of writing, one can select elliptic curve parameters with $p \approx q \approx 2^{256}$ to obtain security equivalent to a work-factor of 2^{128} operations. Hence, in current systems elliptic curve-based systems are preferred over RSA-based ones.

10.6.4 Lattice-based Constructions

A major problem with both RSA and ECC primitives is that they are not secure against quantum computers; namely, Shor's algorithm will break both the RSA and ECC hard problems in polynomial time. Hence, the search is on for public key schemes which would resist the advent of a quantum computer. The National Institute of Standards and Technology (NIST) is currently engaged in a process to determine potential schemes which are *post-quantum* secure, see [976] for more details on this.

The most prominent of these so-called post-quantum schemes are those based on hard problems on lattices. In particular, the NTRU schemes and a variety of schemes based on the Learning With Errors (LWE) problem, and its generalisation to polynomial rings, known as the Ring-LWE problem. There are other proposals based on hard probles in coding theory, on the difficulty of computing isogenies between elliptic curves and other constructs. NIST is currently conducting a program to select potential post-quantum replacements.

10.7 PUBLIC KEY SIGNATURES

[962, c12][963, c16]

Public key encryption assumes that the receivers public key is known to be associated with the physical entity that the sender wishes to communicate with. This binding of public key with an entity is done via means of a so called *digital certificate*. A digital certificate is a signed statement that a given entity is associated with a given public key. This certificate is issued by a certificate authority, and utilizes the second main public key construct; namely a digital signature.

Just as with public key encryption algorithms, modern digital signature algorithms are based either on the RSA problem or a variant of the DLP problem; hence the reader is also directed again to [964] for more advanced details. For post-quantum security, there are a number of proposals based on lattice constructions; but none have yet been widely accepted or deployed at the time of writing this document. Again for PQC signatures we refer to the current NIST process [976].

The prototypical digital signature scheme given in text-books is loosely called RSA-FDH, where FDH stands for Full Domain Hash. The algorithm takes a message m and signs it by outputting

$$s = H(m)^d \pmod N.$$

Verification is then performed by testing whether the following equation holds

$$s^e = H(m) \pmod N.$$

Here, the hash function is assumed to have domain $\{0,1\}^*$ and co-domain $(\mathbb{Z}/N\mathbb{Z})^*$. This scheme comes with a security proof in the random oracle model, but is almost impossible to implement as no standardized hash function has co-domain the whole of $(\mathbb{Z}/N\mathbb{Z})^*$, since N is much bigger than the output of hash functions such as SHA-2.

All standardized hash functions output a value in $\{0,1\}^t$ for some t, thus what is usually done is to take a hash value and then prepend it with some known pre-determined values, and then 'sign' the result. This forms the basic idea behind the Public Key Cryptography Standards (PKCS) v1.5 signature standard. This signs a message by computing

$$s = (0x01 \| 0xFF \ldots 0xFF \| 0x00 \| H(m))^d \pmod N,$$

where enough padding of $0xFF$ bytes is done to ensure the whole padded string is just less than N in size. Despite the close relationship to RSA-FDH, the above signature scheme has no proof of security, and hence a more modern scheme is usually to be preferred.

10.7.1 RSA-PSS

The modern way to use the RSA primitive in a digital signature scheme is via the padding method called PSS (Probabilistic Signature Scheme). This is defined much like RSA-OAEP via the use of two hash functions, one which expands data and one which compresses data:

$$G : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k-k_1-1},$$
$$H : \{0,1\}^* \longrightarrow \{0,1\}^{k_1},$$

where $k = \log_2 N$. From G we define two auxiliary functions

$$G_1 : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k_0}$$

which returns the first k_0 bits of $G(w)$ for $w \in \{0,1\}^{k_1}$,

$$G_2 : \{0,1\}^{k_1} \longrightarrow \{0,1\}^{k-k_0-k_1-1}$$

which returns the last $k - k_0 - k_1 - 1$ bits of $G(w)$ for $w \in \{0,1\}^{k_1}$, i.e. $G(w) = G_1(w) \| G_2(w)$.

To sign a message m the private key holder performs the following steps:

- $r \leftarrow \{0,1\}^{k_0}$.
- $w \leftarrow H(m \| r)$.
- $y \leftarrow 0 \| w \| (G_1(w) \oplus r) \| G_2(w)$.
- $s \leftarrow y^d \pmod N$.

To verify a signature (s, m) the public key holder performs the following

- $y \leftarrow s^e \pmod N$.
- Split y into the components $b \| w \| \alpha \| \gamma$ where b is one bit long, w is k_1 bits long, α is k_0 bits long and γ is $k - k_0 - k_1 - 1$ bits long.
- $r \leftarrow \alpha \oplus G_1(w)$.
- The signature is verified as correct if and only if b is the zero bit, $G_2(w) = \gamma$ and $H(m \| r) = w$.

Despite being more complicated than PKCS-1.5, the RSA-PSS has a number of advantages. It is a randomized signature scheme, i.e. each application of the signing algorithm on the same message will produce a distinct signature, and it has a proof of security in the random oracle model relative to the difficulty of solving the RSA problem.

10.7.2 DSA, EC-DSA and Schnorr Signatures

The standard methodology for performing signatures in the discrete logarithm setting is to adapt interactive verification protocols, using proofs of knowledge of a discrete logarithm (see Sections 10.8.1 and 10.9.3), and then to convert them into a non-interactive signature scheme using a hash function.

The two most well known of these are the DSA (Digital Signature Algorithm) method and a method due to Schnorr. The former has widespread deployment but establishing security via security proofs uses less well-accepted assumptions, whereas the latter is less deployed but has well-established security proofs. The former also has, as we shall see, a more complex signing process. Both cases use a hash function H with co-domain $(\mathbb{Z}/q\mathbb{Z})^*$, unlike RSA-FDH this is easy to construct as q is relatively small.

We will describe both algorithms in the context of elliptic curves. Both make use of a public key of the form $Q = [x] \cdot P$, where x is the secret key. To sign a message m, in both algorithms, one first selects a random value $k \in (\mathbb{Z}/q\mathbb{Z})^*$, and computes $r \leftarrow \mathsf{x} - \mathsf{coord}([k] \cdot P)$. One then computes a hash of the message. In the DSA algorithm, this is done with $e \leftarrow H(m)$, whereas

for the Schnorr algorithm, one computes it via $e \leftarrow H(m\|r)$. Then the signature equation is applied which, in the case of EC-DSA, is

$$s \leftarrow (e + x \cdot r)/k \pmod{q}$$

and, in the case of Schnorr, is

$$s \leftarrow (k + e \cdot x) \pmod{q}.$$

Finally, the output signature is given by (r, s) for EC-DSA and (e, s) for Schnorr.

Verification is done by checking the equation

$$r = \mathrm{x} - \mathrm{coord}([e/s] \cdot P + [r/s] \cdot Q)$$

in the case of EC-DSA, and by checking

$$e = H\left(m\|\mathrm{x} - \mathrm{coord}([s] \cdot P - e \cdot Q)\right)$$

in the case of Schnorr. The key difference in the two algorithms is not the signing and verification equations (although these do affect performance), but the fact that, with the Schnorr scheme, the r value is also entered into the hash function to produce e. This small distinction results in the different provable security properties of the two algorithms.

A key aspect of both EC-DSA and Schnorr signatures is that they are very brittle to exposure of the per-message random nonce k. If only a small number of bits of k leak to the attacker with every signing operation, then the attacker can easily recover the secret key.

10.8 STANDARD PROTOCOLS

[963, c18]

Cryptographic protocols are interactive operations conducted between two or more parties in order to realize some cryptographic goal. Almost all cryptographic protocols make use of the primitives we have already discussed (encryption, message authentication, secret sharing). In this section, we discuss the two most basic forms of protocol, namely authentication and key agreement.

10.8.1 Authentication Protocols

In an authentication protocol, one entity (the Prover) convinces the other entity (the Verifier) that they are who they claim to be, and that they are 'online'; where 'online' means that the verifying party is assured that the proving party is actually responding and it is not a replay. There are three basic types of protocol: Encryption based, Message Authentication based and Zero-Knowledge based.

10.8.1.1 Encryption-Based Protocols

These can operate in the symmetric or public key setting. In the symmetric key setting, both the prover and the verifier hold the same secret key, whilst in the public key setting, the prover holds the private key and the verifier holds the public key. In both settings, the verifier first encrypts a random nonce to the prover, the prover then decrypts this and returns it to the verifier, the verifier checks that the random nonce and the returned value are equivalent.

$$
\begin{array}{ccc}
\underline{\text{Verifier}} & & \underline{\text{Prover}} \\
N \leftarrow \mathcal{M} & & \\
c \leftarrow \mathsf{Enc}(N, \mathfrak{pk}; r) & \xrightarrow{c} & \\
 & \xleftarrow{m} & m \leftarrow \mathsf{Dec}(c, \mathfrak{sk}) \\
N \stackrel{?}{=} m & &
\end{array}
$$

The encryption scheme needs to be IND-CCA secure for the above protocol to be secure against active attacks. The nonce N is used to prevent replay attacks.

10.8.1.2 Message Authentication-Based Protocols

These also operate in the public key or the symmetric setting. In these protocols, the verifier sends a nonce in the clear to the prover, the prover then produces a digital signature (or a MAC in the symmetric key setting) on this nonce and passes it back to the verifier. The verifier then verifies the digital signature (or verifies the MAC). In the following diagram we give the public key/digital signature based variant.

$$
\begin{array}{ccc}
\underline{\text{Verifier}} & & \underline{\text{Prover}} \\
N \leftarrow \mathcal{M} & \xrightarrow{N} & \\
 & \xrightarrow{\sigma} & \sigma \leftarrow \mathsf{Sign}(N, \mathfrak{sk}) \\
\mathsf{Verify}(N, \sigma, \mathfrak{pk}) \stackrel{?}{=} \mathsf{true} & &
\end{array}
$$

10.8.1.3 Zero-Knowledge-Based

Zero-knowledge-based authentication protocols are the simplest examples of zero-knowledge protocols (see Section 10.9.3) available. The basic protocol is a so-called Σ- (or Sigma-) protocol consisting of three message flows; a commitment, a challenge and a response. The simplest example is the Schnorr identification protocol, based on the hardness of computing discrete logarithms. In this protocol, the Prover is assumed to have a long-term secret x and an associated public key $Q = [x] \cdot P$. One should note the similarity of this protocol to the Schnorr signature scheme above.

$$
\begin{array}{ccc}
\underline{\text{Verifier}} & & \underline{\text{Prover}} \\
 & & k \leftarrow \mathbb{Z}/q\mathbb{Z} \\
 & \xleftarrow{r} & R \leftarrow [k] \cdot P \\
e \leftarrow \mathbb{Z}/q\mathbb{Z} & \xrightarrow{e} & \\
 & \xleftarrow{s} & s \leftarrow (k + e \cdot x) \pmod{q} \\
R \stackrel{?}{=} [s] \cdot P - e \cdot Q & &
\end{array}
$$

Indeed, the conversion of the Schnorr authentication protocol into the Schnorr signature scheme is an example of the Fiat–Shamir transform, which transforms any Σ-protocol into a signature scheme. If the underlying Σ-protocol is secure, in the sense of a zero-knowledge proofs of knowledge (see Section 10.9.3), then the resulting signature scheme is UF-CMA.

10.8.2 Key Agreement Protocols

A key agreement protocol allows two parties to agree on a secret key for use in subsequent protocols. The security requirements of key agreement protocols are very subtle, leading to various subtle security properties that many deployed protocols may or may not have. We recap on basic properties of key agreement protocols here, but a more complete discussion can be found in [977]. The basic security requirements are

- The underlying key should be indistinguishable from random to the adversary, or that at least it should be able to be used in the subsequent protocol without the adversary breaking the subsequent protocol.
- Each party is assured that only the other party has access to the secret key. This is so-called mutual authentication. In many application scenarios (e.g. in the standard application of Transport Layer Security (TLS) to web browsing protocol), one only requires this property of one-party, in which case we are said to only have one-way authentication.

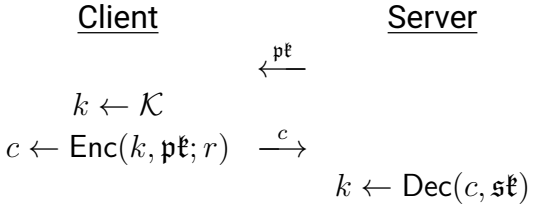
Kerberos is an example of a (usually) symmetric key-based key agreement system. This is a protocol that requires trusted parties to relay and generate secret keys from one party to another. It is most suited to closed corporate networks. On the public internet, protocols like Kerberos are less useful. Thus, here one uses public key-based protocols such as TLS and IPSec. More advanced properties required of modern public key-based protocols are as follows.

- **Key Confirmation:** The parties know that the other party has received the same secret key. Sometimes this can be eliminated as the correct execution of the subsequent protocol using the secret key provides this confirmation. This later process is called *implicit key confirmation*.
- **Forward Secrecy:** The compromise of a participant's long-term secret in the future does not compromise the security of the secret key derived now, i.e. current conversations are still secure in the future.
- **Unknown Key Share Security:** This prevents one party (Alice) sharing a key with Bob, whereas Bob thinks he shares a key with Charlie, despite sharing it with Alice.

Variations on the theme of key agreement protocols include group key agreement, which enables a group of users to agree on a key, or password based key agreement, in which two parties only agree on a (high entropy) key if they also agree on a shared password.

10.8.2.1 Key Transport

The most basic form of key agreement protocol is a form of key transport in which the parties use public key encryption to exchange a random key. In the case of a one-way authenticated protocol, this was the traditional method of TLS operation (up until TLS version 1.2) between a server and a client



This protocol produced the pre-master secret in older versions of TLS (pre-TLS 1.2). To derive the final secret in TLS, further nonces were exchanged between the parties (to ensure that

both parties were alive and the key was fresh). Then, a master secret was derived from the pre-master secret and the nonces. Finally, key confirmation was provided by the entire protocol transcript being hashed and encrypted under the master secret (the so-called *FINISHED* message). In TLS, the resulting key is not indistinguishable from random as the encrypted FINISHED message provides the adversary with a trivial check to determine whether a key is real or not. However, the protocol can be shown to be secure for the purposes of using the master secret to produce a secure bi-directional channel between the server and the client.

A more basic issue with the above protocol is that it is not forward-secure. Any adversary who records a session now, and in the future manages to obtain the server's long-term secret $\mathfrak{sk}$, can obtain the pre-master secret, and hence decrypt the entire session.

10.8.2.2 Diffie–Hellman Key Agreement

To avoid the issues with forward secrecy of RSA-based key transport, modern protocols make use of Diffie–Hellman key exchange. This allows two parties to agree on a uniformly random key, which is indistinguishable from random assuming the Decision Diffie–Hellman problem is hard

$$\begin{array}{cc} \underline{\text{Alice}} & \underline{\text{Bob}} \\ a \leftarrow \mathbb{Z}/q\mathbb{Z} & b \leftarrow \mathbb{Z}/q\mathbb{Z} \\ Q_A \leftarrow [a] \cdot P & \xrightarrow{Q_A} \\ & \xleftarrow{Q_B} \quad Q_B \leftarrow [b] \cdot P \\ K \leftarrow [a] \cdot Q_B & K \leftarrow [b] \cdot Q_A \end{array}$$

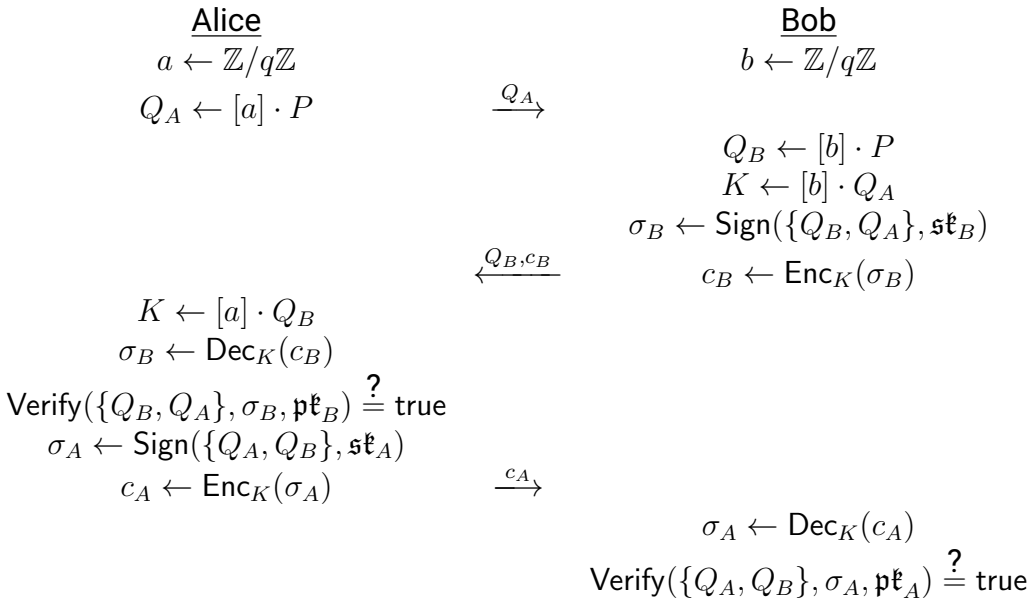
This protocol provides forward secrecy, but provides no form of authentication. Due to this, the protocol suffers from a man-in-the-middle attack. To obtain mutual authentication, the message flow of Q_A is signed by Alice's public key and the message flow of Q_B is signed by Bob's public key. This prevents the man-in-the-middle attack. However, since the signatures are not bound into the message, the signed-Diffie–Hellman protocol suffers from an unknown-key-share attack; an adversary (Charlie) can strip Alice's signature from Q_A and replace it with their signature. The adversary does not learn the secret, but does convince Bob he is talking to another entity.

The one-way authenticated version of Diffie–Hellman key agreement is the preferred method of key agreement in modern TLS deployments, and is the only method of key agreement supported by TLS 1.3. In TLS, the FINISHED message, which hashes the entire transcript, prevents the above unknown-key-share attack. However, it also prevents the protocol from producing keys which are indistinguishable from random, as mentioned above.

10.8.2.3 Station-to-Station Protocol

The Station-to-Station (STS) protocol can be used to prevent unknown-key-share attacks on signed Diffie–Hellman and maintain key indistinguishability. In this protocol, the Diffie–Hellman derived key is used to encrypt the signatures, thus ensuring the signatures cannot be stripped off the messages. In addition, the signatures are applied to the transcript so as to

convince both receiving parties that the other party is 'alive'.



10.9 ADVANCED PROTOCOLS

[963, c20–c22]

Modern cryptography has examined a number of more complex protocols to achieve more complex ends. For example, secure e-voting schemes, secure auctions, data storage and retrieval, etc. Most of these advanced protocols are either based on the simpler components described in this section and/or on the encryption and signature schemes with special properties discussed in the next section. Here we restrict ourselves to discussing three widely needed protocols: Oblivious Transfer, Zero-Knowledge and Multi-Party Computation.

10.9.1 Oblivious Transfer

While Oblivious Transfer (OT) is at the heart of many advanced protocols, it is a surprisingly simple primitive which enables one to accomplish various more complex tasks. In the following, we describe the basic 1-out-of-2 Oblivious Transfer, but extensions to n-out-of-m are immediate. In all cases, the protocol is one executed between a Sender and a Receiver.

In a 1-out-of-2 Oblivious Transfer, the Sender has two messages m_0 and m_1, whilst the Receiver has an input bit b. The output of the protocol should be the message m_b for the Receiver, and the Sender obtains no output. In particular, the Receiver learns nothing about the message m_{1-b}, whilst the Sender learns nothing about the bit b.

This passively secure protocol can be implemented as follows. We assume the Sender's

messages are two elements M_0 and M_1 in an elliptic curve group $E(\mathbb{F}_p)$ of prime order q.

Sender		Receiver
$C \leftarrow E(\mathbb{F}_p)$	$\xrightarrow{C}$	
		$x \leftarrow (\mathbb{Z}/q\mathbb{Z})$
		$Q_b \leftarrow [x] \cdot P$
		$Q_{1-b} \leftarrow C - Q_b$
	$\xleftarrow{Q_0}$	
$Q_1 \leftarrow C - Q_0$		
$k \leftarrow (\mathbb{Z}/q\mathbb{Z})$		
$C_1 \leftarrow [k] \cdot P$		
$E_0 \leftarrow M_0 + [k] \cdot Q_0$		
$E_1 \leftarrow M_1 + [k] \cdot Q_1$		
	$\xrightarrow{C_1, E_0, E_1}$	
		$M_b \leftarrow E_b - [x] \cdot C_1$

The extension to an actively secure protocol is only a little more complex, but beyond the scope of this article.

10.9.2 Private Information Retrieval and ORAM

A Private Information Retrieval (PIR) protocol is one which enables a computer to retrieve data from a server held database, without revealing the exact item which is retrieved. If the server has n data items then this is related to a 1-out-of-n OT protocol. However, in PIR we do not insist that the user does not learn anything else about the servers data, we only care about privacy of the user query. In addition protocols for PIR are meant to be run many times, and we are interested in hiding the total set of access patterns, i.e. even whether a data item is retrieved multiple times. The goal of PIR protocols is to obtain greater efficiency than the trivial solution of the server sending the user the entire database.

An Oblivious Random Access Memory (ORAM) protocol is similar but now we not only allow the user to obliviously read from the server's database, we also allow the user to write to the database. So as to protect the write queries the server held database must now be held in an encrypted form (so what is written cannot be determined by the server). In addition the access patterns, i.e. where data is written to and read from, needs to be hidden from the server.

10.9.3 Zero-Knowledge

A Zero-Knowledge protocol is a protocol executed between a Prover and a Verifier in which the Prover demonstrates that a statement is true, without revealing why the statement is true. The concept is used in many places in cryptography, to construct signature schemes, to attest ones identity, and to construct more advanced protocols. An introduction to the more theoretical aspects of zero-knowledge can be found in [965]. More formally, consider an NP language $\mathcal{L}$ (i.e. a set of statements x which can be verified to be true in polynomial time given a witness or proof w). An interactive proof system for $\mathcal{L}$ is a sequence of protocol executions by an (infinitely powerful) Prover P and a (probabilistic polynomial time) Verifier V, which on joint input x proceeds as follows:

$$
\begin{array}{rcl}
\underline{\text{Verifier}} & & \underline{\text{Prover}} \\
 & \xleftarrow{p_1} & (p_1, s_1') \leftarrow P_1(x) \\
(v_1, s_1) \leftarrow V_1(x, p_1) & \xrightarrow{v_1} & \\
 & \xleftarrow{p_2} & (p_2, s_2') \leftarrow P_2(s_1', v_1) \\
(v_2, s_2) \leftarrow V_2(s_1, p_2) & \xrightarrow{v_2} & \\
 & \xleftarrow{p_3} & (p_3, s_3') \leftarrow P_3(s_2', v_2) \\
\vdots & & \vdots \\
 & \xrightarrow{p_r} & (p_r, s_r') \leftarrow P_r(s_{r_1}, v_{r_1})
\end{array}
$$

By the end of the protocol, the Verifier will output either true or false. An interactive proof system is one which is both *complete* and *sound*

- **Completeness:** If the statement x is true, i.e. $x \in \mathcal{L}$, then if the Prover is honest then the Verifier will output true.
- **Soundness:** If the statement is false, i.e. $x \notin \mathcal{L}$, then no cheating Prover can convince the Verifier with probability greater than p.

Note that even if p is large (say $p = 0.5$) then repeating the proof multiple times can reduce the soundness probability to anything desired. Of course, protocols with small p to start with are going to be more efficient.

For any NP statement, there is a trivial proof system. Namely, the Prover simply sends over the witness w which the Verifier then verifies. However, this reveals the witness. In a zero-knowledge proof, we obtain the same goal, but the Verifier learns nothing bar the fact that $x \in \mathcal{L}$. To formally define zero-knowledge, we insist that there is a (probabilistic polynomial time) *simulator* S which can produce protocol transcripts identical to the transcripts produced between a Verifier and an honest Prover; except the simulator has no access to the Prover. This implies that the Verifier cannot use the transcript to perform any other task, since what it learned from the transcript it could have produced without the Prover by simply running the simulator.

A zero-knowledge proof is said to be *perfect zero-knowledge* if the distribution of transcripts produced by the simulator is identical to those produced between a valid prover and verifier. If the two distributions only cannot be distinguished by an efficient algorithm we say we have *computational zero-knowledge*.

A zero-knowledge proof is said to be a *proof of knowledge* if a Verifier given rewinding access to the prover (i.e. the Verifier can keep resetting the Prover to a previous protocol state and

continue executing) can extract the underlying witness w. This implies that the Prover must 'know' w since we can extract w from it.

A *non-interactive zero-knowledge proof* is one in which there is no message flowing from the Verifier to the Prover, and only one message flowing from the Prover to the Verifier. Such non-interactive proofs require additional setup assumptions, such as a Common Reference String (CRS), or they require one to assume the Random Oracle Model. Traditionally these are applied to specific number theoretic statements, such to show knowledge of a discrete logarithm (see the next section on Σ-protocols), however recently so called Succinct Non-Interactive Arguments of Knowledge (SNARKs) have been developed which enable such non-interactive arguments for more complex statements. Such SNARKs are finding applications in some blockchain systems.

10.9.3.1 Σ-Protocols

The earlier Σ-protocol for identification is a zero-knowledge proof of knowledge.

Verifier		Prover
		$k \leftarrow \mathbb{Z}/q\mathbb{Z}$
	$\xleftarrow{R}$	$R \leftarrow [k] \cdot P$
$e \leftarrow \mathbb{Z}/q\mathbb{Z}$	$\xrightarrow{e}$	
	$\xleftarrow{s}$	$s \leftarrow (k + e \cdot x) \pmod q$
$R \stackrel{?}{=} [s] \cdot P - e \cdot Q$		

The protocol is obviously complete since $Q = [x] \cdot P$, and the soundness error is $1/q$. That it is zero-knowledge follows from the following simulation, which first samples $e, s \leftarrow \mathbb{Z}/q\mathbb{Z}$ and then computes $R = [s]P - e \cdot Q$; the resulting simulated transcript being (R, e, s). Namely, the simulator computes things in the wrong order.

The protocol is also a proof of knowledge since if we execute two protocol runs with the same R value but different e-values (e_1 and e_2) then we obtain two s-values (s_1 and s_2). This is done by rewinding the prover to just after it has sent its first message. If the two obtained transcripts (R, e_1, s_1) and (R, e_2, s_2) are both valid then we have

$$R = [s_1] \cdot P - e_1 \cdot Q = [s_2] \cdot P - e_2 \cdot Q$$

and so

$$[s_1 - s_2] \cdot P = [e_1 - e_2] \cdot Q$$

and hence

$$Q = \left[\frac{s_1 - s_2}{e_1 - e_2}\right] \cdot P$$

and hence we 'extract' the secret x from $x = (s_1 - s_2)/(e_1 - e_2) \pmod q$.

10.9.4 Secure Multi-Party Computation

Multi-Party Computation (MPC) is a technique to enable a set of parties to compute on data, without learning anything about the data. Consider n parties $P_1, \ldots, P_n$ each with input $x_1, \ldots, x_n$. MPC allows these parties to compute *any* function $f(x_1, \ldots, x_n)$ of these inputs without revealing any information about the x_i to each other, bar what can be deduced from the output of the function f. A general introduction to the theory of such protocols can be found in [966].

In an MPC protocol, we assume that a subset of the parties A is corrupt. In *statically* secure protocols, this set is defined at the start of the protocol, but remains unknown to the honest parties. In an *adaptively* secure protocol, the set can be chosen by the adversary as the protocol progresses. An MPC protocol is said to be *passively* secure if the parties in A follow the protocol, but try to learn data about the honest parties' inputs from their joint view. In an *actively* secure protocol, the parties in A can arbitrarily deviate from the protocol.

An MPC protocol should be *correct*, i.e. it outputs the correct answer if all parties follow the protocol. It should also be *secure*, i.e. the dishonest parties should learn nothing about the inputs of the honest parties. In the case of active adversaries, a protocol is said to be *robust* if the honest parties will obtain the correct output, even when the dishonest parties deviate from the protocol. A protocol which is not robust, but which aborts with overwhelming probability when a dishonest party deviates, is said to be an actively secure MPC protocol *with abort*.

MPC protocols are catagorized by whether they utilize information-theoretic primitives (namely secret sharing), or they utilize computationally secure primitives (such as symmetric-key and public-key encryption). They are also further characterized by the properties of the set A. Of particular interest is when the size t of A is bounded by a function of n (so-called threshold schemes). The cases of particular interest are $t < n$, $t < n/2$, and $t < n/3$; the threshold cases of $t < n/2$ and $t < n/3$ can be generalized to Q_2 and Q_3 access structures, as discussed in Section 10.3.2.

In the information-theoretic setting, one can achieve passively secure MPC in the case of $t < n/2$ (or Q_2 access structures). Actively secure robust MPC is possible in the information-theoretic setting when we have $t < n/3$ (or Q_3 access structures). All of these protocols are achieved using secret sharing schemes. A detailed study of secret sharing based MPC protocols is given in [978].

In the computational setting, one can achieve actively secure robust computation when $t < n/2$, using Oblivious Transfer as the basic computational foundation. The interesting case of two party computation is done using the Yao protocol. This protocol has one party (the Circuit Creator, also called the Garbler) 'encrypting' a boolean function gate by gate using a cipher such as AES, the circuit is then sent to the other party (called the Circuit Evaluator). The Evaluator then obtains the 'keys' for their input values from the Creator using Oblivious Transfer, and can then evaluate the circuit. A detailed study of two party Yao based protocols is given in [979].

Modern MPC protocols have looked at active security with abort in the case of $t < n$. The modern protocols are divided into a function-dependent offline phase, which requires public key functionality but which is function independent, then a function-dependent online phase which mainly uses information-theoretic primitives. Since information theoretic primitives are usually very fast, this means the time-critical online phase can be executed as fast as possible.

10.10 PUBLIC KEY ENCRYPTION/SIGNATURES WITH SPECIAL PROPERTIES

[962, c13]

A major part of modern cryptography over the last twenty years has been the construction of encryption and signature algorithms with special properties or advanced functionalities. A number of the following have been deployed in specialized systems (for example, U-PROVE, IDEMIX, attestation protocols and some cryptocurrencies). We recap the main variants below, giving for each one the basic idea behind their construction.

10.10.1 Group Signatures

A group signature scheme defined a group public key $\mathfrak{pk}$, associated to a number of secret keys $\mathfrak{sk}_1, \ldots, \mathfrak{sk}_n$. The public key is usually determined by an entity called a *Group Manager*, during an interaction with the group members. Given a group signature s, one cannot tell which secret key signed it, although one is guaranteed that one did. Thus group signatures provide the anonymity of a Signer. Most group signature algorithms have a special entity called an *Opener* who has some secret information which enables them to revoke the anonymity of a Signer. This last property ensures one can identify group members who act dishonestly in some way.

A group signature scheme can either support *static* or *dynamic* groups. In a static group signature scheme, the group members are fixed at the start of the protocol, when the public key is fixed. In a dynamic group signature scheme the group manager can add members into the group as the protocol proceeds, and (often) revoke members as well.

An example of this type of signature scheme which is currently deployed is the Direct Anonymous Attestation (DAA) protocol; which is essentially a group signature scheme in which the *Opener* is replaced with a form of user controlled linkability; i.e. a signer can decide whether two signatures output by the specific signer can be linked or not.

10.10.2 Ring Signatures

A ring signature scheme is much like a group signature scheme, but in a ring signature there is no group manager. Each user in a ring signature scheme has a public/private key pair $(\mathfrak{pk}_i, \mathfrak{sk}_i)$. At the point of signing, the Signer selects a subset of the public keys (containing this own), which is called a ring of public keys, and then produces a signature. The Receiver knows the signature was produced by someone in the ring, but not which member of the ring.

10.10.3 Blind Signatures

A blind signature scheme is a two party protocol in which a one party (the User) wants to obtain the signature on a message by a second party (the Signer). However, the Signer is not allowed to know which message is being signed. For example, the Signer may be simply notarising that something happened, but does not need to know precisely what. Security requires that the Signer should not learn anything about any message passed to it for signing, and the user should not obtain the signature on any message other than those they submitted for signing.

10.10.4 Identity-Based Encryption

In normal public key encryption, a user obtains a public key $\mathfrak{pk}$, along with a certificate C. The certificate is produced by a trusted third party, and binds the public key to the identity. Usually, a certificate is a digitally signed statement containing the public key and the associated user identity. So, when sending a message to Alice the Sender is sure that Alice is the legitimate holder of public key $\mathfrak{pk}$.

Identity Based Encryption (IBE) is an encryption scheme which dispenses with the need for certificate authorities, and certificates. To encrypt to a user, say Alice, we simply use her identity *Alice* as the public key, plus a global 'system' public key. However, to enable Alice to decrypt, we must have a trusted third party, called a Key Generation Centre, which can provide Alice with her secret key. This third party uses its knowledge of the 'system' secret key to be able to derive Alice's secret key. Whilst dispensing with certificates, an IBE system inherently has a notion of key escrow; the Key Generation Centre can decrypt all messages.

10.10.5 Linearly Homomorphic Encryption

In a linearly homomorphic encryption scheme one can perform a number of linear operations on ciphertexts, which result in a ciphertext encrypting a message having had the same operations performed on the plaintext. Thus, given two encryptions $c_1 \leftarrow \mathsf{Enc}(m_1, \mathfrak{pk}; r_1)$ and $c_2 \leftarrow \mathsf{Enc}(m_2, \mathfrak{pk}; r_2)$ one can form a 'sum' operation $c \leftarrow c_1 \oplus c_2$ such that c decrypts to $m_1 + m_2$. The standard example of such encryption schemes is the Paillier encryption scheme, which encrypts elements $m \in (\mathbb{Z}/N\mathbb{Z})$, for an RSA-modulus N by computing $c \leftarrow (1+N)^m \cdot r^N \pmod{N^2}$ where r is selected in $\mathbb{Z}/N\mathbb{Z}$.

Such encryption algorithms can never be IND-CCA secure, as the homomorphic property produces a trivial malleability which can be exploited by a CCA attacker. However, they can have applications in many interesting areas. For example, one can use a linearly homomorphic encryption scheme to add up votes in a digitally balloted election for two candidates, where each vote is an encryption of either the message zero or one.

10.10.6 Fully Homomorphic Encryption

Fully Homomorphic Encryption (or FHE) is an extension to linearly homomorphic encryption, in that one can not only homomorphically evaluate linear functions, but also non-linear ones. In particular, the ability to homomorphically evaluate both addition *and* multiplication on encrypted data enables one to (theoretically) evaluate any function. Applications of FHE which have been envisioned are things such as performing complex search queries on encrypted medical data etc. Thus, FHE is very interesting in a cloud environment.

All existing FHE schemes are highly inefficient. Thus only very simple functions can be evaluated in suitable time limits. A scheme which can perform homomorphic operations from a restricted class of functions (for example, to homomorphically evaluate all multi-variate polynomials of total degree five) is called a Somewhat Homomorphic Encryption (or SHE) scheme. Obviously, if the set of functions are all multi-variate polynomials of degree one, then the SHE scheme is a linear homomorphic encryption scheme.

10.11 IMPLEMENTATION ASPECTS

There are two aspects one needs to bear in mind with respect to cryptographic implementation. Firstly security and secondly performance.

In terms of security the main concern is one of side-channel attacks. These can be mounted against both hardware implementations, for example cryptographic circuits implemented on smart-cards, or against software implementations running on commodity processors. Any measurable difference which occurs when running an algorithm on one set of inputs versus another can lead to an attack. Such measurements may involve timing differences, power comsumption differences, differences in electromagnetic radiation, or even differences in the sound produced by the fan on the processor. It is even possible to mount remote side-channel attacks where one measures differences in response times from a remote server. A good survey of such attacks, focused on power analysis applied to symmetric algorithms such as AES, can be found in [980].

To protect against such side-channel attacks at the hardware level various techniques have been proposed including utilizing techniques based on secret-sharing (called *masking* in the side-channel community). In the area of software one needs to ensure code is constant-time at the least (i.e. every execution path takes the same amount of time), indeed having multiple execution paths can itself lead to attacks via power-analysis.

To enable increased performance it is becoming increasingly common for processor manufacturers to supply special instructions to enable improvements to cryptographic algorithms. This is similar to the multi-media extensions which have been common place for other applications for some decades. An example of this is special instructions on x86 chips to perform operations related to AES, to perform GCM-mode and to perform some ECC operations.

Public key, i.e. number theoretic constructions, are particularly expensive in terms of computational resources. Thus it is common for these specific algorithms to be implemented in low level machine code, which is tuned to a specific architecture. However, this needs to be done with care so as to take into account the earlier mentioned side-channel attacks.

Finally an implementation can also be prone to fault attacks. These are attacks in which an attacker injects faults (either physical faults on hardware, or datagram faults into a protocol).

Defences against such attacks need to be considered including standard fault tolerent computing approaches in hardware, and full input validation in all protocols. Further details on fault attacks can be found in [981].

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

The two main textbooks below we cross-reference against the main sections here. Further topic specific reading is given by references to the main bibliography.

	[962]	[963]	Other
10.1 Mathematics	c8–c9, App B	c1–c5	[964]
10.2 Cryptographic Security Models	c1–c4	c11	[965, 966, 967, 968]
10.3 Information-theoretically Secure Constructions	c2	c19	[969]
10.4 Symmetric Primitives	c3–c6	c11–c14	[970, 971, 972, 973, 974]
10.5 Symmetric Encryption and Authentication	c3–c4	c13–c14	[974, 975]
10.6 Public Key Encryption	c11	c15–c17	[964, 976]
10.7 Public Key Signatures	c12	c16	[964, 976]
10.8 Standard Protocols	–	c18	[977]
10.9 Advanced Protocols	–	c20–c22	[965, 966, 978, 979]
10.10 Public Key Encryption/Signatures With Special Properties	c13	–	
10.11 Implementation Aspects	–	–	[980, 981]

FURTHER READING

The following two text books are recommended to obtain further information on the topics in this knowledge area. Further topic specific reading is given in the bibliography.

Introduction to Modern Cryptography (J. Katz and Y. Lindell) [962]

A standard modern textbook covering aspects of the design of cryptographic schemes from a provable security perspective.

Cryptography Made Simple (N.P. Smart) [963]

A textbook with less mathematical rigour than the previously mentioned one, but which also covers a wider range of areas (including zero-knowledge and MPC), and touches on aspects related to implementation.

Chapter 11
Operating Systems and Virtualisation

Herbert Bos | Vrije Universiteit Amsterdam

INTRODUCTION

In this Knowledge Area, we introduce the principles, primitives and practices for ensuring security at the operating system and hypervisor levels. We shall see that the challenges related to operating system security have evolved over the past few decades, even if the principles have stayed mostly the same. For instance, when few people had their own computers and most computing was done on multi-user (often mainframe-based) computer systems with limited connectivity, security was mostly focused on isolating users or classes of users from each other[1]. Isolation is still a core principle of security today. Even the entities to isolate have remained, by and large, the same. We will refer to them as security domains. Traditional security domains for operating systems are processes and kernels, and for hypervisors, Virtual Machines (VMs). Although we may have added trusted execution environments and a few other security domains in recent years, we still have the kernel, user processes and virtual machines as the main security domains today. However, the threats have evolved tremendously, and in response, so have the security mechanisms.

As we shall see, some operating systems (e.g., in embedded devices) do not have any notion of security domains whatsoever, but most distinguish between multiple security domains such as the kernel, user processes and trusted execution environments. In this Knowledge Area, we will assume the presence of multiple, mutually non-trusting security domains. Between these security domains, operating systems manage a computer system's resources such as CPU time (through scheduling), memory (through allocations and address space mappings) and disk blocks (via file systems and permissions). However, we shall see that protecting such traditional, coarse-grained resources is not always enough and it may be necessary to explicitly manage the more low-level resources as well. Examples include caches, Transaction Lookaside Buffers (TLBs), and a host of other shared resources. Recall that Saltzer and Schroeder's Principle of Least Common Mechanism [8] states that every mechanism shared between security domains may become a channel through which sensitive data may leak. Indeed, all of the above shared resources have served as side channels to leak sensitive information in attack scenarios.

As the most privileged components, operating systems and hypervisors play a critical role in making systems (in)secure. For brevity, we mainly use the term operating system and processes in the remainder of this knowledge area and refer to hypervisors and VMs explicitly where the distinction is important[2].

While security goes beyond the operating system, the lowest levels of the software stack form the bedrock on which security is built. For instance, the operating system may be capable of executing privileged instructions not available to ordinary user programs and typically offers the means to authenticate users and to isolate the execution and files of different users. While it is up to the application to enforce security beyond this point, the operating system guarantees that non-authorised processes cannot access its files, memory, CPU time, or other resources. These security guarantees are limited by what the hardware can do. For instance, if a CPU's Instruction Set Architecture (ISA) does not have a notion of multiple privilege levels or address space isolation to begin with, shielding the security domains from each other is difficult—although it may still be possible using language-based protection (as in the experimental Singularity operating system [983]).

[1] A situation, incidentally, that is not unlike that of shared clouds today.

[2] Targeted publications about developments in threats and solutions for virtualised environments have appeared elsewhere [982]

The security offered *by* the operating system is also threatened by attacks that aim to evade the system's security mechanisms. For instance, if the operating system is responsible for the separation between processes and the operating system itself gets compromised, the security guarantees are void. Thus, we additionally require security *of* the operating system.

After explaining the threat model for operating system security, we proceed by classifying the different design choices for the underlying operating system structure (monolithic versus microkernel-based, multi-server versus libraryOS, etc.), which we then discuss in relation to fundamental security principles and models. Next, we discuss the core primitives that operating systems use to ensure different security domains are properly isolated and access to sensitive resources is mediated. Finally, we describe important techniques that operating systems employ to harden the system against attacks.

11.1 ATTACKER MODEL

[984, c1-c9][985, c9][986][982]

We assume that attackers are interested in violating the security guarantees provided by the operating system or hypervisor: leak confidential data (e.g., crypto keys), modify data that should not be accessible (e.g., to elevate privileges) or limit the availability of the system and its services (e.g., by crashing the system or hogging its resources). In this knowledge area, we focus on the technical aspects of security, leaving aside insider threats, human behaviour, physical attacks, project management, company policies, etc. Not because they are not important, but because they are beyond OS control and would require a knowledge area of their own. Table 11.1 lists some of the threats and attack methods that we do consider.

The simplest way to compromise the system is to inject a malicious extension into the heart of the operating system. For instance, in monolithic systems such as Linux and Windows, this could be a malicious driver or kernel module, perhaps inadvertently loaded as a Trojan, that has access to all privileged functionality [987]. To maintain their hold on the system in a stealthy manner regardless of what the operating system or hypervisor may do, the attackers may further infect the system's boot process (e.g., by overwriting the master boot record or the Unified Extensible Firmware Interface (UEFI), firmware)—giving the malicious code control over the boot process on every reboot, even *before* the operating system runs, allowing it to bypass any and all operating system level defenses [988].

Besides using Trojans, attackers frequently violate the security properties without any help from the user, by exploiting vulnerabilities. In fact, attackers may use a wide repertoire of methods. For instance, they commonly abuse vulnerabilities in the software, such as memory errors [986] to change code pointers or data in the operating system and violate its integrity, confidentiality or availability. By corrupting a code pointer, they control where the program resumes execution after the call, jump or return instruction that uses the corrupted code pointer. Changing data or data pointers opens up other possibilities, such as elevating the privilege level of an unprivileged process to 'root' (giving all-powerful 'system' privileges) or modifying the page tables to give a process access to arbitrary memory pages. Likewise, they may use such bugs to leak information from the operating system by changing which or how much data is returned for a system call, or a network request.

Attackers may also abuse vulnerabilities in hardware, such as the Rowhammer bug present in many DRAM chips [989]. Since bits in memory chips are organised in rows and packed very

Attack	Description
Malicious extensions	Attacker manages to convince the system to load a malicious driver or kernel module (e.g., as a Trojan).
Bootkit	Attacker compromises the boot process to gain control even before the operating system gets to run.
Memory errors (software)	Spatial and temporal memory errors allow attackers (local or remote) to divert control flow or leak sensitive information.
Memory corruption (hardware)	Vulnerabilities such as Rowhammer in DRAM allow attackers (local or remote) to modify data that they should not be able to access.
Uninitalised data leakage	The operating system returns data to user programs that is not properly initialised and may contain sensitive data.
Concurrency bugs and double fetch	Example: the operating system uses a value from userspace twice (e.g., a size value is used once to allocate a buffer and later to copy into that buffer) and the value changes between the two uses.
Side channels (hardware)	Attackers use access times of shared resources such as caches and TLBs to detect that another security domain has used the resource, allowing them to leak sensitive data.
Side channels (speculative)	Security checks are bypassed in speculative or out-of-order execution and while results are squashed they leave a measurable trace in the micro-architectural state of the machine.
Side channels (software)	Example: when operating systems / hypervisors use features such as memory deduplication, attackers can measure that another security domain has the same content.
Resource depletion (DoS)	By hogging resources (memory, CPU, buses, etc.), attackers prevent other programs from making progress, leading to a denial of service.
Deadlocks/hangs (DoS)	The attacker brings the system to a state where no progress can be made for some part of the software, e.g., due to a deadlock (DoS).

Table 11.1: Known attack methods / threats to security for modern operating systems

closely together, accessing a bit in one row may cause the neighbouring bit in the adjacent row to leak a small amount of charge onto its capacitor—even though that bit is in a completely different page in memory. By repeatedly accessing the row at high frequency ('hammering'), the interference accumulates so that, in some cases, the neighbouring bit may flip. We do not know in advance which, if any, of the bits in a row will flip, but once a bit flips, it will flip again if we repeat the experiment. If attackers succeed in flipping bits in kernel memory, they enable attacks similar to those based on software-based memory corruption. For instance, corrupting page tables to gain access to the memory of other domains.

Another class of attacks is that of concurrency bugs and double fetch [990, 991]. The double fetch is an important problem for an operating system and occurs when it uses a value from userspace twice (e.g., a size value is used once to allocate a buffer and later to copy into that buffer). Security issues such as memory corruption arise if there is a race between the operating system and the attacker, and the attacker changes the userspace value in between the two accesses and makes it smaller. It is similar to a Time Of Check Time Of Use (TOCTOU) attack, except that the value modified is *used* twice.

In addition to direct attacks, adversaries may use side channels to leak information indirectly, for instance by means of cache sidechannels [992]. There are many variants, but a common one consists of attackers filling a cache set with their own data or code and then periodically accessing these addresses. If any of the accesses is significantly slower, they will know that someone else, presumably the victim, also accessed data/code that falls in the same cache set. Now assume that the victim code calls functions in a secret dependent way. For instance, an encryption routine processes a secret key bit by bit and calls function `foo` if the bit is 0, and `bar` if it is 1, where `foo` and `bar` are in different cache sets. By monitoring which cache sets are used by the side channel, the attackers quickly learn the key.

Another famous family of hardware side channels abuses speculative and out-of-order execution [993, 994]. For performance, modern CPUs may execute instructions ahead of time—before the preceding instructions have been completed. For instance, while waiting for the condition of a conditional branch to be resolved, the branch predictor may speculate that the outcome will be 'branch taken' (because that was the outcome for the last n times), and speculatively execute the instructions corresponding to the taken branch. If it turns out that it was wrong, the CPU will squash all the results of the speculatively executed instructions, so that none of the stores survive in registers or memory. However, there may still be traces of the execution in the micro-architectural state (such as the content of caches, TLBs and branch predictors that are not directly visible in the instruction set architecture). For instance, if a speculative instruction in a user program reads a sensitive and normally inaccessible byte from memory in a register and subsequently uses it as an offset in a userspace array, the array element at that offset will be in the cache, even though the value in the register is squashed as soon as the CPU discovers that it should not have allowed the access. The attacker can time the accesses to every element in the array and see if one is significantly faster (in the cache). The offset of that element will be the secret byte. In other words, the attacker can use a cache side channel to extract the data that was accessed speculatively.

More recent attacks show that the hardware vulnerabilities related to speculation and out-of-order execution may be more disastrous than we thought. The Foreshadow attack [995] abuses the fact that Intel CPUs read from the Level 1 cache under speculative execution whenever a memory page is marked as not present—without properly checking the ownership of the data at that physical address. Worse, the vulnerability known as Rogue In-Flight Data (RIDL) [996] (that attackers can exploit without privileges, even from JavaScript in browsers)

and without caring about addresses, shows that Intel CPUs constantly feed speculatively executing instructions with data from arbitrary security domains all the time, via a variety of temporary micro-architectural buffers.

Mitigating these attacks require not just changes in the hardware but also deep and often complex involvement of the operating system. For instance, the operating system may need to flush caches and buffers that could leak data, provide guarantees that no speculation takes place across certain branches, or schedule different security domains on separate cores, etc.

Besides caches, hardware side channels can use all kinds of shared resources, including TLBs, MMUs, and many other components [997]. Indeed, side channels need not be hardware related at all. For instance, memory deduplication and page caches, both implemented in the operating system, are well-known sources for side channels. Focusing on the former for illustration purposes, consider a system that aggressively deduplicates memory pages: whenever it sees two pages with the same content, it adjusts the virtual memory layout so that both virtual pages point to the same physical page. This way, it needs to keep only one of the physical pages to store the content, which it can share in a copy-on-write fashion. In that case, a write to that page takes longer (because the operating system must copy the page again and adjust its page table mappings), which can be measured by an attacker. So, if a write to page takes significantly longer, the attacker knows that some other program also has a copy of that content—a side channel that tells the attacker something about a victim's data. Researchers have shown that attackers may use such coarse-grained side channels to leak even very fine-grained secrets [998]. In many of the side channels, the issue is a lack of isolation between security domains in software *and* in hardware (e.g., there may be no or too little isolation during hardware-implemented speculative execution). It is important to realise that domain isolation issues extend to the hardware/software interface.

For confidentiality in particular, information leaks may be subtle and seemingly innocuous, and still lead to serious security problems. For instance, the physical or even virtual addresses of objects may not look like very sensitive information, until we take into account code reuse [999] or Rowhammer [989] attacks that abuse knowledge of the addresses to divert control flow to specific addresses or flip specific bits.

As for the origin of the attacks, they may be launched from local code running natively on the victim's machine in user space, (malicious) operating system extensions, scripting code fetched across the network and executed locally (such as JavaScript in a browser), malicious peripherals, or even remote systems (where attackers launch their exploit across the network). Clearly, a remote attack is harder to carry out than a local one.

In some cases, we explicitly extend the attacker model to include malicious operating systems or malicious hypervisors as well. These attackers may be relevant in cloud-based systems, where the cloud provider is not trusted, or in cases where the operating system itself has been compromised. In these cases, the goal is to protect the sensitive application (or a fragment thereof), possibly running in special hardware-protected trusted execution environments or enclaves, from the kernel or hypervisor.

A useful metric for estimating the security of a system is the *attack surface* [1000]—all the different points that an attacker can reach and get data to or from in order to try and compromise the system. For instance, for native code running locally, the attack surface includes all the system calls the attacker can execute as well as the system call's arguments and return values, together with all the code implementing the system calls, which the attacker can reach. For remote attackers, the attack surface includes the network device drivers, part of the

network stack, and all the application code handling the request. For malicious devices, the attack surface may include all the memory the device may access using DMA or the code and hardware functions with which the device may interact. Note, however, that the exposure of more code to attackers is only a proxy metric, as the quality of the code differs. In an extreme case, the system is formally verified so that a wide range of common vulnerabilities are no longer possible.

11.2 THE ROLE OF OPERATING SYSTEMS AND THEIR DESIGN IN SECURITY

[985, c1,c7c9][1001, c1]

At a high level, operating systems and hypervisors are tasked with managing the resources of a computer system to guarantee a foundation on which it is possible to build secure applications with respect to confidentiality, integrity and availability.

The main role of these lowest layers of the software stack with respect to security is to provide isolation of security domains and mediation of all operations that may violate the isolation. In the ideal case, the operating system shields any individual process from all other processes. For instance, peripheral processes should not be able to access the memory allocated to the primary process, learn anything about the activities related to that primary process except those which the process chooses to reveal, or prevent the process from using its allocated resources, such as CPU time indefinitely. Some operating systems may even regulate the information flows such that top secret data can never leak to processes without the appropriate clearance, or classified data cannot be modified by processes without the appropriate privilege levels.

Digging a little deeper, we can distinguish between control and data plane operations and we see that isolation in operating systems involves both. In memory isolation, the operating systems operate at the control plane when it configures the MMU (memory management unit), which is then responsible for the isolation without much involvement by the operating system. In most other interactions, for instance when operating on arguments of system calls provided by unprivileged security domains, an operating system operates at both planes. The lack of separation between the planes may easily lead to vulnerabilities—for instance, when the operating system decides to reuse memory pages that previously belonged to one security domain (with access isolation enforced by the MMU) in another domain without properly overwriting the (possibly sensitive) data on that page.

There are many ways to design an operating system. Fig. 11.1 illustrates four extreme design choices. In Fig. 11.1(a), the operating system and the application(s) run in a single security domain and there is no isolation whatsoever. Early operating systems worked this way, but so do many embedded systems today. In this case, there is little to no isolation between the different components in the system and an application can corrupt the activities of the File System (FS), the network stack, drivers, or any other component of the system.

Fig. 11.1(b) shows the configuration of most modern general-purpose operating systems, where most of the operating system resides in a single security domain, strictly isolated from the applications, while each application is also isolated from all other applications. For instance, this is the structure of Windows, Linux, OS X and many of the descendants of the original UNIX [1002]. Since almost every component of the operating system runs in a

single security domain, the model is very efficient because the components interact simply by function calls and shared memory. The model is also safe as long as every component is benign. However, if attackers manage to compromise even a single component, such as a driver, all security is void. In general, device drivers and other operating system *extensions* (e.g., Linux Kernel Modules) are important considerations for the security of a system. Often written by third parties and more buggy than the core operating system code, extensions running in the single security domain of the operating system may compromise the security of the system completely.

Interestingly, the boundary between the kernel and other security domains in such systems is often a bit fuzzier now that operating systems can bypass the kernel for, say, high-speed networking, or implement non performance critical operating system components as user processes. Examples include the File System in User Space (FUSE) in UNIX operating systems and the User Mode Driver Framework (UMDF) in Windows. Even so, most of the operating system functionality still forms a single monolithic security domain.

Fig. 11.1(c) shows the extreme breakup in separate processes of all the components that make up the operating system in a multi-server operating system [1003, 1004]. The configuration is potentially less efficient than the previous model, because all the interactions between different components of the operating system involve Inter-Process Communication (IPC). In addition, the operating system functions as a distributed system and anyone who has ever built a distributed system knows how complicated the problems may get. However, the advantage of a multi-server system is that a compromised driver, say, cannot so easily compromise the rest of the system. Also, while from a conceptual perspective, the multi-server looks like a distributed system, a lot of the complexity of a real distributed system is due to unreliable communication and this does not exist in multi-server systems. The common view is that microkernel-based multi-server designs have security and reliability advantages over monolithic and single-domain designs, but incur somewhat higher overheads—the price of safety.

Finally, Fig. 11.1(d) shows a situation that, at first glance, resembles that of Fig. 11.1(a): on top of a minimal kernel that multiplexes the underlying resources, applications run together with a minimal 'library operating system' (libOS [1005, 1006]). The libOS contains code that is typically part of the operating system, but is directly included with the application. This configuration allows applications to tailor the operating system exactly according to their needs and leave out all the functionality they were not going to use anyway. Library operating systems were first proposed in the 1990s (e.g., in MIT's Exokernel and Cambridge's Nemesis projects). After spending a few years in relative obscurity, they are becoming popular again—especially in virtualised environments where they are commonly referred to as Unikernels [1007]. In terms of security, Unikernels are difficult to compare against, say, microkernel-based multi-server systems. On the one hand, they do not have the extreme separation of operating system components. On the other, they allow the (library) operating system code to be much smaller and less complex—it only has to satisfy the needs of this one application. Moreover, the library cannot compromise isolation: it is part of this application's trusted computing base and no other.

The debate about which design is better goes back to the famous *flame war* between Andrew S. Tanenbaum and Linus Torvalds in 1992. By that time, MINIX [1008], a small UNIX-like operating system developed by Tanenbaum, had been around for half a decade or so, and was gaining traction as an education operating system around the world—especially since Bell Labs' original UNIX was sold as a commercial product with a restrictive license prohibiting

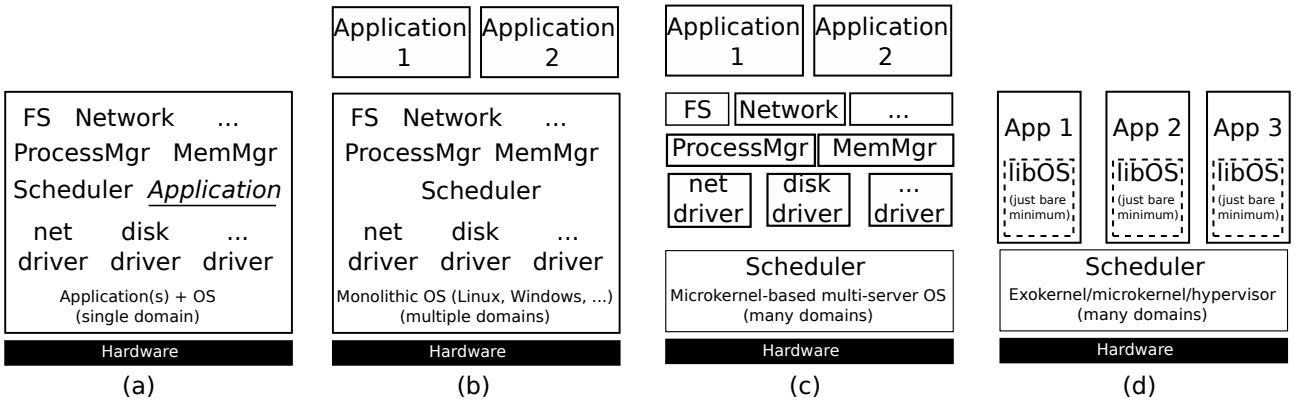


Figure 11.1: Extreme design choices for operating systems: (a) single domain (sometimes used in embedded systems), (b) monolithic OS (Linux, Windows, and many others), (c) microkernel-based multi-server OS (e.g., Minix-3) and (d) Unikernel / Library OS

users from modifying it. One of MINIX's users was Torvalds, then a Finnish student who announced a new operating system kernel in a post in the `comp.os.minix` newsgroup on Usenet. In January 1992, Tanenbaum criticised the design for its lack of portability, and also took aim at Linux's monolithic design, claiming Linux was obsolete from the outset. Torvalds responded with his own criticism of MINIX. This heated exchange contained increasingly sophisticated arguments, many of which still stand today, so much so that the question of who won the debate remains unanswered.

That said, Linux has become wildly popular and few people would consider it obsolete. It is also clear that ideas from multi-server systems such as MINIX have been incorporated into existing operating systems and hypervisor-based systems. Interestingly, at the time of writing even MINIX itself is running in hundreds of millions of Intel processors as a miniature operating system on a separate microprocessor known as the Management Engine. In addition, now that the CPUs in modern systems are increasingly elaborate System on a Chips (SoCs), the hardware itself is starting to look like a distributed system and some researchers explicitly advocate designing the operating system accordingly, with a focus on message passing rather than memory sharing for communication [1009].

The situation for virtualised environments, in general, is comparable to that of operating systems. We have already seen that in one extreme case, the entire virtual machine with the application and a stripped-down operating system can form a single domain. A more common case is to have a hypervisor at the lowest level supporting one or more operating systems such as Linux or Windows in a *virtual machine*. In other words, these hypervisors provide each of the operating systems with the illusion that they run on dedicated hardware. At the other end of the spectrum, we find the entire system decomposed into separate, relatively small, virtual machines. Indeed, some operating systems, such as QubesOS completely integrate the concepts of virtualisation and operating systems by allowing individual user processes to be isolated in their own virtual machines. Finally, as we have already seen, Unikernels are popular in virtualised environments, on top of hypervisors.

Incidentally, one of the drawbacks of virtual machines is that each operating system image uses storage and adds redundancy, as every system will think that it is the king of the hardware mountain, while in reality it is sharing resources. Moreover, each operating system in a virtual machine needs separate maintenance: updates, configuration, testing, etc. A popular alternative is, therefore, to virtualise at the operating system level. In this approach, multiple

environments, known as *containers*, run on top of a single shared operating system. The containers are isolated from each other as much as possible and have their own kernel name spaces, resource limits, etc., but ultimately share the underlying operating system kernel, and often binaries and libraries. Compared to virtual machines, containers are more lightweight. However, if we ignore the management aspects for a moment, virtual machines are often perceived as more secure than containers, as they partition resources quite strictly and share only the hypervisor as a thin layer between the hardware and the software. On the other hand, some people believe that containers are more secure than virtual machines, because they are so lightweight that we can break applications into 'microservices' with well-defined interfaces in containers. Moreover, having fewer things to keep secure reduces the attack surface overall. Early work on containers (or 'operating system level virtualisation" is found in the `chroot` call that was first added to Version 7 Unix in 1979 [1010]. In 2000, FreeBSD released Jails [1011], which went much further in operating system virtualisation. Today, we have many container implementations. A popular one is Docker [1012].

A final class of operating systems explicitly targets small and resource constrained devices such as those found in the Internet of Things (IoT). While everybody has a different opinion on what IoT means and the devices to consider range from smartphones to smart dust, there is a common understanding that the most resource constrained devices should be part of it. For such devices, even stripped down general-purpose operating systems may be too bulky and operating systems are expected to operate in just a few kilobytes. As an extreme example, popular IoT operating systems such as RIOT can be less than 10 KB in size and run on systems ranging from 8-bit microcontrollers to general-purpose 32-bit CPUs, with or without advanced features such as Memory Management Units (MMUs), etc. The abundance of features and application isolation that we demand from operating systems such as Windows and Linux may be absent in these operating systems, but instead there may be support for functionality such as real-time schedule or low-power networking which are important in many embedded systems.

Since we are interested in the security guarantees offered by the operating system, we will assume that there are multiple security domains. In the next section, we will elaborate on the advantages and disadvantages of the different designs from the viewpoint of well-established security principles. Our focus will be on the security of the design and the way in which we can stop attacks, but not before observing that there is more to security at this level. In particular, management and maintainability of the system—with respect to updates, extensions, configuration, etc.—play an important role.

11.3 OPERATING SYSTEM SECURITY PRINCIPLES AND MODELS

[985, c9][1013, c4,c7][8][1014]

Since operating systems (and/or hypervisors) are the foundation upon which rests the security of all higher-level components in the system, it is common to hear their designs debated in terms of security principles such as those of Saltzer and Schroeder (see Table 11.2), and security models such as the Bell-LaPadula [1015] and Biba [1016] access models—the topic of the next few subsections. While Saltzer and Schroeder's security principles are arguably the most well-known, we should mention that others have since added to the list. For instance, important additions that we discuss in this text include the Principle of Minimising the Amount

Principle of...
Economy of mechanism
Fail-safe defaults
Complete mediation
Open design
Separation of privilege
Least privilege / least authority
Least common mechanism
Psychological acceptability

Table 11.2: Saltzer & Schroeder's security principles [8].

of Trusted Code (the Trusted Computing Base) and the Principle of Intentional Use [1017].

11.3.1 Security principles in operating systems

From a security perspective, the walls between different security domains should be as high and as thick as possible—perfect isolation. Any interaction between domains should be subject to rigorous mediation, following the Principle of Complete Mediation, and security domains should have as few mechanisms (especially those involving a shared state such as global variables) in common as possible, adhering to the Principle of Least Common Mechanism. For instance, given a choice between adding a shared procedure with global variables to the operating system kernel and making it available in a user-space library that behaves in an isolated manner for each process, we should choose the latter option, assuming it does not increase the code size too much or violate any other principles or constraints. Moreover, mediation should follow the Principle of Fail-Safe Defaults: the policy for deciding whether domains can access the resources of other domains should be: 'No, unless'. In other words, only explicitly authorised domains should have access to a resource. The principles of Least Common Mechanism and Economy of Mechanism also suggest that we should minimise the amount of code that should be trusted, the *Trusted Computing Base (TCB)*. Since studies have shown that even good programmers introduce between 1 and 6 bugs per 1000 lines of code, assuming the complexity of the code is similar, a small TCB translates to fewer bugs, a smaller attack surface and a better chance of automatically or manually verifying the correctness of the TCB with respect to a formal specification.

With respect to the designs in Fig. 11.1, we note that if there is a single domain, the TCB comprises all the software in the system, including the applications. All mechanisms are 'common' and there is virtually no concept of fail-safe defaults or rigorously enforced mediation. For the monolithic OS design, the situation is a little better, as at least the operating system is shielded from the applications and the applications from each other. However, the operating system itself is still a single security domain, inheriting the disadvantages of Fig. 11.1(a). The extreme decomposition of the multi-server operating system is more amenable to enforcing security: we may enforce mediation between individual operating components in a minimal-size microkernel with fail-safe defaults. Much of the code that is in the operating system's security domain in the other designs, such as driver code, is no longer part of the TCB. Unikernels are an interesting alternative approach: in principle, the operating system code and the application run in a single domain, but the libOS code is as small as possible (Economy of Mechanism) and the mechanism common to different applications is minimised. Resource partitioning can also be mediated completely at the Unikernel level. For a Unikernel application, the TCB consists only of the underlying hypervisor/Exokernel and the OS components it decides to

use. Moreover, the library implementing the OS component is only in this application's TCB, as it is not shared by others.

Another principle, that of Open Design, is perhaps more controversial. In particular, there have been endless discussions about open source (which is one way to adhere to the principle) versus closed source and their merits and demerits with respect to security. The advantage of an open design is that anybody can study it, increasing the probability of finding bugs in general and vulnerabilities in particular[3]. A similar observation was made by Auguste Kerckhoffs about crypto systems and is often translated as that one should not rely on *security by obscurity*. After all, the obscurity is unlikely to last forever and when the bad people find a vulnerability before the good people do, you may have a real problem. The counter argument is that with an open design, the probability of them finding the bug is higher.

In contrast, there is little doubt that a design with a strict decomposition is more in line with the Principle of Least Privilege and the Principle of Privilege Separation than one where most of the code runs in a single security domain. Specifically, a monolithic system has no true separation of privileges of the different operating system components and the operating system always runs with all privileges. In other words, the operating system code responsible for obtaining the process identifier of the current process runs with the power to modify page tables, create root accounts, modify any file on disk, read and write arbitrary network packets, and crash the entire system at any time it sees fit. Multi-server systems are very different and may restrict what calls individual operating system components can make, limiting their powers to just those privileges they need to complete their job, adhering to the Principle Of Least Authority (POLA) with different components having different privileges (Principle of Privilege Separation). Unikernels offer a different and interesting possibility for dealing with this problem. While most of the components run in a single domain (no privilege separation or POLA), the operating system is stripped down to just the parts needed to run the application, and the Unikernel itself could run with just the privileges required for this purpose.

Of course, however important security may be, the Principle of Psychological Acceptability says that in the end the system should still be usable. Given the complexity of operating system security, this is not trivial. While security hardened operating systems such as SELinux and QubesOS offer clear security advantages over many other operating systems, few ordinary users use them and even fewer feel confident to configure the security settings themselves.

11.3.2 Security models in operating systems

An important question in operating systems concerns the flow of information: who can read and write what data? Traditionally, we describe system-wide policies in so-called access control models.

For instance, the Bell-LaPadula model [1015] is a security access model to preserve the confidentiality of information, initially created for the US government. In the 1970s, the US military faced a situation where many users with different clearance levels would all be using the same mainframe computers—requiring a solution known as Multi-Level Security. How could they ensure that sensitive information would never leak to non-authorised personnel? If it adheres to the model designed by David Bell and Leonard LaPadula, a system can handle

[3] On the other hand, researchers have encountered security bugs that are years or sometimes decades old, even in security critical open source software such as OpenSSL or the Linux kernel, suggesting that the common belief that "given enough eyeballs, all bugs are shallow" (also known as Linus' Law) does not always work flawlessly.

multiple levels of sensitive information (e.g., *unclassified*, *secret*, *top secret*) and multiple clearance levels (e.g., the clearance to access *unclassified* and *secret*, but not *top secret* data) and keep control over the flow of sensitive information. Bell-LaPadula is often characterised as 'read down, write up'. In other words, a subject with clearance level *secret* may create *secret* or *top secret documents*, but not *unclassified ones*, as that would risk leaking secret information. Likewise, a user can only read documents at their own security level, or below it. Declassification, or lowering of security levels (e.g., copying data from a *top secret* to a *secret* document) can only be done explicitly by special, 'trusted' subjects. Strict enforcement of this model prevents the leakage of sensitive information to non-authorised users.

Bell-LaPadula only worries about confidentiality. In contrast, the Biba model [1016] arranges the access mode to ensure data *integrity*. Just like in Bell-LaPadula, objects and subjects have a number of levels of integrity and the model ensures that subjects at lower levels cannot modify data at higher levels. This is often characterised as 'read up, write down', the exact opposite of Bell-LaPadula.

Bell-LaPadula and Biba are access control models that the operating system applies when mediating access to resources such as data in memory or files on disk. Specifically, they are Mandatory Access Control (MAC) models, where a system-wide policy determines which users have the clearance level to read or write which specific documents, and users are not able to make information available to other users without the appropriate clearance level, no matter how convenient it would be. A less strict access control model is known as Discretionary Access Control (DAC), where users with access to an object have some say over who else has access to it. For instance, DAC may restrict access to objects based on a user or process identity or group membership. More importantly, DAC allows a user or process with access rights to an object to transfer those rights to other users or processes. Having only this group-based DAC makes it hard to control the flow of information in the system in a structured way. However, it is possible to combine DAC and MAC, by giving users and programs the freedom to transfer access rights to others, within the constraints imposed by MAC policies.

For completeness, we also mention Role-Based Access Control (RBAC) [1018], which restricts access to objects on the basis of roles which may be based on job functions. While intuitively simple, RBAC allows one to implement both DAC and MAC access control policies.

11.4 PRIMITIVES FOR ISOLATION AND MEDIATION

[985, c9][1019, c1-c9],[1014][1013, c4,c7][1020]

In the 1960s, Multics [1021] became the first major operating system designed from the ground up with security in mind. While it never became very popular, many of its security innovations can still be found in the most popular operating systems today. Even if some features were not invented directly by the Multics team, their integration in a single, working, security-oriented OS design was still novel. Multics offered rings of protection, virtual memory, segment-based protection, a hierarchical file system with support for Discretionary Access Control (DAC) *and* mandatory access control (MAC). Indeed, in many ways, the mandatory access control in Multics, added at the request of the military, is a direct software implementation of the Bell-LaPadula security model. Finally, Multics made sure that its many small software components were strongly encapsulated, accessible only via their published interfaces where mediation took place.

If any of this sounds familiar, this is not surprising, as Jerome Saltzer was one of the Multics

team leaders. The Trusted Computer System Evaluation Criteria (TCSEC), better known as the famous *Orange Book* [1014], describes requirements for evaluating the security of a computer system, and is strongly based on Multics. There is no doubt that Multics was very advanced and perhaps ahead even of some modern operating systems, but this was also its downfall—the system became so big and so complex that it arguably violated the Principle of Psychological Acceptability for at least some of its developers. Frustrated, Ken Thomson and Dennis Ritchie decided to write a new and much simpler operating system. As a pun and to contrast it with Multics, they called it 'Unics', later spelt UNIX. Like all major general purpose operating systems in use today, it relied on a small number of core primitives to isolate its different security domains.

So what are these major isolation primitives? First, the operating system has to have some way of authenticating users and security domains so it can decide whether or not they may access certain resources. To isolate the different security domains, the operating system also needs support for access control to objects, such as files. In addition, it needs memory protection to ensure that a security domain cannot simply read data from another domain's memory. Finally, it needs a way to distinguish between privileged code and non-privileged code, so that only the privileged code can configure the desired isolation at the lowest level and guarantee mediation for all operations.

11.4.1 Authentication and identification

Since authentication is the topic of the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14), we will just observe that to determine access rights, an operating system needs to authenticate its users and that there are many ways to do so. Traditionally, only usernames and passwords were used for this purpose, but more and more systems nowadays use other methods (such as smartcards, fingerprints, iris scans, or face recognition)—either instead of passwords or as an additional factor. Multi-factor authentication makes it harder for attackers to masquerade as a legitimate user, especially if the factors are of a different nature, e.g., something you know (like a password), something you own (like a smartcard), and something you 'are' (biometric data such as fingerprints).

For every user thus authenticated, the operating system maintains a unique user id. Moreover, it may also keep other information about users such as in which groups they reside (e.g., student, faculty, and/or administrator). Similarly, most operating systems attach some identity to each of the processes running on behalf of the user and track the ownership and access rights of the files they use. For instance, it gives every running process a unique process id and also registers the id of the users on whose behalf it runs (and thus the groups in which the user resides). Finally, it tracks which user owns the executing binary. Note that the user *owning* the binary and the user *running* the binary need not be the same. For instance, the administrator can create and own a collection of system programs that other users may execute but not modify.

Incidentally, storing credentials in a secure manner is crucial. Several modern operating systems resort to hardware to protect such sensitive data. For instance, they may use a Trusted Platform Module (TPM) to ensure credentials such as disk encryption keys are cryptographically sealed, or employ a separate VM for the credential store, so that even a compromised VM will not get direct access to the credentials.

11.4.2 Access control lists

Given these identities, the operating system is equipped to reason about which user and which process is allowed to perform which operations on a specific object: access control.

When Robert Daley and Peter Neumann first developed the Multics file system, they introduced an Access Control List (ACL) for every block of data in the system [1021, 1022]. Conceptually, an ACL is a table containing users and data blocks that specifies for each data block which users have which kind of access rights. Most modern operating systems have adopted some variant of ACLs, typically for the file system[4]. Let us look at an example. On UNIX-based systems [1002], the default access control is very simple. Every file is owned by a user and a group. Moreover, every user can be in one or more groups. For instance, on a Linux system, user `herbertb` is in nine different groups:

```
herbertb@nordkapp:~$ groups herbertb
herbertb : herbertb adm cdrom sudo dip plugdev lpadmin sambashare cybok
herbertb@nordkapp:~$
```

Per file, a small number of permission bits indicates the access rights for the owning user, the owning group, and everyone else. For instance, let us look at the ACL for a file called `myscript` on Linux:

```
herbertb@nordkapp:~/tmp$ getfacl myscript
# file: home/herbertb/tmp/myscript
# owner: herbertb
# group: cybok
user::rwx
group::rwx
other::r-x
```

We see that `myscript` is owned by user `herbertb` and group `cybok`. The owning user and all users in group `cybok` have permissions to **r**ead, **w**rite, and e**x**ecute the file, while all other users can **r**ead and e**x**ecute (but not write) it.

These basic UNIX file permissions are quite simple, but modern systems (such as Linux and Windows) also allow for more extensive ACLs (e.g., with explicit access rights for multiple users or groups). Whenever someone attempts to read, write or access a file, the operating system verifies whether the appropriate access rights are in the ACL. Moreover, the access control policy in UNIX is typically discretionary, because the owning user is allowed to set these rights for others. For instance, on the above Linux system, user `herbertb` can himself decide to make the file `myscript` writable by all the users (‘`chmod o+w myscript`’).

Besides DAC, Multics also implemented MAC and, while it took a long time to reach this stage, this is now also true for many of the operating systems that took their inspiration from Multics (namely most popular operating systems today). Linux even offers a framework to allow all sorts of access control solutions to be plugged in, by means of so-called ‘reference monitors’ that vet each attempt to execute a security sensitive operation and, indeed, several MAC solutions exist. The best known one is probably Security-Enhanced Linux (SELinux [1023]), a set of Linux patches for security originally developed by the National Security Agency (NSA) in the US and derived from the Flux Advanced Security Kernel (FLASK) [1024].

SELinux gives users and processes a context of three strings: `(username, role, domain)`. While the tuple already (correctly) suggests that SELinux also supports RBAC, there is significant flexibility in what to use and what to avoid. For instance, many deployed systems use only the `domain` string for MAC and set `username` and `role` to the same value for all users. Besides processes, resources such as files, network ports, and hardware resources also have

[4] which in most cases also follows the hierarchical design pioneered in Multics.

such SELinux contexts associated with them. Given this configuration, system administrators may define system-wide policies for access control on their systems. For instance, they may define simply which domains a process have to perform specific operations (read, write, execute, connect) on a resource, but policies may also be much more complicated, with multiple levels of security and strict enforcement of information flow *a la* Bell-LaPadula, Biba, or some custom access-control model.

Mandatory access control in systems such as SELinux revolves around a single system-wide policy that is set by a central administrator and does not change. They do not allow untrusted processes to define and update their own information control policy. In contrast, research operating systems such as Asbestos [1025], HiStar [1026] and Flume [1027] provide exactly that: distributed information flow control. In other words, any process can create security labels and classify and declassify data.

11.4.3 Capabilities

So far, we have assumed that access control is implemented by means of an ACL or access matrix, where all information is kept to decide whether a process P may access a resource R. After authenticating the users and/or looking up their roles or clearance levels, the reference monitor decides whether or not to grant access. However, this is not the only way. A popular alternative is known as *capability* and, stemming from 1966, is almost as old.

In 1966, Jack Dennis and Earl Van Horn, researchers at MIT, proposed the use of capabilities for access control in an operating system [1028]. Unlike ACLs, capabilities do not require a per-object administration with the exact details of who is allowed to perform what operation. Instead, the users present a capability that in itself proves that the requested access is permitted. According to Dennis and Van Horn, a capability should have a unique identifier of the object to which it pertains, as well as the set of access rights provided by the capability. Most textbooks use the intuitive definition by Henry Levy that a capability is a 'token, ticket, or key that gives the possessor permission to access an entity or object in a computer system' [1019]. Possession of the capability grants all the rights specified in it and whenever a process wants to perform an operation on an object, it should present the appropriate capability. Conversely, users do not have access to any resources other than the ones for which they have capabilities.

Moreover, Peter Neumann argues that the access control should be explicit and adhere to the Principle of Intentional Use [1017], by explicitly authorising only what is really intended, rather than something that is overly broad or merely expedient. Adherence to the principle helps to avoid the accidental or unintended use of rights that may lead to security violations in the form of a 'confused deputy' (in which a security domain unintentionally exercises a privilege that it holds legitimately, on behalf of another domain that does not and should not).

Of course, it should be impossible to forge capabilities, lest users give themselves arbitrary access to any object they want. Thus, an operating system should either store the capability in a safe place (for instance, the kernel), or protect it from forgery using dedicated hardware or software-based encryption. For instance, in the former case, the operating system may store a process' capabilities in a table in protected memory, and whenever the process wants to perform an operation on a file, say, it will provide a reference to the capability (e.g., a file descriptor) and never touch the capability directly. In the latter case, the capability may be handled by the process itself, but any attempt to modify it in an inappropriate manner will be detected [1029].

Capabilities are very flexible and allow for convenient delegation policies. For instance, given full ownership of a capability, a process may pass it on to another process, to give that process either the same access rights, or, alternatively, a subset of those rights. Thus, discretionary access control is easy. On the other hand, in some situations, it may not be desirable to have capabilities copied and spread arbitrarily. For this reason, most capability-based systems add a few bits to the capabilities to indicate such restrictions: whether copying is permitted, whether the capability's lifetime should be limited to a procedure invocation, etc.

Comparing ACLs and capabilities, we further observe that ACLs are typically based on users ('the user with id x is allowed to read and write'), while capabilities can be extremely fine-grained. For instance, we may use different capabilities for sending and receiving data. Following the Principle of Least Authority, running every process with the full power of the user, compared to running a process with just the power of the capabilities it acquires, is less secure. Running with the authority of the user who started the program, as is often the case in modern operating systems, is known as a form of *ambient authority* and much more likely to violate the Principle of Least Authority than fine-grained capabilities that equip a process only with the privileges it needs. Moreover, capabilities do not even allow a process to name an object unless it has the appropriate capability, while ACLs should permit the naming of any object by everyone, as the access check only occurs when the process attempts the operation. Finally, ACLs may become very large with growing numbers of users, access rights, and objects.

On the other hand, revoking a particular access right for a particular user in an ACL is easy: just remove a permission in the appropriate table entry. With capabilities, the process is more involved. After all, we may not even know which users/processes have the capability. Adding a level of indirection may help somewhat. For instance, we could make the capabilities point to an indirect object, which in turn points to the real object. To invalidate the capability (for *all* users/processes) the operating system could then invalidate that indirect object. But what to do if we only want to revoke the capability in a subset of processes? While there are solutions, revocation of capabilities remains the most difficult part.

Since the 1960s, many capability-based systems have appeared—initially all supported in hardware [1019] and typically more rigid than the more general capabilities discussed so far. The first was the MIT PDP-1 Timesharing System [1030], followed shortly after by the Chicago Magic Number Machine at the University of Chicago [1031], a very ambitious project with hardware-supported capabilities, which, as is not uncommon for ambitious projects, was never completed. However, it did have a great impact on subsequent work, as Maurice Wilkes of the University of Cambridge learned about capabilities during several visits to Chicago and wrote about it in his book on time-sharing systems. Back in the UK, this book was picked up by an engineer at Plessey which built the fully functional Plessey System 250 (with explicit hardware support for capability-based addressing). Maurice Wilkes himself went on to build the Cambridge CAP computer together with Roger Needham and David Wheeler [1032]. CAP was the first computer to demonstrate the use of secure capabilities. This machine ensured that a process could only access a memory segment or other hardware if it possessed the required capabilities. Another noteworthy capability-based system of that time was CMU's Hydra [1033]—which added explicit support for restricting the use of capabilities or operations on objects (allowing one to specify, for instance, that capabilities must not survive a procedure invocation). Finally, in the 1980s, the Amoeba distributed operating systems explored the use of cryptographically protected capabilities that could be stored and handled by user processes [1029].

Nowadays, many major operating systems also have at least some support for capabilities.

For instance, the L4 microkernel, which is present in many mobile devices today, embraced capability-based security in the version by Gernot Heiser's group at NICTA in 2008[5]. A formally verified kernel called seL4 [1034] from the same group similarly relies on capabilities for access control to resources. In 1997, Linux adopted very limited capability support (sometimes referred to as 'POSIX capabilities'), but this was different from the capabilities defined by Dennis and Van Horn (with less support for copying and transferring capabilities). For instance, Linux capabilities refer only to operations, not objects. Recognising that UNIX file descriptors and Windows handles are almost capabilities already, an interesting effort to merge capabilities and UNIX APIs is the Capsicum project [1035] by the University of Cambridge and Google, where the capabilities are extensions of UNIX file descriptors. FreeBSD adopted Capsicum in version 9.0 in 2012. An outgrowth of Capsicum is Capability Hardware Enhanced RISC Instructions (CHERI), a hardware-software project that transposes the Capsicum capability model into the CPU architecture.

11.4.4 Physical access and secure deletion

It is important to observe that access restrictions at the level of the operating system do not necessarily translate to the same restrictions at the physical level. For instance, the operating system may 'delete' a file simply by removing the corresponding metadata that makes it appear as a file (e.g., when listing the directory), without really removing the *content* of the file on disk. Thus, an attacker that reads raw disk blocks with no regard for the file system may still be able to access the data.

It turns out that securely deleting data on disk is not trivial. Naive deletion, for instance, by overwriting the original content with zeros, is not always sufficient. For instance, on some magnetic disks, data on the disk's tracks leaves (magnetic) traces in areas close to the tracks and a clever attack with sufficient time and resources may use these to recover the content. Moreover, the operating system may have made copies of the file that are not immediately visible to the user, for instance, as a backup or in a cache. All these copies need to be securely deleted. The situation for Solid State Drives (SSDs) is no better, as SSDs have their own firmware that decides what to (over)write and when, beyond the control of the OS. For most operating systems, truly secure deletion, in general, is beyond the operating system's capabilities and we will not discuss it further in this knowledge area, except to say that full disk encryption, a common feature of modern operating systems, helps a lot to prevent file recovery after deletion.

11.4.5 Memory protection and address spaces

Access control is only meaningful if security domains are otherwise isolated from each other. For this, we need separation of the security domains' data according to access rights and a privileged entity that is able to grant or revoke such access rights. We will look at the isolation first and talk about the privileges later, when we introduce protection rings.

A process should not normally be able to read another process' data without going through the appropriate access control check. Multics and nearly all of the operating systems that followed (such as UNIX and Windows) isolate information in processes by giving each process (a) its own processor state (registers, program counter etc.) and (b) its own subset of memory. Whenever the operating system decides to execute process P_2 at the expense of the currently

[5]Other L4 variants, such as the L4 Fiasco kernel from Dresden, also supported capabilities.

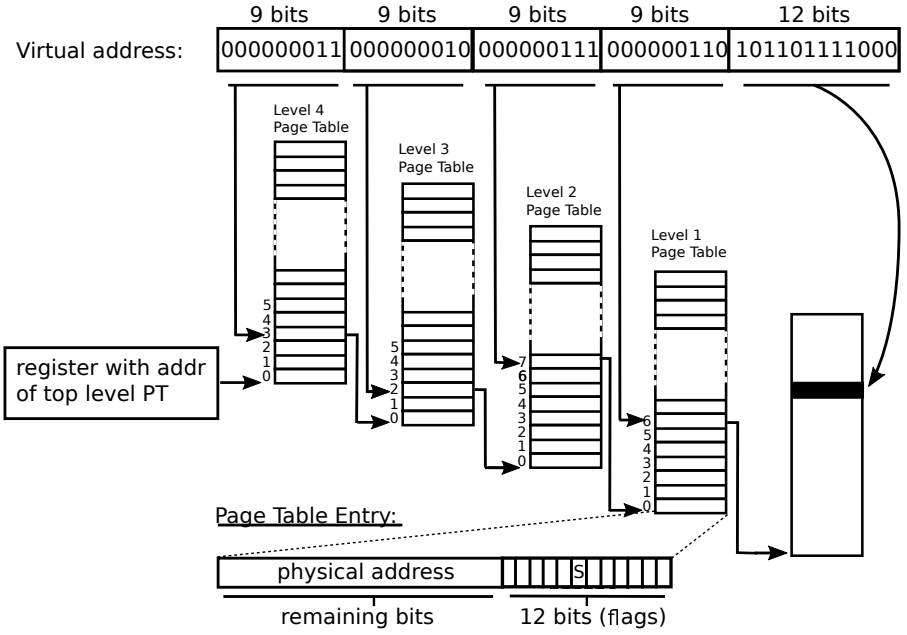


Figure 11.2: Address translation in modern processors. The MMU 'walks' the page tables to find the physical address of the page. Only if a page is 'mapped' on a process' page tables can the process address it, assuming it is present and the process has the appropriate access rights. Specifically, a user process cannot access the page for which the supervisor (*S*) bit is set in the page table entry.

running process P_1 (a so-called context switch), it first stops P_1 and saves all of its processor state in memory in an area inaccessible to other processes. Next, it loads P_2's processor states from memory into the CPU, adjusts the bookkeeping that determines which parts of the physical memory are accessible, and starts executing P_2 at the address indicated by the program counter that it just loaded as part of the processor state. Since user processes cannot directly manipulate the bookkeeping themselves, $P2$ cannot access any of P_1's data in a non-mediated form.

Most modern operating systems keep track of the memory bookkeeping by means of *page tables*, as illustrated in Fig. 11.2. For each process, they maintain a set of page tables (often containing multiple levels organised as a directed acyclic graph[6]), and store a pointer to the top level page table in a register that is part of the processor state and that must be saved and restored on a context switch.

The main use for the page table structure is to give every process its own *virtual* address space, ranging from address 0 to some maximum address (e.g., 2^{48}), even though the amount of physical memory may be much less [1036, 1037, 1038]. Since two processes may both store data at address 0x10000, say, but should not be allowed to access each others' data, there has to be a mapping from the virtual addresses each process uses to the physical addresses used by the hardware. It is like a game of basketball, where each side may have a player with the number 23, but that number is mapped onto a different physical player for each team.

This is where the page tables comes in. We divide each of the virtual address spaces into fixed size pages and use the page table structure to map the address of the first byte of a virtual page onto a physical address. The processor often uses multiple levels of translation.

[6]While it is often helpful to think of page table structures as trees, different branches may point to the same leave nodes.

In the example in Fig. 11.2, it uses the first nine bits of the virtual address as an index in the top level page table (indicated by a control register that is part of the processor state) to find an entry containing the physical address of the next level page table, which is indexed by the next nine bits, and so on, until we reach the last level page table, which contains the physical address of the physical page that contains the virtual address. The last 12 bits of the virtual address are simply the offset in this page and point to the data.

Paging allows the (total) size of the virtual address spaces of the processes to be much larger than the physical memory available in the system. First, a process typically does not use all of its possibly gigantic address space and only virtual pages that are in actual use need backing by physical pages. Second, if a process needs more memory to store some data and no physical pages are free at that moment (for instance, because they are already in use by other processes, or they are backing some other virtual pages of this process), the operating system may swap the content of these pages to disk and then re-use the physical page to store the new data.

A key consequence of this organisation is that a process can only access data in memory if there is a mapping for it in its page tables. Whether this is the case, is controlled by the operating system, which is, therefore, able to decide exactly what memory should be private and what memory should be shared and with whom. The protection itself is enforced by specialised hardware known as the memory management unit (MMU[7]). If the mapping of virtual to physical for a specific address is not in the small but very fast cache known as the Transaction Lookaside Buffer (TLB), the MMU will look for it by walking the page tables and then triggering an interrupt if the page containing the address is not mapped.

The MMU will also trigger interrupts if the page is currently not in memory (swapped to disk), or, more relevant to security, if the user does not have the required privilege to access this memory. Specifically, the last 12 bits of the Page Table Entry (PTE) contain a set of flags and one of these flags, the *S* bit in Fig. 11.2, indicates whether this is a page for supervisor code (say, the operating system running at the highest privilege) or for ordinary user processes. We will have more to say about privileges later.

Page tables are the main way modern operating systems control access to memory. However, some (mostly older) operating systems additionally use another trick: *segmentation*. Not surprisingly, one of the earliest operating systems using both segmentation and paging was Multics [1021, 1038]. Unlike pages, segments have an arbitrary length and start at an arbitrary address. However, both depend on hardware support: an MMU. For instance, processors such as Intel's 32 bits x86 have a set of dedicated registers known as segment selectors: one for code, one for data, etc. Each segment has specific permissions, such as read, write, or execute. Given a virtual address, the MMU uses the current value in the corresponding segment selector as an index in a so-called descriptor table. The entry in the descriptor table contains the start address and length of the segment, as well as protection bits to prevent code without the required privilege level to access it. In case there is only segmentation and no paging, the resulting address is the original virtual address added to the start of the segment and that will be the physical address, and we are done. However, both the GE-645 mainframe computer used for Multics and the more modern x86-32 allow one to combine segmentation and paging. In that case, the virtual address is first translated into a so-called *linear address* using the segment descriptor table and that linear address is then translated into a physical address using the page table structure.

[7] As we shall see later, not all processors have a full-fledged MMU but rather a simpler memory protection unit.

This is as complicated as it sounds; none of the popular modern operating systems still use segmentation. The best known examples of operating systems using segmentation were OS/2 (an ill-fated collaboration between Microsoft and IBM that started in the mid-1980s and that never caught on) and IBM's AS/400 (also launched in the 1980s[8] and still running happily today on a mainframe near you). The Xen hypervisor also used segmentation on 32 bit x86, but on 64 bit systems this was no longer possible. In fact, the 64-bit version of the Intel x86 no longer even supports full segmentation, although some vestiges of its functionality remain. On the other hand, complicated multi-level address translation is still quite common in virtualised environments. Here, the hypervisor tries to give virtual machines the illusion that they are running all by themselves on real hardware, so the MMU translates a virtual address first to what is known as a *guest physical address* (using page tables). However, this is not a real physical address yet, as many virtual machines may have the same idea of using, say, physical address 0x10000. So, instead, the MMU uses a second translation stage (using what Intel refers to as extended page tables, maintained by the hypervisor) to translate the guest physical address to a host physical address ('machine address').

11.4.6 Modern hardware extensions for memory protection

Also, while segmentation is mostly dead, there are many other forms of hardware support for memory protection beyond paging. For instance, many machines have had support for buffer bounds checking and some date back a quarter of a century or more. To illustrate the corresponding primitives, however, we will look at what is available in modern general purpose processors, focusing mostly on the Intel x86 family. The point here is not whether we think this processor is more important or even that feature X or Y will be very important in the future (which is debatable and hard to predict), but rather to illustrate that this is still a very active area for hardware development today.

As a first example, consider the somewhat ill-fated Intel Memory Protection Extensions (MPX) that enhance Intel's workhorse processors with functionality to ensure that array pointers cannot stray beyond the array boundaries (stopping vulnerabilities such as buffer overflows from being exploited). For this purpose, a small set of new registers can store the lower and upper bounds of a small number of arrays, while prior to de-referencing the pointer, new MPX instructions check the value of the array pointer for boundary violations. Even in systems that use MPX only in userspace, the operating system plays a role, for instance, to handle the exception that the hardware throws when it encounters a buffer boundary violation. MPX was heavily criticised for having too few of these bounds registers, leading to much performance overhead. In addition, MPX does not support multi-threading, which may result in data races in legacy code. One might say that MPX is a good example of an attempt by a hardware vendor to add new features for memory safety to their CPUs that is unfortunately not always successful.

More recently, Intel added Memory Protection Keys (MPKs) to its processors[9]. Intel MPK allows one to set four previously unused bits in the PTE (Fig. 11.2) to one of 16 'key' values. In addition, it adds a new 32-bit register containing 2 bits for each key to indicate whether reading and writing is allowed for pages tagged with that key. MPK allows developers to partition the memory in a small number (in this case 16) protection domain and, for instance, allow only a specific crypto library to access cryptographic keys. While unprivileged user processes may

[8] Or even the 1970s, if you want to count the System/38.

[9] Again, Intel was actually late to the party, as similar features existed in a variety of processors since the 1960s.

update the value of the register, only privileged operating system code can tag the memory pages with keys.

Some processor designs support even more advanced memory protection in the form of what, using ARM terminology, we will refer to as memory tagging extensions (MTE[10])[1040]. The idea is simple yet powerful. The processor assigns every aligned chunk of memory (where a chunk is, say, 16 bytes) a so-called "tag" in hardware. Similarly, every pointer also obtains a tag. Tags are generally not very large, say 4 bits, so they can be stored in the top-most byte of the 64-bit pointer value which we do not really use anyway (in fact, ARM supports a `top-byte-ignore` feature that makes the hardware explicitly mask out the top most byte). Whenever the program allocates N bytes of memory, the allocator rounds up the allocation to multiples of 16 bytes and assigns a random tag to it. It also assigns the same tag to the pointer to the memory. From now on, dereferencing the pointer is only permitted if the tag in the pointer matches that of the memory to which it refers—effectively stopping most spatial and temporal memory errors.

Meanwhile, some processors, especially in low-power devices, do not even have a full-fledged MMU at all. Instead, they have a much simpler Memory Protection Unit (MPU) which serves only to protect memory, in a way that resembles the MPK functionality discussed above. In MPU designs, the operating systems define a number of memory regions with specific memory access permissions and memory attributes. For instance, the MPU on ARMv8-M processors supports up to 16 regions. Meanwhile, the MPU monitors all the processor's memory accesses (including instruction fetches and data accesses) and triggers an exception on detecting an access violation.

Note that in the above, we have assumed that the operating system needs protection from untrusted user applications. A special situation arises when the operating itself is not trusted. Perhaps you are running a security-sensitive application on a compromised operating system, or in the cloud, where you are not sure you want to trust the cloud provider. In the general case, you may want to protect your data and applications without trusting any other software. For this purpose, processors may offer hardware support for running extremely sensitive code in a secure, isolated environment, known as a trusted execution environment in ARM's 'TrustZone' or an enclave in Intel's Software Guard Extension (SGX). They offer slightly different primitives. For instance, the code running in an SGX enclave is intended to be a part of a normal user process. The memory it uses is always encrypted as soon as it leaves the processor. Moreover, SGX offers hardware support to perform attestation, so that a (possibly remote) party can verify that the code is running in an enclave and that it is the right code. ARM TrustZone, on the other hand, isolates the 'normal world' that runs the normal operating system and user applications, from a 'secure world' that typically runs its own, smaller operating system as a well as a small number of security sensitive applications. Code in the normal world can call code in the secure world in a way that resembles the way applications call into an operating system. One interesting application of special environments such as ARM TrustZone (or Intel's SMM mode, discussed later) is to use it for runtime monitoring of the integrity of a regular operating system—hopefully detecting whatever stealthy malware or rootkit compromised it before it can do some serious damage. Although aspects of these trusted environments clearly overlap with operating system security, we consider them mostly beyond the scope of this knowledge area. We should also note that in recent years, the security offered by hardware trusted execution environments has been repeatedly pierced by a variety of side channels [995, 996, 1041] that leak information from supposedly secure world.

[10] A similar feature on SPARC processors is known as Application Data Integrity (ADI)[1039]

Switching gears again, it may be the case that the operating system is fine, but the hardware is not. Malicious or faulty hardware may use the system's Direct Memory Access (DMA) to read or overwrite sensitive data in memory that should be inaccessible to them. Moreover, with some standards (such as Thunderbolt over USB-C), a computer's PCIe links may be directly exposed to devices that a user plugs into a computer. Unfortunately for the user, it is hard to be sure that what looks like, say, a display cable or power adapter, does not also contain some malicious circuitry designed to compromise the computer [1042]. As a partial remedy, most architectures nowadays come with a special MMU for data transferred to and from devices. This hardware, called an IOMMU, serves to map device virtual addresses to physical addresses, mimicking exactly the page-based protection illustrated in Fig. 11.2, but now for DMA devices. In other words, devices may access a virtual memory address, which the IOMMU translates to an actual physical address, checks for permissions and stops if the page is not mapped in for the device, or the protection bits do not match the requested access. While doing so provides some measure of protection against malicious devices (or indeed drivers), it is important to realise that the IOMMU was designed to facilitate virtualisation and really should not be seen as a proper security solution. There are many things that may go wrong [1043]. For instance, perhaps the administrator wants to revoke a device's access rights to a memory page. Since updating the IOMMU page tables is a slow operation, it is not uncommon for operating systems to delay this operation and batch it with other operations. The result is that there may be a small window of time during which the device still has access to the memory page even though it appears that these rights have already been revoked.

Finally, we can observe that the increasing number of transistors per surface area enables a CPU vendor to place more and more hardware extensions onto their chips, and the ones discussed above are by no means the only security-related ones in modern processors. Additional examples include cryptographic units, memory encryption, instructions to switch extended page tables efficiently, and pointer authentication (where the hardware detects modification of pointer values). There is no doubt that more features will emerge in future generations and operating systems will have to adapt in order to use them in a meaningful way. A broader view of these issues is found in the Hardware Security Knowledge Area (Chapter 20).

11.4.7 Protection rings

Among the most revolutionary ideas introduced by Multics was the notion of *protection rings*—a hierarchical layering of privilege where the inner ring (ring 0) is the most privileged and the outer ring is the least privileged [1021]. Accordingly, untrusted user processes execute in the outer ring, while the trusted and privileged kernel that interacts directly with the hardware executes in ring 0, and the other rings could be used for more or less privileged system processes.

Protection rings typically assume hardware support, something most general purpose processors offer today, although the number of rings may differ. For instance, the Honeywell 6180 supported as many as eight rings, Intel's x86 four, ARM v7 three (plus an extra one for TrustZone) and PowerPC two. However, as we shall see, the story becomes slightly confusing, because some modern processors have also introduced more and different processor modes. For now, we simply observe that most regular operating systems use only two rings: one for the operating system and one for the user processes.

Whenever less privileged code needs a function that requires more privileges, it 'calls into' the lower ring to request the execution of this function as a service. Thus, only trusted, privileged

code may execute the most sensitive instructions or manipulate the most sensitive data. Unless a process with fewer privileges tricks more privileged code into doing something that it should not be doing (as a confused deputy), the rings provide powerful protection. The original idea in Multics was that transitioning between rings would occur via special *call gates* that enforce strict control and mediation. For instance, the code in the outer ring cannot make a call to just any instruction in the inner ring, but only to predefined entry points where the call is first vetted to see if it and its arguments do not violate any security policy.

While processors such as the x86 still support call gates, few operating systems use them, as they are relatively slow. Instead, user processes transition into the operating system kernel (a 'system call') by executing a software interrupt (a 'trap') which the operating system handles, or more commonly, by means of a special, highly efficient system call instruction (with names such as SYSCALL, SYSENTER, SVC, SCALL etc., depending on the architecture). Many operating systems place the arguments to the system call in a predefined set of registers. Like the call gates, the traps and system call instructions also ensure that the execution continues at a predefined address in the operating system, where the code inspects the arguments and then calls the appropriate system call function.

Besides the user process calling into the operating system, most operating systems also allow the kernel to call into the user process. For instance, UNIX-based systems support *signals* which the operating system uses to notify the user program about 'something interesting': an error, an expired timer, an interrupt, a message from another process etc. If the user process registered a handler for the signal, the operating system will stop the current execution of the process, storing all its processor states on the process' stack in a so-called signal frame, and continue execution at the signal handler. When the signal handler returns, the process executes a `sigreturn` system call that makes the operating system take over, restore the processor state that is on the stack and continue executing the process.

The boundary between security domains, such as the operating system kernel and userspace processes is a good place to check both the system calls themselves and their arguments for security violations. For instance, in capability-based operating systems, the kernel will validate the capabilities [1034], and in operating systems such as MINIX 3 [1004], specific processes are only allowed to make specific calls, so that any attempt to make a call that is not on the pre-approved list is marked as a violation. Likewise, Windows and UNIX-based operating systems have to check the arguments of many system calls. Consider, for instance, the common `read` and `write` system calls, by which a user requests the reading of data from a file or socket into a buffer, or the writing of data from a buffer into a file or socket, respectively. Before doing so, the operating system should check if the memory to write from or read into is actually owned by the process.

After executing the system call, the operating system returns control to the process. Here also, the operating system must take care not to return results that jeopordise the system's security. For instance, if a process uses the `mmap` system call to request the operating system to map more memory into its address space, the operating system should ensure that the memory pages it returns no longer contain sensitive data from another process (e.g., by initialising every byte to zero first [1044]).

Zero intialisation problems can be very subtle. For instance, compilers often introduce padding bytes for alignment purposes in data structures. Since these padding bytes are not visible at the programming language level at all, the compiler may see no reason to zero initialise them. However, a security violation occurs when the operating system returns such a data structure in response to a system call and the unitialised padding contains sensitive data

from the kernel or another process.

Incidentally, even the signalling subsystem in UNIX systems that we mentioned earlier is an interesting case for security. Recall that the `sigreturn` takes whatever processor state is on the stack and restores that. Now assume that attackers are able to corrupt the stack of the process and store a fake signal frame on the stack. If the attackers are then also able to trigger a `sigreturn`, they can set the entire processor state (with all the register values) in one fell swoop. Doing so provides a powerful primitive in the hands of a skilled attacker and is known as Sigreturn-Oriented Programming (SROP) [1045].

11.4.8 One ring to rule them all. And another. And another.

As also mentioned earlier, the situation regarding the protection rings is slightly more confusing these days, as recent CPUs offer virtualisation instructions for a hypervisor, allowing them to control the hardware accesses at ring 0. To do so, they have added what, at first sight, looks like an extra ring at the bottom. Since on x86 processors, the term 'ring 0' has become synomymous with 'operating system kernel' (and 'ring " with 'user processes'), this new hypervisor ring is commonly referred to as *'ring –1'*. It also indicates that operating systems in their respective virtual machines can keep executing ring 0 instructions natively. However, strictly speaking, it serves a very different purpose from the original rings, and while the name ring –1 has stuck, it is perhaps a bit of a misnomer.

For the sake of completeness, we should mention that things may get even more complex, as some modern processors still have other modes. For instance, x86 offers what is known as System Management Mode (SMM). When a system boots, the firmware is in control of the hardware and prepares the system for the operating system to take over. However, when SMM is enabled, the firmware regains control when a specific interrupt is sent to the CPU. For instance, the firmware can indicate that it wants to receive an interrupt whenever the power button is pressed. In that case, the regular execution stops, and the firmware takes over. It may, for instance, save the processor state, do whatever it needs to do and then resume the operating system for an orderly shutdown. In a way, SMM is sometimes seen as a level lower than the other rings (*ring –2*).

Finally, Intel even added a *ring –3* in the form of the Intel Management Engine (ME). ME is a completely autonomous system that is now in almost all of Intel's chipsets; it runs a secret and completely independent firmware on a separate microprocessor and is *always* active: during the booting process, while the machine is running, while it is asleep, and even when it is powered off. As long as the computer is *connected* to power, it is possible to communicate with the ME over the network and, say, install updates. While very powerful, its functionality is largely unknown except that it runs its own small operating system[11] which researcher found contained vulnerabilities. The additional processors that accompany the main CPU (be it the ME or related ones such as Apple's T2 and Google's Titan chips) raise an interesting point: is the operating system running on the main CPU even capable of meeting today's security requirements? At least, the trend appears to augment it with special-purpose systems (hardware and software) for security.

[11] Version 11 of the ME, at the time of writing, is based on MINIX-3.

11.4.9 Low-end devices and the IoT

Many of the features described above are found, one way or another, in most general-purpose processor architectures. However, this is not necessarily true in the IoT, or embedded systems in general, and tailored operating systems are commonly used [1046]. Simple microcontrollers typically have no MMUs, and sometimes not even MPUs, protection rings, or any of the advanced features we rely on in common operating systems. The systems are generally small (reducing attack surface) and the applications trusted (and possibly verified). Nevertheless, the embedded nature of the devices makes it hard to check or even test their security and, wherever they play a role in security sensitive activities, security by means of isolation/containment and mediation should be enforced externally, by the environment. Wider IoT issues are addressed in the Cyber-Physical Systems Security Knowledge Area (Chapter 21).

11.5 OPERATING SYSTEM HARDENING

[999, 1013, 1047, 1048]

The best way to secure operating systems and virtual machines is to have no vulnerabilities at all: security by design. For instance, we can use formal verification to ensure that certain classes of bugs cannot be present in the software or hardware, and that the system is functionally correct [1034]. Scaling the verification to very large systems is still challenging, but the field is advancing rapidly and we have now reached the stage that important components such as a microkernel, file systems and compilers have been verified against a formal specification. Moreover, it is not necessary to verify all the components of a system: guaranteeting isolation simply requires a verified microkernel/hypervisor and a few more verified components. Verification of other components may be desirable, but is not essential for isolation. Of course, the verification itself is only as good as the underlying specification. If you get that wrong, it does not matter if you have verified it, you may still be vulnerable.

Despite our best efforts, however, we have not been able to eradicate all security bugs from large, real-world systems. To guard themselves against the types of attacks described in the threats model, modern operating systems employ a variety of solutions to complement the above isolation and mediation primitives. We distinguish between five different classes of protection: information hiding, control flow restrictions, partitioning, code and data integrity checks, and anomaly detection.

11.5.1 Information hiding

One of the main lines of defense in most current operating systems consists of hiding whatever the attackers may be interested in. Specifically, by randomising the location of all relevant memory areas (in code, heap, global data and stack), attackers will not know where to divert the control flow, nor will they be able to spot which addresses contain sensitive data, etc. The term Address Space Layout Randomization (ASLR) was coined around the release of the PaX security patch, which implemented this randomisation for the Linux kernel in 2001 [1049]—see also the discussion in the Software Security Knowledge Area (Section 15.4.2). Soon, similar efforts appeared in other operating systems and the first mainstream operating systems to have ASLR enabled by default were OpenBSD in 2003 and Linux in 2005. Windows and MacOS followed in 2007. However, these early implementations only randomised the address space in user programs and randomisation did not reach the kernel of major operating systems,

under the name of Kernel ASLR (KASLR), until approximately a decade after it was enabled by default in user programs.

The idea of KASLR is simple, but there are many non-trivial design decisions to make. For instance, how random is random? In particular, what portion of the address do we randomise? Say your Linux kernel has an address range of 1GB (=2^{30}) for the code, and the code should be aligned to 2MB (=2^{21}) boundaries. The number of bits available for randomisation (the *entropy*) is $30 - 21 = 9$ bits. In other words, we need at most 512 guesses to find the kernel code. If attackers find a vulnerability to divert the kernel's control flow to a guessed address from a userspace program and each wrong guess leads to a system crash, it would suffice to have userspace access to a few hundred machines to get it right at least once with high probability (although many machines will crash in the process).

Another important decision is what to randomise. Most implementations today employ coarse-grained randomisation: they randomise the *base* location of the code, heap or stack, but within each of these areas, each element is at a fixed offset from the base. This is simple and very fast. However, once attackers manage to get hold of even a single code pointer via an information leak, they know the addresses for every instruction. The same is true, *mutatis mutandis*, for the heap, stack etc. It is no surprise that these information leaks are highly valued targets for attackers today.

Finer-grained randomisation is also possible. For instance, it is possible to randomise at the page level or the function level. If we shuffle the order of functions in a memory area, even knowing the base of the kernel code is not sufficient for an attacker. Indeed, we can go more fine-grained still, and shuffle basic blocks, instructions (possibly with junk instructions that never execute or have no effect) or even the register allocations. Many fine-grained randomisation techniques come at the cost of space and time overheads, for instance, due to reduced locality and fragmentation.

Besides the code, fine-grained randomisation is also possible for data. For instance, research has shown that heap allocations, globals and even variables on the stack can be scattered around memory. Of course, doing so will incur a cost in terms of performance and memory.

Considering KASLR, and especially coarse-grained KASLR, as our first line of defense against memory error exploits would not be far off the mark. Unfortunately, it is also a very weak defense. Numerous publications have shown that KASLR can be broken fairly easily, by leaking data and/or code pointers from memory, side channels, etc.

11.5.2 Control-flow restrictions

An orthogonal line of defense is to regulate the operating system's control flow. By ensuring that attackers cannot divert control to code of their choosing, we make it much harder to exploit memory errors, even if we do not remove them. The best example is known as Control-Flow Integrity (CFI) [1048], which is now supported by many compiler toolchains (such as LLVM and Microsoft's Visual Studio) and incorporated in the Windows kernel under the name of Control Flow Guard as of 2017 — see also the Software Security Knowledge Area (Chapter 15).

Conceptually, CFI is really simple: we ensure that the control flow in the code always follows the static control flow graph. For instance, a function's return instruction should only be allowed to return to its callsite, and an indirect call using a function pointer in C, or a virtual function in C++, should only be able to target the entry point of the legitimate functions that it should be able to call. To implement this protection, we can label all the legitimate targets

for an indirect control transfer instruction (returns, indirect calls and indirect jumps) and add these labels to a set that is specific for this instruction. At runtime, we check whether the control transfer the instruction is about to make is to a target that is in the set. If not, CFI raises an alarm and/or crashes the program.

Like ASLR, CFI comes in many flavours, from coarse-grained to fine-grained, and from context sensitive to context insensitive. And just like in ASLR, most implementations today employ only the simplest, most coarse-grained protection. Coarse-grained CFI means relaxing the rules a little, in the interest of performance. For instance, rather than restricting a function's return instruction to target-only legitimate call sites that could have called this function, it may target *any* call site. While less secure than fine-grained CFI [1050], it still restricts the attackers' wiggle room tremendously, and has a much faster runtime check.

On modern machines, some forms of CFI are (or will be) even supported by hardware. For instance, Intel Control-Flow Enforcement Technology (CET) supports shadow stacks and indirect branch tracking to help enforce the integrity of returns and forward-edge control transfers (in a very coarse-grained way), respectively. Not to be outdone, ARM provides pointer authentication to prevent illegitimate modification of pointer values—essentially by using the upper bits of a pointer to store a Pointer Authentication Code (PAC), which functions like a cryptographic signature on the pointer value (and unless you get the PAC right, your pointer is not valid).

Unfortunately, CFI only helps against attacks that change the control flow—by corrupting control data such as return addresses, function pointers and jump targets—but is powerless against non-control data attacks. For instance, it cannot stop a memory corruption that overwrites the privilege level of the current process and sets it to 'root' (e.g., by setting the effective user id to that of the root user). However, if restrictions on the control flow are such a success in practice, you may wonder if similar restrictions are also possible on data flow. Indeed they are, which is called Data-Flow Integrity (DFI) [1051]. In DFI, we determine statically for each load instruction (i.e., an instruction that reads from memory) which store instructions may legitimately have produced the data, and we label these instructions and save these labels in a set. At runtime we remember, for each byte in memory, the label of the last store to that location. When we encounter a load instruction, we check if the last store to that address is in the set of legitimate stores, and if not, we raise an alarm. Unlike CFI, DFI has not been widely adopted in practice, presumably because of the significant performance overheads.

11.5.3 Partitioning.

Besides the structural decomposition of a system in different security domains (e.g, into processes and the kernel) protected by isolation primitives with or without hardware support, there are many additional techniques that operating systems employ to make it harder for attackers to compromise the TCB. In this section, we discuss the most prominent ones.

W⊕X memory. To prevent code injection attacks, whereby the attackers transfer control to a sequence of instructions they have stored in memory areas that are not meant to contain code such as the stack or the heap, operating systems today draw a hard line between code and data [1047]. Every page of memory is either executable (code pages) or writable, but not both at the same time. The policy, frequently referred to as W⊕X ('write xor execute'), prevents the execution of instructions in the data area, but also the modification of existing code. In the absence of code injection, attackers interested in diverting the of the program are forced to reuse code that is already present. Similar mechanisms are used to make sensitive

data in the kernel (such as the system call table, the interrupt vector table, etc.) read-only after initialisation. All major operating systems support this mechanism, typically relying on hardware support (the NX bit in modern processors[12])—even if the details differ slightly, and the name may vary from operating system to operating system. For instance, Microsoft refers to its implementation by the name Data Execution Prevention (DEP). *Preventing the kernel from accessing userspace.* We have already seen that operating systems use the CPU's protection rings to ensure that user processes cannot access arbitrary data or execute code in the operating system, in accordance with the security principles by Saltzer & Schroeder, which prescribe that all such accesses be mediated. However, sometimes we also need to protect the other direction and prevent the kernel from blindly accessing (or worse, executing) things in userspace.

To see why this may be bad, consider an operating system where the kernel is mapped into every process' address space and whenever it executes a system call, it executes the kernel code using the process' page tables. This is how Linux worked from its inception in 1991 until December 2017. The reason is that doing so is efficient, as there is no need to switch page tables when executing a system call, while the kernel can efficiently access all the memory. Also since the kernel pages have the supervisor (*S*) bit set, there is no risk that the user process will access the kernel memory. However, suppose the kernel has a bug that causes it to de-reference a function pointer that under specific circumstances happens to be NULL. The most likely thing to happen is that the kernel crashes. After all, the kernel is trying to execute code on a page that is not valid. But what if a malicious process deliberately maps a page at address 0, and fills it with code that changes the privileges of the current process to that of root? In that case, the kernel will execute the code, with kernel privileges. This is bad.

It should now be clear that the kernel should probably not blindly execute process code. Nor should it read blindly from user data. After all, an attacker could use it to feed malicious data to the kernel instructions. To prevent such accesses, we need even more isolation than that provided by the default rings. For this reason, many CPUs today provide Supervisor Mode Execution Protection (SMEP) and Supervisor Mode Access Protection (SMAP)[13]. SMEP and SMAP are enabled by setting the appropriate bits in a control register. As soon as they are on, any attempt to access or transfer control to user memory will result in a page fault. Of course, this also means that SMAP should be turned off explicitly whenever the kernel *needs* to access user memory.

Some operating systems, including Linux, got SMEP-like restrictions 'for free' on systems vulnerable to the Meltdown vulnerability in 2017 [993], which forced them to adopt an alternative design, which came with a price tag. In particular, they were forced to abandon the single address space (where the kernel executes in the address space of the process), because of the Meltdown out-of-order execution side channel from Table 11.1. To recap, the Meltdown (and related Spectre) attacks consist of attackers abusing the CPU's (over-)optimism about what happens in the instructions it executes out-of-order or speculatively. For instance, it wrongly assumes that load instructions have the privilege to read the data they access, the outcome of a branch is the same as the previous time a branch at a similar address was executed, or the data needed for a load instruction is probably the data in this temporary CPU buffer that was just written. However, even if any of these assumptions are wrong, the CPU can recover by squashing the results of the code that was executed out-of-order or speculatively.

[12] NX (no execute) is how AMD originally called the feature in its x86 compatible CPUs. Intel calls it Execute Disable (XD) and ARM Execute Never (XN).

[13] Again, this is x86 terminology. On ARM similar features are called Privileged Access Never (PAN) and Privileged Execute Never (PXN).

In a Meltdown-like attack, the attackers' process executes an out-of-order instruction to read a byte at a (supposedly inaccessible) kernel address, and the CPU optimistically assumes all is well and simply accesses the byte. Before the CPU realises things are *not* well after all and this byte should not be accessible, the attackers have already used the byte to read a particular element in a large array in their own process' address space . Although the CPU will eventually squash all the results, the damage is already done: even though the byte cannot be read directly, the index of the array element that is in the cache (and is, therefore, measurably faster to access than the other elements) must be the kernel byte.

To remedy this problem on somewhat older processors that do not have a hardware fix for this vulnerability, operating systems such as Linux use a design that completely separates the page tables of the kernel from those of the processes. In other words, the kernel also runs in its own address space, and any attempt by an out-of-order instruction to read a kernel address will fail. The kernel can still map in the pages of the user process and thus access them if needed, but the permissions can be different. Specifically, if they are mapped in as non-executable, we basically get SMEP functionality for free.

For other vulnerabilities based on speculative execution (such as Spectre and RIDL), the fix is more problematic. Often, multiple different spot solutions are used to patch the most serious issues. For instance, after a bounds check that could be influenced by untrusted users, we may want to insert special instructions to stop speculation completely. Likewise, operating systems such as Windows try to "gang schedule" only code that belongs to the same security domain on the same core (so that leaking from on thread to another on the same core is less of an issue), while others such as OpenBSD disable hyperthreading altogether on Intel processors. However, it is unclear how complete the set of patches will be, while we are waiting for the hardware to be fixed.

Partitioning micro-architectural states Sophisticated side channel attacks build on the aggressive resource sharing in modern computer systems. Multiple security domains share the same cache, the same TLB, the same branch predictor state, the same arithmetic units, etc. Sharing is good for efficiency, but, as indicated by the Principle of Least Common Mechanism, they also give rise to side channels. To prevent such attacks, operating systems may need to sacrifice some of the efficiency and partition resources even at fine granularity. For instance, by means of *page colouring* in software or hardware-based cache allocation technology, an operating system may give different processes access to wholly disjointed portions of the cache (e.g., separating the cache sets or separating the ways within a cache set). Unfortunately, partitioning is not always straightforward and currently not supported for many low-level resources.

11.5.4 Code and data integrity checks

One way to reduce the exploitability of code in an operating system, is to ensure that the code and/or data is unmodified and provided by a trusted vendor. For instance, for many years Windows has embraced *driver signing*. Some newer versions have taken this a step further and use a combination of hardware and software security features to lock a machine down, ensuring that it runs only trusted code/apps—a process referred to by Microsoft as 'Device Guard'. Even privileged malware cannot easily get non-authorised apps to run, as the machinery to check whether to allow an app to run sits in a hardware-assisted virtualised environment. Most code signing solutions associate digital signatures associated with the operating system extensions allow the operating system to check whether the code's integrity

is intact and the vendor is legitimate. A similar process is popularly used for updates.

However, what about the code that checks the signature and, indeed, the operating system itself—are we sure that this has not been tampered with by a malicious bootkit? Ensuring the integrity of the system software that is loaded during the booting involves a number of steps, mostly related to the multiple steps in the boot process itself. From the earliest commercial computers onward, booting involved multiple stages. Even the IBM 701, a popular computer in the early 1950s with as many as 19 installations, already had such a multi-stage booting procedure that started with pressing a special 'Load' button to make the system load a single 36-bit word from, typically, a punched card. It would execute (part of) this word to load even more instructions, and then start executing these instructions as the "boot program".

In general, securely booting devices starts with an initial 'root of trust' which initiates the booting process and is typically based in hardware, for instance, a microcontroller that starts executing software from internal, immutable memory, or from internal flash memory that cannot be reprogrammed at all, or only with strict authentication and authorisation checks. As an example, modern Apple computers use a separate processor, the T2 Security Chip, to provide the hardware root of trust for secure boot among other things, while Google has also developed a custom processor for this called the Titan. We will now discuss how a hardware-root of trust helps to verify that a system booted securely.

Booting general-purpose computers typically starts with the firmware which initiates a sequence of stages that ends with a fully booted system. For instance, the firmware may load a special *bootloader* program which then loads the operating system kernel which in turn may load additional boot drivers until finally the operating system is fully initialised and ready to interact with the user or applications. All of these stages need protection. For instance,the Unified Extensible Firmware Interface (UEFI) can protect the first stage (i.e., verify the integrity of the bootloader), by means of Secure Boot. Secure boot verifies whether the boot loaders were signed with the appropriate key, i.e., using keys that agree with the key information that is stored in the firmware. This will prevent loaders and drivers without the appropriate signatures from gaining control of the system. The bootloader can now verify the digital signature of the operating system kernel before loading it. Next, the kernel verifies all other components of the operating system (such as boot drivers and possibly integrated anti-malware software) before starting them. By starting the anti-malware program before other drivers, it can subsequently check all these later components, and extend the chain of trust to a fully initialised operating system.

The next problem is: how do we *know* that this is the case? In other words, how do we know that the system really did boot securely and we can trust whatever is displayed on the screen? The trick here is to use attestation, whereby a (remote) party can detect any changes that have been made to our system. Remote attestation typically uses special hardware such as a Trusted Platform Module (TPM) that serves as a root of trust and consists of verifying, in steps, whether the system was loaded with the 'right' kind of software. In particular, a TPM is a cryptograhic hardware module that supports a range of cryptographic functions, key generation and management, secure storage (e.g., for keys and other security-sensitive information), and importantly, integrity measurements. See the Hardware Security Knowledge Area (Chapter 20) for further discussion.

For the integrity measurements, TPMs have a set of Platform Configuration Registers called PCR-0, PCR-1, ..., that are set to a known value on every boot. These registers are not for writing to directly, but rather for *extending*. So, if the current value of the PCR-0 register is X and we want to extend it with Y, the TPM calculates $hash(X, Y)$ and stores the outcome in

PCR-0. Now, if we want to extend it further, say with Z, the TPM again calculates the hash of Z and the value currently in PCR-0 and stores the outcome in PCR-0. In other words, it will calculate $hash(Z, hash(X, Y))$. We can now extend this further and create an arbitrarily long "hash chain".

The values in the PCRs can serve as *evidence* that the system is in a trustworthy state. Specifically, the first code that executes when you boot your system is firmware boot code that is sometimes referred to as the Core Root of Trust for Measurements (CRTM) or BIOS boot block. This code will 'measure' the full firmware by generating a hash of its content which it sends to the TPM to extend PCR-0, before it starts executing it. Next, the firmware that is now executing will measure the next component of the boot process and again store the value in a PCR of the TPM (e.g., by extending PCR-0), before executing it. After a number of these stages, the PCR register(s) contain a hash chain of all steps that the system took to boot. A remote party can now verify whether the system booted securely by asking the TPM for a 'quote': a report of a set of PCR values currently in PCRs (together with a nonce supplied by the remote party), that is signed with the TPM's private Attestation Identity Key that never leaves the TPM (and derives from a hardcoded key that was created at manufacturing time). As the public key is well-known, anyone can verify that the quote came from the TPM. Upon receiving the quote and after verifying that it came from the TPM and that it was fresh, the remote party knows that the booting process could only have followed the steps that created these hashes in the PCRs. If they correspond to the hashes of known and trusted code, the remote party knows that the system booted securely.

Code and data integrity checking may well continue at runtime. For instance, the hypervisor may provide functionality to perform introspection of its virtual machines: is the code still the same, do the data structures still make sense? This technique is known as Virtual Machine Introspection (VMI). The VMI functionality may reside in the hypervisor itself, although it could be in a separate application. Besides the code, common things to check in VMI solutions include the process list (is any rootkit trying to hide?), the system call table (is anybody hijacking specific system calls?), the interrupt vector table, etc.

11.5.5 Anomaly detection

A monitor, be it in the hypervisor or in the operating system, can also be used to monitor the system for unusual events—anomaly detection [1052]. For instance, a system that crashes hundreds of times in a row could be under attack by someone who is trying to break the system's address space layout randomisation. Of course, there is no hard evidence and just because an anomaly occurred does not mean there is an attack. Anomaly detection systems must strike a balance between raising too many false alarms, which are costly to process, and raising too few, which means it missed an actual attack.

11.6 OPERATING SYSTEMS, HYPERVISORS—WHAT ABOUT RELATED AREAS?

[1013, c4,c7]

The problems that we encounter at the operating system and hypervisor levels resurface in other systems areas and the solutions are sometimes similar. In this section, we briefly discuss databases as an example of how operating system security principles, issues and solutions are applied to other domains [1053]. Security in database systems follows similar principles as those in operating systems with authentication, privileges, access control and so on as prime concerns. The same is true for access control, where many databases offer discretionary access control by default, and role-based and mandatory access control for stricter control to more sensitive data. Representing each user as a security domain, the questions we need to answer concern, for instance, the user's privileges, the operations that should be logged for auditing, and the resource limits such as disk quota, CPU processing time, etc. A user's privileges consist of the right to connect to the database, create tables, insert rows in tables, or retrieve information from other users' tables, and so on. Note that sometimes users who do not have access to a database except by means of a specific SQL query may craft malicious inputs to elevate their privileges in so-called SQL injection attacks [1054].

While database-level access control limits who gets access to which elements of a database, it does not prevent accesses at the operating system level to the data on disk. For this reason, many databases support transparent data encryption of sensitive table columns on disk—often storing the encryption keys in a module outside the database. In an extreme case, the data in the database may be encrypted while only the clients hold the keys.

Querying such encrypted data is not trivial [454]. While sophisticated cryptographic solutions (such as homomorphic encryption) exist, they are quite expensive and simpler solutions are commonly used. For instance, sometimes it is sufficient to store the hash of a credit card number, say, instead of the actual number and then query the database for the hash. Of course, in that case, only exact matches are possible—as we cannot query to see if the value in the database is greater than, smaller than, or similar to some other value (nor are aggregated values such as averages or sums possible). The problem of querying encrypted databases is an active field of research and beyond the scope of this Knowledge Area.

While security and access control in regular databases is non-trivial already, things get even more complex in the case of Outsourced Databases (ODBs), where organisations outsource their data management to external service providers [1055]. Specifically, the data owner creates and updates the data at an external database provider, which then deals with the client's queries. In addition to our earlier concerns about confidentiality and encryption, questions that arise concern the amount of trust to place in the provider. Can the data owner or the querying client trust the provider to provide data that was created by the original data owner (authenticity), unmodified (integrity), and fresh results to the queries? Conceptually, it is possible to guarantee integrity and authenticity by means of signatures. For instance, the data owner may sign entire tables, rows/records in a table, or even individual attributes in a row, depending on the desired granularity and overhead. More advanced solutions based on authenticated data structures are also commonly advocated, such as Merkle hash trees. In Merkle hash trees, originally used to distribute authenticated public keys, leaf nodes in the tree contain a hash of their data value (the database record), each non-leaf node contains a hash of the hashes of its children, and the root node's hash is signed and published. All that is

needed to verify if a value in a leaf node is indeed part of the original signed hash tree is the hashes of the intermediate nodes, which the client can quickly verify with a number of hashes proportional to the logarithm of the size of the tree. Of course, range queries and aggregation are more involved and researchers have proposed much more complex schemes than Merkle hash trees, but these are beyond the scope of this knowledge area. The take-away message is that with some effort we can guarantee authenticity, integrity and freshness, even in ODBs.

11.7 EMBRACING SECURITY

[985, c9][1013, c1-c21]

Increasingly advanced attacks are leading to increasingly advanced defenses. Interestingly, many of these innovations in security do not originally come from the operating system vendors or large open source kernel teams, but rather 'from the outside'—sometimes academic researchers, but in the case of operating system security, also often from independent groups such as *GRSecurity* and *the PaX Team*. For instance, the PaX Team introduced ASLR as early as 2001, played a pioneering role in making data areas non-executable and executable sections non-writable, as well as in ensuring the kernel cannot access/execute user memory. Surprisingly, where you might think that the major operating systems would embrace these innovations enthusiastically, the opposite is often true and security measures are adopted inconsistently.

The main reason is that nothing is free and a slow-down or increase in power consumption because of a security measure is not very popular. The Linux kernel developers in particular have been accused of being obsessed with performance and having too little regard for security. However, when the situation is sufficiently pressing, there is no other way than to deal with the problem, even if it is costly. In operating systems, this performance versus security trade-off has become increasingly important. Research often focuses on methods that significantly raise the bar for attackers, at an acceptable overhead.

CONCLUSION

In this Knowledge Area, we addressed security issues at the lowest levels of the software stack: the operating system and the hypervisor. Operating system / hypervisor security involves both the security *of* the operating system / hypervisor and the security guarantees offered *by* the operating system / hypervisor. As the most privileged components, operating systems and hypervisors play a critical role in making systems (in)secure. Unfortunately, the attack surface of a modern operating system or hypervisor is often large and threats of increasing sophistication involving both software and hardware call for increasingly powerful defenses also involving software and hardware. Starting from security principles and fundamentals, we showed that the system's security is influenced by the design of the system (e.g., in the isolation of security domains), and the available security primitives and mechanisms to enforce the principles (e.g., memory isolation, capabilities, protection rings). Many of the principles of operating system design are useful across many application domains and are commonly applied in other areas, such as database management systems. As with most domains, we saw that design decisions at the operating system/hypervisor level are a trade-off between security and performance—a balancing act that often slows down the adoption of security measures.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[985]	[1013]	[984]	[986]	[982]	[1001]	[8]	[1014]	[1019]	[1020]
11.1 Attacker model	c9		c1-c9	[986]	[982]	c1				
11.2 OS design and security	c1-c12				[982]	c1	[8]	[1014]		[1020]
11.3 Principles and models	c9	c4,c7					[8]	[1014]		
11.4 primitives	c3,c9	c4,c7					[8]	[1014]	c1-c9	[1020]
11.5 OS hardening	c9		c1-c9	[986]						
11.6 Security in databases		c4,c6		[986]						
11.7 Embracing Security	c9	c1-c21		[986]						

Chapter 12
Distributed Systems Security

Neeraj Suri | Lancaster University

INTRODUCTION

A distributed system is typically a composition of geo-dispersed resources (computing and communication) that collectively (a) provides services that link dispersed data producers and consumers, (b) provides on-demand, highly reliable, highly available, and consistent resource access, often using replication schemas to handle resource failures, and (c) enables a collective aggregated capability (computational or services) from the distributed resources to provide (an illusion of) a logically centralised/coordinated resource or service.

Expanding on the above, the distributed resources are typically dispersed (for example, in an Azure or Amazon Cloud, in Peer-to-Peer Systems such as Gnutella or BitTorrent, or in a Blockchain implementation such as Bitcoin or Ethereum) to provide various features to the users. These include geo-proximate and low-latency access to computing elements, high-bandwidth and high-performance resource access, and especially highly-available uninterrupted services in the case of resource failure or deliberate breaches. The overall technical needs in a distributed system consequently relate to the orchestration of the distributed resources such that the user can transparently access the enhanced services arising from the distribution of resources without having to deal with the technical mechanisms providing the varied forms of distributed resource and service orchestrations.

To support these functionalities, a distributed system commonly entails a progression of four elements. These include (a) data flows across the collection of authorised inputs (regulated via Access/Admission Control), (b) transportation of the data to/across the distributed resources (Data Transport functionality), (c) a resource coordination schema (Coordination Services), and (d) property based (e.g., time or event based ordering, consensus, virtualisation) data management to support the desired applications such as transactions, databases, storage, control, and computing.

Consequently, distributed systems security addresses the threats arising from the exploitation of vulnerabilities in the attack surfaces created across the resource structure and functionalities of the distributed system. This covers the risks to the data flows that can compromise the integrity of the distributed system's resources/structure, access control mechanisms (for resource and data accesses), the data transport mechanisms, the middleware resource coordination services characterising the distributed system model (replication, failure handling, transactional processing, and data consistency), and finally the distributed applications based on them (e.g., web services, storage, databases and ledgers).

This Knowledge Area first introduces the different classes of distributed systems categorising them into two broad categories of decentralised distributed systems (without central coordination) and the coordinated resource/services type of distributed systems. Subsequently, each of these distributed system categories is expounded for the conceptual mechanisms providing their characteristic functionalities prior to discussing the security issues pertinent to these systems. As security breaches in a distributed system typically arise from breaches in the elements related to distribution (dispersion, access, communication, coordination, etc.), the KA emphasises the conceptual underpinnings of how distributed systems function. The better one understands how functionality is distributed, the better one can understand how systems can be compromised and how to mitigate the breaches. The KA also discusses some technology aspects as appropriate along with providing references for following up the topics in greater depth.

CONTENT

12.1 CLASSES OF DISTRIBUTED SYSTEMS AND VULNERABILITIES

[1056, c2][1057, c5][1058, c18]

12.1.1 Classes of Distributed Systems

A diversity of viewpoints, models, and deployments exist for characterising distributed systems. These include defining a distributed system at the level of the aggregation of physical resources (e.g., Peer to Peer or Cloud systems), defining it at the Middleware level (e.g., Publish-Subscribe, distributed object platforms, or Web services), or defining it in terms of the services a distributed system provides (e.g., Databases or Ledgers). While a spectrum of definitions exists in literature, distributed systems can be broadly classified by the coordination schema linking the resources or by the specification of the services utilising them. One broad class is of *decentralised control* where the individual resources primarily interact with their "neighbouring" resources. The other broad category links the distributed resources via communication processes, such as message passing, to realise varied forms of *virtual centralised/coordinated control*. Thus, based on such communication and coordination models, distributed systems can be categorised into the following two broad classes.

1. *Decentralised point-to-point interactions across distributed entities without a centralised coordination service*: Peer to Peer (P2P) systems represent this class of distributed systems. Decentralised un-timed control is a prominent characteristic of such systems. For example, systems such as Kademlia, Napster, Gnutella, and many other distributed file and music sharing/storage systems, wireless sensor networks as well as online gaming systems fall in this category.
2. *Coordinated clustering across distributed resources and services*: This is a broad class that is best understood when sub-divided into two coordination sub-classes, namely (a) the coordination of resources and (b) the coordination of services. We will utilise these two coordination abstractions throughout this chapter. The spectrum of distributed systems includes Client-Server models, n-Tier Multi-tenancy Models, elastic on-demand geo-dispersed aggregation of resources (Clouds – public, private, hybrid, multi-Cloud, Big Data services, High Performance Computing), and transactional services such as Databases, Ledgers, Storage Systems, or Key Value Store (KVS). The Google File System, Amazon Web Services, Azure, and Apache Cassandra are simple examples of this class. While this class may appear to be both broad and diverse, the coordination abstraction (for either resources or services) directly characterises the type of distributed system into these two sub-classes. In both cases, these systems are typically coordinated via communication exchanges and coordination services with the intended outcome of providing a "virtually centralised system" where properties such as causality, ordering of tasks, replication handling, and consistency are ensured. There are discrete definitions in the literature for Client-Server systems, Cloud Computing, Mobile Computing, Distributed Databases, etc., though the provisioning of virtual "centralised/coordinated" behaviour is a common characteristic across them.

Notes: There are many nuances of security in distributed systems. One viewpoint focuses on the concepts and mechanisms to provide security in a distributed system where the resources and services are dispersed. The other viewpoint considers using distribution as a means of providing security, e.g., the dispersal of keys versus a centralised key store or the use of Virtual Machines (VMs) to partition and isolate resources and applications. This KA focuses on the former category of "security in a distributed system". However, it also discusses the latter viewpoints given that the dispersed security mechanisms typically execute on dispersed resources logically resulting in the need for the above mentioned classes of Decentralised or Coordinated clustering.

It is worth highlighting that a distributed system architecture is often an aggregation of multiple layers where each layer builds upon the services provided by the layer below and coordinated services offered across the distribution. At the lowest level, resources within a particular device (memory, computation, storage, communication) are accessed through the Operating System primitives provided on that device. Distributed services e.g., naming, time synchronisation, distributed file systems are assembled through the interaction of different components and services running on individual devices. Higher layers build upon the lower layers and services to provide additional functionalities and applications. Interactions across the different components of the distributed system at each level are provided by middleware frameworks that support many different communication styles: message passing, Remote Procedure Calls (RPCs), distributed object platforms, publish-subscribe architectures, enterprise service bus. Distributed applications are thus realised in a layered (or tiered) fashion through the interactions and coordination of distributed components and services. Within these architectures, decentralisation and coordination at each layer may differ resulting in hybrid compositions of decentralisation and coordination patterns. We refer the reader to the Operating Systems & Virtualisation Knowledge Area (Chapter 11) for issues concerning access to basic resources and the books [1013, 1058, 1059, 1060, 1061] for further reading on distributed systems architectures and middleware.

12.1.2 Classes of Vulnerabilities & Threats

Vulnerabilities refer to design or operational weaknesses that allow a system to be potentially compromised by an attacker. Analogously, a threat reflects the potential or likelihood of an attacker causing damage or compromising the system. Furthermore, security is an end-to-end systems property. Consequently, the vulnerabilities of a distributed system are broadly grouped based on the functional blocks therein defining the distributed system. Logically, these functional blocks and their operations also constitute the threat/attack surface for the systems where an attacker/adversary can exploit a vulnerability to compromise the system. At a high level, the attack surface relates to the compromises of the physical resources, the communication schema, the coordination mechanisms, the provided services themselves, and the usage policies on the data underlying the services.

The following outlines the general functionalities that will be progressively detailed in the subsequent sections as relevant to the specific distributed system model.

12.1.2.1 Access/Admission Control & ID Management

Access or Admission control determines the authorised participation of a resource, a user, or a service within a distributed system. This can include the sourcing of data and the access rights to read/write and use data over the lifetime of a service. The potential threats and consequent attacks include masquerading or spoofing of identity to gain access rights to the data. They can also involve Denial of Service (DoS) attacks that detrimentally limit access (e.g., depletion of computing resources and communication channels) leading to the inaccessibility and unavailability of the distributed resources/services. It is worth emphasising that resource distribution often entails more points for access control, and also more information transported in the system to support access control thus increasing the attack surface of the system (see the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14) for a discussion of authentication and authorisation in distributed systems).

A distributed system entity (resource, service, user, or data element) participates in a distributed system with a physical or logical identity. The identity, statically or dynamically allocated, can be a resource identifier such as an ID name or a number[1]. Here, authorisation may be specified in terms of the user and/or resource identity including the use of login names and passwords. Thus, an activity that involves tampering with the identity constitutes a likely threat.

12.1.2.2 Data Transportation

The network level threats span routing, message passing, the publish-subscribe modalities of resource interaction, event based response triggering, and threats across the middleware stack. Moreover, these can be passive (eavesdropping) or active attacks (data modification). A typical example is the Man In the Middle (attack) (MITM) attack where the attacker inserts itself between the victim's browser and the web server to establish two separate connections between them. This enables the attacker to actively record all messages and selectively modify data without triggering a suspicious activity alarm if the system does not enforce endpoint authentication. We refer the reader to [1059, 1060] for detailed coverage of these topics, and to the Network Security Knowledge Area (Chapter 19).

[1] [1013] provides an excellent discourse on naming issues in Chapter 6.

12.1.2.3 Resource Management and Coordination Services

This critical group encompasses the spectrum of threats to the mechanisms (typically middleware protocols) that provide the coordination of resources. This includes, among others, the aspects of synchronisation, replication management, view changes, time/event ordering, linearisability, consensus, and transactional commit.

12.1.2.4 Data Security

As a distributed system essentially operates on data (at rest or in motion) over the facets of data-sourcing, data-distribution, data-storage, or data-usage in services, the classical CIA (Confidentiality, Integrity and Availability) properties directly apply to each element (and interfaces) of this data chain. The threats to confidentiality include information leakage threats such as Side Channel Attacks or Covert Channel Attacks. Any delay or denial of data access constitutes a threat to Availability. Integrity aspects concern any compromise of data correctness such as the violation of data consistency as observed by the distributed participants. This includes the different types of consistency (strong, weak, relaxed, eventual, etc.) over storage and transactional services. Consequently, addressing the security of the data elements of a distributed system requires consideration of the threats mentioned above across resources, access control, data transportation, and coordination services as well as data threats in the form of malicious applications, code, and viruses (see the Malware & Attack Technologies Knowledge Area (Chapter 6)).

Section Organisation Based on this overview, the subsequent sections progressively outline the security approaches for distributed systems as split into the above mentioned classes of decentralised and coordination based systems. In order to understand the security issues relevant to each class, the sections also provide a basic overview of the underlying distributed system concepts along with pointers for further reading. Section 12.2 presents the commonly used models for decentralised P2P systems. Section 12.3 then elaborates the corresponding security threats for the P2P systems. This is followed by the exposition of coordinated distributed system models in Section 12.4, and by a discussion of the corresponding security aspects in Section 12.5.

12.2 DISTRIBUTED SYSTEMS: DECENTRALISED P2P MODELS

[1062, c11-12][1057, c25]

Peer to Peer (P2P) systems constitute a decentralised variant of distributed systems. Their popularity is driven by the characteristic P2P features of scalability, decentralised coordination, and low cost. Scalability implies that no changes to the protocol design are needed with increasing numbers of peers. Whereas a Client-Server architecture typically entails increasing back-end (Server) resources with increasing numbers of (Client) requests, this is not the case in P2P systems due to their inherent decentralised architecture. Furthermore, the decentralised P2P system designs promote inherent resilience against individual peer failures or other disruptions. The peer population itself represents the service provisioning infrastructure

of the system. Thereby, potential service consumers are required to partake in resource provisioning avoiding the need for dedicated data centres. Over the past two decades, a multitude of P2P models have emerged. Regardless of their specific realisation, they usually combine the following five principles: (1) symmetry of interfaces as peers can take interchangeable duties as both servers and clients, (2) resilience to perturbations in the underlying communication network substrate and to peer failures, (3) data and service survivability through replication schemes, (4) usage of peer resources at the network's edge, imposing potentially low infrastructure costs and fostering scalability as well as decentralisation, and (5) address variance of resource provisioning among peers.

These five principles make P2P a vital foundation for a diverse set of applications. Originally, P2P systems were (in)famous for their support of file sharing applications such as eMule or KaZaA, though their usage is now common in applications such as social networks, multimedia content distribution, online games, internet telephony services, instant messaging, the Internet of Things, Car-to-car communication, Supervisory Control and Data Acquisition (SCADA) systems, and wide area monitoring systems. As discussed in later sections, distributed ledgers also utilise some aspects of P2P operations.

P2P Protocol Categories The two major P2P paradigms are *unstructured* and *structured* systems. These system designs directly correlate with the application categories introduced in the previous section, i.e., unstructured protocols are mostly suitable for (large scale and scalable) data dissemination, whereas structured ones are usually applied for efficiency of data discovery. The emergent hybrid P2P protocol designs combine aspects from both unstructured and structured ones within an integrated P2P system.

Additionally, hierarchical P2P systems also exist. These partly contradict the conceptual P2P principle that considers all peers as *equal* in the sense of service provisioning. These hierarchical systems can be considered as layered systems, e.g., composition of multiple overlays consisting of front-end and back-end peers.

Regardless of the type of P2P system, it is important to note that the basic P2P operations are based on three elements, namely (a) identification or naming of peer nodes, (b) routing schemas across peers, and (c) discovery of peers as a function of their identifiers and routing.

In order to support the discussion of security in P2P systems, the next subsections provide an introductory level technical overview on P2P protocols. We provide a brief overview of the P2P protocol categories in regard of the overlay topology, resources discovery, and message passing. The reader is referred to [1063] for a comprehensive discussion on P2P operations.

12.2.1 Unstructured P2P Protocols

Representatives of the unstructured P2P protocol class such as Freenet[2] or Gnutella [1064, 1065] are mainly used for data dissemination applications such as censorship-free[3] communication or file sharing. While the set of peers do not have any characteristic topology linking them, their implicit topology is usually embedded within the physical communication underlay network topology and often unveils tree or mesh like sub-graphs, which allow for low latency

[2] **https://freenetproject.org/**

[3] In the sense that data and information is stored and exchanged with integrity and privacy preserving techniques to address freedom of expression and speech concerns.

message exchange, e.g., to address timeliness requirements of data dissemination applications. Tree topologies can be found, e.g., in single source streaming media data dissemination with various consumers as leaf nodes. Meshes are the more generic case, for example, in applications with multiple sources and sinks such as in file sharing applications.

Unstructured P2P protocols typically search for resources (i.e., peers and data) by name or labels, and do not use a structured addressing scheme. This feature supports scalable dissemination but scales poorly for resource discovery or reproducible routing paths. Peers nevertheless maintain an identifier to allow independence of the underlay network address. Resources are discovered using search algorithms on the overlay graph. Examples of search algorithms include breadth-first search, depth-first search, random walks, or expanding ring searches. These options are often combined according to the requirements of the application.

The communication across peers is via messages. Message passing may be direct, i.e., using an underlay network connection between two peers, but this usually requires that the peers explicitly know the peer address and route. When the destination peer for the message to be sent is unknown, messages are piggybacked alongside a resource discovery operation.

All peers maintain lists (direct routing tables with addresses or hashed addresses) with contact information about other peers. Hence, messaging works efficiently and the network does not suffocate from address-search messages. The efficiency of such lists depends on the liveness of the peers. Hence, the listed peers are periodically pinged for liveness and removed when no reply is received. The periodicity is dynamically adjusted based on the relevant churn, i.e., the rate of peer joins and departures.

12.2.2 Structured P2P Protocols

Structured P2P protocols such as Chord, Pastry, Tapestry, Kademlia, CAN etc. [1066, 1067, 1068, 1069] are typically used for data discovery applications where the structure of the topology aids efficient searches. Their topology graphs usually show small-world properties, i.e., there exists a path between any two peers with a relatively small number of edges. Structured topologies often appear as ring structures with shortcuts, which forms a basis for scalable and efficient operations such as resource discovery and message passing. Some protocols have more exotic topologies, e.g., butterfly graphs, fixed-degree graphs, or a multi-torus. The salient characteristics are efficiency of node discovery and efficiency of routing that uses information on the P2P structure and topology. As this aspect has security implications, we briefly detail these operations.

Unlike unstructured P2P's open addressing schemas, in structured P2P protocols, pointers to resources such as peers or data are stored in a distributed data structure which is called a *Distributed Hash Table (DHT)*. The overlay's *address space* is usually an integer scale in the range of $[0, \ldots, 2^w - 1]$ with w being 128 or 160 in general. Usually, a *distance function* $d(a, b)$ is defined which allows distance computations between any two identifiers a and b in the address space. Distance computations are crucial for the lookup mechanism and data storage responsibilities. The distance function and its properties differ among protocol implementations. Data discovery is realised by computing the key of an easy-to-grasp resource identifier such as a distinctive name/key and subsequently requesting that key and its data from one of the responsible peers.

Messages – for example to request the data for a given key – are exchanged in most structured protocols directly, i.e., using an underlay network connection between two peers. If peers

do not know each other, then no direct connection can be set up and the destination peer's location needs to be determined to conduct routing. To this end, an overlay lookup mechanism aims to steadily decrease the address space distance towards the destination on each iteration of the lookup algorithm until the identifier can be resolved. This design approach turns out to be very efficient and promotes scalability. Once the lookup has successfully retrieved the destination's underlay network address, messages can be exchanged. Lookup variants include iterative or recursive algorithms as well as parallelised queries to a set of closest neighbour peers.

Routing tables usually store $k \cdot w$ entries with k being a protocol specific constant. Moreover, for the i^{th} portion of k entries with $i \in [0 \dots w]$, the peer stores contact information of peers that share i common prefix bits of the peer's key. In other words, routing tables usually provide more storage for closer peers than more distant ones. Moreover, routing tables keep only information about live and reachable peers, therefore peers are periodically pinged. In structured protocols, maintenance is more expensive as the topological structure needs to be retained, e.g., newly joined peers have to be put into the appropriate peer's routing tables or leaving/unresponsive peers have to be replaced by live ones in many peers' routing tables.

12.2.3 Hybrid P2P Protocols

Hybrid variants of P2P protocols integrate elements from unstructured and structured schemas, as their principal intent is data discovery and data dissemination. Prominent hybrid protocol examples include file sharing services such as Napster and BitTorrent [1070]. BitTorrent was originally a classical unstructured protocol but now has been extended with structured P2P features to provide a fully decentralised data discovery mechanism. Consequently, BitTorrent could abandon the concept of so called "tracker servers" (that facilitated peer discovery) and improve its availability. On the other hand, architectural requirements often need to be considered to fully utilise the capacity of hybrid P2P protocols. An example would be establishing how the data discovery is transmitted among the servers and how it is reported back to the user [1071]. Similar considerations apply to other streaming overlay approaches.

12.2.4 Hierarchical P2P Protocols

Typically, all the peers in a P2P system are considered to be *equal* in terms of the client-server services they can provide. Yet, for some application scenarios it turns out that a hierarchical P2P design can be advantageous. These can include a layered design of structured and unstructured overlays. In hierarchical designs, peers are further categorised based on their bandwidth, latency, storage, or computation cycles provisioning with some (super) peers taking a coordinating role. Usually, the category with fewer peers represented the back-end part of the hierarchical system, whereas the multitude of peers act as front-end peers that process service requests at the first level and only forward requests to the back-end when they cannot fulfill the service request in the first place. This improves the look-up performance and also generates fewer messages in the network. Furthermore, popular content can be cached locally to reduce download delays [1072]. This design has proven successful, for example, in the eDonkey file sharing system or in Super P2P models such as KaZaA where a selected peer acts as a server to a subset of clients.

12.3 DISTRIBUTED SYSTEMS: ATTACKING P2P SYSTEMS

[1058, c16][1073, c5]

We present security attacks corresponding to the above mentioned classes of P2P systems. To facilitate this discussion, we outline the functional elements of a P2P system that help the reader relate the security implications for specific systems or application cases. Subsequently, we assess the risks stemming from attacks to plan the requisite mitigation. The P2P functional elements that need protection broadly include:

1. P2P Operations (P-OP) such as discovery, query, routing, download, etc. that are accessible through the service interface of the P2P protocol. This functionality relates to the network level.
2. P2P Data Structures (P-DS), e.g., data stored in a peer's routing table or resources that are shared with other peers of the overlay network. This functional element may be accessible at either the network level or locally on the peer's host machine.

We will refer to these two elements as P-OP and P-DS, in the following subsections where we discuss the specific P2P attacks. We use the established security notions of [1074] for Confidentiality, Integrity and Availability. Whenever a definition refers to authentication, we assume that peers are implicitly authenticated on joining the overlay network. P2P protocols may may use admission control systems or may be open to arbitrary peers.

Note that we focus on attacks *against* P2P systems (e.g., denial of service or routing disruptions) and do not consider attacks that are prepared or conducted *using* P2P systems in order to harm non-P2P systems (e.g., using a P2P system to coordinate distributed denial of service attacks).

12.3.1 Attack Types

We now present the different attacks that are specific to P2P systems. Broadly, the attacks correspond to attacking the functional elements, P-OP and P-DS, either by (a) disrupting their connectivity or access to other nodes for dissemination/discovery/routing or (b) corrupting their data structures. Besides the well known (distributed) denial of service attacks which apply to P2P as well as to other systems, most attacks exploit fundamental P2P features such as message exchange based decentralised coordination and especially that each peer has only a partial (local) view of the entire system. Consequently, attackers aim to trick other peers by providing incorrect data or collude to create partitions that hide views of the system from good nodes. This includes example scenarios such as (a) to mislead peers in terms of routing, (b) to take advantage of access to resources, (c) to overcome limitations in voting systems or games, or (d) to hide information in the overlay among others. We refer the reader to the survey articles [1075, 1076] for a fuller exposition of P2P security. We now enumerate some representative security attacks and relate them to their corresponding impact on Confidentiality, Integrity and Availability (CIA). Some examples of attacks are further discussed in Section 12.3.2 along with corresponding mitigation approaches.

- *Denial of service attacks* (DoS) [1074], Distributed Denial of Service (DDoS), or *disruption attacks* [1077] manifest as resource exhaustion by limiting access to a node or a communication route. In P2P architectures, the attacker aims to decrease the overlay network's service availability by excessively sending messages to a specific set of peers and thereby negatively

affecting the P-OP functionality. This could affect the peer join/leave mechanism, or other arbitrary P2P service aspects, e.g., damaging the routing put/get operations in a DHT. For example, benign peers may be impaired by an excessive maintenance workload. Moreover, DoS and DDoS attacks can have a negative impact on bandwidth usage and resource provisioning which may result in degraded services. For instance, GitHub was hit with a sudden onslaught of traffic that reached 1.35 terabits per second[4]. The traffic was traced back to "over a thousand different Autonmous Systems (ASNs) across tens of thousands of unique endpoints" participating in the attack.

- *Collusion attacks* [1078] aim to compromise the availability, integrity, or confidentiality of P2P networks. Collusion refers to the fact that a sufficiently large subset of peers colludes to carry out a strategy which targets the P2P services and thereby negatively affects the P-OP functionality. The typical attack aims to override control mechanisms such as those for reputation or trust management, or bandwidth provisioning. The Sybil and Eclipse attacks, discussed later on, are based on attackers colluding to create network partitions to hide system state information from good nodes.

- *Pollution attacks* [1079, 1080] or *index poisoning* [1081] aim to compromise the P2P system's integrity and its P-DS functionality by adding incorrect information. Consequences of pollution attacks are the proliferation of polluted content resulting in service impairments. An example is the typhoid ad-ware attack where the attacker partially alters the content, e.g., adding advertisement at a single peer that subsequently spreads this polluted content to other peers.

- *White washing* [1080] or *censorship attacks* aim to compromise the availability or integrity of P2P systems. This includes either illicit changing of, deletion of or denying access to data. Therefore, these attacks endanger the P-DS functionality. White washing attacks are especially dangerous for P2P systems that use reputation based systems since they allow a peer with a bad reputation to leave the system, and subsequently re-join as a benign user.

- *Routing attacks* [1077, 1082] aim to compromise the availability or integrity of P2P networks. Routing attacks play an important role in composite attacks, such as the Eclipse attack which obstructs a good node's view of the rest of the system. In routing attacks, a malicious peer undermines the message passing mechanism, e.g., by dropping or delaying messages. Another routing attack variant is *Routing Table Poisoning (RTP)* [1082]. In this attack, an attacker deliberately modifies its own or other peers' routing tables, e.g., by returning bogus information to benign peer lookup requests. *Attraction and repulsion* [1077] are specific variants of routing attacks which either increase (attraction) or decrease (repulsion) the attractiveness of peers, e.g., during path selection or routing table maintenance tasks. These attacks negatively affect the P-DS functionality. The compromise of the routing table in Pastry, often used in online social networks, is a typical routing attack.

- *Buffer map cheating attacks* [1083] aim to decrease the availability of P2P networks, particularly those used for media streaming applications. Through this attack, adversaries reduce the outgoing traffic load of their peers by lying about their data provisioning. This is also an infringement on integrity and affects the P-OP functionality. This attack is especially relevant in streaming media P2P applications which rely on the collaboration of peers. Omission, Fake Reporting, Fake Blocks, incorrect Neighbour Selection are related implications of such attacks.

- *Sybil attacks* [1084] aim to compromise the availability or confidentiality (via spoofing) of P2P networks and can be regarded as a specific version of *node/peer insertion attacks*. They consider the insertion into the overlay of peers that are controlled by one or several adversaries.

[4]https://www.wired.com/story/github-ddos-memcached

This could happen at specific or arbitrary locations of the overlay's topology, depending on the attacker's aim. Furthermore, P2P applications may consider system users as legal entities and consequently restrict the amount of peers per user to the amount of allowed votes for that entity. Hence, an imbalance results in terms of the expected amount of peers per user. Sybil attacks may be a precursor for many of the previously described attacks. Sybil attacks affect the P-OP functionality of the system. Prominent Sybil attacks include the compromise of the BitTorrent DHT and the Sybil attack on the Tor anonymisation network.

- *Eclipse attacks* [1085] aim to decrease the availability, integrity and confidentiality of P2P networks. Essentially, a good peer is surrounded by a colluding group of malicious peers that either partially or fully block the peer's view of the rest of the system. The consequence is that the malicious nodes can either mask or spoof the node's external interactions. This is a composite attack that may involve routing table poisoning, DoS/DDoS, Sybil attacks, collusion, white washing, or censorship. Consequently, these attacks have an impact on both the P-OP and P-DS functionality. Variants of Eclipse attacks include Localised Eclipse Attacks (LEA), Topology Aware Localised Eclipse Attacks (taLEA) and Outgoing Eclipse Attacks (OEA) attacks among others. An example of an Eclipse attack on Bitcoin is discussed in Section 5.

Attack	Availability	Integrity	Confidentiality	Functionality
DoS/DDoS	✓	✗	✗	P-OP
Collusion	✓	✓	✓	P-OP
Pollution	✗	✓	✗	P-DS
White washing & censorship	✓	✓	✗	P-DS
Routing	✓	✓	✗	P-DS
Buffer map cheating	✓	✓	✗	P-OP
Sybil	✓	✗	✓	P-OP
Eclipse	✓	✓	✓	P-DS, P-OP

Table 12.1: P2P Attacks, Security Goals and Affected Functionality

12.3.1.1 Summary

Table 12.1 summarises attacks on the P2P functional elements that entail modifications of the P2P system to either degrade or compromise the P2P operations. The adversarial collusion of malicious peers is a key factor to launch these attacks resulting in significant disruption. In many cases, the inherent design choices of P2P, which foster scalability and fault tolerance, are exploited. Attacks against P2P systems usually show an impact in terms of the system's confidentiality, integrity, or availability. Several of the observed attacks are known from other system architectures such as client-server models while others are new ones or compositions of various attacks. The difference from comparable attacks in client-server system architectures is that P2P overlay networks may grow very large and adversaries have to correspondingly adapt their efforts, i.e., they need to scale up the fraction of malicious peers accordingly, thereby requiring a substantial amount of coordination to execute an effective collusion strategy. These attacks vary depending upon whether the attacker has direct or indirect network access via a P2P overlay. The latter requires attackers to properly join the network prior to the attack. Thus, this may entail malicious peers making, e.g., a proper announcement in the overlay network, before they can launch their adversarial behaviour.

Supplemental Observations:

- Denial of service attacks degrade or prevent a system from correct service delivery [1086,

1087]. The more sophisticated Sybil attack [1087, 1088, 1089] can be used as a potential precursor for an Eclipse attack [1087, 1088].

- If either secure storage, secure routing, or authentication mechanisms cannot be provided, a set of attacks including omission, content forgery, content pollution, censorship, or routing table poisoning may be the consequence [1087, 1089].

- Churn relates to the effects of peers joining and leaving in an overlay. Churn attacks consider artificially induced churn with potentially high peer join/leave rates to cause bandwidth consumption due to the effort needed to maintain the overlay structure. This can lead to partial or complete denial of service [1089].

- Varied cheating attack strategies exist (for observing or corrupting player information and activities) in Massive Multiplayer Online Games (MMOG) built upon P2P architectures [1089].

12.3.2 Attacks and their Mitigation

We present some example attacks along with the approaches used to mitigate them. For a comprehensive coverage, we refer the reader to the surveys of [1075, 1076].

Basic PoS and P-DS Based Scenarios: The prominent P2P protocol security mechanisms are authentication mechanisms, secure storage, and secure routing. These three mechanisms allow the implementation of various downstream mechanisms. Authentication mechanisms [1087, 1090] help to maintain a benign peer population and provide the technical basis for downstream mechanisms like secure admission, secure storage, or secure routing. Secure storage is vital for data centric applications in order to prevent attackers from conducting illicit data modifications [1086, 1088, 1090, 1091]. In a broader sense, illicit data modification in online games is considered as cheating [1089]. The use of secure routing is typically advocated as an approach to facilitate the identification of peers conducting improper message forwarding [1088, 1090, 1091]. Limiting the number of routing paths and/or protecting the paths using (high overhead) cryptographic approaches are alternate approaches to mitigating routing attacks.

Sybil and Eclipse Scenarios: Sybil attacks occur where the attacker could launch an attack with a small set of malicious peers and subsequently gather multiple addresses, which allows malicious peers to fake being a larger set of peers. Using Sybil attacks, a LEA can be launched via a chain of Sybil/malicious nodes. However, the attack relies on the assumption of the existence of a single path towards the victim that can be manipulated by the attacker. Alternately, a LEA can be launched using Sybil peers.

In such attacks, mitigation relies on using a centralised authority that handles peer enrolments or admission. Extending this concept, adding certificates (issued by a common Certificate Authority) to peers' network IDs while joining the network is another possibility. Other mitigation techniques to prevent malicious entities from selecting their own network IDs could entail a signing entity using public key cryptography.

Buffer Map Cheating Scenarios: Other disruptions could be used to attack the KAD P2P network [1069], which is a Kademlia based network, through flooding peer index tables close to the victim with false information as a simplistic taLEA variant. A KAD network crawler is introduced to monitor the network status and detect malicious peers during a LEA. However, a high overhead is incurred if each peer uses such a mechanism to detect malicious entities. This becomes impractical as the overlay size increases.

Divergent lookups have been proposed as an alternate taLEA mitigation technique where the disjoint path lookups avoid searching the destination peer's proximity to skip the wasteful querying of malicious peers under taLEA assumptions.

Routing Scenarios: Mitigation mechanisms to handle routing attacks consider assigning multiple paths for each lookup using disjoint paths though at the cost of high message overhead. Alternatives include the use of cryptographic schemes to protect the paths. However, P2P is a decentralised coordination environment where implementing a centralised service to support the coordination of system wide cryptographic signatures is hard to realise.

The aforementioned security mechanisms increase the resilience of P2P systems against the various attacks. Naturally, these mechanisms are resilient only until a critical mass of colluding malicious peers is reached. In addition, some of these mechanisms require cryptographic support or the identification of peers. These requirements may interfere with application requirements such as anonymity, heterogeneity, or resource frugality.

12.4 DISTRIBUTED SYSTEMS: COORDINATED RESOURCE CLUSTERING

[1062, c5,7,12,25][1056, 3][1057, c5,c14] [1058, c16-17,c19]

Contrasting with the decentralised-control of P2P systems, a multitude of distributed systems exist where the interactions across the distributed resources and services are orchestrated using varied coordination mechanisms that provide the illusion of a logically centralised and coordinated system or service. The coordination can simply be a scheduler/resource manager, a discrete coordinator or a coordination group, and include ordering in time (causality) or varied precedence orders across distributed transactions. While it is tempting to define each type of distributed system discretely (i.e., differing from decentralised control in P2P), the large and diverse group of distributed systems/services share a common abstraction of "coordination" although its realisation and resultant properties for each system will vary.

Firstly, there is the case where a service is replicated on a distributed resources platform (or infrastructure) to enable geo-dispersed access to users while sustaining the required type of consistency specifications on the service. The Cloud and many distributed Client-Server systems fall in this category.

The alternate approach addresses distributed services (versus platforms) where the dispersed service participants interact to yield the collective distributed service for given consistency requirements. For example, transactional databases and distributed ledgers fall in such a category of strong consistency. Web crawlers, searches, or logistics applications may well work with weak consistency specifications.

Overall, these constitute the two broad classes of distributed systems in the coordinated resource pooling mode, namely the classes of *resource-coordination* and *service-coordination*, as based on their characteristic coordination schema although their functionality and definitions often overlap.

In the subsequent subsections, in order to contextualise distributed systems security, we first detail the basic distributed concepts along with the coordination schema based on them. This is followed by outlining the characteristic systems in each of the resource and service coordination models. This forms the basis behind the general set of disruptions/vulnerabilities

relevant to both classes of coordinated distributed systems. We then outline the threats and security implications specific to each class of systems. We refer the reader to the excellent texts of [1056, 1057, 1062] for a comprehensive and rigorous treatise of these issues.

A Note on Technologies Underlying Distributed Platforms: The introduction emphasised that the focus of this KA is on security in distributed systems rather than the use of distribution towards providing security. Expanding on this topic, it is worth commenting on alternate perspectives related to the "design and realisation" of distributed platforms and services. This design oriented perspective tends to emphasise the architecture of distributed systems, distributed services and their construction. This perspective typically focuses on (a) establishing security requirements, (b) realisation approaches on how to meet given security requirements at each level of abstraction, and (c) considers a distributed system as a layered architecture where each layer builds upon the primitives offered at the layer below and from distributed services. In this perspective, centralised (coordinated) and decentralised patterns are often combined, differently and at different layers. Also from this perspective, the security requirements of the applications must be met by complementing and building upon what is offered at the lower layers and services.

This is a construction and compositional approach where the security properties (requirements) at the application level, or at a given layer, drive the selection of solutions and subsystems that must be assembled (e.g., authentication, authorisation, accountability, non-repudiation etc.). The composition of such subsystems/solutions is often achieved through the use of trade-offs (and also threat) analysis that tend to cover some and not all of the requirements and thus determining relative strengths and weaknesses. For example, blockchain applications, further discussed in Section 12.5.2, emphasise non-repudiation and decentralisation as their main properties.

This layered and compositional approach can often be encountered in the literature such as [982, 1057, 1058, 1059, 1060, 1061, 1092] and many others. As the architectures and realisation fundamentally underlie the KA premise of providing security in distributed systems, the reader is encouraged to refer to this literature. The following section returns the focus back on distributed system concepts, and especially the fundamental concepts of the coordination class of distributed systems.

Distributed Concepts, Classes of Coordination

As mentioned in the introduction, a distributed system is a collation of geo-dispersed computing resources that collectively interact to provide (a) services linking dispersed data producers and consumers, (b) high-availability via fault tolerant replication to cover resource (computing and communication) failures, or (c) a collective aggregated capability (computational or services) from the distributed resources to provide (an illusion of) a logically centralised/coordinated resource or service.

Distributed systems are often structured in terms of services to be delivered to clients. Each service comprises and executes on one or more servers and exports operations that the clients invoke by making requests. Although using a single, centralised server appears tempting, the resulting service resident on a server can only be as fault tolerant as the server hosting it. Typically, in order to accommodate server failures, the servers are replicated, either physically or logically, to ensure some degree of independence across server failures with such isolation. Subsequently, replica management protocols are used to coordinate client interactions across

these server replicas. Naturally, the handling of client failures or client compromises (including their role in launching attacks via malicious code or viruses) also needs to be considered.

We now outline a basic set of distributed system concepts that also constitute the basis of the security considerations therein. The concepts are presented at an informal level to communicate the intuitions, and the reader is referred to [1056, 1057, 1058, 1062] for a comprehensive treatise on the topics.

12.4.1 Systems Coordination Styles

In order for the distributed resources and services to meaningfully interact, the synchronisation basis across them, in physical time or in logical order, needs to be specified. The synchronisation applies at both the network and process levels. We refer the reader to [1056, 1057, 1058, 1062] for more details. At a high level, the synchronisation types include the following:

1. **Synchronous:** All components of a distributed system are coordinated in time (as lock step or rounds) to be synchronised with each other. Causality is explicitly obtained. Examples include typical safety-critical systems such as aircraft fly-by-wire control where predictability and guaranteed real-time responsiveness is desired.
2. **Asynchronous:** Separate entities take steps in arbitrary order and operate at different speeds. The ordering of events needs to be ensured through collective interactions. Typical examples are transactional systems, databases, web crawlers, etc.
3. **Partially synchronous:** Some restrictions apply on ordering of actions but no lock-step synchronisation is present. Typical examples are SCADA control systems or high-value transactional stock systems where timeliness has implications on the service correctness.

12.4.2 Reliable and Secure Group Communication

Group communication addresses the communication schema available to ensure reliable delivery of messages across the distributed entities. These can be simple point-to-point direct messaging supported by appropriate acknowledgements (ACKS and NACKS) for reliable delivery. Alternately, reliable and secure multicast (atomic, best-effort, regular, uniform, logged, stubborn, probabilistic, causal, etc.) to provide redundant channels or ordering of messages can be used along with the more sophisticated publish-subscribe forms of group communication [1057, 1058]. In these approaches, the channels and messages can be encrypted or cryptographically signed though this entails higher transmission and processing overheads. The range of credential management, symmetric/asymmetric cryptography techniques, PKI cryptosystems, secure key distribution [1093] also fall in this category. The reader is referred to [1056, 1057, 1058, 1094] for a comprehensive coverage of group communication primitives.

12.4.3 Coordination Properties

The utility of a distributed system comes from a coordinated orchestration of the dispersed resources to yield a collectively meaningful capability. Prior to discussing the variety of commonly used coordination schemas in Section 12.4.4, we first present the base definitions of *Consensus*, *Group Membership* and *Consistency*.

Consensus

Informally, consensus pertains to achieving an agreement on values. For example, the values could be data or process IDs. Consensus requires the following properties to hold:

1. **Agreement:** All good processes agree on the same value.
2. **Validity:** The agreed upon value is a good/valid value.
3. **Termination:** A decision is eventually achieved.

The specific type of consensus depends upon the semantics of the faults (*crash*, *omission*, *Byzantine*, etc.) to be addressed. The faults types are discussed in Section 12.5.

Group Membership and Consistency:

Membership is a key "service" property in distributed systems that determines the set of constituent resources and also the nature of the agreement achieved on the set of valid participants (static, dynamic, quorum membership) and the data. From a security perspective, this often relates to the integrity property for the service. Consistency has varied nuances and the prominent types are listed below with fuller details presented in [1056, 1057, 1058, 1062, 1094, 1095]. Note that the underlying assumption is always that the constituent processes can be modelled as deterministic state machines. That is, performing a specific sequence of actions always leads to the same state.

- **Strong consistency models:** In these models the participants must agree on one consistent order of actions to take. Hence, the processes are guaranteed to reach a consistent state under the assumption of determinism.
    1. *Strict Consistency:* In strict consistency there are no constraints on the observed order of actions as long as it is consistent across all the participants.
    2. *Linearisability:* The linearisability model is essentially strict consistency with the additional constraint that the observed order of actions corresponds to their real time order.

  Strong consistency models are widely used in high risk contexts where any inconsistencies in the data may lead to dire consequences. In these situations, consistency is more valued than availability and enforcing strong consistency constraints results in more delays in the systems due to the frequent synchronisation. Traditional relational database systems such as MySQL [1096] or Microsoft's SQL Server [1097] but also modern NoSQL databases such as MongoDB [1098] or Google's Chubby lock service [1099] are popular examples that implement these strong consistency models.
- **Weak Consistency Models:** In these models, the participants do not necessarily observe the same order of actions. This can lead to inconsistent states depending on the nature of the additional constraints that the observed orders have to satisfy. Naturally, this can

lead to inconsistent states that can be dealt with through conflict resolution mechanisms [1100].

1. *Sequential Consistency:* Sequential consistency is met if the order in which the actions are performed by a certain process corresponds to their original order. In order words, the sequential execution order of every process is preserved.
2. *Causal Consistency:* Causal consistency is achieved by categorising actions into those causally related/dependent and those that are not. In this case only the order of causally related actions has to be preserved. Two events are causally related if they both access the same data object and at least one of them is a write event.
3. *Eventual Consistency:* In eventual consistency there are no special constraints that have to be satisfied by the order of observer actions. The idea behind this concept is that the participants will *eventually* converge to a consistent state either by observing equivalent orders of actions or by resorting to *costly* conflict resolution mechanisms.

Systems with weaker consistency models became popular with the advent of the Internet where wide scale web servers had to accommodate a large number of users. To achieve this, such systems sacrifice strong consistency guarantees to achieve higher availability for their user base. Systems like Amazon's Dynamo [1101], Facebook's Cassandra [1102] are widely known examples of systems with weak consistency guarantees.

12.4.4 Replication Management and Coordination Schema: The Basis Behind Attack Mitigation

A fundamental challenge for developing reliable distributed systems is to support the cooperation of the dispersed entities required to execute a common task, even when some of these entities, or the communication across them, fails. There is a need to ensure ordering of the service actions and to avoid partitions of the distributed resources in order to result in an overall "coordinated" group of resources.

The state machine replication or state machine approach [1103] is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas. The approach also provides a framework for understanding and designing replication management protocols. The essential system abstraction is that of a state machine such that the outputs of the state machine are fully determined by the sequence of requests it processes independent of time or other activity in the system. Replication can be active, semi-active, passive, or lazy [1058].

It should be noted that ideally one would like to collectively attain high availability, consistency and also full coordination to eliminate any partitioning of the set of distributed resources. However, the CAP assertion comes into play as:

CAP

Any network shared data system (e.g. Web) can provide only 2 of the 3 possible properties [1104] as:

1. **Consistency (C):** equivalent to having a single up-to-date copy of the data, i.e., each server returns the right response to each request.
2. **Availability (A):** of the data where each request eventually receives a response.
3. **Partition (P):** Network partition tolerance such that servers cannot get partitioned into non-communicating groups.

Naturally, security attacks attempt to compromise these elements of CAP.

Replication and Coordination

In order to provide coherent and consistent behaviour (in value and order), distributed resources use various types of replica management, i.e., the coordination schema. This is a key coordination mechanism that characterises the functionality of any distributed system. The factors determining the specific mechanism depend on the type of system synchronisation model, the type of group communication and especially the nature of the perturbations (faults or attacks) being considered. The mechanisms can be simple voting or leader election processes (e.g., Ring Algorithms, Bully) or more complex consensus approaches to deal with crashes or Byzantine[5] behaviour. The commit protocols for database transactions are relevant here as are the schemes for credential management and PKI infrastructures providing verified access control. We briefly describe a set of widely used schema, and the reader is referred to [1056, 1057, 1062] for complete coverage. Authorisation and Authentication in distributed systems are also discussed in the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14).

Paxos

To avoid the situation of distributed entities conducting uncoordinated actions or failing to respond, Paxos [1105], a group of implicit leader-election protocols for solving consensus in an asynchronous setup, has been developed. Paxos solves the consensus problem by giving all the participants the possibility to propose a value to agree upon in an initial phase. In the second phase, if a majority agrees on a certain value, the process that had proposed the value implicitly becomes the leader, and agreement is achieved. The same process is repeated for the next value to achieve consensus on a sequence of values.

The protocol is known not to provide liveness only under very specific circumstances as described in [1105]. In this case, processes continue to propose values indefinitely and remain blocked in the initial phase as no majority can be formed and progress is never made. However, this situation rarely occurs in practice and Paxos remains one of most widely used coordination protocols.

Since only a majority is necessary in the second phase to reach consensus, the protocol is additionally tolerant to crashes even in the case of recovery. This is remarkable since, as long as the majority of the processes has not failed, consensus can be reached. The paper [1106]

[5] Byzantine behaviour happens when an entity/attacker sends different (albeit valid) information to different recipients.

is an excellent read of the experiences of implementing Paxos at Google for the Chubby file system.

While there exists a variety of implementations of the Paxos protocol, it is notoriously known for being hard to implement and build middleware upon it due to its inherent complexity. For this purpose, RAFT, a protocol similar to Paxos that provides the same guarantees, has been proposed. RAFT has recently gained in popularity due to its simpler design. Paper [1107] explains the motivation behind the development of the RAFT protocol and how it works by comparing it with Paxos.

Byzantine Fault Tolerance (BFT)

Attacks and other deliberate disruptions do not necessarily follow the semantics of benign omissions, timing or crashes. In order to tolerate arbitrarily malicious behavior, Byzantine Fault Tolerance (BFT) protocols use coordinated replication to guarantee the correct execution of operations as long as at most a third of processes is compromised and exhibits arbitrary (i.e., Byzantine, cf. Section 12.5) behavior.

In BFT, processes exchange the values they have received from each other in rounds. The number of rounds necessary to reach consensus is determined by the number of compromised participants there are in the system [1108]. Note that since the protocol operates in rounds, it is classified as a synchronous coordination protocol. It has been shown in [1109] as the FLP impossibility result that it is impossible to reach consensus in the case of asynchronous communication. Due to the necessity of synchronous communication and the rather higher overhead of message exchange required to deal with Byzantine failures, BFT protocols are applied mostly in specific critical applications. However, there are multiple ongoing attempts for practical BFT optimisations by strengthening some basic assumptions on synchronisation, determinism, and number of compromises [1110, 1111, 1112]. The Google File System (Chubby) and Amazon Web Services (AWS) implement Paxos and also partial BFT functionality. It is also important to emphasize that BFT is expensive not only for the message complexity over the number of rounds needed. It is also expensive for the number of nodes needed ($> 3f$) to handle f malicious failures, i.e., f being the number of nodes controlled by an adversary. The generalisation of adversarial structures to quorum systems is discussed in [1094].

From a security viewpoint, for its ability to tolerate arbitrary malicious behaviors, the BFT protocols constitute an appealing building block for the construction of intrusion tolerant systems. It is worth making the observation that these protocols consider the number of compromised entities. When faced with a malicious attacker identical replicas are not sufficient because they exhibit the same vulnerabilities. A malicious adversary who can compromise one replica can easily compromise the others if they are identical. Replication and diversity (or distinct protection methodologies) are needed. We refer the reader to the discussions in [1056, 1057, 1062, 1094, 1108, 1113, 1114].

Commit Protocols

A number of applications, e.g., databases, require ordering across replicated data or operations where either all participants agree on conducting the same correct result (i.e., commit) or do nothing – the atomicity property. Hence, as a specialised form of consensus, a distributed coordinator directed algorithm is required to coordinate all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction.

The Two-Phase Commit (2PC) is a straightforward example of such atomic commitment protocols. The protocol proceeds with a broadcast query from a leader to all the clients to commit. This is followed by an acknowledgment (commit or abort) from each client. On receiving all responses, the leader notifies all clients on an atomic decision to either commit or abort [1057, 1059, 1060]. The protocol achieves its goal even in many cases of failure (involving either process, network node, or communication failures among others), and is thus widely used. An approach based on logging protocol states is used to support recovery. The classical 2PC protocol provides limited support for the coordinator failure that can lead to inconsistencies.

To solve this problem the three-phase commit (3PC) protocol has been developed. The 3PC protocol is essentially an extension of the BFT protocol and adds a third communication phase to assist the leader with the decision for an abort. This entails a higher messaging and logging overhead to support recovery. While 3PC is a more robust protocol compared to BFT, it is not widely used due to the messaging overhead and its sensitivity to network partitioning (i.e., the P in CAP). In practice, systems use either BFT for its simplicity or the Paxos protocol for its robustness.

12.5 DISTRIBUTED SYSTEMS: COORDINATION CLASSES AND ATTACKABILITY

[1062, c3][1056, c5,c6][1057, c19] [1058, c18][1073, c3]

The General Class of Disruptions

The attack surface [1073, 1115] in distributed systems involves the disruption of the resources, communication, interfaces, and/or data that either impairs the resource availability or disrupts the communication layer interconnecting the resources to impact Confidentiality, Availability, or Integrity of the overall system and its services. The disruptions can be from improper design, arising from operational conditions or deliberate attacks. Resource compromises or disruptions form the basic attack targets. However, the functionality of a distributed system emerges from the interactions across the distributed resources. As referenced in Section 12.1.2, the resources and services (including replication management) in a distributed system are primarily linked via communication infrastructures. These span the range of direct message exchanges or via middleware architectures such as pub-sub or event based triggering among others.

A number of varied terminologies exist to cover the range of operational and deliberate perturbations from crashes, omissions, timing, value disruptions, spoofing, viruses, trapdoors, and many others. We refer the reader to [1074] for a comprehensive discussion on the topic.

As the distributed systems primarily rely on message passing for both data transportation and coordination, we group the perturbations at the level of message delivery[6]. The term "perturbation or disruption" is deliberately used as the anomalous operation can result from operational issues (dependability) or from a malicious intent (security). The manifestation of these perturbations on the system operations results in deviations from the specified behavior of the system. Complementing the vulnerabilities mentioned in Section 12.1.2 of access control, data distribution, interfaces, the communication level perturbations can be broadly grouped as:

1. **Timing Based**: This spans the omission of messages, early, delayed, or out-of-order messaging. Crashes and denial-of-service also fall in this group as they typically manifest as disruptions of the proper temporal delivery of messages by obstructing access to the communication channels or resources.
2. **Value/Information Based:** Spoofing attacks, mimicking, duplication, information leakage such as a Covert Channel Attack or Side Channel Attack, and content manipulation attacks broadly fall in this category. The manipulation of the content of messages manifests as Byzantine behavior. This attack is only viable if a set of resources use the exchange messages to build their global view of the system. A malicious entity can send deliberately modulated information (e.g., a mixture of correct and incorrect values) to different groups of resources to result in partitions of system state views. Thus, based on different values received by different nodes, the individual nodes are unable to constitute a "consistent" and correct view of the system state. The degree of breach of consistency (strong – full agreement by all on value and order – weak, partial, eventual) constitutes the degree of disruption. The nature of the underlying transactional service (e.g., distributed ledgers in Blockchains) determines the type of breach of the functionality. Relating to the groups of vulnerabilities, a Byzantine attack can abuse access control, message delivery and coordination services, or the data itself (viruses, compromised mobile code, worms) to compromise the system.

It should be noted that a perturbation also includes the property of persistence, i.e., the duration of a perturbation can be transient, episodic, intermittent, or permanent in nature. Furthermore, attacks often entail multiple simultaneous occurrences that involve a combination of timing, value, persistence, and dispersed locations, potentially due to collusion between multiple attacking entities.

Attacks and Implications

On this general background, we now detail the two prominent classes of distributed systems as based on the coordination schema (resource- and service-coordination). This will also form the system grouping for considering the security manifestations of attacks.

We use the classical CIA (Confidentiality, Integrity, and Availability) terminology though the implications of these terms often differ according to the type of system and services. For each class, the specification of its functionality determines the type of attack and the resultant compromise that detrimentally affects the delivery of services.

As mentioned in Section 12.1.2, the threat surfaces of a distributed system comprise attacks

[6]The provisioning of message integrity by techniques such as coding, cryptographic primitives, message acknowledgements, retries, secure group communication, etc. are discussed in [1057, 1058] and the Cryptography Knowledge Area (Chapter 10).

on the resources, admission control, the communication architectures, the coordination mechanisms, and the data. Similarly, attacks aim to subvert the assumptions behind the functionality of resources, the services, and the underlying coordination schema.

In the following subsection, we enumerate some attack scenarios for the resources/infrastructure and services/application classes of coordination. Given the immense diversity of types of resource and services based distributed systems, the purpose of these examples is only to illustrate some potential scenarios. It is also worth highlighting that often a resource attack does not harm the resource *per se* but primarily affects the service executing on the resource.

12.5.1 The Resource Coordination Class – Infrastructure View

This class of "virtualised resource access" primarily deals with the coordination of a group of computing and communication resources to provide an ensemble of highly-available, highly-reliable "platform" of diverse shared resources to the user. This is an infrastructure (vs applications) view where the user specifies the operational requirements for the desired service (e.g., computational capabilities, number of Virtual Machines (VMs), storage, bandwidth constraints, etc.) but is agnostic to the actual mechanisms providing the on-demand access to the resources, scalability, physical characteristics, and geo-location/distribution of the underlying resources.

Overall, the key characteristic of this coordination model is the provisioning of high-reliability, high-availability access to resources. The basic resource replication simply provides a pool of resources to support high-availability access. However, the resource replication schema provides only the "capabilities" to support the services executing on it. Integrity is relevant corresponding to the service specifications. For instance, VMs need to provide the specified level of isolation without information leakage. Similarly, a web server is typically replicated across machines both for reliability and for low-latency localised geo-dispersed access. Each replicated server has the same set of data, and any time the data is updated, a copy is updated across the replicated servers to provide consistency on data. It is the nature of the service (as executing on the resources platform) that determines the type of desired coordination, perhaps as consistency (strong, weak, eventual, causal). This will be the basis of the Service Coordination class discussed later on.

We briefly present the Cloud and Client-Server models that constitute prominent examples of the class of distributed resources.

The Cloud Model

The Cloud, in all its manifestations, is representative of the resource coordination model as essentially a "resources platform" for services to execute on. There are multiple types of Clouds offering varied types of services ranging across emphasis on high-performance, low-latency access or high-availability amongst many other properties. It is the specific resource coordination schema dictated by the specifications of the desired services based on which the Cloud "platform" provides structured access to the Cloud resources. The chosen coordination schema correspondingly supports the type of desired capabilities, for example, access to specialised computing resources and/or resource containers such as physical or virtual machines each offering differing isolation guarantees across the containers. The user specified services execute on the Cloud resources, which are managed by the Cloud service provider. The coordination schema, as a centralised or distributed resource manager,

handles the mapping and scheduling of tasks to resources, invoking VMs, health monitoring of resources, fault-handling of failed resources such that the user transparently obtains sustained access to the resources as per the contractual Service Level Agreements (SLAs) specified on the Cloud resources. The ENISA [1116], NIST [53], and ISO [1117] specifications of Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) are representations of "resources/platforms/infrastructures supporting the services". The multitude of Cloud models, architectures, and services existing in practice makes it difficult to project a single notion of Cloud security. Each specific resource coordination model is characterized by the types of resource types in the Cloud model, the type of computing architecture as well as the desired functionalities within the Cloud. These include, as a non-exhaustive list, the desired types of resource fault handling, the chosen approach for handling of service bursts, the type of schemas implemented for resource federation and migration, for task orchestration, scheduling, the desired degree of concurrent access, the supported levels of multi-tenancy etc.

However, from a security perspective, it is useful to de-construct the Cloud into its architectural and functional components that result in the Cloud's attack surface to consider. Analogous to the infrastructure view of a data center being an aggregation of computing and storage resources, the Cloud is an aggregation of geo-dispersed resources that are available on-demand to the user. The user has resource-location and resource-composition agnostic transparent access to highly-scalable, highly-available, highly-reliable resource and service virtualisation. The user specifies the operational attributes of interest (termed as Service Level Objectives) as (a) performance specifications, (b) reliability, (c) replication and isolation characteristics as types and number of VMs, (d) latency, (e) security as the level/degree of encryption and other mechanisms at the computing or communication level and (f) cost parameters for delivery or non-delivery of services in the form of contracts known as Service Level Agreements. The exact composition of the resources, their location or the mechanisms collating the aggregated resources is *transparent* to the user. The functional blocks of the Cloud include authentication, access control, admission control, resource brokering, VM invocation, schedulers, monitors, reconfiguration mechanisms, load balancers, communication infrastructures, user interfaces, storage, and many other functions under the PaaS and IaaS paradigms [53, 1116, 1117]. These functional blocks, the physical Cloud resources along with the interfaces across them directly constitute the attack surface of the Cloud.

The Client-Server Model

Resource groups where a set of dedicated entities (servers – service providers) provide a specified service (e.g., Web services – file system servers, name servers, databases, data miners, web crawlers, etc.) to a set of data consumers (clients). A communication infrastructure, such as the public Internet, a local network, or a combination thereof, links the servers to the clients. This can be monolithic, layered, or hierarchical. Both servers and clients are replicated to either provide a characteristic collective distributed service or for fault tolerance. Note that we are referring to Client-Server architecture as a resources platform or infrastructure and not the Client-Server services per se. The functionality of a Client-Server infrastructure is derived from the specifications of the services using the Client-Server model and from the requisite coordination schema underlying it.

Attackability Implications (and Mitigation Approaches) on Resource Coordination

We now outline some example scenarios for the Cloud though they analogously apply to the Client-Server and other resource models as well. The reader is referred to [1118, 1119] for an insightful discussion relating security and functionality issues in the Cloud.

- *Compromise of Resources:* Such attacks impact the Availability of the basic resources.

Mitigation: Protection can be obtained by using access control schemes (including Firewalls) to limit external access to services and network resources. Authorisation processes are set up for granting of rights along with access control mechanisms that verify the actual rights of access [1120]. Other approaches to resource protection include the sandboxing resources or having a tamper-resistant Trusted Computing Base (TCB) that conducts coordination handling [1057, 1058] and enforces resource accesses. While the resource class primarily considers attacks on the infrastructure, data at-rest or in-motion (as in a data storage facility) can also be considered as a resource. Consequently, it can be protected using techniques such as encryption. As the specification of a distributed service includes the specification of both normal and anomalous behavior on the use of the data providing the service, this protection is considered under the services class.

Other manifestation of resource attacks, including on communication channels, aim to partition resources (and overlying services). The implication here is on Availability for the resources and on Integrity for the services.

- *Compromise of Access/Admission Control:* This comprises the broad categories of Masquerading, Spoofing, and ID management attacks. The implication on the resources is on Availability, though both the Integrity and Confidentiality of the data/service are affected. In case of a DoS attack, the consequence is on resource Availability.

Mitigation: Intrusion Detection System (IDS) constitute typical mitigation approaches. These are complemented by periodic or random ID authentication queries. The periodic checking of system state is used to establish the sanity of IDs.

- *Compromise of VM:* The typical manifestation is of information leakage from the VM via a Covert Channel Attack or Side Channel Attack or similar attacks. The consequence is the violation of Integrity and Confidentiality of the services provisioned by the VM.

Mitigation: A variety of schemes for VM's protection are detailed in [982] (also see the Operating Systems & Virtualisation Knowledge Area (Chapter 11)). There are three aspects to be considered here as the detection of leakage, the system level where the leakage transpires, and the handling of leakage. Taint analysis is a powerful technique for data level detection. As covert/side-channel attacks often happen at the hardware level and are influenced by the schedulers, the use of detectors employing hardware performance counters is a generally used technique as advocated in [1121]. System level handling of VM compromises often starts from the level of tightening the specification of trust assumptions and validating them being upheld using analytical, formal, or experimental stress techniques. Hypervisors are commonly used for the enforcement of VM operations.

- *Compromise of Scheduler:* There are two manifestations of such attacks. When the scheduler is affected and this results in an anomalous task or resource allocation, such a deviation (on an incorrect resource allocation) can be detected through Access Control. In the case of a malicious takeover of the scheduler, the likely resultant inconsistencies across the

system state or resource-task bindings can be filtered by the coordination schema whose job is to maintain a consistent state. Such attacks typically impact Availability and Integrity. Confidentiality is not breached.

Mitigation: As mentioned in the attack description, Access Control and coordination constructs are used to check the consistency of the system state for any observed mis-match to the legitimate or allowed set of resource allocations. This can be used identify corruptions of the scheduler.

- *Compromise of Broker:* This occurrence, within a Cloud resource manager/broker or an inter-Cloud broker, primarily impacts resource Availability.

Mitigation: Approaches similar to scheduler compromise mitigation are used here. If backup brokers are part of the design, that is a typical fall back, otherwise, system stops are often the solution.

- *Compromise on Communication:* As communication is a core functionality to achieve resource coordination, this has strong implications on the resources to stay coordinated and directly impacts Availability. The consequent inability to support replication, resource to task allocation, etc. fundamentally compromises the functionality of the system.

Mitigation: A variety of communication protection techniques are presented in the Network Security Knowledge Area (Chapter 19). These include retries, ACK/NACK based schemes, cryptographically secured channels among others.

- *Compromise on Monitoring and Accounting:* With incorrect information on the state of the system and/or services, this can lead to compromise of Confidentiality, Integrity, and Availability.

Mitigation: State consistency schemes are the typical mechanism utilised here. It is worth mentioning that the replication and coordination concepts presented in Sections 12.4 and 12.4.4 form the basis of the mitigation approaches. The very purpose of the replication management is to obtain consistent system states to circumvent disruptions.

12.5.2 The Services Coordination Class – Applications View

The service coordination model focuses on the specific characteristics of the services that determine the degree/type of coordination relevant to supporting that service. For example, a database hosted on a Cloud necessarily requires the provision of integrity in the form of ACID[7] properties along with liveness. Distributed storage, such as KVS (Key Value Store) or transactional database services, may require varied levels of consistency or linearisability where the desired level of integrity may depend on the level of data-access latency feasible in the system. The broad class of Web services to include Web crawlers and search engines may require weak or partial consistency as per CAP. On the other hand, Blockchains or ledger queries, that provide distributed crypto based consensus, have strong consistency (and traceable auditing) as a key requirement with lesser demands on latency. Thus, it is the specification of the service (KVS, Database, Blockchain) that determines the nature of the coordination schema for the distributed resources platform.

We present some characteristic examples of the services class as:

[7] A stands for atomic, C for consistent, I for isolated and D for durable.

Web Services: These cover the spectrum of data mining, web crawlers, information servers, support for e-transactions, etc. This is a fairly broad, and generic, category, which encompasses a wide variety of services. It is useful to note that many of these services utilise the Client-Server paradigm though our interest here is at the services level.

Key Distribution: This is a broad class of (Authorisation & Authentication) services such as Kerberos, PKI, etc. Such services typically enable authentication (either proving server authenticity to a client, or mutually authenticating both client and server) over insecure networks, based on various cryptographic protocols. Authentication services commonly act as trusted third party for interacting entities in a distributed system. For further details see the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14).

Storage/KVS

This is a diverse set of services starting from register level distributed read-writes that entail strong consistency with very low latency. Another general model is Key Value Store (KVS) where data is accessed via keys/pointers/maps with simple read, write, delete types of semantics. In KVS, the data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection with a focus on fast access times (up to a constant access time). The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as extensions with specified properties. For example, an ordered model can be developed that maintains the keys in a lexicographic order to efficiently retrieve selective key ranges. Key-value stores can use consistency models ranging from eventual consistency to strict consistency. The security issues requires dealing with data-at-rest (static storage) and data-in-transit (dynamic R/W ops).

Transactional Services, Databases

This is a wide class of services covering databases and general transactional services (retrieval, informational data mining, banking and stock transactions, etc.). The requirements are consistency as in banking where all the debit and credit transactions are (strongly or weakly) serializable for consistency. More generally, a database adheres to all of the stipulated ACID properties.

On the other hand, a number of data mining and information lookup transactions only require weaker nuances of consistency. For example, an information lookup process can work with physically partitioned data centers resulting in stale or inconsistent information as long as they are eventually reconcilable within some specification of the service requirements. The specification of the type and degree of perturbations and level of consistency the services are designed to be resilient to determines the specific coordination schema to use. Additionally, in the case of weaker consistency models, the user is required to deal with any stale data that might have been retrieved from the database.

Blockchains/Cryptocurrencies

The concept of a ledger provides for consistent bookkeeping on transactions. This is problematic to achieve in a distributed system where the participating entities do not trust each other and are potentially untrustworthy. Blockchains provide a decentralised, distributed, and public ledger that is used to record transactions across many computers so that the record cannot be altered retroactively without also altering all subsequent blocks. Such alterations require the consensus of the network and can therefore not be performed unilaterally by an attacker. This also allows the participants to verify and audit transactions inexpensively. Blockchains form the foundation for numerous cryptocurrencies, most notable Bitcoin.

In technical terms, a Blockchain is a list of records or blocks. The aforementioned properties arise from the fact that each block incorporates a cryptographic hash of the previous block and a timestamp. If a block in the chain is altered without also altering all subsequent blocks, the hash of the following block will no longer match, making the tampering on the Blockchain detectable.

When used as distributed ledgers, Blockchains are typically managed by peer-to-peer networks. Peers in such a network participate in a protocol for validating newly submitted blocks. Blockchains are also examples of widely deployed systems exhibiting high tolerance to Byzantine failures.

The generic Blockchain concept allows participation by any entity (permission-less systems, public blockhains) and does not include any access restrictions. This is the case for the blockchains underlying many widely used cryptocurrencies such as Bitcoin. However, a more restrictive participation model (permissioned systems, private blockchains) is also possible, where a “validating authority” grants permission for participation.

In order to deter denial of service attacks and other service abuses such as spam on a network, the concept of Proof-of-Work (PoW) (i.e., spending processing time to perform computationally expensive tasks) is specified as a requirement for participation. This is effective as a means of preventing service abuses such as spam since the required work is typically hard to perform but easy to verify, leading to asymmetric requirements for service requester and provider. However, PoW schemes also lead to high energy usage and, depending on the chosen work requirement, may lead to unreasonably high barriers of entry. This is the case, for instance, in certain cryptocurrencies, where meaningful participation requires custom hardware designed for the specific type of work required. To avoid these shortcomings, alternative approaches relying on Proof-of-Stake (PoS) are in development but not as mature as PoW-based schemes and not widely deployed.

A comprehensive discussion on Blockchain issues appears in [1122, 1123]. As a note, Blockchains represent an interesting combination of decentralised resources using the P2P model for the resource coordination and the coordination schema of consensus for its service functionality.

Overall, service integrity, in terms of consensus as supported by requisite liveness, is the key characteristic of the service coordination model. This contrasts with the resource coordination class where resource accessibility and availability were the dominant drivers/considerations.

Attackability Implications (and Mitigation Approaches) on Service Coordination

The services and applications constitute a very broad class to cover, both for the type of attacks and the diversity of services where the functional specification of the service determines the type and degree of the impact on security. In most cases the breach on Integrity, along with on Confidentiality, is the first class impact with impact on Availability following as a consequence. Some examples of breaches for the coordination schema and service types are mentioned below.

Note: The mitigation schemes applicable here are the same as described in Section 12.5.1 that essentially result from the basic replication management and coordination concepts presented in Sections 12.4 and 12.4.4. The very purpose of replication based coordination, at the resource or the service level, is to prevent compromises by discrete attacks up to the

threshold of severity type and the number of disruptions the replication schema is designed to handle.

Compromise of Key distribution in PKI: The authentication processes supporting the distribution of public keys is compromised affecting service Integrity and Confidentiality.

Compromise of Data at Rest: This is analogous to the breach of resources in the resource coordination model as applicable to storage systems.

Compromise of Data in Motion: This has varied consistency and latency consequences that compromise the Integrity depending on the specifications of the services. We present a very simplistic enumeration using transactions classes as:

Short transactions: (Storage/KVS, etc.) The major driver for this class is both consistency and low latency (e.g., linearisability). As both liveness and safety are violated, the Integrity of the transaction is compromised. It is worth noting that a DoS attack may not affect consistency. However, as latency is affected, the service Integrity is lost.

Large transactions: Ledgers (Blockchain, etc.) lie in this category where, although latency is important, it is the Integrity (as defined by the consistency of the ledger) that is the primary property to preserve. As Ledgers constitute a popular service, we discuss it to illustrate aspects of both attack surfaces and assumptions.

To recapitulate from Section 12.5.2, Blockchains constitute a ledger of information that is dispersed across a distributed system. Blockchains ensure the security of data by not providing a single point of attack. The ledger is stored in multiple copies on a network of computers. Each time an authorised participant (for example in a permissioned system) submits a transaction to the ledger, the other participants conduct checks to ensure that the transaction is valid, and such valid transactions (as blocks) are added to the ledger chain. Consensus ensures a consistent view of the sequence of transactions and the collated outcome. The cryptographic basis of the hash, on each block, is expected to avoid tampering, and the Proof of Work notion is designed to mitigate the effect of DoS attacks.

What makes this system theoretically tamper proof are two aspects: (a) an unforgeable cryptographic hash linking the blocks, and (b) attack-resilient consensus by which the distributed participants agree on a shared history of transactions.

Compromising these involves the compromise of stored cryptographic keys and the hash. While theoretically safe, such systems may turn out to be vulnerable to emergent technologies such as quantum computing. Moreover, while Proof of Work requirements (i.e., “to demonstrate” a greater than 50% participant agreement) can make collusion attacks prohibitively expensive in sufficiently large systems, they can be feasible on systems with fewer participants.

Similarly, the consensus property can be compromised via an Eclipse attack [1124] for Bitcoin, and also in general cases where there exists the potential to trick nodes into wasting computing power. Nodes on the Blockchain must remain in constant communication in order to compare data. An attacker that can take control of a node’s communication and spoof other nodes into accepting false data to result in wasted computing or confirming fake transactions can potentially breach consensus. The work in [1123] provides useful reading on such compromises.

Mixed transactions: As implied in the label, this combines short and large transactions. The security implications depend on the type of services. As an example, we outline two service

groups, namely:

- E-commerce supporting transactions: The core requirements here are ACID properties that entail strong consistency and no partitions. Any compromises affect the Integrity of the service.

- Informational systems: Services such as Webcrawlers, Data Retrieval for applications such as Uber or informational queries for shopping can handle (both network and data) partitions of data to operate on stale cached data. The attack may lead to redundant computations on the searches or slightly stale information but Integrity is not violated as long as the semantics of Weak, Relaxed, or Eventual consistency, as applicable for the service specification, are sustained. Also informational queries have mixed latency requirements. For example, the small latency within a local data center and higher-tolerable latency across geo-dispersed data centers may define the degree of attack tolerance until both Availability and Integrity are compromised.

CONCLUSIONS

The intent of this chapter has been to outline how distributed systems work, and how the mechanisms supporting the operations of such systems open security issues in them. Very often the expectation is that classical security techniques will directly apply in a distributed systems context as well. However, this is often not the case and the better one understands the conceptual basis of a distributed system, the better one can understand and provide for its security. The KA discussed the functional categorisation of distributed systems into two major classes: decentralised and coordinated control. The operations for each class were elaborated leading to the security implications resulting from the different specifics underlying distributed systems.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[1062]	[1056]	[1057]	[1058]	[1073]
12.1 Classes of Distributed Systems and Vulnerabilities		c2	c5	c18	
12.2 Distributed Systems: Decentralised P2P Models	c11,c12		c25		
12.3 Distributed Systems: Attacking P2P Systems				c16	c5
12.4 Distributed Systems: Coordinated Resource Clustering	c5-7, c12, c25	c3	c5,c14	c16, c17, c19	
12.5 Distributed Systems: Coordination Classes and Attackability	c3	c5-6	c19	c18	c3

FURTHER READING

The following books are recommended for a deeper coverage of the distributed system and security concepts.

Distributed Algorithms Concepts Lynch [1062] — The book lays out the essential concepts of distributed systems. The focus is on synchronisation and consensus though it provides a comprehensive and mathematically rigorous coverage of distributed systems concepts from an algorithms viewpoint.

Reliable & Secure Distributed Programming Cachin, Guerraoui, Rodrigues [1056] — Coming from a distributed programming viewpoint, this is another rigorous book that covers both fault tolerance and security. It also provides an excellent coverage of cryptographic primitives. Although it predates the development of Ledgers, most of the concepts behind them are covered in this book.

Group Communication & Replication Birman [1057] — This is an excellent book that combines concepts with an emphasis on the actual development of distributed systems. The case studies provide valuable insights on practical issues and solutions. An insightful coverage of P2P systems also appears in this book.

Security Engineering Anderson [1013] — This book makes for excellent reading on the realisation of distributed system from a security perspective especially for naming services and multi-level security. The reader is also encouraged to read the texts [1061, 1092] that detail complementary coverage on CORBA and Web services.

Threat Modeling Swiderski, Snyder [1073] — The coverage is on the basics of threat modeling from a software life cycle and application security viewpoint. While not a distributed systems book, it still provides valuable insights on how threat modeling is conducted in practice.

Chapter 13
Formal Methods for Security

David Basin | ETH Zurich

INTRODUCTION

This Knowledge Area surveys the most relevant topics in formal methods for security. As a discipline, formal methods address foundations, methods and tools, based on mathematics and logic, for rigorously developing and reasoning about computer systems, whether they be software, hardware, or a combination of the two. The application of formal methods to security has emerged over recent decades as a well-established research area focused on the specification and proof of security properties of systems, their components, and protocols. This requires a precise specification of:

- the **system** at an appropriate level of abstraction, such as design or code,
- the **adversarial environment** that the system operates in, and
- the **properties**, including the **security properties**, that the system should satisfy.

Formal reasoning allows us to prove that a system satisfies the specified properties in an adversarial environment or, alternatively, to identify vulnerabilities in the context of a well-defined class of adversaries.

Formal methods have a wide scope of applicability. Hence this Knowledge Area is relevant to many other KAs as it encompasses general approaches to modelling, analysis, and verification that relate to many technical aspects of cybersecurity. Moreover, as formal methods for security have applications across the entire system stack, this KA leverages background knowledge of hardware and software security from other chapters. As specific prerequisites, this KA benefits from background knowledge of logic, discrete mathematics, theorem proving, formal languages, and programming semantics, at a Computer Science undergraduate level, and provides references covering these topics; some relevant text books include [1125, 1126, 1127]. Modelling and abstraction are cornerstones of formal methods. This KA covers their application across security topics, including access control, secure information flow, security protocols, and program correctness. These can be considered with respect to requirements such as authentication, confidentiality, anonymity, and integrity, and in the context of specific attacker models that capture different classes of attacker capabilities, giving rise to threats to the system.

We cover a variety of approaches to formal analysis and verification, including those founded on semantics, games, simulation, equivalence, and refinement. This includes both logic-based approaches (where requirements are expressed as logical statements) and behavioural approaches (given by models of secure behaviour). Our emphasis is on state-of-the-art practical applications of formal methods, enabled by tool support. Hence, the KA covers representative examples of mature tools for these approaches as applied in practice, including general-purpose theorem provers such as Isabelle/HOL and Coq, satisfiability solvers such as Z3, model checkers such as SPIN, FDR, and PRISM, and more specialised security-specific verification tools such as Tamarin, ProVerif, CryptoVerif, and EasyCrypt. We also cover representative examples of tool support for program development and code analysis.

Finally, we provide real-world examples where formal methods have been effective in verifying the security of systems and their components or where analysis has been instrumental in identifying vulnerabilities.

Structure. We have structured this KA along three dimensions: foundations and methods for modelling systems, types of systems, and level of abstraction. After motivating the use of formal methods for security, we survey the different foundations and methods, along with associated tools. We subsequently consider their application to different kinds of systems, as well as differentiating between levels of abstraction, such as programs versus designs. In particular, we explore hardware, protocols, software and large-scale systems, and system configuration. These categories overlap, of course (e.g., systems are built from hardware and software) and a formal method may be used across multiple categories (e.g., methods for the analysis of side-channels or secure information flow are used for both hardware and software). Nevertheless, these categories serve as a convenient structure to introduce the different approaches, and to highlight their scope and reach with respect to different formal analysis problems.

CONTENT

13.1 MOTIVATION

Formal methods have come a long way since the pioneering work of Floyd, Dijkstra, and Hoare on assigning meaning to programs and using deductive methods for the formal verification of small imperative programs.

Despite the undecidability of the underlying verification problems, formal methods are being used with increasing success to improve the security of large-scale, real-world systems. Examples of the successful industrial usage of formal methods are found at companies such as Microsoft [1128], Amazon [1129, 1130], and Google [1131]. Concrete examples are given later in this KA.

There are several motivations for the use of formal methods in security, as discussed next.

13.1.1 Inadequacy of Traditional Development Methods

System development often follows the traditional cycle of code, test and fix. For security-critical systems, “test and fix” becomes “penetrate and patch” as errors can lead to security vulnerabilities. For example, a program missing a check when writing into memory can be exploited by a buffer-overflow attack. Alternatively, a design that fails to check all access requests can lead to unauthorised resource usage. The effects of even small errors can be disastrous in practice. As explained in the Secure Software Lifecycle Knowledge Area (Section 17.1), some attackers are very skilled at finding and exploiting even obscure bugs. Formal methods offer the possibility of improving this cycle by rigorously establishing that systems meet their specification or are free of certain kinds of bugs.

As we shall see, formal methods can be used during different phases of system development and operation, and can be applied to different kinds of system artifacts.

- **System design**: They can be used to specify and verify (or find errors in) designs of all kinds. A good example is security protocols, considered in Section 13.4, where formal methods have made great strides in improving their security.
- **Code level**: As explained in the Software Security Knowledge Area (Chapter 15), programs have bugs, which often represent security vulnerabilities. Indeed, with the right tools,

finding bugs is easy [1132, 1133]. As we describe in Section 13.2.3, formal methods tools for code range from simple static analysers that verify "shallow" properties (e.g., ensuring the absence of specific kinds of security bugs) in very large code bases, to full-fledged interactive verification that can verify deep properties, but usually in smaller programs. We consider the application of formal methods to programs of different kinds in Section 13.5.

- **Configuration level**: The security of systems also depends on how they are configured. For example, access control mechanisms require a specification of who is authorised to do what. We address the application of formal methods to configurations in Section 13.6.

13.1.2 Towards More Scientific Development Methods

The previous discussion positions formal methods as a way to reduce security-relevant bugs. However, one can approach the problem from the other end: as a quest to place the development of secure systems on a firm mathematical ground. This quest for mathematical methods to develop programs (more generally, systems) that behave as they should, rather than merely to catch bugs, was a driving motivation for the early pioneers in formal methods. The essence of this position is that programs and systems can be viewed as *mathematical objects* that can be specified and reasoned about using mathematics and logic. For example, imperative programs can be given semantics — operational, denotational, or axiomatic [1134] — which can be embedded in a theorem prover and used to formalise and reason about programs and their properties [1127]. Under this view, specifications are essential for documenting what programs should do and proofs are essential for ensuring that the programs do it.

Both motivations for formal methods, in this subsection and the previous one, are standard and independent of security. But security adds several challenges. First and foremost, the environment is adversarial: one assumes it contains *adversaries* (also known as *attackers*) who try to attack the system and possibly violate its intended security properties. See the Adversarial Behaviours Knowledge Area (Chapter 7) for a discussion on the kinds of adversaries and adversarial behaviours one finds in practice. Indeed, security is only meaningfully understood with respect to an *adversary model* that specifies a class of attackers in terms of their capabilities or behaviours. To underscore this point, consider protecting information on a computer's disk. Network and file system access control might protect the information on the disk against a remote network attacker, but fail to protect it against an attacker with physical access to the computer who can remove the disk, read out its contents, and even forensically analyse it to reconstruct deleted files. Hence when reasoning about systems and their security, part of the specification must include a model of the adversary; either this is made explicit (for example, when reasoning about security protocol in Section 13.4), or it is implicit and part of the argument used to justify the analysis method.

Second, some security properties differ from those properties considered in traditional program correctness in that they are not trace properties. A *trace property* is a property (or set) of individual system executions. In contrast, some security properties are *hyperproperties* [1135] defined as properties of sets of executions. Note that interest in hyperproperties is not unique to security; hyperproperties have been studied in other contexts such as process calculi. However, hyperproperties play a particularly prominent role in security in specifying central notions like confidentiality or integrity. We will return to the distinction between properties and hyperproperties in Section 13.2.

In the quest to raise the level of rigour for security arguments, some researchers have called

for the development of a *science of security*. Such a science could embrace not only rich specifications and verification methods, but also laws for reasoning about security that relate systems and adversaries with security properties (or policies). Taking an example from [1136]:

> "For generality, we should prefer laws that relate classes of attacks, classes of defenses, and classes of policies, where the classification exposes essential characteristics. Then we can look forward to having laws like "Defenses in class $\mathcal{D}$ enforce policies in class $\mathcal{P}$ despite attacks from class A" or "By composing defenses from class $\mathcal{D}'$ and class $\mathcal{D}''$, a defense is constructed that resists the same attacks as defenses from class $\mathcal{D}$."

The question is still open on how such a science should look and how it would relate to traditional sciences, with their notions of theories about the empirical world and refutation via experimentation and observation. Indeed, an in-depth analysis and critique of different approaches to developing a science of security was provided by [1137] who note, for example, that the history lessons from other sciences first need to be better accounted for. In addition, formal verification, while unquestionably valuable, is only one part of a security rationale. One often fails to validate the mapping between models and assumptions to actual systems and adversaries in the real world [1138]; in fact, this validation between mathematical abstractions and the real world falls outside of the domain of formal proof [1139]. A real-world adversary will not be deterred from attacking a system just because a mathematical theorem apparently rules this out.

13.1.3 Limitations

Section 13.1.2 touches upon a central limitation of formal methods: the faithfulness of the models. Namely, one verifies properties of models, not the actual systems used. Hence the quality of the results depends on the quality of the model. This is a general problem for all formal methods, although runtime verification methods, discussed in Section 13.2.3.4, partially mitigate this problem as they analyse the actual running system rather than a separate model of the system. The use of executable system models, supporting simulation and model animation, can also help uncover system formalisation errors. For security, adversary modelling is particularly critical and the capabilities of adversaries are notoriously difficult to model accurately given human skill and creativity in attacking systems.

A second limitation of formal methods is that one may simply fail to specify all relevant system requirements, including relevant security requirements. Requirements may also be specified incorrectly. Formal methods that return counter examples, such as model checkers, can to some extent help counter the problem of incorrect property specifications (or mistakes in the system model), but they will not help in revealing overlooked system properties. Bridging requirements, which are in the heads of the different system stakeholders, with formal models is a difficult problem that is ultimately outside of the domain of formal methods themselves.

To illustrate the above two points, consider full-stack verification, where the idea is to verify as much of the entire system as possible, rather than just fragments of it. In particular, the aim is to bridge layers of the software-hardware stack such as high-level programs, low-level assembly programs, the operating system that programs run on, and the underlying hardware. In Section 13.5.6 we give an example of such a system stack where all levels are linked by theorems about the translations provided by compilers and assemblers. The theorems and their proofs provide rigourous mathematical guarantees about the stack's contents (e.g., programs, assembly code, operating system, and hardware) and the relationships between

the stack's layers. But the two limitations mentioned above, concerning the top and bottom of the stack, remain. That is, are the relevant properties correctly specified for the high-level programs and does the low-level hardware actually conform to its model?

Finally, there are theoretical and practical limitations regarding the verification methods and tools themselves. Rice's theorem tells us that any nontrivial property about Turing complete languages is undecidable. So we should not hope for a push-button verification tool that will always terminate and can verify all relevant properties of arbitrary systems in a sound and complete way. In practice, compromises must be made. The tools may approximate behaviours (giving false negatives or false positives depending on the approximation, while still being useful), or target only specific classes of programs or properties, or require human interaction, as is the case of interactive theorem proving. Another drawback is that many tools are demanding in terms of the expertise and effort required, and these demands may be hard to satisfy in projects with limited resources for quality assurance.

13.2 FOUNDATIONS, METHODS, AND TOOLS

[1135, 1140, 1141, 1142, 1143, 1144]

In this section we will take a broad, high-level view of approaches to reason about the security of systems. Note that by *system*, we include hardware and software at all levels of the system stack, and at different levels of abstraction. In some cases, we will move between the terms system, program, and process, to reflect the terminology used in the relevant literature.

13.2.1 Properties of Systems and Their Executions

13.2.1.1 Trace Properties

In formal methods, it is common to take an abstract view of systems and their behaviours. System executions are modelled as finite or (for non-terminating systems) infinite sequences

$$\pi \triangleq s_0\, s_1\, s_2\, \cdots,$$

where the s_i belong to an alphabet Σ. Possible interpretations of the s_i are system states, atomic actions, or even state-action pairs. Such a sequence is called a *trace*.

Under this view, a system defines a set of traces. A *property* can also be defined by a set of traces, and is, unsurprisingly, called a *trace property*. System correctness can then be defined in terms of set inclusion: the traces of the system are included in the traces of the property, i.e., the system's behaviour satisfies the property.

In security, many relevant properties are *safety properties*. Informally they stipulate that something "bad" or "undesirable" does not happen during system execution [1140, 1145]. An invariant is a standard example of a safety property, expressing that the system never reaches an undesired state. In more detail, an invariant is a property of states that should hold of every reachable system state. It is defined by a subset of Σ (the "good" states) and can be extended to traces $s_0\, s_1\, s_2\, \cdots$, whereby a trace satisfies an invariant when every s_i satisfies it. An example is the invariant "only the owner of a file may view its content"; this is a safety property where the bad thing is reaching a system state where a file is being viewed by someone other than its owner. Another example of a safety property, expressed directly as a trace property and

formalisable in a temporal logic like linear temporal logic (LTL), would be that "withdrawing funds requires first entering a PIN." In temporal logic this might be expressed as:

$$\Box(\text{FundsWithdraw} \rightarrow \blacklozenge\text{EnterPIN}) ,$$

where the two propositions are interpreted as the actions of withdrawing funds and PIN entry and $\Box$ and $\blacklozenge$ are the LTL operators for "always" and "sometime in the past", respectively. Note that this safety property is not an invariant as it is not a property of states.

The notion that something bad never happens seems to be a good fit for many security properties, where the bad thing might be some unauthorised action. Safety properties also have the attractive feature that, when they are violated, they are finitely falsifiable. Namely, a finite trace is enough to witness a violation since once a safety property is violated, the breach can never be rectified by some extension of the trace.

Not all trace properties are safety properties. In particular, *liveness properties* do not constrain the finite behaviour of a system but rather impose requirements on its infinite behaviours. A typical example is that some event occurs infinitely often or that, possibly under some conditions, something "good" eventually happens. In practice, liveness properties are much less common in security than safety properties as they abstract from *when* events occur. Although this is meaningful in verification, e.g., to rule out infinite loops, in security we often need more concrete guarantees. For example, if an authentication server should respond to requests, it is not that helpful to know that it will *eventually* do so. In practice some upper bound on the response time is required, which thereby turns the property into a safety property. Finally, note that some trace properties are neither safety nor liveness, but rather the conjunction of a safety and liveness property [1140].

Trace properties can be established using model checkers, notably when the systems generating them are finite-state, or using theorem provers. For safety properties, their finite falsifiability also makes them well-suited to runtime monitoring. We describe this in more detail in Section 13.2.3.

13.2.1.2 Hyperproperties

Some security properties are not properties of traces but rather hyperproperties, which are properties of *sets* of traces. To determine whether a system satisfies a hyperproperty it is no longer sufficient to examine each of the system's traces individually, one must instead examine the entire set of traces.

Let us start with a simple illustrating example, from the domain of side-channel analysis, also discussed in Section 13.3.2. Consider timing side-channels in a setting where adversaries cannot only observe system input and output, but also how much time functions, like encryption, take to be executed on their input. This could be modelled using *timed traces*: events modelling function computations are augmented with the time required for the computation. If a function's computation does not admit a timing side-channel, then the time required to compute the function should be independent of any secret input. In other words, the time taken to execute on any secret is the same as the time taken to execute on any other secret. Analysing any individual trace π would be insufficient to establish this. One must examine the set of all of the system's traces. Actually, in this particular example, it would suffice to examine all *pairs* of system traces. This is an example of what is known as a *2-safety hyperproperty* [1135].

Hyperproperties were first studied, without being so named, in the 1980s when researchers started investigating noninterference [1141] and related secure information flow properties [1146, 1147]. Hyperproperties emerged almost 30 years later [1135] as a general formalism capable of specifying such notions. As such, hyperproperties represent a generally useful contribution from the security community to the formal methods community. Let us consider this in more detail focusing on noninterference, the prevailing semantic notion for secure information flow [1148]; see also Section 13.5.1 for more on this topic.

Noninterference is studied in a setting where users and the data they input into systems have different classifications and the actions of privileged users should not affect, or influence, those of less privileged users. In other words, the secret (or high) inputs of privileged users should not interfere with the public (low) outputs observed by non-privileged users. Put more simply, the public outputs are independent of the secret inputs.

To see how this can be formulated as a hyperproperty, consider a system with inputs I that receives high inputs h and low inputs $i \in I \backslash \{h\}$ and produces low outputs o. In this context, we might formalise noninterference by requiring that all system traces π and π' whose inputs differ only in h, have, at all times, the same outputs o. This is a hyperproperty whose formalization involves two traces (technically, it is another example of a 2-safety hyperproperty) and it can be formalised in a hyper-temporal logic supporting explicit quantification over traces, like hyperLTL [1149, 1150] as follows:[1]

$$\forall \pi . \forall \pi' . \Box (\bigwedge_{i \in I \backslash \{h\}} i_\pi = i_{\pi'}) \Rightarrow \Box (o_\pi = o_{\pi'}) \, .$$

The original formalisation of noninterference by Goguen and Meseguer [1141] was in the spirit of the above definition, although somewhat different in its details. It was stated in terms of deterministic state-transition systems and the purging of high inputs. Namely, the commands entered by users with high clearances can be removed without affecting the outputs learned by those users with low clearances. Since this original formulation, researchers have proposed numerous generalisations and variants [1148], which can naturally be cast as hyperproperties. These include observational determinism [1151] (stating that every pair of traces π and π' with the same initial low observation are indistinguishable for low users, i.e., the system appears deterministic to these users), generalised noninterference [1152] (which allows for nondeterminism in the low outputs, but requires that they are unaffected by high inputs) and numerous variants of noninterference based on interleavings of traces, such as those considered by McLean [1153] and other researchers.

Hyperproperties have their own associated theory with notions like hypersafety, which generalises safety properties to sets of traces, and hyperliveness. For example, a hyperproperty is hypersafety when, to disprove it, it suffices to show a counterexample set of finite traces. There are also model checkers explicitly built for hyper-temporal logics, discussed further in Section 13.2.3.

[1] In this hyperLTL formulation, taken from [1150], the π and π' are path variables (semantically, they range over the infinite traces of a given Kripke structure) and atomic formulas of the form a_π refer to the occurrence of an atomic proposition a at the current (starting) position of the path π.

13.2.1.3 Relations on Systems

The previous notions focused on properties expressed in terms of system behaviours, i.e., their execution traces. One may alternatively define relations directly between systems themselves. A standard setting for this is where a system, possibly concurrent or distributed, is represented as a *labelled transition system* (LTS) or some variant thereof. In its simplest form, an LTS is a triple $\langle Q, A, \rightarrow \rangle$ consisting of a set of states Q, a set of actions (or events) A, and a transition relation $\rightarrow \subseteq Q \times A \times Q$, where $q \xrightarrow{a} q'$ represents the transition from state q to q' taking place when the action a occurs. An LTS may be extended to a *labelled Kripke structure* by additionally labelling states by predicates, from a given set P, that hold in that state. When P is a singleton (or omitted) then the labelled Kripke structure is simply called a *Kripke structure*. Moreover, when the labelled Kripke structure has an initial state, it is sometimes called a *process graph*. Finally, it is called a *(nondeterministic)* finite automaton when the states and set of actions are finite and the single predicate denotes acceptance.

There has been substantial research in the formal methods community on *process theory*: how to represent systems by processes and define and verify statements about the relationship between processes. In addition to directly representing processes as variants of labelled transition systems, one can use richer or more specialised languages that emphasise aspects like concurrency or communication. Options here include structures such as Petri nets, event structures, or process calculi like CCS and CSP.

Numerous relations have been defined and studied to compare processes. They capture different notions like two processes are interchangeable or one process implements another, for instance. Such relations range from being very strong, like the isomorphism of process graphs, to being much weaker, like the equivalence of their traces, viewing them as automata, as well as relations in between like simulation and bisimulation [1154, 1155, 1156]. Demanding that two systems have the same structure (isomorphism) is usually too strong in practice as this is not something that a system's user, or an attacker, can directly observe. The weaker properties of trace equivalence or trace containment are relevant for checking safety properties, where one LTS represents an implementation and the other a specification. Bisimulation is stronger than trace equivalence as it additionally identifies systems with the same branching structure.

Formalising properties as process equivalences is common in security. Different equivalences are used, such as observational equivalence (discussed below), testing equivalence, and trace equivalence [1157, 1158, 1159]. A standard cryptographic notion is *indistinguishability*, which formalises that an adversary cannot distinguish between two protocols, usually either the same protocol using different secrets, or a real protocol and a simulation of the protocol using random data. Observational equivalence formalises a similar notion in the context of processes, whereby two processes P and P' are observationally equivalent if, intuitively speaking, an observer, who may be an attacker, cannot tell them apart. The processes may compute with different data and the way they compute internally may be completely different, but they appear identical to an external observer. Observational equivalence is also attractive as it is a congruence relation that supports compositional proofs: whenever a process P is equivalent to another process P', then P can be replaced by P' in other, more complex processes.

Observational equivalence for processes in the applied pi calculus [1159] has been extensively studied in the formal methods for security community. This calculus is a process calculus, based on Milner's pi calculus, that supports a term algebra used to model cryptographic operations used, for example, in cryptographic protocols. It is worth noting that robust verification tools exist for this calculus like the security protocol model checker ProVerif [1160], discussed

in Section 13.4. For the applied pi calculus, observational equivalence amounts to a form of labelled bisimulation, which provides proof techniques for establishing process equivalence. Moreover, observational equivalence can be naturally used to formalise privacy-style properties. Applications include formalising the resistance of protocols to guessing attacks, strong secrecy (the attacker cannot distinguish between different values of a secret), and anonymity and unlinkability properties (for example, the attacker cannot distinguish between two processes implementing an election protocol that are otherwise identical, except that one swaps the votes of two arbitrary voters) [1160, 1161, 1162].

13.2.2 Logics and Specification Languages

Formal methods require one or more languages to specify systems and their properties. In practice, one rarely specifies systems directly as labelled transition systems as it is easier to use higher-level languages. Even programming languages or hardware description languages can be used, where programs are given operational semantics in terms of transition systems and there is support for translating between high-level and low-level descriptions. Alternatively, one might work with a specialised language like a process calculus that emphasises particular aspects of systems, such as concurrency and interaction. We have already mentioned examples of specialised languages, such as the applied pi calculus, which is well suited for specifying security protocols.

In many cases, the specification language is the language of a logic (or a logical theory) that also provides a sound way to reason about statements in the language, in particular, to determine their validity or satisfiability. The specific language used and statements made depend on what is being formalised and what should be proven.

There is no general agreement on what is the ideal logic or language to use for program specification; one's choice of logic and associated verification system is partly a matter of taste, expressiveness, desire for automation, and education. The specification formalisms used in practice range from weak, relatively inexpressive logics that have decision procedures, to stronger, expressive logics that usually require interactive theorem proving. Weaker logics include propositional logic and propositional temporal logics. Stronger expressive logics include higher-order logics, both classical and constructive. An intermediate option would be to use a logic based on (some fragment of) first-order logic with equality and background theories.

The weaker logics are limited in what they can formalise. For example, propositional logic cannot talk about relations and functions. This is possible in first-order logic, but only for some relations and functions. For instance, one cannot formulate inductively defined relations, which are relevant when reasoning about the operational semantics of systems or protocols, e.g., to capture the reachable states. When the logic used is insufficiently expressive, then such notions must be approximated. For example, using the technique of bounded model checking, one can encode in propositional logic the traces of a system up to some given length and then use decision procedures based on satisfiability or SMT solvers to verify properties of these bounded-length traces [1163].

A middle ground is to use a specification language based on a particular fragment of first-order logic or a given first-order theory such as primitive recursive arithmetic or Peano arithmetic, which would allow formalising recursive computations over the natural numbers and other data structures. The ACL2 theorem prover [1164] is an example of a theorem prover built on such a logic that has been used to formalise algorithms and systems represented in pure Lisp.

The logic used enables a high degree of automation in constructing proofs, in particular for proofs by mathematical induction. ACL2 has been applied to verify the functional correctness and security properties of numerous industry-scale systems [1165].

Alternatively, one might prefer a richer logic, trading off proof automation for expressiveness. Higher-order logic is a popular example. It can either be based on classical higher-order logic, formalised, for example, in the Isabelle/HOL [1166] or HOL-light [1167] systems, or a constructive type theory like the calculus of inductive constructions, implemented in the Coq tool [1168, 1169]. In these richer logics, one can formalise (essentially) all mathematical and logical notions relevant for reasoning about systems. These include systems' operational semantics, properties, hyperproperties, and other correctness criteria, as well as relations between systems like refinement relations. Nipkow and Klein's textbook on concrete semantics [1127] gives a good overview of how programming language semantics, programming logics, and type systems can be formalised in Isabelle/HOL, with applications, for example, to information flow control.

13.2.3 Property Checking

Arguments about systems' security properties can be machine supported. This ranges from tool-supported audits, used to detect potential vulnerabilities, to proofs that systems are secure. We will explore a range of different options in this section, including those based on proofs, static analysis, and dynamic analysis. All of these improve upon human-based system audits or pencil-paper based security proofs in their precision, automation and (to varying degrees) scalability.

13.2.3.1 Interactive Theorem Proving

Interactive theorem proving is the method of choice for proving theorems when using expressive logics like higher-order logic. In general, interactive theorem proving is used for logics and formalisms where deduction cannot easily be automated, and therefore, humans must interactively (or in a batch mode) guide a theorem prover to construct proofs. This is often the case when inference problems are undecidable or of sufficient computational complexity that human assistance is required to construct proofs in practice.

In contrast with the other methods described in this section, interactive theorem proving is substantially more labor-intensive. Some of the more tedious forms of interaction (arithmetic reasoning, simple kinds of logical reasoning, applying lemmas, etc.) can be offset by integrating decision procedures and other inference procedures into theorem provers. Moreover, many provers are extensible in that users may write tactics, which are programs that implement proof construction strategies and thereby automate parts of proof construction [1170]. Nevertheless, even with automation support, it may still be necessary to guide the theorem prover by establishing lemmas leading up to a proof. For example, for program verification, one may need to provide loop invariants, or appropriate generalisations for proving inductive theorems. This effort, although time-consuming, has a side benefit: the human carrying out the proof gains insight into why the theorem holds.

Interactive theorem proving has been applied to numerous large-scale verification projects where the verification time is measured in person-years. We will present one example of this, the verification of the seL4 microkernel, in Section 13.5.4.

13.2.3.2 Decision Procedures

Advances in decision procedures and other forms of algorithmic verification have played an essential role in increasing the usage and acceptance of formal methods across a large range of industry applications. Even humble propositional logic is relevant here, as it often suffices to encode and reason about (finite) system behaviours, system configurations, and other system artifacts. There now exist highly effective constraint solvers for the satisfiability problem (SAT) in propositional logic. Due to advances in algorithms, data structures, and heuristics such as conflict-driven backtracking, SAT solvers like Chaff [1171], Grasp [1172], and MiniSAT [1173] can determine the satisfiability of formulas with millions of clauses and solve real problems in program analysis.

SAT-based procedures (as well as other procedures, such as those used for computer algebra [1174]) have also been successfully extended to reason about the satisfiability of fragments of first-order logic, so-called satisfiability modulo theories (SMT) [1142, 1175] as embodied in tools like Z3 [1176], CVC4 [1177], and Yices [1178]. The languages that such tools handle are fragments of first-order logic restricted syntactically or semantically, e.g., by syntactically restricting the function or predicate symbols allowed or by fixing their interpretation. Such restrictions can lead to decidable satisfiability problems and allow for specialised algorithms that exploit properties of the given fragment. The resulting decision procedures may work very well in practice, even for fragments with high worst-case computational complexity. Examples of theories supported by SMT solvers include linear and nonlinear arithmetic (over the reals or integers), arrays, lists, bit-vectors, IEEE standard floating-point arithmetic, and other data structures.

SAT and SMT procedures have numerous applications for reasoning about properties of systems. These include determining the validity of verification conditions, symbolic execution, bounded and unbounded model checking, software model checking, predicate abstraction, and static analysis, to name but a few examples. We shall see examples in other sections of this chapter of how decision procedures are used to support other deduction methods.

Many security properties can be expressed as temporal properties, in particular safety properties. When systems, or their abstractions as models, are finite-state, then model checking algorithms provide a decision procedure for determining that the system model satisfies the specified properties [1179, 1180]. For small state spaces, one may explicitly represent and exhaustively search them. For larger state spaces, more sophisticated techniques have been developed. For example, both the system and property can be represented as automata and efficient algorithms are used to determine if the system specification conforms to the property, for notions of conformance that correspond to language inclusion, different refinement orderings, or observational equivalence. The transition systems of the automata can be represented symbolically, e.g., using binary decision diagrams [1181], and strategies like partial-order reduction [1182] can be used to reduce the search through the state-space of concurrently executing automata.

Numerous successful model-checkers have been developed. Aside from incorporating different algorithms and data structures, they employ different system and property specification languages. System specification languages include variants of automata, process algebra, and higher-level programming languages for describing concurrently executing processes. Property specification languages include temporal logics like LTL, CTL, and probabilistic variants. Examples of effective, general purpose model checkers are the explicit-state LTL model checker SPIN [1183], the symbolic model checker NuSMV supporting both LTL and CTL [1184], the model checker FDR2 supporting programs and properties specified in the process calculus

CSP and different refinement relations [1185], and the model checker PRISM for modelling and reasoning about systems with random or probabilistic behaviours [1186].

Particularly relevant for security are algorithms and tools that have been developed for model checking systems with respect to hyperproperties [1150]. Also relevant are the specialised model checkers developed for security protocols, described in Section 13.4.1.2. These tools support protocol specifications involving cryptographic functions and effectively handle the non-determinism introduced by an active network adversary.

13.2.3.3 Static Analysis

Static analysis (cf. Malware & Attack Technologies Knowledge Area (Section 6.3.1.1) and Software Security Knowledge Area (Section 15.3)) refers to a broad class of automated methods that analyse programs statically, that is without actually executing them. Most static analysis methods operate on a program's source code or some kind of object code, rather than at the level of designs. Despite the undecidability or intractability of most questions about program behaviour, static analysis attempts to efficiently approximate this behaviour to either compute sound guarantees or to find bugs. The behavioural properties analysed are usually limited to specific problems like the detection of type errors or division-by-zero. For security, the properties might focus on vulnerabilities such as injection attacks or memory corruption problems. In contrast to verification using theorem provers, the emphasis is on completely push-button techniques that scale to industry-sized code bases. Indeed, in the interest of minimal user interaction (and thereby greater acceptance), many static analysis methods do not even require a formal specification but instead target certain kinds of "generic" problems that often lead to runtime exceptions or security vulnerabilities in practice. In other words, they target "shallow", but meaningful, general properties analysed on huge code bases rather than "deep" system-specific properties analysed on relatively small programs and designs.

The methods used vary from tool to tool. As observed in the Software Security Knowledge Area (Section 15.3) a major divide in the tools, and also in the research community, concerns soundness. Some tools are heuristic-based checkers and explicitly reject the requirement for soundness [1133]. For other tools, soundness is imperative and, while false positives may be tolerated, false negatives are not acceptable.

Many of the sound methods can be cast as some kind of abstract interpretation [1143] where the program's control-flow graph is used to propagate a set of abstract values through the different program locations. The abstract values are approximate representations of sets of concrete values — for example, the abstract values may be given by types, intervals, or formulas – and are determined by an abstraction function, which maps concrete values to abstract values. The propagation of abstract values is continued until these values cease to change. Mathematically, this can be understood as the iterative application of a monotone function until a fixed point is reached.

Industrial-strength static analysis tools are extremely successful and widespread. Early examples of such tools, from the 1970s, were type checkers and programs like LINT, which enforced the typing rules of C even more strictly than C compilers would. More modern tools leverage theoretical advances in abstract interpretation, constraint solving, and algorithms for inter-procedural analysis. Examples of such tools include Grammatech's CodeSonar, Coverity's Prevent (for an early account, see [1133]), or Micro Focus' Fortify tool.

As a concrete example of a successful tool, the Fortify tool classifies security bugs into different categories and employs specialised analysis techniques for each category. As

an example, one category concerns input validation, and includes vulnerabilities such as buffer overflows, command injections, cross-site scripting, HTTP response splitting, path manipulation, reflection abuse, and improper XML validation, to name but a few. One static analysis technique that the Fortify tool applies to these problems with considerable success is *taint analysis* [2], which tracks how possibly untrusted inputs can flow unchecked through the program and thereby affect (or taint) arguments to function calls or outputs to users. For instance, if user input could flow through the program and be used as part of an SQL query, then this would represent a potential SQL injection vulnerability. Other categories, with associated analysis techniques, focus on API abuse, improper use of security features, or encapsulation issues.

Two modern examples of the large-scale applications of formal methods are the use of static analysis (and other) tools at Google and Amazon. Google incorporates a variety of analysis tools into their developer workflow [1131]. The tools are, by the authors' own account, "not complex" and include bug finding tools that extend the compiler (including abstract-syntax-tree pattern-matching tools, type-based checks, and unused variable analysis) and more sophisticated tools that support code audits. Unlike the compiler-based tools, the audit tools incorporate abstract-interpretation techniques that may generate false positives, that is, they sometimes report on potential problems that are not actual problems. The Google tools also suggest possible fixes to the problems they report.

Amazon [1130, 1187] has similarly had considerable success applying formal methods to prevent subtle but serious bugs from entering production. In contrast to Google's practices, as reported in [1131], Amazon uses a combination of automated static analysis tools alongside techniques based on explicit specification, model checking and theorem proving. So push-button techniques are combined with more time-intensive specification and verification for critical components or properties. For example, multiple Amazon teams are using TLA+ (which is a language based on a combination of set theory and temporal logic, with associated tools) to specify systems and carry out proofs about their properties. [1188] reports that these tools are extensively used to reason about security-critical systems and components, including cryptographic protocols, cryptographic libraries, hypervisors, boot-loaders, firmware and network designs.

13.2.3.4 Dynamic Analysis

Runtime verification (cf. Malware & Attack Technologies Knowledge Area (Section 6.3.1.2) and Software Security Knowledge Area (Section 15.3.2)) is a general approach to verifying properties of systems at runtime [1144]. In this approach, one specifies the property ϕ that a system should satisfy and uses a tool to verify that ϕ holds for the actual behaviour(s) observed during execution. The tool may work by running alongside the system and checking that its observed behaviour agrees with the property, or the checks may even be woven into the system itself to be checked as the system executes [1189]. When the system fails to satisfy the property ϕ, a relevant part of the current execution trace may be output as a counterexample, witnessing ϕ's failure.

In more detail, runtime verification is often implemented by a monitoring program. This program observes the behaviour of a target system, which is a finite, evolving trace $\pi = s_0 \cdots s_{n-1}$ at some level of abstraction, e.g., the program's inputs and outputs, or perhaps

[2] Taint analysis is also implemented in some tools as a dynamic analysis technique where taint information is tracked at runtime.

(parts of) its internal states. Each new event s_n produced by the system extends the trace π to $s_0 \cdots s_n$ and the monitor determines whether this extended trace satisfies or violates ϕ, and violations are output to the user. Depending on ϕ's structure, violations can be determined immediately once they occur (e.g., for formulas in past-time temporal logics). Alternatively, for other formulas (e.g., those involving future-time operators), the monitor may not be able to report a violation caused by the current event s_n until some time in the future.

Runtime verification has trade-offs compared with other techniques like model checking and theorem proving. Rather than checking whether *all* system behaviours satisfy the desired property ϕ, one checks just that the observed behaviour satisfies ϕ. In this sense, runtime verification amounts to *model checking a trace*. Unlike the previously discussed verification approaches, runtime verification methods produce relatively weak verification guarantees: they can only establish ϕ for the executions they witness. However, in contrast to model checking and theorem proving, these are guarantees with respect to the actual system executing in its real environment, rather than with respect to some model of the system and its environment. Moreover, at least for safety properties, runtime verification does not produce false positives like static analysis does. Finally, the methods used are completely automated and often efficient and scalable in practice.

Recently, progress has been made on runtime monitoring techniques for hyperproperties [1190]. In this setting, one may need to give up some of the advantages of runtime verification for trace properties. In particular, since the monitor observes just one trace, but a hyperproperty refers to sets of traces, the monitor may have false positives due to the need to approximate the other unobserved traces.

Runtime verification has been successfully deployed in numerous safety and security-critical settings. NASA is a case in point, where different approaches to runtime verification have been used for the real-time monitoring of Java programs [1191]. [1192, 1193] provide examples of using the MonPoly tool [1194] to monitor security policies and data protection policies of distributed systems.

13.3 HARDWARE

[1195, 1196, 1197, 1198]

Hardware security and attacks on hardware were presented in the Hardware Security Knowledge Area (Chapter 20). Formal methods are now widely used when developing hardware [1195] and are indispensable for assurance. As explained in the Hardware Security Knowledge Area (Section 20.2), the Common Criteria standard can be used to guide hardware security evaluations, and its higher Evaluation Assurance Levels mandate the use of formal models and formal methods in this process. More generally, hardware verification is a well-developed research area, see for example the surveys [1199, 1200]. We highlight here some security-relevant topics in this field.

13.3.1 Hardware Verification

Microprocessors and other hardware components can be formulated at different levels of abstraction ranging from high-level Hardware Description Languages (HDLs) at the register transfer level (e.g., in HDLs like Verilog or VHDL) to low-level descriptions at the transistor level. Once the description languages used are given a semantics, one may use decision procedures and theorem provers for property verification.

Historically, much of the progress in decision procedures and model checking was driven by hardware verification problems. For example, equivalence checking for combinational circuits led to advances in propositional satisfiability procedures, e.g., data structures like ordered binary decision diagrams [1181]. Verifying temporal properties of sequential circuits was a central motivation behind the development of efficient model checking algorithms. Moreover, by giving HDLs a semantics in a language for which verification tools exist, one can directly utilise existing tools for those languages. For example, [1201] shows how designs in the Verilog HDL can be translated into equivalent ANSI-C programs. The property of interest is later instrumented into the C program as an assertion. Standard verification techniques can be used to validate the assertion such as bounded model checking, path-based symbolic execution, and abstract interpretation.

For large-scale hardware systems, some form of semi-automated or interactive theorem proving is usually required. For example, ACL2 has been used to verify a number of microprocessors due to its support for automated inductive reasoning (useful to establish invariants about system behaviour) and because designs can be efficiently executed (useful both for simulation and for proofs). The ACL2 verification of a microprocessor is described in [1202], which features complex control mechanisms such as out-of-order issue and completion of instructions, speculative execution with branch prediction, and memory optimisation such as load-bypassing and load-forwarding. Such verification can be part of a Common Criteria certification. For example, [1203] reports on the certification of the AAMP7G Microprocessor, where the ACL2 theorem prover was used to verify that its microcode, implementing a separation microkernel, was in accordance with EAL 7 requirements (the highest evaluation level in the Common Criteria) and guarantees security-relevant aspects such as space partitioning.

13.3.2 Side-Channels

Particularly relevant from the standpoint of hardware security is the detection and removal of side-channels. A side-channel is an unintended communication channel where information presumed secret, such as cryptographic keys, can be leaked. Examples of hardware side-channels are presented in the Hardware Security Knowledge Area (Chapter 20), including side-channels and attacks based on timing and power analysis.

Abstractly, detecting side-channels requires analysing hardware (or programs) to determine if one can observe differences in behaviour, depending on values of secret data. This amounts to checking noninterference of secret values on observable outputs, where the observations can include measurements like timing or power usage. Note that analysis of noninterference and other secure information flow notions are typically studied for software (see Section 13.5.1 for more on this) but some techniques are cross-cutting and have also been applied to hardware. One challenge in reasoning about noninterference for both hardware and software is that, even in well-designed and well-implemented algorithms, often some information can leak. For instance, the failure to decrypt a message or the failure to log-in with a guessed password reveals some limited information about the secret, namely that the guessed key or password

was incorrect. One option here is to explicitly specify where information is allowed to leak in controlled ways. This is called *declassification* [1204]. Another option, which we explore further below, is to measure *how much* information can be leaked through a side-channel.

There is a large body of work on side-channel analysis, focusing on foundations and analysis methods as well as attacks and countermeasures. Early research, like [1205, 1206], developed generic models and frameworks for proving that hardware implementations are free of side-channels, for measuring the amount of information that can be leaked when side-channels exist, and for exploring possible countermeasures to eliminate leakages. As an example, [1207] proposes a mathematical model, based on abstract virtual-memory computers, that can be used to develop provably secure cryptography in the presence of bounded side-channel leakage. This model was further specialised into a framework for evaluating side-channel attacks by [1196], which measures information-theoretic leakage with respect to adversaries who interact with the system in *non-adaptive* ways.

While the above research focused more on foundations, subsequent research moved closer to tool-supported formal methods. In [1208], Köpf et. al. propose a framework that, like [1196], uses information-theoretic metrics to measure side-channel leakage. However in this work, the metrics are used to quantify the information revealed to an *adaptive* adversary who can make timing or power measurements, based on the number of interactions with the system under attack. The essential idea is to search over all adversary attack strategies, which are the adaptive decisions that the adversary can make, and use information-theoretic entropy measures to express the adversary's expected uncertainty about the secret after interacting with the system using each strategy. This framework is instantiated with a hardware description environment, thereby supporting the automated detection and quantification of information leakages in hardware algorithms, like those implementing cryptography.

More recent research incorporates ideas from program analysis to track the observations that can be made by adversaries. For example, for reasoning about micro-architectural side-channels, the cache audit tool [1197] uses an abstract interpretation framework, with a suitable abstract domain, to track information about the possible cache states. This is used to quantify the information leakage possible by measuring the timings associated with cache hits and misses that can occur during execution on different control flow paths. The tool takes as input a program binary and a cache configuration and derives formal, quantitative security guarantees against adversaries that can observe cache states, traces of hits and misses, and execution times.

Researchers have also developed techniques that use formal models and program analysis to either *generate* programs that are, guaranteed to execute in constant time (independent of secret data) or *repair* programs with timing leaks by ensuring that all branches take the same time [1209, 1210, 1211, 1212]. There is, however, still considerable research left to do. Most current work stops at the level of intermediate representation languages, rather than going all the way down to machine code. Another deficit lies in the fact that the models are not detailed enough to say what happens under speculative or out-of-order execution at the hardware level. Overcoming these limitations remains an important challenge.

13.3.3 API Attacks on Security Hardware

Even when hardware implements the "right" functionality and is free of side-channels, it may still be vulnerable to attacks that leak secrets. An active research area has been on attacking security hardware by abusing its API, effectively by sending the hardware valid commands but in an unanticipated sequence that violates the designer's intended security policy [1198]. Such attacks have been carried out against a wide variety of security-critical hardware including cryptographic co-processors and more specialised processors designed for automated teller machines, pay-TV, and utility metering. The problem of API attacks is cross-cutting as they are also applicable to software libraries. API attacks can also be seen as a protocol problem in how the API is used; hence formal methods and tools for security protocols, described in Section 13.4, are relevant for their analysis.

One particularly successful application of formal methods has been the use of protocol model checkers to detect attacks on security tokens over their RSA PKCS#11 APIs. As [1213] shows, given a token it is possible to reverse-engineer a model of its API in an automated way. This model can then be input to a security protocol model checker to search for attacks where sensitive keys are exposed to the adversary.

13.4 CRYPTOGRAPHIC PROTOCOLS

[1214, 1215, 1216, 1217, 1218, 1219]

Security protocols are a superb showcase for formal methods and how their use can improve the security of critical systems and their components. As noted in [1218], security protocols play a role analogous to fruit flies for genetic research: they are small and seemingly simple. However, for protocols, this simplicity is deceptive as they are easy to get wrong. A classic example of this is the attack that Lowe found in the 1990s on a simple three step entity authentication protocol that is part of the Needham Schroeder Public Key protocol [1217].

Formal methods for security protocols have been the subject of considerable research for over four decades; see [1220] for a survey on security protocol model checking and [1221] for a more general survey on computer-aided cryptography. Our focus in this section is primarily on using formal methods to analyse security protocol *designs*; we return to *implementations*, in particular of cryptography, in Section 13.5.2. Design verification has high practical relevance as if designs are not secure, then no (design-conform) implementation will be secure either. In the past, this issue was not taken seriously enough and most protocols were standardised without a clear formulation of either their properties or the intended adversary [1222]. This situation is now improving, albeit slowly. Protocol verification tools are having a practical impact on the design of security protocols that matter, such as ISO/IEC Protocols for Entity Authentication [1223], TLS 1.3 [1224, 1225, 1226], 5G [1227], and the EMV standard for credit-card payments [1228, 1229].

13.4.1 Symbolic Methods

Symbolic models are the basis of many successful formal methods approaches to reasoning about protocols and applications that use cryptography. In the symbolic setting, messages are represented by terms in a term algebra containing cryptographic operators. The adversary is modelled as being active: he controls all network traffic and can manipulate terms, e.g., he can concatenate terms together or decompose concatenated terms into their subterms. Properties of cryptography are typically formalised with equations or inference rules. For example, for symmetric encryption the equation

$$decrypt(encrypt(m, k), k) = m$$

would formalise that all parties (including the adversary) can decrypt any message m encrypted with a key k, by using the same key for decryption. Cryptography is assumed to work perfectly, in an idealised way, in that the adversary cannot do anything more than is specified by such equations. So either the adversary can decrypt a cipher text $c = encrypt(m, k)$ yielding the plain text message m, if he possesses the right decryption key k, or no information about the plain text is leaked. This kind of model of an active network adversary who has access to all message terms, can manipulate them, but cannot "break" cryptography is sometimes called the “Dolev-Yao adversary” after the seminal work of Dolev and Yao [1230].

A security protocol is specified by a set of parameterised processes, called *roles*, that describe the actions taken by the agents executing the role. For example, in a key agreement protocol there might be an initiator role, a responder role, and a key-server role. Protocols are given an operational semantics where agents may play in multiple roles. For example, Alice may be the initiator in one protocol run and a responder in a second run. So in general there may be arbitrarily many role instances. This setup gives rise to a transition-system semantics with an associated notion of trace. The transition system is given by the parallel composition of unboundedly many role instances together with a process describing the Dolev-Yao adversary. The transition system is infinite-state as there can be arbitrarily many processes. Also note that the adversary himself is an infinite-state process; this reflects that he is very prolific in that he can produce and send infinitely many different messages to the other agents running in the different roles. For example, if the adversary has seen a message m, he can always concatenate it with itself (or other messages he has seen) arbitrarily many times, hash it, repeatedly encrypt it, etc. Finally, security definitions are formulated in terms of trace properties, which are often simple invariants. For example, the adversary never learns a key presumed to be secret.

13.4.1.1 Theorem Proving

One way to establish a safety property P about the traces of an infinite-state transition system is to show that P is inductive: it holds for the empty trace and, assuming it holds for an arbitrary (finite) trace π, it holds for all extensions of π by an event s, i.e., $P(\pi\, s)$ follows from $P(\pi)$.

This is the essence of the approach taken by Paulson [1216] where protocols and the actions of the Dolev-Yao adversary are encoded as inductive definitions in higher-order logic. The properties of protocols are also formalised in higher-order logic as properties of finite traces.[3] For example, a common class of properties is that one event s always precedes another

[3] The restriction to finite traces is necessary as protocols are defined inductively as sets of finite traces (rather than co-inductively as infinite traces). However, this is not a restriction in practice, unless one wishes to reason about liveness properties, which is rarely the case for security protocols.

event s'. This is relevant for authentication and agreement properties [1231], for example, when Bob finishes a protocol run in his role then Alice previously finished in her role. Another important class of properties are secrecy properties, e.g., that a session key established in a key exchange protocol is secret even after the subsequent compromise of any long-term keys (perfect forward secrecy).

Given a specification of a protocol, including the adversary and the protocol's intended properties, Paulson then proves by induction, for each property P, that every trace in the protocol's inductive definition satisfies P. The inductive proofs are constructed using Isabelle/HOL [1166]. Theorems are proven interactively, which may require considerable work, say to establish the auxiliary invariants needed for proofs. [1216] reports that the time needed for an expert to construct proofs ranged from several days for small academic protocols to several weeks for a protocol like the handshake of TLS v1.3. Despite this effort, numerous nontrivial protocol were proven this way, such as the TLS v1.3 handshake, Kerberos v1.4, and sub-protocols of SET (Secure Electronic Transactions [1232]).

This inductive method was further refined by [1233, 1234], who use derived proof rules to mimic the backwards search from an attack state to an initial state. Proofs are again constructed in Isabelle/HOL, but the proof rules also support substantial automation. The method reduces the time required for proofs in comparison to [1216] by several orders of magnitude, whereby the security of "academic" protocols can often be proven completely automatically.

In contrast to post-hoc verification, one can use theorem provers to *develop* protocols hand-in-hand with their correctness proofs using step-wise refinement. Refinement [1235, 1236] allows one to decompose the complexity of verifying properties of a complex (transition) system into verifying the properties of much simpler, more abstract systems, and establishing refinement relationships between them. Namely, one starts with a very abstract transition system model M_0 of the desired (distributed) system and refines it into a sequence of increasingly concrete models $M_1, M_2, \ldots M_n$, proving properties of the system at the highest possible level of abstraction, and establishing a refinement relation between each pair of models M_i and M_{i+1}, for $i \in \{0, \ldots n-1\}$. The existence of transitive refinement relations ensure that properties proven for the more abstract models also hold for the more concrete models.

The refinement approach has been applied to security protocols in [1237, 1238], where a refinement strategy is given that transforms abstract security goals about secrecy and authentication into protocols that are secure when operating in an environment controlled by a Dolev-Yao-style adversary. This approach simplifies the proofs of properties and provides insights on why protocols are secure via the proven invariants. Furthermore, since a model may be refined in multiple ways, the refinement steps can be structured as a tree, rather than as a sequence. This fosters the development of families of protocols, given by the models at the leaves of the tree, which share common structure and properties. The refinement method is implemented in Isabelle/HOL and used, for example, to develop a family of entity authentication and key establishment protocols that include practically relevant features such as key confirmation, replay caches, and encrypted tickets.

13.4.1.2 Model Checking Trace Properties

Despite the undecidability of the underlying question of whether a protocol is secure in the symbolic model [1239], it is still possible to build effective analysis tools. Early tools, such as Millen's Interrogator [1240], the Longley-Rigby search tool [1241], and the NRL Protocol Analyzer [1242], were search procedures that exhaustively searched the problem space or some part of it. While this would normally not provide correctness guarantees, these tools were still very effective in finding subtle attacks on security protocols.

One way to approach model checking is to reduce it to a problem that can be tackled using standard algorithmic-verification tools. We give two examples of this. The first is exemplified by the use of CSP and FDR [1218]. CSP is a general language and theory for modelling systems consisting of interacting processes and FDR (and its more recent incarnations like FDR2) is a model checker for CSP. The basic idea is that the system analysed consists of processes describing the different protocol roles which are run in parallel with a process describing the adversary. These processes are either formalised directly in CSP or, compiled from a more abstract description [1243]. The security property is also formalised as a CSP process. FDR is then used to establish, via trace refinement, that the language of the system is contained in the language of the property. Since the FDR tool only handles processes with finite alphabets and finite state spaces, these must be bounded, e.g., by bounding the number of role instances and the size of messages that agents and the adversary can generate. This is a practical limitation since by introducing bounds, one may miss attacks.

A second example of using standard deductive tools is the SATMC model checker [1244]. SATMC takes as input a protocol and property description, given in a high-level input language [1245], and translates them into a propositional logic satisfiability problem, essentially encoding a bounded reachability (or planning) problem: can the adversary drive the protocol into an attack state? This is then solved using an off-the-shelf SAT solver. As with the use of CSP and FDR, the main challenge and limitation is the restriction to a finite state space.

More recent state-of-the-art tools, such as ProVerif [1160, 1215] and Tamarin [1219, 1246], use specialised algorithms to efficiently manage and search the infinite state space defined by the protocol and adversary. Both tools support user-defined cryptographic primitives specified by equations and the verification of trace properties and privacy properties, the latter specified using observational equivalence. ProVerif takes as input a protocol description in a process calculus, called the applied pi calculus, cf.Section 13.2.1.3. It then translates the protocol description into a set of Horn clauses, applying domain specific approximations and abstractions in the process. For example, individual fresh values (used for modelling secrets) are abstracted into sets of fresh values and each action in a process can be executed multiple times. Given the abstracted protocol model, ProVerif uses resolution to determine whether relevant properties hold; these include secrecy and correspondence properties [1247] or observational equivalence [1248]. The ProVerif tool is widely used and is also part of other tool chains. For example, it is used in [1249] to analyse protocol implementations written in F#, which can be directly executed.

In Tamarin, protocols are specified using multiset rewriting rules and properties are specified in a fragment of first-order logic that supports specifications over traces. Proofs are constructed by a backwards search using constraint solving to perform an exhaustive, symbolic search for executions with satisfying traces. The states of the search are constraint systems and one starts with a system specifying the negation of the desired properties; hence one uses Tamarin to systematically find attacks. More generally, a constraint can express that some multiset rewriting step occurs in an execution or that one step occurs before another step.

Formulas can also be used as constraints to express that some behaviour does not occur in an execution. Applications of constraint reduction rules, such as simplifications or case distinctions, correspond to the incremental construction of a satisfying trace.

Tamarin's constraint reduction rules are sound and complete. This means that if one arrives at contradictions in all leaves of the derivation, then no satisfying trace exists; so we have a proof that there is no counterexample to the desired property, i.e, the property holds. Alternatively, if no further rule can be applied, but there is no contradiction in a leaf, then we can construct a trace that represents a counterexample to the desired property, i.e., we have an attack. Rule application can be carried out automatically or with user guidance, which may be necessary when automated rule application fails to terminate. Tamarin has special built-in support for equational theories relevant to security protocols, such as Diffie-Hellman exponentiation. Tamarin also supports, in contrast to ProVerif, the specification of protocols with persistent global state, e.g., counters, memory cells, etc., that are shared between protocol participants. Finally, Tamarin's specification language can be used to specify a wide variety of adversary models in the symbolic setting [1250].

13.4.1.3 Model Checking Non-trace Properties

As described in Sections 13.2.1.2 and 13.2.1.3, privacy-style properties are typically not trace properties, but can be expressed as hyperproperties or as observational equivalences between processes. Several model checkers support proving limited kinds of equivalences between protocols.

In the ProVerif model checker, one can formulate and prove a special kind of equivalence called *diff-equivalence* between processes [1159, 1251, 1252]. Diff-equivalence is a property that is stronger than observational equivalence and hence proofs of diff-equivalence suffice to establish observational equivalence and can be used to verify many privacy-style properties. The key insight is that diff-equivalence limits the structural differences between processes, which simplifies automating equivalence proofs. In more detail: two processes, P and Q, are specified by a new kind of process B, called a *biprocess*, which is a process in the applied pi calculus except that it may contain subterms of the form *diff*$[M, M']$ (where M and M' are terms or expressions) called *diff-terms*. P is then the process, where each diff-term in B is replaced by its first projection (e.g., M in the above example). Similarly Q is the process constructed by replacing each diff-term with its second projection (e.g., M'). Hence P and Q have the same structure but differ just in those places distinguished by a diff-term in B. ProVerif contains support to automatically establish the diff-equivalence of P and Q. Both Tamarin [1253] and Maude-NPA [1254] have subsequently also added support to verify variants of diff-equivalence.

Static equivalence [1159] is another property, related to indistinguishability, which is supported by different tools. Static equivalence is defined with respect to an underlying equational theory. It states, roughly speaking, that two terms are statically equivalent when they satisfy the same equations. This essentially amounts to a special case of observational equivalence that does not allow for the continued interaction between a system and an observer: the observer gets data once and conducts experiments on its own. Decision procedures for static equivalence have been implemented by tools including YAPA [1255], KISS [1256], and FAST [1257].

Static equivalence is useful for a variety of modelling problems in computer security. For example, it can be used to model off-line guessing attacks, where the adversary tries to guess a secret and verify his guess, without further communication with the system. This problem

is studied in [1258, 1259], who formulate the absence of off-line guessing attacks by using static equivalence to express that the adversary cannot distinguish between two versions of the same symbolic trace: one corresponding to a correct guess and the other corresponding to an incorrect guess. Decision procedures, like those listed above, are then used to answer this question.

13.4.2 Stochastic Methods

In the symbolic setting, there are no probabilities, only non-determinism, and security definitions are possibilistic (the adversary cannot do something bad) rather than probabilistic (he can do something bad only with negligible probability). Probabilities, however, are part of standard cryptographic definitions of security, cf. Section 13.4.3. They also arise when protocols are based on randomised algorithms or the guarantees themselves are probabilistic. Different options exist for augmenting both transition systems [1260] and logics with probabilities, and the corresponding model checking algorithms combine methods from conventional model checking, methods for numerical linear algebra, and standard methods for Markov chains [1261].

An example of a popular and powerful model checking tool is PRISM [1186, 1262, 1263], which supports the specification and analysis of a different type of probabilistic models. These include discrete and continuous-time Markov chains, Markov decision processes, and probabilistic automata and timed variants thereof. PRISM supports different property specification languages including Probabilistic Computation Tree Logic (PCTL) and Continuous Stochastic Logic (CSL), both based on the temporal logic CTL.

Probabilistic model checking has been successfully applied to a variety of protocols for security and privacy, where the protocols incorporate randomisation and the properties are probabilistic. For example, [1264] used PRISM to model the protocol underlying the Crowds anonymity system [1265], which is a peer-to-peer protocol for anonymous group communication based on random message routing among members of a “crowd”. [1264] models the behaviour of the group members and the adversary as a discrete-time Markov chain and the system’s anonymity properties are formalised in PCTL. PRISM is used to analyse the system, showing, for example, how certain forms of probabilistic anonymity change as the group size increases or random routing paths are rebuilt. A second example is the use of PRISM to analyse a probabilistic model of a randomised non-repudiation protocol that guarantees fairness without resorting to a trusted third party [1266]. Here the model checker is used to estimate the probability of a malicious user breaking the non-repudiation property, as a function of different protocol parameters. Two additional examples are the use of PRISM to analyse the probabilistic fairness guarantees provided by a randomised contract signing protocol [1267] and the probabilistic modeling and analysis of security-performance trade-offs in Software Defined Networking [1268].

In Section 13.2.1.2 we discussed the relevance of hyperproperties for security. Model checkers have also been developed to support the verification of such properties. Namely, hyperproperties are specified in the logics HyperLTL or HyperCTL* and automata-based algorithms are used to check finite-state systems with respect to these specifications [1149, 1150]. Recent work has also shown how to specify and reason about hyperproperties in the context of Markov Decision Processes [1269, 1270]. For example, in [1269], a specification language is given called Probabilistic Hyper Logic that combines a classic probabilistic logic with quantification over schedulers and traces. This logic can express a variety of security-relevant

properties such as probabilistic noninterference and even differential privacy. Although the model checking problem is undecidable in this case, it is nevertheless possible to provide methods for both proving and refuting formulas in relevant fragments of the logic that include many hyperproperties of practical interest.

13.4.3 Computational Methods

Cryptography has a history going back thousands of years. However, until the last century it was approached more as an art than a science. This changed with the development of public key cryptography and the idea of *provable security*, where reduction arguments are used to put the security of cryptographic schemes on a firm scientific footing. In contrast to symbolic models, the definitions and proofs involved are at a much lower level of abstraction, sometimes called the *computational* model. In this model, agents manipulate bit strings rather than terms and the adversary's capabilities are modelled by a probabilistic polynomial-time Turing machine rather than a process in a process calculus or an inductive definition in higher-order logic. Additionally, security definitions are probabilistic rather than possibilistic. For example, they are formulated in terms of the adversary's chances of winning some game, cf. Cryptography Knowledge Area (Section 10.2.2).

There is a trade-off between computational and symbolic models: the computational models are more detailed and accurate, but substantially more effort is involved in formalising protocols and in proving their properties. This trade-off is not surprising. In fact it is standard from both the formal methods and the security perspective: concrete models provide a more precise description of how a system works and what the adversary can do, but formalisation and proof construction is correspondingly more difficult. The difficulty in applying these ideas in practice is substantial and some researchers have even questioned whether the gains are worth it or, given the complexity, whether one can trust the results. [1271] contains a brief history of the topic and surveys some of the past controversies around this question.

To increase confidence in the validity of cryptographic proofs in the computational setting, various proposals have been made. For instance different structuring techniques and abstractions were proposed to simplify constructing proofs and understanding the resulting security arguments. Prominent examples are game-based proofs [1272, 1273], Canetti's universal composability framework [1274], and Maurer's constructive cryptography [1275]. In addition, in response to Halevi's call [1276] for increased rigour in cryptographic proofs, formal-methods tools were developed to help cryptographers make the transition from pen-and-paper proofs to mechanically checked security proofs in the computational setting. There has also been some success in bridging symbolic models and computational models via computational soundness results for symbolic abstractions [1277, 1278, 1279, 1280, 1281]; these results enable symbolic proofs, but with stronger, computational guarantees.

In the following, we expand on representative approaches and associated tools for carrying out game-based and simulation-based proofs.

13.4.3.1 Game-based Proofs

Games and game transformations can be used to structure cryptographic proofs. A game describes the interaction between different parties (typically the adversary and some other entity), which are modelled as probabilistic processes. Proofs are structured as a sequence of games where the initial game represents the security goal (for example, formalising indistinguishability under chosen-ciphertext attacks) and the final game is one where the adversary's advantage is zero by inspection. The transitions between games are based on small transformations that preserve, or only slightly alter, the overall security, e.g., by transforming one expression into an equivalent one or applying a transformation based on some computational hardness assumption (cf. Cryptography Knowledge Area (Section 10.2.3)). This is expressed in terms of the probability of events occurring in each of the games; for example, the probability of an event A happening in a game G_i is close to the probability of some event A' happening in game G_{i+1}. A good overview and tutorial on constructing such proofs is [1273].

CryptoVerif [1214, 1282] was the first tool developed to support game-based proofs. In CryptoVerif, games are modelled as processes in a language inspired by the pi calculus and transitions are justified by process-algebraic notions like bisimulations. The tool can prove secrecy and correspondence properties, which are relevant, for example, for authentication and key agreement protocols. Moreover, CryptoVerif is highly automated: the tool can automatically decompose games into reductions and oracles and even discover intermediate games, rather than having them given explicitly by the user. To achieve this, CryptoVerif imposes various restrictions. For example, the game transformation strategies are hard-coded and users must trust that CryptoVerif correctly implements them and that the transformations' pen-and-paper justifications are correct. The tool also lacks extensibility as it supports just a fixed set of language primitives and game transformations. Nevertheless, CryptoVerif is quite powerful and has been applied to both protocols and primitives, including Kerberos, and the Full-Domain Hash signature scheme.

An alternative approach to mechanising game-based proofs is to build the foundations for these proofs directly within an expressive logic like a type theory (for example, the calculus of inductive constructions) or higher-order logic. This is the approach taken by tools like CertiCrypt [1283] and FCF [1284], which are implemented in Coq, and CryptHol [1285], implemented in Isabelle/HOL. These approaches are considered *foundational* in that they start with a formalised semantics. In particular, the tools provide a language for probabilistic programs with a semantics formalised within the logic. On top of this, one can

express relevant concepts for game-based proofs like discrete probability distributions, failure events, games, reductions, oracles, and adversaries. The semantics is used to formally derive proof rules for reasoning about probabilistic programs and game transformations, e.g., to replace subprograms by equivalent subprograms. Deriving these rules ensures the consistency of the resulting theory and one thereby has stronger guarantees than is possible by directly axiomatising proof rules. Moreover, the underlying logic (e.g., HOL) can directly be used to formulate and prove the correctness of arbitrary games and transformations between them. Examples of the kinds of theorems proven using the above tools are the semantic security of OAEP (with a bound that improves upon existing published results) and a proof of the existential unforgeability of FDH signatures (in CertiCrypt), the security of an efficient scheme for searchable semantic encryption (in FCF), and the IND-CCA security argument for a symmetric encryption scheme (in CryptHol).

Proof construction using these tools usually requires significant manual effort since the user has a free hand in structuring games, their transformations, and justifications. The EasyCrypt

tool [1286] shows, however, that some automation is possible by using appropriate programming logics and deductive machinery. EasyCrypt provides a language for formalising the probabilistic programs used to model security goals and hardness assumptions, as well as a probabilistic Hoare logic and a relational version thereof for reasoning about procedures (games) and pairs of procedures (game transformations). Proof construction can be automated using tactics, which are programs that build proofs, along with SMT solvers. EasyCrypt users interact with the system by providing proof sketches for game-hopping proofs and the above deductive machinery is then used to construct the resulting proofs.

13.4.3.2 Simulation-based Proofs

Simulation-based security proofs are based on a general paradigm for formalising security definitions and establishing the correctness of systems with respect to them. The main idea is that the security of a real system (or protocol) is abstractly defined in terms of an ideal system, otherwise known as an ideal functionality. For example, semantic security for encryption could be formalised by comparing what an adversary can learn when receiving a real ciphertext with what the adversary can learn when receiving just a random value. A real system securely realises the ideal system if every attack on it can be translated into an "equivalent" attack on the ideal system. Here, the notion of equivalence is specified based on the environment that tries to distinguish the real attack from the ideal one. This approach is attractive due to the general way that security can be defined and also because it yields strong composability properties. Security is preserved even if the systems are used as components of an arbitrary (polynomially bounded) distributed system. This supports the modular design and analysis of secure systems. A good overview and a tutorial on constructing such proofs is [1287].

Numerous proposals have been made for formalizing simulation-based security, see e.g., [1288], all with different scopes and properties. These include universal composability [1274], black box simulatability [1289], inexhaustible interactive Turing machines [1290], constructive cryptography [1275], and notions found in process calculi, both with [1291] and without [1292] probabilities. Support for formalising proofs in these frameworks is, however, not yet as advanced as it is for game-based proofs and progress has been more modest. For example, [1293] formalised a computationally-sound symbolic version of blackbox simulatability in Isabelle/HOL and used it to verify simple authentication protocols. [1294, 1295] have carried out simulation proofs in Isabelle/HOL, building on CryptHOL, for example verifying multi-party computation protocols. There has also been some progress on automating simulation-based proofs using the EasyCrypt system to mechanise protocol security proofs within a variant of the universally composable security framework [1296].

13.5 SOFTWARE AND LARGE-SCALE SYSTEMS

[1297, 1298, 1299, 1300, 1301, 1302, 1303]

There are many well-established methods and tools for verifying functional properties of programs, for example, verifying that a program has a given input/output relation or satisfies a given trace property. As explained in Section 13.2.1, this is relevant for security. Also of direct relevance are the numerous tools employing both sound and unsound analysis procedures that can help identify common security problems, such as memory corruptions, injection attacks and race conditions. We refer to [1304] for a general survey of formal methods techniques, [1305] for a survey on automated methods, and to [1306] for an excellent overview of some

```
h := h mod 2
l := 0
if (h = 1)
  then l := 1
  else skip
```

Figure 13.1: Example of Information Flow

of the successes and challenges in applying formal methods to large-scale systems like operating systems and distributed systems.

In the following subsections we examine methods, tools, and applications in the context of general software, cryptographic libraries, and other kinds of systems. We start with information flow control, as it provides a foundation for enforcing end-to-end security policies.

13.5.1 Information Flow Control

Information flow control (IFC) concerns the enforcement of confidentiality or integrity guarantees during a system's execution. The basic setup distinguishes different security levels, such as high and low or, more generally, a lattice of levels. For confidentiality policies, information should not flow from high to low and dually for integrity properties, cf. the Biba model from the Operating Systems & Virtualisation Knowledge Area (Section 11.3) or taint tracking, where information should not flow from tainted to untainted. As explained in Section 13.2.1.2, noninterference and related hyperproperties can often be used to formulate such policies.

Figure 13.1, taken from [1297], contains a simple example of a program that violates a secure information flow policy for confidentiality that prohibits information from the high variable h from flowing to the low variable l. Due to the if-branching in this example there is an implicit flow where the value of h may flow to l.

Information flow control is of particular importance for program and system verification addressing *end-to-end* security guarantees: information flows must be restricted throughout the entire system, from start to finish, to prevent violations of confidentiality or integrity policies. Real-world programs are of course much more complex than this and it may not be so trivial to determine whether the high variable affects the value of low variables anywhere throughout the program. When the property holds, the guarantee is a strong one. This is in contrast to what is normally achieved by conventional security mechanisms like access control, which controls the context in which data can be released, but not how the data is used after its release.

Below we give examples of popular approaches to verifying that programs ensure secure information flow in practice.

13.5.1.1 Static Analysis and Typing

Early work on IFC focused on preventing information flows using static analysis [1307] and specialised typing disciplines [1308]. Specifically, in [1308], a type system was developed that soundly enforces a secure information flow property inspired by noninterference. The rough idea is that each program expression has a type, which includes not just a conventional type, like Int or Bool, but also a security label. Typing rules associate types with expressions and make statements about expressions being well typed based on the types of their subexpressions. For example, in a lattice with just the labels high and low, any expression can have the type high, whereas it has type low if all its variables are associated with the type low. Moreover, an expression e can only be assigned to a low variable l, representing a direct information flow, if e has type low. The typing rules also prevent indirect information flows by making sure that low variables cannot be updated in high contexts. For example, an if-then-else expression branching on a Boolean b would allow low assignments in its branches only if b is low typeable. This rules out programs like that in Figure 13.1. For further details see, e.g., [1297, 1309].

Various languages and compilers now support security type systems. These include Jif (Java + Information Flow) [1298, 1310], Flow Caml [1311, 1312], which is an information flow analyser for the Caml language, the SPARK programming language built on top of Ada [1313] (based in part on the seminal work of [1314]), and JOANA for Java [1315]. For example, Jif extends Java with security types to support information flow control and access control, enforced both statically at compile time and at run time. In Jif, the variable declaration

$$\text{int } \{\text{Alice} \rightarrow \text{Bob}\}\ x;$$

expresses both that x is an integer and that Alice requires that the information in x can only be seen by Bob. In contrast,

$$\text{int } \{\text{Alice} \leftarrow \text{Bob}\}\ x;$$

expresses Alice's requirement that information in x can only be influenced by Bob, which is an integrity policy. The Jif compiler analyses these labels to determine whether the given Java program respects them.

The research of [1315] illustrates how richer secure information flow properties can be handled in full Java with concurrency. Traditional information flow control focuses on possibilistic leaks: can secret (high) data directly or indirectly leak to non-secret (low) data. However, this is insufficient when programs have parts that may execute concurrently since the scheduling of their threads can effect which values are output or their ordering. [1315] investigates probabilistic noninterference, which is relevant for concurrent programs as it accounts for probabilistic leaks where, for some scheduling strategies, the probability of some publicly visible behaviour depends on secret data. In particular, [1315] develops sophisticated program analysis techniques to enforce a version of probabilistic noninterference, called *low-security observational determinism*, which requires that execution order conflicts between low events are disallowed if they may be influenced by high events. The resulting tool, JOANA, takes Java programs, where all inputs and outputs are labelled high or low and determines whether this property holds. JOANA can handle full Java byte code, it scales to programs with an arbitrary number of threads and 50k+ lines of code, and its (conservative) analysis is of high precision.

13.5.1.2 Self-composition and Product Programs

As explained in Section 13.2.1.2, information flow properties are hyperproperties that relate pairs of executions of a program P. A natural idea, rather than reasoning about pairs of executions, is to reason about a *single* execution of a program P', derived by composing P with a renaming of itself [1316]. Depending on the programming language and the properties involved, this self-composition may be just the sequential composition of programs. Namely P' is simply $P; R(P)$, where $R(P)$ is a version of P with renamed variable. Alternatively P' may be a more complex product construction, for example, based on P's transition system representation [1317].

This idea is powerful and has wide scope. For example, one can systematically reduce different notions of secure information flow for a program P to establishing a safety property for the self-composition of P. Furthermore, the idea generalises beyond secure information flow: hyperproperties covering both safety and liveness can be formalised and reasoned about using forms of self-composition [1135, 1318].

From the deductive perspective, a strong advantage of both self-composition and product programs is that one can reuse off-the-shelf verifiers to reason about the resulting program. This includes using weakest-precondition calculi and Hoare logics [1319, 1320], separation logics, and temporal logics. When the program is finite-state, temporal logic model checkers can also be employed as decision procedures. Indeed, self-composition provides a basis for LTL model checking for hyperproperties [1149]. Note that an alternative proposal is not to combine programs but to use a specialised *relational logic* to explicitly reason about pairs of programs [1321, 1322].

Overall, the above ideas have seen wide applicability: from reasoning about secure information flow to detecting, or proving the absence of, timing and other side-channels, cf. the example with timed traces in Section 13.2.1.2. An example of the practical application of these ideas is [1299]. The authors present a state-of-the-art theorem proving environment for realistic programming languages and extend it to support the verification of arbitrary safety hyperproperties, including secure information flow. The key idea is a specialised product construction that supports the specification of procedures and the modular (compositional) reasoning about procedure calls. This product construction is implemented within the Viper verification framework [1323], which is an off-the-shelf verifier supporting different language front-ends (such as Java, Rust, and Python), an intermediate language (that is a simple imperative language with a mutable heap and built-in support for specifications) and automation backends that leverage symbolic execution, verification condition generation, and SMT solvers. The tool has been used to reason about information flow properties, including those involving declassification, of challenging programs. The programs combine complex language features, including mutable state on the heap, arrays, procedure calls, as well as timing and termination channels. User interaction with Viper is mainly limited to adding pre-conditions, post-conditions, and loop invariants and Viper completes the proofs in reasonable time.

13.5.2 Cryptographic Libraries

Reasoning about the implementations of cryptographic primitives and protocols involves challenges that go beyond the verification of their designs, as discussed in Section 13.4. Specifically, the methods and tools for code-level verification must work directly with the implementations themselves. Moreover, the properties to be verified must also account for new problems that can arise in the implementation [1300], such as the following.

- **Freedom from side-channel attacks:** This includes that secrets are not revealed (secure information flow) by software or hardware artifacts including assignment, branching, memory access patterns, cache behaviour, or power consumption.
- **Memory safety:** Only valid memory locations are read and written, for example there are no dangling pointers that reference memory that has been deallocated.
- **Cryptographic security:** The code for each cryptographic construction implements a function that is secure with respect to some standard (computational or information-theoretic) security definition, possibly under cryptographic assumptions on its building blocks.

Note that for programs written in low-level languages like C or assembler, memory safety is particularly important since a memory error in any part of the code could compromise the secrets it computes with or lead to the adversary taking control of the application. This necessitates the ability to reason effectively about low-level code.

One approach to the verification of cryptographic libraries is that taken by HACL* [1300], which is a verified version of the C cryptographic library NaCL. The NaCL library is a minimalistic, but fully-functional library of modern cryptographic primitives including those for encryption, signatures, hashing and MACs. Since C is difficult to reason about, the library is reimplemented in F* [1324], which is a functional programming language designed to support program verification. F* features a rich type system and supports dependent types and refinement types, which can be used to specify both functional correctness and security properties. Type checking is mechanised using an SMT solver and manual proofs. The NaCL library is written in a subset of F* that can be translated to C and subsequently to assembler using either conventional or verified compilers. The NaCL library is verified with respect to its functional correctness, memory safety, and the absence of timing side-channels where secrets could be leaked; cryptographic security was not covered however. For example, memory safety is proven by explicitly formulating properties of programs that manipulate heap-allocated data structures. Functional correctness is proven with respect to executable specifications that are themselves checked using test vectors.

An alternative to working with a high-level language like F* is to work with a lower-level language that is an assembly language or reasonably close to one. This eliminates or reduces the need to trust that compilers will respect the properties proven of the high-level code. Moreover, the implementations can be highly optimised, which is often a requirement for cryptography in practice. Two representative examples of this are Vale and Jasmin.

The Vale tool [1325] is used to verify cryptographic functions written in a low-level language, also called Vale, designed for expressing and verifying high-performance assembly code. The Vale language is generic, but targets assembly code by providing suitable language constructions such as operations on registers and memory locations. Moreover, Vale programs can be annotated with pre-conditions, post-conditions, assertions, and loop invariants to aid proofs. Vale code is translated into the Dafny logical framework [1326] and the operational

semantics of the assembly language, e.g., x86, is written directly in Dafny. Since the operational semantics can account for the timing of operations and memory accesses, one can establish the absence of (i) timing and (ii) cache side-channels by showing that (i) no operations with variable latency depend on secrets and that (ii) there is no secret-dependent memory access, respectively. The Vale tool implements a specialised static analysis procedure to verify this noninterference. Dafny's off-the-shelf verification tool is additionally used to verify functional correctness, where proofs are partially automated using the Z3 SMT solver.

Jasmin [1327] (and also CT-Verif [1328]) is an alternative approach based on a low-level language for writing portable assembler programs, also called Jasmin, and a compiler for this language. The Jasmin language was designed as a generic assembly language and the compiler platform (currently limited to produce x86_64 assembly) has been verified in Coq. The language and compiler are designed to allow fast implementations, where precise guarantees can be made about the timing of instructions. This enables verifying the absence of timing and cache-based attacks, in addition to functional correctness and memory safety. Verification leverages a tool chain similarly to Vale, using Dafny and the Z3 SMT solver. Jasmin has been used, for example, to verify assembler code for Curve25519, an elliptic-curve variant of Diffie-Hellman key exchange.

Finally, we mention Cryptol[4], an industrial-strength domain-specific language for formalising and reasoning about efficiently-implementable hardware. Cryptol features an executable specification language, a refinement methodology for bridging high-level specifications with low-level implementations, e.g., on FPGAs, verification tools involving SAT and SMT solvers, as well as a verifying compiler that generates code along with a formal proof that the output is functionally equivalent to the input.

13.5.3 Low-level Code

Cryptographic libraries are just one example of security-critical software, often written in assembler for performance reasons. Operating systems and device drivers typically also contain assembly code and similar verification techniques are needed to those just described. Here, we briefly survey some additional, representative, approaches and tools that have been used to verify low-level code.

As with HACL*, Vale, and Jasmin, a common starting point is a generic low-level language that can be specialised to different platforms, rather than starting with a concrete assembly language like x86. The language chosen can then be given a formal semantics, by embedding it in a higher-order logic like Coq, e.g., as was done with Jasmin. Such an embedding enables one to (1) prove that the translation to assembler preserves desired functional or security properties[5] and (2) derive proof rules for reasoning about low-level programs. For example, [1331] builds on separation logic [1332], which is a powerful logic for reasoning about imperative programs that manipulate pointers, to build a verification environment for low-level code. Namely, on top of a formalisation of separation logic in Coq, the authors of [1331] formalise an assembler and derive proof rules for the verification of assembly code. The Bedrock project [1301, 1333, 1334] provides another Coq framework for verifying low-level code using separation logics. A final example is given by the Igloo project [1335], which links a refinement-based formalism for developing programs in Isabelle/HOL with the Viper verification environment,

[4] **https://www.cryptol.net/**

[5] More generally, one may employ techniques for *secure compilation* [1329, 1330], whereby compilers are guaranteed to preserve security properties of the source program in the target compiled program.

which also supports the verification of low-level code using a separation logic.

In contrast, one may proceed in the reverse direction, by translating assembler into a higher-level programming language for which verification tools already exist. This is the approach taken by Vx86 [1336]. It takes as input x86 assembler code that has been manually annotated with verification conditions — pre-conditions, post-conditions, and loop invariants — and translates it to annotated C code, which makes the machine model explicit. The C code is then analysed by VCC, Microsoft's Verifying C Compiler [1337], which employs its own verification tool chain involving Z3.

13.5.4 Operating Systems

Operating systems are critical for security as they provide services that applications rely on, such as I/O and networking, file system access, and process isolation. Security bugs in the operating system can therefore undermine the security of everything running at higher layers of the software stack. Moreover, operating systems are ubiquitous and control everything from small embedded devices to supercomputers. Formal methods are important in ensuring their functional correctness as well as the absence of different classes of security problems.

13.5.4.1 Functional Correctness of Kernel Components

Verification of functional correctness is an essential part of verifying the security of kernel components. If parts of the kernel do not do their job properly, then it will be impossible to secure applications running on top. For example, it is essential that a multitasking kernel enforces *data separation* so that applications in one partition cannot read or modify the data in other partitions. There is also *temporal separation* where processes use resources sequentially and resources assigned to one application are properly sanitised before being assigned to another. Also important is *damage limitation* whereby the kernel limits the effect of compromises so that the compromise of one component does not impact others, or the operating system itself. These requirements are mandated by standards like the Common Criteria and ISO/IEC 15408 and are therefore demanded for certifying high-assurance systems like operating systems. Establishing them is an important part of functional verification.

We note in this regard that the distinction between functional requirements and non-functional requirements, which typically include security requirements (such as information flow control or absence of particular bug classes), is somewhat fuzzy. Functional requirements describe what the system should do. Generally they can be described via goal or I/O-oriented descriptions defining the system's purpose, or in terms of system states and traces. In contrast, non-functional requirements are usually seen as constraints, pertaining to aspects like usability, scalability, maintainability and testability. But the boundary is particularly blurred when it comes to security. As an example, memory safety is not the goal of an operating system. However, without memory safety the functions that the operating system should perform will not always work correctly, especially in the presence of an adversary who can find and exploit memory corruption vulnerabilities.

The seL4 project is the first, and probably best known, operating system verification effort, which formally verified a complete, general-purpose operating system kernel. The seL4 microkernel is a member of the L4 family, comprising 8,700 lines of C code and 600 lines of assembler. It was verified [1302] from its abstract specification all the way down to its C implementation. Functional correctness for seL4 states that the implementation strictly fol-

lows the abstract specification that precisely describes the kernel's behaviour. This includes standard safety properties such as requiring that the kernel will never crash or perform unsafe operations.

The verification of seL4 was carried out in Isabelle/HOL and uses refinement to establish a formal correspondence between three specifications of the system at different levels of abstraction. The first specification is an *abstract specification* that completely describes the kernel's behaviour by specifying the kernel's outer interface, e.g., system calls and their effects, as well as the effects of interrupts and system faults. The second specification is an *executable specification* written in a functional programming language and the third specification is seL4's actual *C implementation*. All three specifications are given in higher-order logic and for the third specification this necessitates also specifying a formal semantics for the subset of C used. At the heart of the proof are the refinement steps, showing that the executable specification refines the abstract specification and that the C specification refines the executable one. This guarantees that all Hoare logic properties established for the abstract specification also hold for the refined models, in particular the kernel's source code.

Verifying seL4 was a major effort, requiring roughly 20 person-years. This included ca. 9 person-years for developing formal language frameworks, proof tools, support for proof automation, and libraries. The seL4-specific proof took ca. 11 person-years. Subsequent work later extended the formalisation to also prove strong information flow control properties [1338].

While seL4 is exemplary, it is by no means a lone example of what is possible with enough time and sweat. There have been dozens of formal methods projects that specify and verify substantial parts of operating systems and different variants thereof, e.g., hypervisors, separation kernels, partitioning kernels, security kernels, and microkernels [1339]. Other projects that have proved the correctness of simple hypervisor-like OS kernels are CertiKOS [1340] and ExpressOS [1341]. In contrast, other projects focus on functional correctness for just selected critical parts of the kernel. For example, Jitk [1342] is an infrastructure for building in-kernel interpreters using a just-in-time (JIT) compiler that translates high-level languages into native code like x86. Jitk uses CompCert, a verified compiler infrastructure [1343] to guarantee that the translation is semantics-preserving. Other examples are FSCQ [1344] and Cogent [1345], which focus on the verification of file systems, and Microsoft's SLAM model checker [1128], which uses predicate abstraction techniques to verify the correctness of Windows device drivers.

13.5.4.2 Absence of Bug Classes

Rather than verifying all or parts of an operating system (or any program, for that matter), formal methods may instead target eliminating specific kinds of security-critical vulnerabilities. Important examples, formulated positively in terms of desirable properties (which guarantee the absence of the vulnerabilities) are ensuring type and memory safety, or control flow integrity. We shall give examples of each of these below. These properties are non-negotiable prerequisites for software that needs any sort of integrity or reliability at all. One gets considerable benefits proving such properties, even if one is not aiming for full functional correctness.

Both type and memory safety guarantees are provided for the Verve operating system [1346]. Type safety ensures that all operations are well typed; for example a stored Boolean value is not later retrieved as an integer. Memory safety, discussed in Section 13.5.2 ensures that the

operations do not overflow assigned memory or deference dangling pointers. These properties rule out common security problems such as buffer overflows. All of Verge's assembly language instructions (except those for the boot loader) are mechanically verified for safety; this includes device drivers, garbage collectors, schedulers, etc. To achieve this, Verve is written in a safe dialect of C# that is compiled to a typed assembly language (TAL) that runs on x86 hardware, and verification is based on a Hoare verifier (Boogie) that leverages the Z3 SMT solver for automation. Verve was later used as part of the Ironclad project [1347] to build secure applications.

Control-Flow Integrity (CFI) was discussed in the Operating Systems & Virtualisation Knowledge Area (Section 11.5.2) and the Software Security Knowledge Area (Section 15.4). CFI techniques are a form of dynamic analysis (cf. Section 13.2.3.4) where at runtime a program is monitored and regulated to prevent adversaries from altering a program's control flow to execute code of their choosing, e.g., for return-oriented programming. Efficient algorithms for dynamic analysis have made their way into modern compiler tool chains like GCC and LLVM [1348]. Additionally, in the SVA [1349] and KCoFI [1350] projects, CFI and memory safety protection have been implemented into execution environments that can be used to protect commodity operating systems.

13.5.5 Web-based Applications

The Web & Mobile Security Knowledge Area (Chapter 16) introduces Web and mobile security. One key takeaway is that the Web ecosystem is tremendously complex. It comprises browsers, browser extensions, servers, protocols, and specialised languages, all of which interact in complex ways. Hence an important role of formal methods is to formalise these different parts of the Web and provide analysis techniques to help understand the properties of the individual components and their interaction. In light of the Web's complexity, various approaches have emerged to tackle different aspects of this problem; see [1351] for a survey of the scope and applications of formal methods in this domain.

13.5.5.1 Web Programming

The JavaScript programming language is a core Web technology that runs in browsers and supports interactive Web pages. JavaScript is known to be rather quirky [1352] and a source of security bugs. Given this complexity, an important application of formal methods is to provide JavaScript with a formal semantics. This was accomplished by [1353], who formalised the JavaScript (ECMA) standard in the Coq proof assistant. Using this formalization, they verified reference interpreter in Coq, from which they extracted an executable OCaml interpreter. Moreover, by running the interpreter on existing JavaScript test suites, they were able to clarify and improve ambiguous parts of the standard.

Formal methods can also be used to provide support for developing JavaScript applications in a disciplined way. This idea is pursued in [1354] who propose developing secure JavaScript applications in F* and using a verified compiler to produce JavaScript. As previously explained, F* is a dependently typed language with a clear semantics and verification support. The authors of [1354] verify a full-abstraction result that allows programmers to reason about their programs with respect to F*'s semantics, rather than that of JavaScript. An alternative approach to improving the security of JavaScript is to enhance JavaScript interpreters to enforce information flow control and thereby ensure that sensitive data does not leave the browser [1355].

Finally, one can use formal methods to explore alternative programming languages for the Web. WebAssembly is a prominent example [1356, 1357]. This language features a formal semantics and there is support for verifying functional properties, information flow security, and timing side-channel resistance of WebAssembly programs.

13.5.5.2 Web Components

Formal methods can be used to analyse the properties of individual Web components and improve their security. We give two examples: Web browsers and Web protocols.

An executable, operational semantics of the core functionality of a web browser is presented in [1358]. Their model accurate captures asynchronous execution within the browser, browser windows, DOM trees, cookies, HTTP requests and responses, and a scripting language with first-class functions, dynamic evaluation, and AJAX requests. Such a model provides a valuable basis for formalizing and experimenting with different security policies and mechanisms, such as solutions to web-script security problems like cross-site scripting.

One may also focus on the security of the underlying protocols. For example, [1359] modelled a Web-based single sign-on protocol given by the OASIS Security Assertion Markup Language (SAML) 2.0 Web Browser SSO Profile. This protocol is used by identity providers such as Google for single sign-on, on the Web. However, analysis of the protocol's authenticity properties using the SATMC model checker revealed severe flaws that allowed a dishonest service provider to impersonate users [1359]. Here, as is often the case when using formal methods, the attack suggested a fix, which was subsequently verified.

13.5.5.3 Component Interaction

In contrast to focusing on individual Web components, one may consider the interaction of the Web's components by modelling subsets of the Web ecosystem, at an appropriate level of abstraction. [1360] create a model of key Web components and their interactions in Alloy, a modelling tool based on first-order logic. The components include browsers, servers, scripts, HTTP, and DNS. Additionally they model different adversary capabilities like Web attackers who can create malicious web-sites or active network attackers. Finally, they specify different invariants and properties that are typically required for secure browsing; for example, when a server receives an HTTP request, it often wishes to ensure that the request was generated by a trusted principal and not an attacker.

Specifications like the above bring us into the standard setting of formal methods for security: given a model of the system and an attacker, one wishes to verify that all behaviours satisfy the given property. [1360] uses Alloy's analyser to answer this and thereby find bugs in existing security mechanisms and analyse proposed improvements to the Web. For example, they analyze and find attacks on: a proposed cross-site request forgery defense based on the Origin header, Cross-Origin Resource Sharing; a proposal to use Referrer validation to prevent cross-site scripting; a new functionality in the HTML5 form element; and WebAuth, a Kerberos-based single sign-on system. An analogous approach is taken by [1361], who provide a detailed model of core security aspects of the Web infrastructure, staying as close as possible to the underlying Web standards. The model was used to provide a rigourous security analysis of the BrowserID system.

13.5.6 Full-stack Verification

We have considered various options for using formal methods to improve security across the system stack. Ideally, the entire stack, or as much of it as is possible, comes with guarantees that are linked across levels.

The first such stack was specified in pure Lisp and verified with the Boyer-Moore (NQTHM) prover, a predecessor of the ACL2 theorem prover. The focus was on the hardware and operating system layers [1303] and establishing functional correctness, not security per se. The components verified included a code generator for a high-level language, an assembler and linking loader, a simple operating system, and a microprocessor. These layers were connected by downward and upward functions, mapping down and up the system stack. The downward functions formalised that: the compiler maps high-level language states to assembly language states; an assembler, linker and loader that, when composed, map the assembly language states to states of their machine architecture; and a function that maps architecture states to register transfer level (RTL) states. The upward maps formalised the high-level data represented by a region of a low-level state. Theorems were proved in NQTHM that related these layers. Finally, pre-conditions and post-conditions for the applications programs were defined and the programs were proved correct with respect to the high-level language semantics. All these parts were put together in a “warranty theorem” that informally states the following [1303]:

> “Suppose one has a high-level language state satisfying the pre-condition of a given application program [...]. Construct the RTL state obtained from the high-level state by successively compiling, assembling, linking, and loading. Run the RTL machine on the constructed state. Map certain data regions up, using the partial inverse functions, producing some high-level data objects. Then the postcondition holds of that high-level data."

This theorem was formalised and proved using the NQTHM theorem prover, thereby linking all the layers. This stack, albeit somewhat simplistic, captures the essence of full-stack verification.

The vision of full-stack verification is being pursued on a larger scale within the DeepSpec [1362] project. This project has the goal of developing fully-verified software and hardware, including operating system kernels [1340], cryptographic primitives and protocols [1363], a message system [1364], and a networked server [1365]. Verification uses the Verified Software Toolchain (VST) [1366], which is a framework for verifying C programs via a separation logic embedded in Coq.

13.6 CONFIGURATION

[1367, 1368, 1369]

The security of a system depends not only on its implementation, but also its configuration and operation. When configuring a system, accounts must be created, credentials and certificates must be distributed or configured, and authorisation policies must be specified for numerous systems at different levels of the stack, e.g., the operating system, middleware, applications, and network devices. Clearly this must also be done correctly and formal methods can help to determine whether this is the case, or even assist with the configuration process, for example, by compiling high-level policies to low-level configurations.

We will expand on this in the following subsections, with a primary focus on the correctness (with respect to high-level *properties*) of the authorisations (also called the *policy*) configured to enforce access control. This problem is important as access control mechanisms are ubiquitous, the specification and management of access control policies is challenging, and misconfiguration is the source of many security problems in practice. See the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14) for more on the challenges in this regard, as well as for a discussion on logics for access control and delegation.

13.6.1 Policy Analysis

As observed in the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14), there is a wide spectrum of policy languages in which administrators can specify access control policies. Many of these languages are low-level and not particularly user friendly. Moreover, organisations and their IT setup can be complex, which is reflected in their access control policies. A large-scale organisation may have millions of users and access may need to be regulated across numerous and diverse resources including applications, files and data sets. Even when industry-standard policy languages are used, such as for role-based access control (RBAC) or attribute-based access control (ABAC), the resulting policies are voluminous and detailed. Take an RBAC setup in a typical organisation: there may be hundreds of roles and thousands of rules and it is extremely challenging to get this right, especially as authorisations evolve over time. The problem here is analogous to program verification: how does one know whether what is written is really what is wanted?

Early research on formalising access control, like the Harrison, Ruzzo, Ullman (HRU) model described in the Authentication, Authorisation & Accountability Knowledge Area (Section 14.3.3.1), investigated notions of *safety* within the context of access control matrices and updates to them. The authors of this seminal work posed the problem of *safety analysis*, asking whether access rights can be leaked in undesirable ways. Since then, researchers have looked into both:

1. handling different, more realistic policy languages than those based on the access-control matrices used in HRU, and
2. reasoning about more general properties (safety is interesting, albeit rather specialised).

One generalisation of safety analysis is *security analysis* [1370], where the property concerns whether an access control system preserves invariants that encode desired security properties, i.e., whether these properties are preserved across state changes. [1371] explores this problem in the context of role-based access control. The authors study the complexity of this problem (showing it is in PSPACE) and provide algorithmic solutions based, for example, on model checking.

In [1367], the authors show how to generalise both (1) the policy language and (2) the properties, simultaneously. They show that many policy languages, including various RBAC extensions and ABAC, can be directly expressed in a fragment of first-order logic, which the authors identify and call FORBAC. Moreover, the properties that one expects configurations to satisfy can also be formalised within FORBAC. Hence one can use first-order deduction to analyse whether configurations satisfy their desired properties. For FORBAC, [1367] shows that SMT solvers suffice, that this approach scales in practice, and it can be used to verify or find errors in industrial-scale access control configurations.

A related problem to policy analysis is understanding policy changes, given that policies are

maintained and evolve over time. This problem is known as *change-impact analysis*. [1368] presents Margrave, which is such an analysis tool for access-control policies written in the XACML standard and another version of Margrave handles firewall policy languages [1372]. Like the tools mentioned above, given an XACML policy and a property, Margrave can determine whether the policy satisfies the property. In addition, for change-impact analysis, it can take two policies and summarise the semantic differences between them — what is allowed in one but not the other — by computing a compact representation in the form of a multi-terminal Binary Decision Diagram.

Another approach to finding errors in policies, in the context of network firewalls, is taken by [1373]. Firewall rules are generally written in low level, platform-specific languages, their control flow may be non-trivial, and they are further complicated by Network Address Translation (NAT). Unsurprisingly, it is easy for administrators to make mistakes when specifying configurations. The idea behind [1373] is to reverse the compilation procedure and use *decompilation* to obtain a more high-level description of a given firewall configuration. Specifically, their tool, called FireWall Synthesizer, translates configurations from different popular firewall systems into a common intermediate language; from configurations in the intermediate language, the Z3 SMT solver is used to synthesise an abstract firewall policy, in tabular form. The generated specification plays the role of a property, in that it is much easier for system administrators to understand than the original policy. The tool also supports the comparison of different configuration versions.

13.6.2 Specification-based Synthesis

Rather than having administrators directly configure policies and afterwards, if at all, prove that they satisfy desired properties, an alternative is to specify the properties and synthesise policy configurations from them. The advantage of this approach is that if the property language is sufficiently high-level, then it is presumably easier to express correctly the property than the policy that implements it. This is analogous to the advantages of using high-level programming languages over assembler, or the synthesis of controllers from declarative specifications of their properties.

One example of this is the tool [1369] that translates policies written in a high-level policy language to policies in XACML, which is a low-level, machine-oriented, industrial standard policy language. The high-level language used is a first-order policy specification language that allows authorisation rules to be defined based on arbitrary conditions. The translation to XACML combines syntax-driven translation with query answering. It exemplifies a common translation-oriented approach of using simple deductive reasoning to expand a policy written succinctly in a high-level language to a more verbose one written in a lower-level (but, equally expressive) language.

A second, rather different example is found in [1374] and [1375], both of which provide methods to construct runtime monitors for workflow enforcement engines that enforce access control policies involving separation of duty, which is the security principle that requires different users to carry out different tasks in a work flow. For example, the user who is responsible for financial transactions in a bank should be different from the user who audits the transactions. Roughly speaking, both approaches require that one specifies which pairs (or sets) of tasks in a workflow must be handled by different users, along with other access-control constraints. The problem is then to generate a runtime monitor that enforces all these constraints, while being as liberal as possible so that the workflow can successfully complete. [1374] solves

the problem using techniques based on symbolic model checking. In contrast, [1375] tackles the problem by providing a specification language for access-control and separation of duty constraints based on the process algebra CSP. Namely, given a CSP specification ϕ, an execution monitor (cf. Authentication, Authorisation & Accountability Knowledge Area (Section 14.3.3.2)) is synthesised corresponding to the transition system for ϕ defined by CSP's operational semantics.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Topics	Cites
13.1 Motivation	
13.2 Foundations, Methods and Tools	[1135, 1140, 1141, 1142, 1143, 1144]
13.3 Hardware	[1195, 1196, 1197, 1198]
13.4 Cryptographic Protocols	[1214, 1215, 1216, 1217, 1218, 1219]
13.5 Software and Large-Scale Systems	[1297, 1298, 1299, 1300, 1301, 1302, 1303]
13.6 Configuration	[1367, 1368, 1369]

ACKNOWLEDGEMENTS

The author would like to specially thank Christoph Sprenger and Kenny Paterson for their valuable input.

Chapter 14 Authentication, Authorisation & Accountability

Dieter Gollmann | Hamburg University of Technology & Nanyang Technological University Singapore

Abstract

Access control builds on authorisation and authentication. This KA[1] will present the general foundations of access control and some significant instantiations that have emerged as IT kept spreading into new application areas. It will survey modes of user authentication and the way they are currently deployed, authentication protocols for the web, noting how new use cases have led to a shift from authentication to authorisation protocols, and the formalisation of authentication properties as used in today's protocol analysis tools. On accountability, the focus is on the management and protection of audit logs. The surveillance of logs to detect attacks or inappropriate behaviour is described in the Security Operations & Incident Management Knowledge Area (Chapter 8) while the examination of evidence following a breach of policy or attack is covered in the Forensics Knowledge Area (Chapter 9). Throughout the KA, we will flag technical terms that appear in more than one meaning in the academic and the trade literature.

14.1 INTRODUCTION

> "All science is either physics or stamp collecting." [Ernest Rutherford]

In some cases, IT systems may guarantee – by design – that no undesirable behaviour is possible. In other cases, IT systems exhibit such a degree of flexibility – also by design – that additional measures need to be taken to limit undesirable behaviour in accordance with the given circumstances. As noted by Lessig, this can be done by code in the system that excludes behaviour, which will violate certain rules, or it can be done by codes of conduct that the users of the system are expected to adhere to [1376]. In the latter case, disciplinary or legal processes deal with those that had broken the rules. This is the context for authentication, authorisation, and accountability.

Readers acquainted with the mores of academic writing may now expect definitions of core terms, maybe some refinement of terminology, and then an overview of the latest approaches in achieving authentication, authorisation, and accountability. As will be shown, this approach fails at the first hurdle. These three terms are overloaded to an extent that provides ample space for confusion and dispute. For example, *authorisation* stands both for the setting of rules and for checking compliance with those very rules. Readers should thus be cautious when studying the literature on this Knowledge Area.

Changes in the way IT is being used create their own challenges for taxonomies. How closely should terms be tied to the environment in which they first emerged? There is a habit in the trade and research literature of linking terms exclusively to a notional 'traditional' instantiation of some generic concept, and inventing new fashionable terms for new environments, even though the underlying concepts have not changed.

[1]The title of this KA, 'Authentication, Authorisation & Accountability' is abbreviated as AAA in CyBOK related documents and past versions of this KA. Please note that the acronym 'AAA' is also frequently used in other contexts to refer to 'Authentication, Authorisation, and Accounting', for example in connection with network access protocols such as Diameter.

14.2 CONTENT

This KA first addresses authorisation in the context of access control and presents the main flavours of access control in use today. The section on access control in distributed systems explains concepts used when implementing access control across different sites. The KA then moves to authentication, touching on user authentication and on authentication in distributed systems, and concludes with a discussion of logging services that support accountability.

14.3 AUTHORISATION

[1377, 1378, 1379, 1380, 1381]

In their seminal paper [1378], Lampson et al. postulate *access control = authentication + authorisation*. We will follow this lead and present authorisation in the context of access control, starting with an introduction to the concepts fundamental for this domain, followed by an overview of different policy types. Libicki's dictum, "connotation, not denotation, is the problem" [1382] also applies here, so we will pay particular attention to the attributes used when setting access rules, and to the nature of the entities governed by those rules. Code-based access control, mobile security, and Digital Rights Management will introduce new paradigms to access control, without changing its substance. We will then present design options for policy enforcement and discuss delegation and some important theoretical foundations of access control.

14.3.1 Access Control

Access control is "the process of granting or denying specific requests . . . " [1383]. This process needs the following inputs

- Who issued the request?
- What is requested?
- Which rules are applicable when deciding on the request?

"Who" in the first question is dangerous. The word suggests that requests always come from a person. This is inaccurate for two reasons. First, the source of a request could be a particular machine, a machine in a particular configuration, or a particular program, e.g. a particular Android app. Secondly, at a technical level, requests in a machine are issued by a process, not by a person. The question thus becomes, "for whom or what is the process speaking for when making the request?" "What is requested" is frequently given as a combination of an action to be performed and the object on which the action is to be performed. The rules are logical expressions that evaluate to a decision. In the elementary case, the decision is *permit* or *deny*. When policies get more elaborate, there may be reasons for adding an *indeterminate* decision. A decision may also prescribe further actions to be performed, sometimes called *obligations*.

14.3.1.1 Core Concepts

The term 'security policy' is used both for the general rules within an organisation that stipulate how sensitive resources should be protected, and for the rules enforced by IT systems on the resources they manage. Sterne had coined the terms *organisational policies* and *automated policies* to distinguish these two levels of discourse [1377].

When setting security policies, *principal* stands for the active entity in an access request. When policies directly refer to users, as was the case in the early stages of IT security, *user identities* serve as principals. Access control based on user identities is known as *Identity-Based Access Control (IBAC)*. In security policies that refer to concepts such as *roles* or to the program that issues a request, the principal is a role or a program. Principal may then generally stand for any security attribute associated with the issuer of a request. With this generalisation, any flavour of access control is by definition *attribute-based access control* (see Section 14.3.1.4).

Subject stands for the active entity making a request when a system executes some program. A subject *speaks for* a principal when the runtime environment associates the subject with the principal in an unforgeable manner. The original example for creating a subject that speaks for a principal is user log-in, spawning a process running under the user identity of the person that had been authenticated. The research literature does not always maintain this distinction between principals and subjects and one may find security policies referring to subjects. When policies refer to attributes of a user but not to the user's identity, user identities become a layer of indirection between principals and subjects [1384].

A subject is created, e.g., at log-in, and can be terminated, e.g. at log-out. Similarly, user identities are created through some administrative action and can be terminated, e.g., by deleting a user account. In practice, subjects have considerably shorter lifetimes than user identities. Processes that control industrial plants are a rare example of subjects that could live forever, but could be killed by system crashes.

Object is the passive entity in an access request. *Access operations* define how an object may be accessed by a subject. Access operations can be as elementary as *read*, *write*, *execute* in Linux, they can be programs such as *setuid programs* in Linux, and they can be entire workflows as in some flavours of UCON (Section 14.3.1.8).

Access rights express how a principal may access an object. In situations where there is a direct match between access operations and access rights, the conceptual distinction between access operations and access rights may not be maintained. *Permission* is frequently used as a synonym for access right. *Privilege* may also be used as a synonym for access right, e.g., Oracle9i Database Concepts Release 2 (9.2) states:

> "A privilege is permission to access a named object in a prescribed manner . . . "

Other systems, such as Windows, make a distinction between access rights and privileges, using privilege specifically for the right to access system resources and to perform system-related tasks. Operating systems and databases often have a range of *system privileges* that are required for system administration.

The *reference monitor* (more details in Section 14.3.2.2) is the component that decides on access requests according to the given policy.

14.3.1.2 Security Policies

Automated *security policies* are a collection of rules. The rules specify the access rights a principal has on an object. Conceptually, a policy could then be expressed as an *Access Control Matrix* with rows indexed by principals and columns indexed by objects [1120]. *Access Control Lists (ACLs)* stored with the objects correspond to the columns of this matrix; *capabilities* stored with principals correspond to the rows of this matrix (also see the Operating Systems & Virtualisation Knowledge Area (Chapter 11)).

Discretionary Access Control (DAC) and *Mandatory Access Control (MAC)* are two core policies formulated in the 1970s in the context of the US defence sector. Discretionary access control policies assign the right to access protected resources to individual user identities, at the discretion of the resource owner. In the literature, DAC may generically refer to policies set by resource owners but also to policies referring directly to user identities, i.e., to IBAC.

Mandatory access control policies label subjects and objects with *security levels*. The set of security levels is partially ordered, with a least upper bound and a greatest lower bound operator. The security levels thus form a *lattice*. In the literature, MAC may generically refer to policies mandated by the system as, e.g., in Security-Enhanced Linux (SELinux) [1385, 1386] and in Security-Enhanced (SE) Android [1387], or to policies based on security levels as in past products such as Trusted Xenix or Trusted Oracle. Policies of the latter type are also known as *multi-level security policies* and as *lattice-based policies*.

14.3.1.3 Role-based Access Control

In *Role-Based Access Control (RBAC)*, *roles* are an intermediate layer between users and the permissions to execute certain operations. Operations can be well-formed transactions with built-in integrity checks that mediate the access to objects. Users are assigned roles and are authorised to execute the operations linked to their active role. *Separation of Duties (SoD)* refers to policies that stop single users from becoming too powerful. Examples for SoD are rules stating that more than one user must be involved to complete some transaction, rules stating that a user permitted to perform one set of transactions is not permitted to perform some other set of transactions, the separation between front office and back office in financial trading firms is an example, or rules stating that policy administrators may not assign permissions to themselves. Static SoD rules are considered during user-role assignment, dynamic SoD must be enforced when a role is activated. The NIST RBAC model [1380] distinguishes between:

- *Flat RBAC*: users are assigned to roles and roles to permissions to operations; users get permissions to execute procedures via role membership; user-role reviews are supported.
- *Hierarchical RBAC*: adds support for role hierarchies.
- *Constrained RBAC*: adds separation of duties.
- *Symmetric RBAC*: adds support for permission-role reviews, which may be difficult to achieve in large distributed systems.

Many commercial systems support some flavour of role-based access control, without necessarily adhering to the formal specifications of RBAC published in the research literature. RBAC is an elegant and intuitive concept, but may become quite messy in deployment as suggested by comments in an empirical study on the use of RBAC [1388]. Practitioners note

that RBAC works as long as every user has only one role, or that "the enormous effort required for designing the role structure and populating role data" constitutes an inhibitor for RBAC.

14.3.1.4 Attribute-based Access Control

Attribute-Based Access Control (ABAC) is defined in [1389] as a *"logical access control methodology where authorisation to perform a set of operations is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes"*. This is a generic definition of access control that no longer reserves a special place to the user or to the user's role, reflecting how the use of IT systems has changed over time.

Access control may be performed in an application or in the infrastructure supporting the application. Access control in an infrastructure uses generic attributes and operations. The Linux access control system may serve as an example. Access control in an application uses application-specific attributes and operations. In this distinction, ABAC can be viewed as a synonym for application-level access control.

14.3.1.5 Code-based Access Control

Code-Based Access Control (CBAC) assigns access rights to executables. Policies may refer to code origin, to code identity (e.g., the hash of an executable), or to other properties of the executable, rather than to the identity of the user who had launched the executable. Origin can subsume the domain the code was obtained from, the identity of the code signer, a specific name space (.NET had experimented with *strong names*, i.e. bare public keys serving as names for name spaces), and more. CBAC can be found in the Java security model [1390] and in Microsoft's .NET architecture [1391].

The reference monitor in CBAC typically performs a *stack walk* to check that all callers have been granted the required access rights. The stack walk addresses the *confused deputy problem* [1392], where an unprivileged attacker manipulates a system via calls to privileged code (the confused deputy). Controlled invocation is implemented through *assert* statements; a stack walk for an access right will stop at a caller that asserts this right.

14.3.1.6 Mobile Security

Smartphones typically have a single owner, hold private user data, offer communication functions ranging from cell phone to NFC, can observe their surroundings via camera and microphone, and can determine their location, e.g., via GPS. On smartphones, apps are the principals for access control. The objects of access control are the sensitive data stored on a phone and the sensitive device functions on a phone.

Access control on a smartphone addresses the privacy requirements of the owner and the integrity requirements of the platform. Android, for example, divides permission groups into normal, dangerous, and signature permissions. Normal permissions do not raise privacy or platform integrity concerns; apps do not need approval when asserting such permissions. Dangerous permissions can impact privacy and need user approval. Up to Android 6.0, users had to decide whether to authorise a requested permission when installing an app. User studies showed that permissions were authorised too freely due to a general lack of understanding and risk awareness, see e.g. [1393]. Since Android 6.0, users are asked to authorise a permission

when it is first needed. Signature permissions have an impact on platform integrity and can only be used by apps authorised by the platform provider; app and permission have to be signed by the same private key. For further details see the Web & Mobile Security Knowledge Area (Chapter 16).

14.3.1.7 Digital Rights Management

Digital Rights Management (DRM) has its origin in the entertainment sector. Uncontrolled copying of digital content such as games, movies or music would seriously impair the business models of content producers and distributors. These parties hence have an interest in controlling how their content can be accessed and used on their customers' devices. Policies can regulate the number of times content can be accessed, how long content can be sampled for free, the number of devices it can be accessed from, or the pricing of content access.

DRM turns the familiar access control paradigm on its head. DRM imposes the security policy of an external party on the system owner rather than protecting the system owner from external parties. *Superdistribution* captures the scenario where data are distributed in protected containers and can be freely redistributed. Labels specifying the terms of use are attached to the containers. The data can only be used on machines equipped with a so-called Superdistribution Label Reader that can unpack the container and track (and report) the usage of data, and enforce the terms of use [1394]. The search for such a tamper resistant enforcement mechanism was one of the driving forces of Trusted Computing.

The level of tamper resistance required depends on the anticipated threats. Trusted Platform Modules are a hardware solution giving a high degree of assurance. Enclaves in Intel SGX are a solution in system software. Document readers that do not permit copying implement this concept within an application. *Sticky policies* pursue a related idea [1395]; policies stick to an object and are evaluated whenever the object is accessed.

Attestation provides trustworthy information about a platform's configuration. *Direct anonymous attestation* implements this service in a way that protects user privacy [1396]. Remote attestation can be used with security policies that are predicated on the software running on a remote machine. For example, a content owner could check the software configuration at a destination before releasing content. In the FIDO Universal Authentication Framework (FIDO UAF) just the model of the authenticator device is attested. All devices of a given model hold the same private attestation key [1397].

For a brief period, it was fashionable to use Digital Rights Management as the generic term subsuming 'traditional' access control as a special case.

14.3.1.8 Usage Control

Usage Control (UCON) was proposed as a framework encompassing authorisations based on the attributes of subject and object, *obligations*, and *conditions* [1379]. In [1379], obligations are additional actions a user has to perform to be granted access, e.g., clicking on a link to agree to the terms of use. In today's use of the term, obligations may also be actions the system has to perform, e.g., logging an access request. Such actions may have to be performed before, during or after an access happens. Conditions are aspects independent of subject and object, e.g., time of day when a policy permits access only during office hours or the location of the machine access is requested from. Examples for the latter are policies permitting certain requests only when issued from the system console, giving access only

from machines in the local network, or policies that consider the country attributed to the IP address a request comes from. Many concepts from UCON have been adopted in the XACML 3.0 standard [1398].

Usage control may also include provisions for what happens after an object is accessed, e.g., that a document can be read but its content cannot be copied or adjustment of attributes after an access has been performed, e.g., decrementing the counter of free articles a visitor may access. In another interpretation, 'traditional' access control deals with the elementary access operations found at an infrastructure level while usage control addresses entire work flows at the application level. In telecom services, usage control may put limits on traffic volume.

14.3.2 Enforcing Access Control

To enforce a security policy, this policy first has to be set. For a given request, a decision has to be made about whether the request complies with the policy, which may need additional information from other sources. Finally, the decision has to be conveyed to the component that manages the resource requested. In the terminology of XACML, this involves

- Policy Administration Points where policies are set,
- Policy Decision Points where decisions are made,
- Policy Information Points that can be queried for further inputs to the decision algorithm,
- Policy Enforcement Points that execute the decision.

14.3.2.1 Delegation and Revocation

Delegation and *granting* of access rights both refer to situations where a principal, or a subject, gets an access right from someone else. The research literature does not have firm definitions for those terms, and the trade literature even less so. Granting tends to be used in a generic sense; *granted access rights* often refer to the current access rights of a subject that delivers a request to a reference monitor. Delegation is sometimes, but not always, used more narrowly for granting short-lived access rights during the execution of a process. For example, XACML distinguishes between *policy administration* and *dynamic delegation* that *"permits some users to create policies of limited duration to delegate certain capabilities to others"* [1399].

A second possible distinction lets delegation refer only to the granting of access rights held by the delegator, while granting access also includes situations where a managing principal assigns access rights to others but is not permitted to exercise those rights itself.

Rights may not always be granted in perpetuity. The grantor may set an expiry date on the delegation, a right may be valid only for the current session, or there may be a *revocation* mechanism such as the Online Certificate Status Protocol (OCSP) for X.509 certificates (see Section 14.4.1). OCSP is supported by all major browsers. Revocation lists are suitable when online checks are not feasible and when it is known in advance where a granted right may be consumed.

14.3.2.2 Reference Monitor

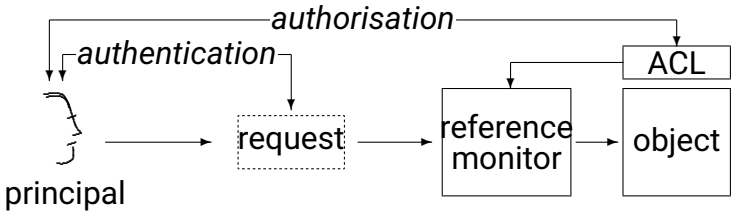


Figure 14.1: Access Control = Authentication + Authorisation.

In its original definition, the *reference monitor* was the abstract machine mediating all accesses by subjects to objects. The *security kernel* was a trustworthy implementation of the reference monitor. The *Trusted Computing Base (TCB)* was the totality of protection mechanisms within a computer system responsible for enforcing a security policy (definitions follow [1400]). There has been some interpretation creep since. The reference monitor component in current operating systems, e.g., the Security Reference Monitor in Windows, would actually be the security kernel from above, and TCB is today sometimes used in a limited sense to stand only for the security kernel. A reference monitor performs two tasks.

- It *authenticates* any evidence supplied by the subject with an access request. Traditionally, the user identity the subject was speaking for was authenticated.
- It evaluates the request with respect to the given policy. The early literature on refers to this task as *authorisation* (of the request), see e.g., [1378].

However, *authorisation* also stands for the process of setting a security policy; principals are authorised to access certain resources. This overloads the term *authorisation*, applying it both to principals and to requests, but with different meanings. A convention that refers to "authorised principals" and "approved requests" would resolve this issue. Figure 14.1 represents the view of access control adopted in operating systems research around 1990. In Section 14.5.3.4, *authorisation* will stand for the granting of access rights to principals.

The *decision algorithm* executed by the reference monitor has to identify the applicable policies and rules, and try to collect the evidence those rules refer to from *Policy Information Points*. For situations where more than one rule is applicable for a given request, *rule combining algorithms* specify the final decision.

14.3.2.3 Types of Reference Monitors

Schneider describes three types of reference monitors [1381]:

- Reference monitors that only see the system calls to protected resources, but not the entire program executed. This type of reference monitor, called *execution monitor* in [1381], is implemented in many operating systems.
- Reference monitors that can see the entire program and analyse its future behaviour before making an access control decision.
- Instructions guarding all security relevant operations are in-lined into the program; in all other respects the in-lined program should behave as before. This type of reference monitor, called *in-line reference monitor* in [1381], is mostly used to deal with software security issues, see e.g. [1189].

14.3.3 Theory

14.3.3.1 Security Models

Security models are high-level specifications of systems intended to enforce certain security policies. Such models can be used in a formal security analysis to show that a lower-level specification faithfully implements the model.

The Bell-LaPadula (BLP) model [1015] is a state machine model for discretionary and mandatory access control policies that adapted existing rules governing access to classified data to IT systems. The mandatory access control policies state that a subject can only read objects at its own or at a lower level (no read up). To prevent unauthorised declassification of data, a subject may only write to objects at its own or at a higher level ($*$-property, no write down). The SeaView model extends the BLP policies to multi-level secure relational databases [1401]. *Polyinstantiation* of database entries, i.e., keeping separate entries at the different security levels, is used to prevent integrity checks from causing information leaks. BLP was highly influential in computer security into the 1990s.

The Biba model captures integrity policies based on integrity levels [1016]. The access rules are the dual of the BLP model, no read down and no write up, but have no predecessors in the world of paper documents. The *low watermark policies* in Biba introduce dynamic policies (mutable in the terminology of UCON, Section 14.3.1.8) that adapt the integrity level of an object depending on the integrity level of the subject performing the access operation.

The Clark-Wilson model [1402] places *well-formed transactions* as an intermediate layer between principals and objects; *constrained data items* can only be accessed via those transactions; users (principals) are 'labelled' with the transactions they are authorised to execute. This model captures the way data are processed in enterprise systems. The Chinese Wall model [1403] formalises dynamic *conflict of interest policies* that apply in financial consultancy businesses when working for clients that are commercial competitors. Hence, the act of accessing data for one client dynamically removes permissions to access data from other clients in the relevant conflict-of-interest class.

The Harrison, Ruzzo, and Ullman model (HRU) [1404] provides a context for examining a core question in policy management: is it possible to decide whether an access right may *leak* to a subject through some sequence of commands? This problem is undecidable in general, but may be decidable under certain restrictions.

14.3.3.2 Enforceable Policies

Schneider has examined the relationship between different kinds of and different kinds of reference monitors [1381]. Security policies are defined as predicates over execution traces, taking a broader view than Section 14.3.1.2 where rules applied to individual access operations. Policies that only consider the given execution trace are called *properties*. Information flow policies that require an execution trace to be indistinguishable from some benign execution trace are thus not properties. It is shown that only *safety properties* can be enforced by execution monitors (see Section 14.3.2.3).

14.3.3.3 Access Control Logics

Access control and delegation logics [1405] specify calculi for reasoning about composite principals in distributed systems. The calculus for in distributed systems [1406] was developed as a formal specification for parts of the Digital Distributed Systems Security Architecture. In such an architecture, cryptographically secured sessions can be established between parties. For example, when a session is established with a principal on some other machine, the session key can be treated as a subject for access control that *speaks for* that principal.

14.4 ACCESS CONTROL IN DISTRIBUTED SYSTEMS

[1407, 1408, 1409, 1410]

Access control in distributed systems deals with technology issues and with organisational issues. Any distributed system needs mechanisms for securely transmitting access requests, attributes, policies, and decisions between nodes. These mechanisms are largely based on cryptography. The requirement for mechanisms that identify and retrieve all policies relevant for a given request may become more pronounced than in centralised settings.

In federated systems where several organisations collaborate, can be set by different parties. This demands some common understanding of the names of principals, attributes, and attribute values so that policies issued by one party can be used in decisions by some other party. Arriving at such a common understanding adds to the practical challenges for RBAC listed in Section 14.3.1.3.

We first introduce core concepts for this domain. We will then cover origin-based access control, examining cross-site scripting from the viewpoint of access control. Federated Access Control and the use of cryptography in access control are explored further.

14.4.1 Core Concepts

The literature on access control in distributed systems uses the following related terms, but the distinction between those terms is fluid.

- A *certificate* is a digitally signed data structure, created by an issuer, binding a subject (not to be confused with the term subject as introduced earlier) to some further attributes. The emphasis is on protection by a digital signature.
- A *credential* is something presented to gain access. Examples for credentials are passwords or fingerprints. In distributed systems, a credential can be a data structure containing attributes of the subject. The emphasis is on evidence submitted to the decision algorithm.
- A *token* records ('encapsulates') the result of some authorisation decision. For example, in operating systems the access token contains the security credentials for a login session. The emphasis is on conveying the result of an access decision to some enforcement point. So-called *bearer tokens* are not tied to a specific subject and can be used by anyone in possession of the token.

X.509 certificates [1411] can be used for implementing user-centric access control. Identity certificates bind user identities to public verification keys. Attribute certificates bind user identities to access rights. Attribute certificates correspond closely to ACL entries.

14.4.2 Origin-based Policies

In web applications, clients and servers communicate via the HTTP protocol. The client browser sends HTTP requests; the server returns result pages. The browser represents the page internally in the `document` object in the Document Object Model (DOM). Security policies specify which resources a script in a web page is allowed to access, or which servers an `XMLHttpRequest` may refer to. Web applications are thus the principals in access control. By convention, principal names are the domain names of the server hosting an application; the policy decision point (cf. Section 14.3.2) at the client side is located in the browser.

The prototype policy for web applications is the *Same-Origin-Policy (SOP)*, stating that a script may only connect back to the origin it came from or that an HTTP cookie is only included in requests to the domain that had placed the cookie. Two pages have the same origin if they share protocol, host name and port number. Certain actions may be exempt from the same origin policy. For example, a web page may contain links to images from other domains, reflecting a view that images are innocuous data without malign side effects. There exist variations of the SOP, e.g., policies for cookies that also consider the directory path. There is also the option to set the `HttpOnly` flag in a `Set-Cookie` HTTP response header so that the cookie cannot be accessed by client side scripts.

Sender Policy Framework (SPF) [1412] implements origin-based access control in the email system as a measure against spoofing the sending domain of an email. Domain owners publish SPF policies in their DNS zone. An SMTP server can then use the domain part of the MAIL FROM identity to look up the policy and consult this policy to check whether the IP address of the SMTP client is authorised to send mail from that domain.

14.4.2.1 Cross-site Scripting

Cross-site scripting attacks on web applications can be treated as cases of failed authentication in access control. The browser lets all scripts that arrive in a web page speak for the origin of that page. A browser would then run a script injected by the attacker in the context of an origin other than the attacker's. Content Security Policy (CSP) refines SOP-based access control. The web server conveys a policy to the browser that characterises the scripts authorised to speak for that server [1409]. Typically, this is done by specifying a directory path on the web server where authorised scripts (and other web elements) will be placed.

The use of CSP in practice has been examined in [1413], observing that the `unsafe-inline` directive disabling CSP for all pages from a given domain was widely used. This is a familiar policy management issue. A new security mechanism is deployed but quickly disabled because it interferes too much with established practices. Moreover, CSP had an inherent vulnerability related to *callbacks*. Callbacks are names of scripts passed as arguments to other (authorised) scripts, but arguments are not covered by CSP. In *strict* CSP policies, the server declares a nonce in the CSP policy it sends to the client as the script source. The server also includes this nonce as an attribute in all scripts fetched by the client. The client's browser only accepts scripts that contain this nonce as an attribute. Nonces must only be used once and must be unpredictable.

14.4.2.2 Cross-origin Resource Sharing

When *mashups* of web applications became popular, this exposed another limitation of the same origin policy: there was no built-in mechanism for specifying exceptions to the SOP. Mashup designers initially had to find ways of circumventing the SOP enforced by the browsers. The Cross-Origin Resource Sharing (CORS) protocol was then introduced to support policies for sharing resources cross-origin [1414]. When a script requests a connection to a target other than its own origin, the browser asks the target to authorise the connection request. The decision at the target considers evidence supplied by the browser, such as the origin of the script or user credentials associated with the request.

CORS specifies a set of HTTP headers to facilitate this exchange. *Preflight Requests* include an `Access-Control-Request-Method` header that informs the target about the access intended. The response lists methods and headers the target grants to the given origin. CORS requests are by default sent without user credentials. The target can set the `Access-Control-Allow-Credentials: true` header to indicate that user credentials may be provided with requests to access a resource. The target must also specify an origin in the `Access-Control-Allow-Origin` header. Otherwise, the browser will not pass on the target's response to the script that had made the request.

14.4.3 Federated Access Control

When organisations join to form a federated security domain, the import of identities, credentials, policy rules, and decisions from different contexts (name spaces) becomes an important security issue. A federation may have several *Policy Administration Points* where policies are defined, *Policy Decision Points* where decisions on access requests are made, *Policy Enforcement Points* where the decisions are enforced, and *Policy Information Points* where additional evidence required for evaluating an access request can be obtained.

Trust management as originally conceived in PolicyMaker [1407] refers to access control systems for such scenarios. *Federated identity management* deals with the management of digital identities in a federation, and in particular with single sign-on in a federation. In Web Services, related standards for authentication (SAML, Section 14.5.3.3) and access control (XACML) have been defined. OAuth 2.0 and OpenID Connect (Section 14.5.3.4) provide user authentication and authorisation via access tokens.

Binder is an instance of a federated access control system [1408]. The Binder policy language is based on Datalog. Policies are logical clauses. Binder *contexts* are identified by public keys and export statements by signing them with the corresponding private key. The decision algorithm is *monotonic*; presenting more evidence cannot reduce the access rights granted.

14.4.4 Cryptography and Access Control

Access control mechanisms in an operating system implement a logical defence. Access requests passed via the reference monitor will be policed. This includes requests for direct memory access. However, data are stored in the clear and a party with physical access to the storage medium can retrieve the data and thus bypass logical access control. When solutions for the protection of unclassified but sensitive data were evaluated in the U.S. in the 1970s, it was decided that encrypting the data was the best way forward. Access control would then be applied to the keys needed to unlock the data.

14.4.4.1 Attribute-Based Encryption

Cloud computing has raised the interest in access control on encrypted data over the past decade. Storing data in encrypted form protects their confidentiality but creates a key management challenge. *Attribute-Based Encryption (ABE)* addresses this challenge by constructing encryption schemes that enforce attribute-based decryption policies. Policies are logical predicates over attributes, represented as access structures. The Key Generator is a Trusted Third Party that generates private keys and has to check a user's policy / attributes before issuing a private key. The Key Generator is thus in a position to recreate private keys.

Key-Policy Attribute-Based Encryption (KP-ABE) works with policies that define a user's access rights [1410]. From the corresponding access structure, the Key Generator creates a private decryption key. Documents are encrypted under a set of attributes. In Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [1415], the policy refers to the document and the access structure is used for encryption. The user's private key created by the Key Generator depends on the user's attribute set. In both variants, decryption is possible if and only if the given attribute set satisfies the given access structure.

A study of the feasibility of ABE in realistic dynamic settings had concluded that the overheads incurred by those schemes were still prohibitive [1416]. Efficient encryption and decryption do not necessarily imply an efficient access control system.

14.4.4.2 Key-centric Access Control

In distributed systems, access requests may be digitally signed. Access rights could then be granted directly to the public verification key without the need to bind the public key to some other principal. SPKI/SDSI uses authorisation certificates for implementing key centric access control, where (names of) public keys are bound to access rights [1417]. The right to further delegate an access right is controlled by a delegation flag.

Cryptographic keys are rarely suitable principals for access control, however. They would need to have an obvious meaning in the application domain that provides the context for a given security policy. In most cases, cryptographic keys would be subjects speaking for some principal. *Constrained delegation* refines the basic delegation mechanism of SPKI/SDSI so that separation of duties policies can be enforced [1418].

14.5 AUTHENTICATION

[1217, 1419, 1420, 1421, 1422, 1423]

Authentication in a narrow sense verifies the identity of a user logging in – locally or remotely – and binds the corresponding user identity to a subject. User authentication based on passwords is a common method. Some applications have adopted biometric authentication as an alternative. Authentication in distributed systems often entails key establishment. Some security taxonomies thus reduce authentication to a 'heartbeat' property to separate authentication from key establishment. The design of authentication protocols is a mature area in security research with good tool support for formal analysis. Standard protocols such as Kerberos, SAML, or OAuth are deployed widely today.

We will give a brief overview of identity management before moving to password-based and biometric user authentication. We then cover authentication protocols from the Needham-Schroeder protocol via Kerberos and SAML to OAuth 2.0, observing that OAuth 2.0 is more of an authorisation protocol than an authentication protocol. We conclude with an overview of formalisations of authentication properties that serve as the basis for a formal analysis of authentication protocols.

14.5.1 Identity Management

Following NIST, *"identity management systems are responsible for the creation, use, and termination of electronic identities"*. This includes operational aspects when creating and deleting electronic identities. On creation, one question is how strongly electronic identities must be linked to persons. In some sensitive areas, strong links have to be established and documented. For example, money laundering rules may demand a thorough verification of an account holder's identity. In other areas, electronic identities need not to be tied to a person. *Privacy by design* implies that such applications should use electronic identities that cannot be linked to persons. Identity management may also link access rights to an electronic identity, either directly or via some layer of indirection such as a role. Electronic identities should be terminated when they are no longer required, e.g. when a person leaves an organisation. Care has to be taken that this is done on all systems where this identity had been registered.

Electronic identities exist at different layers. There are identities for internal system purposes, such as user identities in an operating system. These identities must be locally unique and could be created by system administrators (Linux). This can lead to problems when an identity is taken out of use and re-assigned later. The new user may get unintended access to resources the predecessor had access to. When organisations merge, collisions between identities may arise that identity management then must address. Alternatively, identities could be long random strings (Windows). The probability for one of the problems just mentioned to arise is then negligible, but when a user account is re-created, a new random identity is assigned so access rights have to be reassigned from scratch.

Electronic identities such as user names and email addresses could be random strings, but it is often preferable to assign understandable identities. There is, for example, merit in communicating with meaningful email addresses. Email addresses can be taken out of use and re-assigned later, but a user may then receive emails intended for its previous owner.

Web applications often use email addresses as electronic identities. This is convenient for contacting the user, and it is convenient for users as they do not have to remember a new

identity. There are alternatives, such as FIDO UAF (Section 14.5.2.3), where electronic identities are randomly created public keys and a back channel for resetting passwords is not required as no passwords are used.

Identity management can also be viewed from a person's perspective. A person using different identities with different organisations may want to manage how identities are revealed to other parties.

14.5.2 User Authentication

Access requests are issued by subjects. Subjects can be associated with security attributes when they are created or during their lifetime. Authentication can then be viewed as the service that validates the security attributes of a subject when it is created. When subjects are created due to some user action, and when their security attributes depend on the corresponding user identity, *user authentication* has to give a reasonable degree of assurance that the user identity linked to the subject belongs to the user who had triggered the creation of the subject. The degree of assurance (strength of authentication) should be commensurate with the severity of the risk one wants to mitigate. The term *risk-based authentication* thus states the obvious.

User authentication can also support accountability, as further elaborated in Section 14.6. *Authentication ceremony* refers to the steps a user has to go through to be authenticated.

There are access control systems where the security attributes of a subject persist throughout the lifetime of that subject. Many operating systems adopt this approach. Policy changes do not affect active processes, but the lifetime of subjects is limited, which limits the period when the new policy is not applied consistently. Alternatively, the attributes of a subject are checked each time it issues a request. For example, a user already logged in to a banking application is authenticated again when requesting a funds transfer. When the focus moves from the subject to individual requests, authentication can be viewed as the service that checks the validity of the security attributes submitted with the request to the decision algorithm.

14.5.2.1 Passwords

When passwords are employed for user authentication, protective measures at the system side include the storing of hashed (Unix, Linux) or encrypted (Windows) passwords, the salting of passwords, and shadow password files that move sensitive data out of world-readable password files. Protective measures at the user side include guidance on the proper choice and handling of passwords, and security awareness programs that try to instil behaviour that assures the link between a person and a principal. Recommendations in this area are changing. The Digital Identity Guidelines published by NIST build on assessments of the observed effectiveness of previous password rules and reflect the fact that users today have to manage passwords for multiple accounts [1423]. The new recommendations advise

- against automatic password expiry; passwords should only be changed when there is a reason;
- against rules for complex passwords; password length matters more than complexity;
- against password hints or knowledge-based authentication; in an era of social networks too much information about a person can be found in public sources;
- to enable "show password while typing" and to allow paste-in password fields.

Password-based protocols for remote authentication are RADIUS, DIAMETER (both covered in the Network Security Knowledge Area (Chapter 19)), HTTP Digest Authentication, and to some extent Kerberos (Section 14.5.3.2). Password guidance is further discussed in the Human Factors Knowledge Area (Chapter 4).

14.5.2.2 Biometrics for Authentication

Numerous well-rehearsed arguments explain why passwords work poorly in practice. Biometrics are an alternative that avoids the cognitive load attached to password-based authentication. Fingerprint and face recognition are the two main methods deployed for biometric user authentication, known as *verification* in that domain.

Biometric features must be sufficiently unique to distinguish between users, but fingerprints or faces cannot be considered as secrets. Fingerprints are left in many places, for example. Biometric features are thus better treated as public information when conducting a security analysis and the process of capturing the features during authentication has to offer an adequate level of *liveness detection*, be it through supervision of that process or through device features. Employing biometrics for user authentication makes the following assumptions:

- The biometric features uniquely identify a person; face, fingerprints, and iris patterns may serve as examples.
- The features are stable; the effects of aging on fingerprint recognition are surveyed, e.g., in [1424].
- The features can be conveniently captured in operational settings.
- The features cannot be spoofed during user authentication.

User authentication, known as *verification* in biometrics, starts from a *template* captured by a device. From the template, a *feature vector* is extracted. For example, the template may be the image of a fingerprint, the features are the positions of so-called *minutiae* (ridge endings, bifurcations, whorls, etc.). Users initially register a reference feature vector. During authentication, a new template is captured, features are extracted and compared with the reference values. A user is authenticated if the number of matching features exceeds a given threshold. This process may fail for various reasons:

- Failure to capture: this may happen at registration when it is not possible to extract a sufficient number of features, or during authentication.
- False rejects: the genuine user is rejected because the number of matches between reference features and extracted features is insufficient.
- False accepts: a wrong user is accepted as the matching threshold is exceeded.
- Spoofing: to deceive the device capturing the template, some object carrying the user's features is presented. Liveness detection tries to ensure that templates are captured from the very person that is being authenticated [1425].

Biometric authentication based on face recognition or fingerprints is used increasingly at automated border control gates [1426]. It has also become a feature on mobile devices, see e.g. [1427]. A survey of the current state-of-the-art approaches to biometric authentication is given in [1428].

14.5.2.3 Authentication Tokens

Authentication by password relies on "something you know". Biometric authentication builds on "who you are". In a further alternative, users are issued with a device (a.k.a. *token* or *security key*, not to be confused with a cryptographic key) that computes a OTP synchronised with the authenticator, or a response to a challenge set by the authenticator. Possession of the device is then necessary for successful authentication, which is thus based on "something you have".

A token could be a small hand-held device with an LED display for showing an OTP that the user enters in a log-in form; RSA SecureID and YubiKey are examples for this type of token. A token could come with a numeric keypad in addition to the LED display and with a 'sign' button. The holder could then receive a challenge, e.g., an 8-digit number, enter it at the keypad, press 'sign' to ask the token to compute and display the response, and then enter the response in a log-in form. Some e-banking services use this type of token for account holder authentication. With PhotoTAN devices, the challenge is sent as a QR code to the user's computer and scanned from the screen by the PhotoTAN device. When authentication is based on a secret shared between token and server, different tokens must be used for different servers.

The FIDO authenticator is a token that can create public key / private key pairs; public keys serve as identifiers, private keys are used for generating digital signatures [1397]. In FIDO UAF, users register a public key with a server. The same token can be used for different servers, but with different keys. User authentication is based on a challenge-response pattern (Section 14.5.4.1), where the user's authenticator digitally signs the response to the server's challenge. The response is verified using the public key registered with the server.

In some applications, possession of the token is sufficient for user authentication. In other applications, authentication is a two-stage process. First, the token authenticates the user, e.g., based on a PIN or a fingerprint. In a second stage, the server authenticates the token. It will depend on the threat model whether 'weak' authentication in the first stage and 'strong' authentication in the second stage can provide adequate security.

Apps on smartphones can provide the same functionality as authentication tokens, but smartphones are not dedicated security devices. User authentication may then be compromised via attacks on the smartphone. This may become even easier when smartphones come with a secondary authentication mechanism for use when a device is partially locked, with a less onerous but also less secure authentication ceremony. This creates a conflict between the interests of smartphone manufacturers who value ease-of-use of a communications device, and the interests of the providers of sensitive applications searching for a security token.

14.5.2.4 Behavioural Authentication

Behavioural authentication analyses "what you do", lending itself naturally to *continuous authentication*. Keystroke dynamics [1429, 1430] can be captured without dedicated equipment. Characteristic features of hand writing are writing speed and pen pressure [1431]. Here, special pens or writing pads need to be deployed. Voice recognition needs a microphone. Smartphones come with various sensors such as touch screens and microphones that are being utilised for behavioural authentication today. The requirements on behavioural authentication are the same as those listed in Section 14.5.2.2:

- The behavioural features uniquely identify a person.
- The features are stable and unaffected by temporary impairments.

- The features can be conveniently captured in operational settings.
- The features cannot be spoofed during user authentication.

Advocates of continuous authentication promise *minimum friction, maximum security*. Behavioural authentication does not inconvenience the user with authentication ceremonies, but variations in user behaviour may cause false rejects. For example, how will a severe cold affect voice recognition? There needs to be a smooth fall-back when behavioural authentication fails. Security depends on the strength of liveness detection. For example, will voice recognition detect synthesised speech or a very proficient human voice imitator? Without a precise threat model, behavioural authentication can only offer uncertain security guarantees. There is a growing research literature on different modes of behavioural authentication. Criteria for assessing the actual contributions of this research include sample size and composition, whether longitudinal studies have been performed, the existence of an explicit threat model and resistance to targeted impersonation attempts.

14.5.2.5 Two-factor Authentication 2FA

Multi-factor authentication combines several user authentication methods for increased security. The European Payment Services Directive 2 (PSD2, Directive (EU) 2015/2366), written for the regulation of financial service providers, prescribes 2FA for online payments (with a few exceptions). PSD2 thus is a case study on rolling out large scale 2FA solutions.

The two factors could be a password and an authentication token for computing Transaction Authentication Numbers (TANs) uniquely tied to the content of a transaction. The token could be a separate device; if the device is tied to one payment service only, customers would have to carry multiple devices with them. For devices that can be used with several services, some level of prior standardisation is required. The FIDO alliance has been promoting its standards for PSD2 compliant two-factor authentication.

The token could be a smartphone registered with the service; customers could then install apps for several services on the same device. This approach has been favoured by many banks. However, when passwords (or PINs) and TANs are handled by the same device, the two mechanisms are no longer independent, reducing the security gains claimed for 2FA.

In contrast to the European Trust Services and Electronic identification regulation (eID Directive - Regulation (EU) No 910/2014) that specifies requirements on secure signature creation devices, PSD2 does not impose security requirements on the devices used for user authentication but wants “to allow for the use of all common types of devices (such as computers, tablets and mobile phones) for carrying out different payment services”. PSD2 and the eID Directive thus strike different balances between ease-of-use and security, a trade-off notoriously difficult to get right.

14.5.3 Authentication in Distributed Systems

When methods for user authentication in distributed systems were first designed, an authenticated *session* took the place of a process speaking for the user. Authenticated sessions were constructed on the basis of cryptographic keys. In the terminology of Section 14.3.1, those session keys became the subjects of access control, and *key establishment* became a core feature of the user authentication process.

14.5.3.1 Needham-Schroeder Protocol

The Needham-Schroeder protocol is a key establishment protocol that employs an authentication server as an intermediary between a client and a server [1419]. Client and server share secret keys with the authentication server respectively. Nonces, values that are used only once, are used as a defence against replay attacks. The client does not have to share individual long term secrets with all servers it wants to access, it needs just one shared secret with the authentication server. The authentication server issues a session key to client and server, and has to be trusted to properly authenticate the client and the server.

14.5.3.2 Kerberos

The Kerberos protocol [1420] adapted the Needham-Schroeder protocol for user authentication at the MIT campus. Most major operating systems have since adopted (variations of) Kerberos for user authentication.

Users share a password with a Kerberos Authentication Server (KAS) they are registered with. From this password, the client and the KAS derive a symmetric encryption key. In its response to a client request ① the KAS sends an encrypted session key to the client, together with a *ticket* containing that session key encrypted under a key shared between the KAS and the server ②. If the correct password is entered at the client, the session key can be decrypted. The ticket is forwarded to the server ③. Client and server now share the session key, and the server can return an authenticator constructed with the session key ④.

A Ticket Granting Server (TGS) may provide a further layer of indirection between client and server. The KAS would issue a session key for the TGS and a Ticket Granting Ticket (TGT) to the client. With the session key and the TGT, the client then requests a ticket for the resource server. The TGS checks the TGT and can apply an access control policy to decide whether to issue a ticket for use with the server. If the request is approved, the TGS issues another session key and a ticket encrypted under a secret key shared between TGS and server.

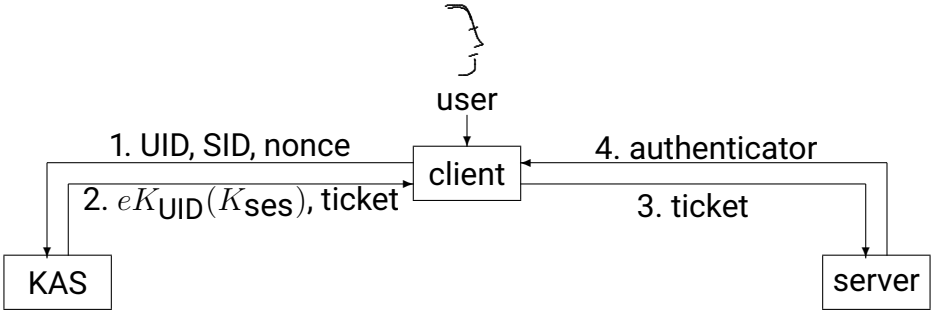


Figure 14.2: Message flow in the Kerberos protocol.

14.5.3.3 SAML

The Security Assertion Markup Language (SAML) v2.0 defines meta-protocols for authentication in web services [1421]. Meta-protocols specify high-level message flows that can be bound to various underlying protocols such as Kerberos. Applications that use SAML for authentication then need not be aware of the underlying protocol used. Many cloud service providers, e.g., AWS, Azure, IBM, are using SAML for user authentication via a browser.

Security tokens containing *assertions* are used to pass information about a principal (usually an end user) between a SAML authority (a.k.a. Identity Provider (IdP) or asserting party), and a SAML consumer (a.k.a. Service Provider (SP) or relying party). Assertions can be passed via the client to the relying party (browser POST profile, Figure 14.3) or be pulled by the relying party from the asserting party via a handle (artefact) passed via the client (browser artefact profile, Figure 14.4). The specification of SAML messages and assertions is based on XML.

An authentication assertion has to include the name of the identity provider and the user identity, but this is insufficient. This was shown to be the case by an attack against the implementation of Service Provider-initiated single sign-on with Redirect/POST bindings used at that time in Google Applications [1359]. In this implementation, authentication assertions included just the two aforementioned fields. A malicious Service Provider could ask a user for authentication at a specific Identity Provider (step (0) in Figure 14.3) and then re-use the assertion to impersonate the user with another Service Provider that relied on the chosen Identity Provider and where the user was known by the same user identity, e.g., an email address.

The specification of SAML Redirect/POST Bindings includes the Service Provider's ID and a request ID issued by the Service Provider in the authentication assertion. Hence, a Service Provider would only accept an assertion issued in reaction to a pending authentication request.

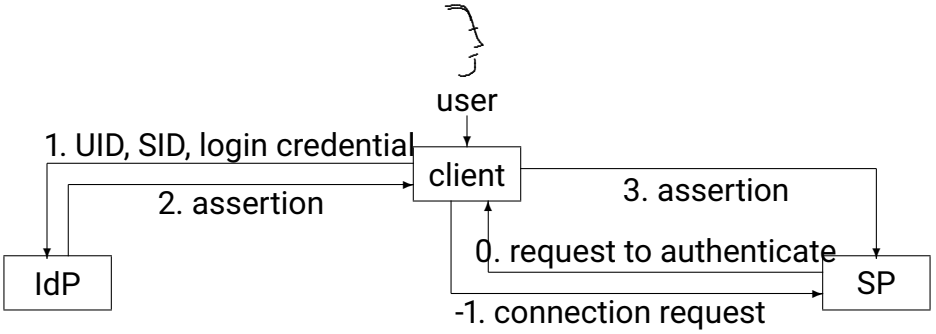


Figure 14.3: Message flow in the SAML POST profile.

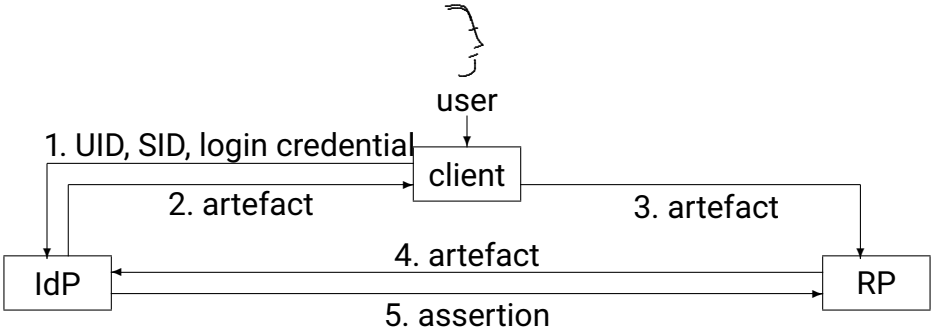


Figure 14.4: Message flow in the SAML artefact profile.

SAML was introduced as a meta-protocol to isolate web services from underlying authentication protocols and from different underlying communication protocols. It was conceived as a

federated single sign-on protocol where the relying party decides how to use assertions when making decisions according to its own security policy.

In the practical deployment of SAML, parsing XML documents – the price to be paid for employing a meta-protocol – can create non-trivial overheads and can introduce security vulnerabilities. Furthermore, the advent of smartphones has made it easier to access the internet from mobile user devices, removing one of the reasons for introducing a meta-protocol between web services and the underlying IT systems.

14.5.3.4 OAuth 2 – OpenID Connect

Newer protocols such as OAuth 2.0 [1422] and OpenID Connect [1432] run directly over HTTP and provide authentication and authorisation. The parties involved include a user who owns resources, the resource owner, a resource server that stores the user's resources, a so-called client application that wants to be granted access to those resources, and an Authorisation Server (AS) that can authenticate users and client applications.

Clients have to be registered with the AS. They will receive a public client ID and a client secret shared with the AS. This secret is used for establishing secure sessions between the client and the AS. The client also registers redirect_URIs with the AS. The AS will redirect a user agent only to those registered redirect_URIs. Proper definition of the redirect_URIs is primarily a matter for the client, and can also be enforced by the AS. Weak settings are open to exploitation by attackers.

In an OAuth protocol run (a high level overview is given in Figure 14.5), the user agent (browser) has opened a window for the client application. In the client window, an *authorisation request* can be triggered ①; the request also contains a redirect_URI. The user agent then typically conveys the authorisation request and the user's authorisation to the AS ②. A secure session between the user agent and the AS is required, and may already exist if the user has logged in previously at the AS. If authorisation is granted, an *authorisation grant* is returned to the user agent ③, which will pass it on to the redirect_URI given by the client ④. The client then posts the *authorisation grant* and a redirect URI to the AS ⑤. It is assumed that the AS can authenticate this message as coming from the client. If the request is valid, the AS returns an *access token* to the *redirect URI* provided, where the token can be used to retrieve the resource from the resource server ⑥.

Authorisation requests and *authorisation grants*indexauthorisation grant are linked via a request ID, called *state* in OAuth. Omitting the request ID or using a fixed value had introduced vulnerabilities in applications using OAuth, see e.g. [1433, 1434].

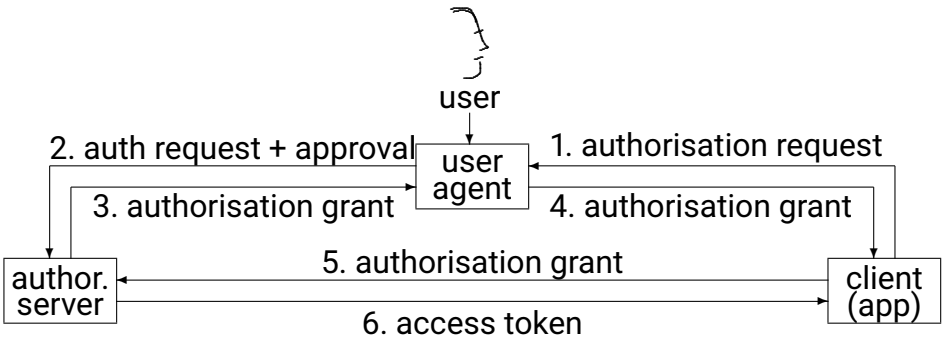


Figure 14.5: Message flow in OAuth 2.0.

There is a fundamental switch in focus compared to SSO protocols such as Kerberos and SAML

despite a considerable degree of similarity in the message flows. In an OAuth 2.0 protocol run the user is no longer the party requesting access to a resource owned by someone else, but the party granting access to resources owned by the user. OAuth 2.0 has thus become an authorisation protocol. Several assumptions about pre-existing trust relationships between parties have to be met for OAuth to be secure. Conversely, one cannot take for granted that the OAuth security properties still hold when the protocol is deployed in a new setting.

OpenID Connect puts user authentication back into the OAuth 2.0 message flow. The client application now doubles as a relying party, and the authorisation server becomes an authentication & authorisation server that issues digitally signed *id tokens* (authentication assertions in SAML diction). An id token contains the name of the issuer, the name of the authenticated user (called *subject*), the intended relying party (called *audience*), the nonce that had been sent with the authentication request, an indicator of authentication strength, and other fields.

14.5.4 Facets of Authentication

We have sketched how user authentication in distributed systems first integrated session and key establishment with the process of verifying a user's identity, and later established authorisation practices to access a user's resources. In communication security, *peer entity authentication* refers to the process of verifying the identity of the peer in a *connection* and *data origin authentication* to the process of verifying the origin of individual data items.

User authentication, whether relating to a local system or to a remote system, entails three aspects:

- creating a new subject, e.g. a new process or a new session with a *fresh* session key,
- linking an internal entity, e.g. a user ID, to the subject,
- linking an external entity, e.g. a person, to an internal identity.

To differentiate between these aspects, the term *key establishment* was introduced in communication security towards the end of the 1980s for the first aspect. *Entity authentication* stood for what was left. Quoting ISO/IEC 9798, *"entity authentication mechanisms allow the verification, of an entity's claimed identity, by another entity. The authenticity of the entity can be ascertained only for the instance of the authentication exchange"*. This property is related to *dead peer detection* and to the *heartbeat extension* in RFC 6250 [1435]. Note that this definition does not distinguish between internal and external entities.

14.5.4.1 Patterns for Entity Authentication

Entity authentication according to the definition in ISO/IEC 9798 can be implemented with challenge response-mechanisms. When *prover* and *verifier* share a secret, the verifier sends an unpredictable challenge to the prover who constructs its response as a function of the challenge and the shared secret. For example, HTTP digest authentication uses the hash of the challenge, a password, and further data that binds authentication to a particular HTTP request.

When public key cryptography is used, the verifier needs the prover's public key. With a digital signature scheme, the verifier could send the challenge in the clear and the prover could respond with the signed challenge. With a public key encryption scheme, the verifier could encrypt the challenge under the prover's public key; a response constructed from the decrypted

challenge would authenticate the prover. The latter mechanism is used with Trusted Platform Modules (TPMs) where successful decryption of data encrypted under the public endorsement key of a TPM authenticates the TPM. In both cases, the verifier needs an authentic copy of the prover's public verification key. When users are identified by arbitrary public keys, no Public Key Infrastructure is required and the public key could be set directly in a registration phase.

14.5.4.2 Correspondence Properties

The Public-Key Needham-Schroeder protocol uses public key encryption with its challenge-response mechanism [1419]. In this protocol, a malicious prover could decrypt a challenge and reuse it in a protocol run with a third party pretending to be the original verifier; the third party would then respond to the verifier although the verifier is not engaged in a protocol run with the third party [1217]. This scenario would amount to an attack if the mismatch in the assumptions about a protocol run is security relevant. The attack would be detected if the identities of prover and verifier are included in all messages. Note that in this 'attack' the verifier still correctly concludes that the prover is alive.

Matches in the assumptions about aspects of a protocol run held by the peers on completion of a run can be captured by *correspondence properties*, as proposed in [1436] and further elaborated in [1231]:

- *Aliveness*: whenever the verifier (initiator) concludes a protocol run, the prover had also been engaged in a protocol run.
- *Weak agreement*: whenever the verifier (initiator) concludes a protocol run apparently with a given prover, the prover had also been engaged in a protocol run, apparently with that verifier.
- *Non-injective agreement*: whenever the verifier (initiator) concludes a protocol run apparently with a given prover, the prover had also been engaged in a protocol run, apparently with that verifier, and responder and receiver agree on a specified set of data items pertaining to a protocol run.
- *Agreement*: whenever the verifier (initiator) concludes a protocol run apparently with a given prover, the prover had also been engaged in a protocol run, apparently with that verifier, and responder and receiver agree on a specified set of data items pertaining to a protocol run, and each protocol run of the verifier corresponds to a unique protocol run of the prover.

In the vulnerable Redirect/POST Binding in Google Applications there is no agreement on the service provider an authentication assertion is intended for ([1359], Section 14.5.3.3). Flawed implementations of OAuth that use a fixed value for the state variable do not even guarantee aliveness ([1433], Section 14.5.3.4).

Correspondence properties are *intensional* properties well suited for protocol analysis using model checking. This line of research had reversed the earlier decision to separate pure entity authentication from agreeing on session keys and again added agreement on certain data items to authentication. TAMARIN [1437] and ProVerif [1438] are examples for tools that support the automated analysis of authentication protocols.

14.5.4.3 Authentication as Verified Association

Returning to a holistic view on authentication, one could use this as a general term for mechanisms that create a new subject and associate it with evidence relevant for access decisions. If this route is taken, verifying the identity of a user becomes just a special case of authentication.

There would, furthermore, be merit in distinguishing between association with internal and external entities. The latter case is an instance of the 'difficult and error prone' problem of faithfully representing aspects of the physical world within an IT system. The veracity of such representations cannot be guaranteed by cryptographic means alone.

For example, access control in distributed systems may make use of public key cryptography (Section 14.4.4.2). Public keys can then be interpreted as subjects for the purpose of access control. The checks performed by a certificate authority before it issues a certificate would then amount to authentication of an external entity.

14.5.4.4 Authentication for Credit or for Responsibility

Authentication may serve the purpose of giving *credit* to an entity for actions it has performed, or of establishing which entity is *responsible* for an action [1439]. In the first case, an attack amounts to earning undeserved credits and authentication is broken if the attacker succeeds in making a victim perform actions under the attacker's identity. In the second case, an attack amounts to deflecting responsibility to someone else and authentication is broken if the attacker succeeds in performing actions under the victim's identity.

14.6 ACCOUNTABILITY

[1440, ch. 24], [1441, ch. 18]

Accountability has been defined as *"the security goal that generates the requirement for actions of an entity to be traced uniquely to that entity. This supports non-repudiation, deterrence, fault isolation, intrusion detection and prevention, and after-action recovery and legal action"* [1383].

This definition invites investigations into psychology to determine what makes an effective deterrent, investigations into legal matters to determine the standard of evidence demanded in a court of law, and technical investigations into the collection, protection, and analysis of evidence. This Knowledge Area will focus on those technical aspects.

We will cover the technical prerequisites for accountability. We will briefly explore potential conflicts between privacy and accountability, describe current activities in distributed logging of events, and refer to some related terms that overlap with accountability.

14.6.1 Technical Aspects

Accountability supports processes that are launched after events have occurred. Such a process may be a regular audit that checks whether an organisation complies with existing regulations. It might represent a technical audit that scans logs in search for signs of a cyber attack. It may also be an investigation triggered by an incident that tries to identify the vulnerabilities exploited, or an investigation that tries to identify the parties responsible. In all cases, the quality of the evidence is decisive.

The aforementioned processes make use of logs of events. Such logs may be kept by the operating system, by networking devices, or by applications (Section 14.6.2 will give an example). The nature of the events depends on the activity that is being monitored.

14.6.1.1 Audit Policies

Accountability is only as strong as the quality of evidence collected during operations. System administrators may set audit policies that define which events will be logged. Examples for such events are successful and failed authentication attempts, and decisions on sensitive access requests. Operating systems and audit tools provide menus to guide administrators through this task. Access control policies that specify as obligations that certain requests must be logged also influence which evidence is collected.

14.6.1.2 Preserving the Evidence

Accountability is only as strong as the protection of the evidence collected during operations. Attackers could try to hide their traces by deleting incriminating log entries once they have acquired sufficient privileges. They could then modify audit policies so that future actions are not recorded, but should not be able to tamper with the evidence already collected.

Tamper resistance could rely on physical measures like printing the log on an endless paper reel or writing the log to WORM (Write-Once, Read-Many) memory like an optical disk. Tamper resistance could be supported by cryptography. Storing the log as a hash chain [1442, 1443] makes it evident when entries have been removed, but does not guarantee that entries cannot be lost.

Audit policies have to address situations where logging is disrupted, e.g., because the log file has run out of space. Is it then acceptable to overwrite old entries or should the system be stopped until proper auditing is again enabled? This conflict between availability and accountability has to be resolved.

14.6.1.3 Analysing the Evidence

Audit logs can create large volumes of data and many entries are not security relevant so that automated processing is required. Known attack patterns can be detected by their signatures. Machine learning techniques can help to detect anomalies. Lessons learned when applying this approach to network intrusion detection are discussed in [1444]. Visualisation techniques try to draw the administrators' attention to the most relevant events.

14.6.1.4 Assessing the Evidence

Accountability is only as strong as the method of user authentication when legal or disciplinary actions are to be supported. This relates to technical aspects of the authentication mechanism and also to user resilience to phishing and social engineering attacks. Telling users not to fall for obvious phishing attacks is easy, but a well-designed spear phishing attack will not be obvious.

Accountability is only as strong as the organisational security policies on connecting devices, e.g. USB tokens, to internal systems, and policies on access to external web sites. Accountability is only as strong as the defences against software vulnerabilities that can be exploited to run code under a user identity without the user being aware of that fact, e.g. so-called drive-by-downloads.

14.6.2 Privacy and Accountability

Privacy rules can have an impact on the events that may be logged. Employment law may, for example, limit how closely a company monitors its employees, which might make it difficult to achieve accountability when rules have been broken.

Sometimes, there are technical resolutions to such conflicts between legal goals. Take the example of a company that is not permitted to log which external websites employees connect to: when an external site is attacked from within the company network, it is desirable that the perpetrator can be held accountable. To achieve both goals, the company gateway would log for outgoing requests only the internal IP address and the port number used with the global IP address. There is thus no record of visited websites. If an attack is reported, the website affected can provide the port number the attack came from, establishing the link between the internal IP address and the visited site.

Conversely, logging may have unintended privacy impacts. Take *Certificate Transparency* [RFC 6962] as an example. Certificate Transparency is a logging service for the issuers of TLS certificates. Participating Certificate Authorities record the issuance of certificates with this service. Domain owners can scan the log for certificates for their domain that they had not asked for, i.e., detect authentication failures at issuers. This service was introduced in reaction to attacks where such misissued certificates had been used to impersonate the domain affected, and makes issuers accountable to domain owners.

Private subdomains are subdomains created for internal use only. When a certificate for a private subdomain is requested, the certificate will be recorded in the Certificate Transparency log disclosing the existence of the private subdomain to the public [1445].

14.6.3 Distributed Logs

Logs may be kept to hold the users of a system accountable. Logs may be kept to hold the owner of a system accountable. In the latter case, auditors may require that the logging device is sealed, i.e., rely on a physical root of trust. Alternatively, logs could be kept in a distributed system run by independent nodes where there are sufficient barriers to forming alliances that can take over the system.

The nodes maintaining the log need to synchronise their versions of the log. The overheads for synchronisation, or *consensus*, depend on the failure model for the nodes and for the communication network, and on the rules for joining the distributed system. Systems may be open for anyone, or be governed by a membership service. The recent interest in *blockchains* extends to this type of logging solutions.

14.6.4 Related Concepts

The definition at the start of Section 14.6 refers to non-repudiation and intrusion detection. Non-repudiation has a specific meaning in communication security, viz. providing *unforgeable* evidence that a specific action occurred. This goal is not necessarily achieved by logging mechanisms; they may protect the entries recorded, but may record entries that have already been manipulated.

Intrusion detection (see the Security Operations & Incident Management Knowledge Area (Chapter 8)) is an area of its own with overlapping goals. Intrusion detection does not have the requirement *for actions of an entity to be traced uniquely to that entity*. The focus will be more on detecting attacks than detecting the attacker.

The definition given subsumes both the accountability of legal persons and technical investigations into security breaches. The standards of evidence may be higher in the first case. Tracing actions uniquely to an entity leads to *cyber attribution*, the process of tracking and identifying the perpetrators of a cyber attack. Circumstantial evidence such as similarity in malware may be used in this process, and mis-attribution due to *false flag operations* is an issue. Calling for DRM to protect the intellectual property of content owners, because digital content can be copied so easily, but assuming that malware cannot be copied would be incongruous.

APPLYING THE KNOWLEDGE

IT security mechanisms should not be deployed for their own sake but for a reason. The reason has to come from an application in need of protection. An organisational policy would capture the protection requirements and then be implemented by an automated policy (Section 14.3.1.1). Sometimes, this process can start from a clearly defined organisational policy. The policies governing access to classified paper documents are an example. There, the translation into automated policies did not have to bridge a wide conceptual gap, although there were unanticipated twists, e.g., the no write-up policy of the BLP model. These specific circumstances may have raised the unwarranted expectation that this approach would work in general. The fact that these policies were applied in highly hierarchical organisations and were fairly stable are further points worth noting.

Sinclair et al. paint a picture of a very different world [1446]. Their observations can be summarized under the headings of *translation* (from organisational to automated policies) and *automation*. Any translation has to start from a source document, in our case an organisational policy. The translator will face problems when the source is ambiguous or inconsistent. This situation is more likely to arise in organisations with a matrixed structure, where several entities are setting policies, than in strictly hierarchical organisations. Moreover, the wider the language gap between the source document and the destination document, the more difficult translation becomes, and the more difficult it is to ascertain that the translation meets the spirit of the source. The latter step is a prerequisite for *policy certification*, i.e., management approval of a given automated policy.

Organisational policies may intentionally leave decisions at the discretion of caseworkers, e.g., for handling situations where none of the existing rules is directly applicable or where competing rules apply. It is a purpose of automation to remove discretion. Removing discretion adds rules that do not have a counterpart in the organisational policy. Creating an automated policy is then more than translation, it becomes an exercise in creative writing in the spirit of the organisational policy. To do this job well, the writer needs a good understanding of the applications and their workflows, on top of proficiency in the target language (the domain of IT experts). Automated policies based on naïve assumptions easily become denial-of-service attacks on the user. As a related point, there is a tension between the competing goals of keeping a policy simple – which may be feasible in an organisational policy that leaves room for discretion – and of requiring the (automated) policy to cater for a variety of different contexts. This explains why in many cases the number of rules created to cater for exceptions to the general rules ends up being overwhelming. Points that span organisational and automated policies are the handling of dynamic policy changes and the analysis of the side-effects of policy rules in highly complex systems.

The literature on security operations has to say more about the points raised in this section than the research literature on IT security, which has a habit of abstracting problems to a point where much of the awkward issues encountered in real life have disappeared [1446], and then confusing its simplified models with reality.

Similar disconnects between application experts and infrastructure experts exist within the IT domain. Dynamically configurable applications are running foul of well-intended policies such as SOP (Section 14.4.2) and CSP (Section 14.4.2.1). Organisations may then opt for open policies that provide no protection but allow the dynamic applications to run, or applications writers may explore workarounds accepted by the automated policy but still defeating its spirit.

CONCLUSIONS

Access control has kept adapting to the changing applications of IT systems. Access control was originally conceived for the protection of sensitive data in multi-user and multi-level secure systems. Access control without user identities was literally unthinkable. Applications have since changed and some require new modes of access control. One could then reserve 'access control' for the original setting and invent new terms for each new flavour of access control. DRM may serve as an example. This KA has not taken this route but applied 'access control', 'authentication', and 'authorisation' more generally while staying true to the generic meanings of these terms. User identities have lost their prominence along this way. Code (apps) and web domains have taken their place.

Authentication originally stood for the service that links external entities like human users to internal actions in the IT system; today, it may also denote the service that verifies evidence associated with access requests that are submitted for evaluation by a decision algorithm. Design and analysis of cryptographic authentication protocols for distributed systems is a mature knowledge area. Cryptographic solutions for other aspects of access control are often more of academic than of practical interest.

Accountability services build on tamper resistant records of events. The evidence collected may serve as input for technical investigations that try to establish how an attack was conducted and to identify its effects. The evidence collected may also be used in disciplinary processes that deal with situations where rules were broken at the level of the persons implicated. Privacy rules may put limits on the events that are recorded, and the nature of the events recorded may reduce privacy in ways not anticipated.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Section	Cites
14.3 Authorisation	[1377, 1378, 1379, 1380, 1381]
14.3.1 Access Control	[1377, 1379, 1380],
14.3.2 Enforcing Access Control	[1378]
14.3.3 Theory	[1381]
14.4 Access Control in Distributed Systems	[1407, 1408, 1409, 1410]
14.4.1 Core Concepts	
14.4.2 Origin-based Policies	[1409]
14.4.3 Federated Access Control	[1407, 1408]
14.4.4 Cryptography and Access Control	[1410]
14.5 Authentication	[1217, 1419, 1420, 1421, 1422, 1423]
14.5.1 Identity Management	[1423]
14.5.3 Authentication in Distributed Systems	[1419, 1420, 1421, 1422]
14.5.4 Facets of Authentication	[1217]
14.6 Accountability	[1440, ch. 24], [1441, ch. 18]
14.6.1 Technical Aspects	[1440, ch. 24], [1441, ch. 18]
14.6.2 Privacy and Accountability	
14.6.3 Distributed Logs	
14.6.4 Related Concepts	

IV Software & Platform Security

Chapter 15
Software Security

Frank Piessens | KU Leuven

INTRODUCTION

The purpose of this Software Security chapter is to provide a structured overview of known categories of software implementation vulnerabilities, and of techniques that can be used to prevent or detect such vulnerabilities, or to mitigate their exploitation. This overview is intended to be useful to academic staff for course and curricula design in the area of software security, as well as to industry professionals for the verification of skills and the design of job descriptions in this area.

Let us start by defining some terms and concepts, and by defining the scope of this chapter. A first key issue is what it means for software to be *secure*? One possible definition is that a software system is secure if it satisfies a specified or implied security objective. This security objective specifies *confidentiality*, *integrity* and *availability* requirements[1] for the system's data and functionality. Consider, for instance, a social networking service. The security objective of such a system could include the following requirements:

- Pictures posted by a user can only be seen by that user's friends (confidentiality)
- A user can like any given post at most once (integrity)
- The service is operational more than 99.9% of the time on average (availability)

Different security requirements can be at odds with each other, for instance, locking down a system on the appearance of an attack is good for confidentiality and integrity of the system, but bad for availability.

A *security failure* is a scenario where the software system does not achieve its security objective, and a *vulnerability* is the underlying cause of such a failure. The determination of an underlying cause is usually not absolute: there are no objective criteria to determine *what* vulnerability is responsible for a given security failure or *where* it is located in the code. One might say that the vulnerability is in the part of the code that has to be fixed to avoid this specific security failure, but fixes can be required in multiple places, and often multiple mitigation strategies are possible where each mitigation strategy requires a different fix or set of fixes.

The definitions of "security" and "vulnerability" above assume the existence of a security objective. In practice however, most software systems do not have precise explicit security objectives, and even if they do, these objectives are not absolute and have to be traded off against other objectives such as performance or usability of the software system. Hence, software security is often about avoiding known classes of bugs that enable specific attack techniques. There are well-understood classes of software implementation bugs that, when triggered by an attacker, can lead to a substantial disruption in the behaviour of the software, and are thus likely to break whatever security objective the software might have. These bugs are called *implementation vulnerabilities* even if they are relatively independent from application- or domain-specific security objectives like the example objectives above.

This document, the Software Security KA, covers such implementation vulnerabilities, as well as countermeasures for them. Many other aspects are relevant for the security of software based systems, including human factors, physical security, secure deployment and procedural aspects, but they are not covered in this chapter. The impact of security on the various

[1] Other common information security requirements like non-repudiation or data authentication can be seen as instances or refinements of integrity from a software perspective. But from other perspectives, for instance from a legal perspective, the semantics of these requirements can be more involved.

phases of the software lifecycle is discussed in the Secure Software Lifecycle Knowledge Area (Chapter 17). Security issues specific to software running on the web or mobile platforms are discussed in the Web & Mobile Security Knowledge Area (Chapter 16).

The remainder of this chapter is structured as follows. Topic 15.1 (Categories) discusses widely relevant categories of implementation vulnerabilities, but without the ambition of describing a complete taxonomy. Instead, the topic discusses how categories of vulnerabilities can often be defined as violations of a partial specification of the software system, and it is unlikely that a useful complete taxonomy of such partial specifications would exist. The discussion of countermeasures for implementation vulnerabilities is structured in terms of where in the lifecycle of the software system they are applicable. Topic 15.2 (Prevention) discusses how programming language and Application Programing Interface (API) design can prevent vulnerabilities from being introduced during development in software programmed in that language and using that API. In addition, defensive coding practices can contribute to the prevention of vulnerabilities. Topic 15.3 (Detection) covers techniques to detect vulnerabilities in existing source code, for instance, during development and testing. Topic 15.4 (Mitigation) discusses how the impact of remaining vulnerabilities can be mitigated at runtime. It is important to note, however, that some countermeasure techniques could in principle be applied in all three phases, so this is not an orthogonal classification. For instance, a specific dynamic check (say, an array bounds check) could be mandated by the language specification (Prevention, the countermeasure is built in by the language designer), could be used as a testing oracle (Detection, the countermeasure is used by the software tester) or could be inlined in the program to block attacks at run-time (Mitigation, the countermeasure is applied on deployment).

CONTENT

15.1 CATEGORIES OF VULNERABILITIES

[1447][1448, c4,c5,c6,c7,c10,c11][1449, c6,c9] [1013, c17][1450, c5,c9,c11,c13,c17]

As discussed in the Introduction, we use the term *implementation vulnerability* (sometimes also called a *security bug*) both for bugs that make it possible for an attacker to violate a security objective, as well as for classes of bugs that enable specific attack techniques.

Implementation vulnerabilities play an important role in cybersecurity and come in many forms. The Common Vulnerabilities and Exposures (CVE) is a publicly available list of entries in a standardised form describing vulnerabilities in widely-used software components, and it lists close to a hundred thousand such vulnerabilities at the time of writing. Implementation vulnerabilities are often caused by insecure programming practices and influenced by the programming language or APIs used by the developer. This first topic covers important categories of implementation vulnerabilities that can be attributed to such insecure programming practices.

Existing classifications of vulnerabilities, such as the Common Weakness Enumeration (CWE), a community-developed list of vulnerability categories, are useful as a baseline for vulnerability identification, mitigation and prevention, but none of the existing classifications have succeeded in coming up with a complete taxonomy. Hence, the categories discussed in this first topic should be seen as examples of important classes of vulnerabilities, and not as an ex-

haustive list. They were selected with the intention to cover the most common implementation vulnerabilities, but this selection is at least to some extent subjective.

Specific categories of implementation vulnerabilities can often be described as violations of a (formal or informal) specification of some sub-component of the software system. Such a specification takes the form of a contract that makes explicit what the sub-component expects of, and provides to its clients. On violation of such a contract, the software system enters an error-state, and the further behaviour of the software system is typically behaviour that has not been considered by the system developers and is dependent on system implementation details. Attackers of the system can study the implementation details and exploit them to make the system behave in a way that is desirable for the attacker.

15.1.1 Memory Management Vulnerabilities

Imperative programming languages support mutable state, i.e., these languages have constructs for allocating memory cells that can subsequently be assigned to, or read from by the program, and then deallocated again. The programming language definition specifies how to use these constructs correctly: for instance, allocation of n memory cells will return a reference to an array of cells that can then be accessed with indices 0 to $n-1$ until the reference is deallocated (freed) again. This specification can be seen as a contract for the memory management sub-component. Some programming languages implement this contract defensively, and will throw an exception if a client program accesses memory incorrectly. Other programming languages (most notably, C and C++) leave the responsibility for correctly allocating, accessing and deallocating memory in the hands of the programmer, and say that the behaviour of programs that access or manage memory incorrectly is *undefined*. Such languages are sometimes called *memory unsafe* languages, and bugs related to memory management (*memory management vulnerabilities*) are a notorious source of security bugs in these languages.

- A *spatial* vulnerability is a bug where the program is indexing into a valid contiguous range of memory cells, but the index is out-of-bounds. The archetypical example is a *buffer overflow vulnerability* where the program accesses an array (a buffer) with an out-of-bounds index.
- A *temporal* vulnerability is a bug where the program accesses memory that was once allocated to the program, but has since been deallocated. A typical example is dereferencing a dangling pointer.

The C and C++ language specifications leave the behaviour of a program with a memory management vulnerability *undefined*. As such, the observed behaviour of a program with a vulnerability will depend on the actual implementation of the language. Memory management vulnerabilities are particularly dangerous from a security point of view, because in many implementations mutable memory cells allocated to the program are part of the same memory address space where also compiled program code, and runtime metadata such as the call stack are stored. In such implementations, a memory access by the program that violates the memory management contract can result in an access to compiled program code or runtime metadata, and hence can cause corruption of program code, program control flow and program data. There exists a wide range of powerful attack techniques to exploit memory management vulnerabilities [1447].

An attack consists of providing input to the program to trigger the vulnerability, which makes

the program violate the memory management contract. The attacker chooses the input such that the program accesses a memory cell of interest to the attacker:

- In a *code corruption attack*, the invalid memory access modifies compiled program code to attacker specified code.
- In a *control-flow hijack attack*, the invalid memory access modifies a code pointer (for instance, a return address on the stack, or a function pointer) to make the processor execute attacker-provided code (a *direct code injection attack*), or to make the processor reuse existing code of the program in unexpected ways (a *code-reuse attack*, also known as an *indirect code injection attack*, such as a *return-to-libc attack*, or a *return-oriented-programming attack*).
- In a *data-only attack*, the invalid memory access modifies other data variables of the program, possibly resulting in increased privileges for the attacker.
- In an *information leak attack*, the invalid memory access is a read access, possibly resulting in the exfiltration of information, either application secrets such as cryptographic keys, or runtime metadata such as addresses which assist prediction of the exact layout of memory and hence may enable other attacks.

Because of the practical importance of these classes of attacks, mitigation techniques have been developed that counter specific attack techniques, and we discuss these in Topic 15.4.

15.1.2 Structured Output Generation Vulnerabilities

Programs often have to dynamically construct structured output that will then be consumed by another program. Examples include: the construction of SQL queries to be consumed by a database, or the construction of HTML pages to be consumed by a web browser. One can think of the code that generates the structured output as a sub-component. The intended structure of the output, and how input to the sub-component should be used within the output, can be thought of as a contract to which that sub-component should adhere. For instance, when provided with a name and password as input, the intended output is a SQL query that selects the user with the given name and password from the users database table.

A common insecure programming practice is to construct such structured output by means of string manipulation. The output is constructed as a concatenation of strings where some of these strings are derived (directly or indirectly) from input to the program. This practice is dangerous, because it leaves the intended structure of the output string implicit, and maliciously chosen values for input strings can cause the program to generate unintended output. For instance, a programmer can construct a SQL query as:

```
query = "select * from users where name='" + name
      + "'" and pw = '" + password + "'"
```

with the intention of constructing a SQL query that checks for name and password in the where clause. However, if the `name` string is provided by an attacker, the attacker can set name to `"John' --"`, and this would remove the password check from the query (note that `--` starts a comment in SQL).

A *structured output generation vulnerability* is a bug where the program constructs such unintended output. This is particularly dangerous in the case where the structured output represents *code* that is intended to include provided input as *data*. Maliciously chosen input

data can then influence the generated output code in unintended ways. These vulnerabilities are also known as *injection vulnerabilities* (e.g., SQL injection, or script injection). The name 'injection' refers to the fact that exploitation of these vulnerabilities will often provide data inputs that cause the structured output to contain additional code statements, i.e. exploitation *injects* unintended new statements in the output. Structured output generation vulnerabilities are relevant for many different kinds of structured outputs:

- A *SQL injection vulnerability* is a structured output generation vulnerability where the structured output consists of SQL code. These vulnerabilities are particularly relevant for server-side web application software, where it is common for the application to interact with a back-end database by constructing queries partially based on input provided through web forms.
- A *command injection vulnerability* is a structured output generation vulnerability where the structured output is a shell command sent by the application to the operating system shell.
- A *script injection vulnerability*, sometimes also called a *Cross-Site Scripting (XSS) vulnerability* is a structured output generation vulnerability where the structured output is JavaScript code sent to a web browser for client-side execution.

This list is by no means exhaustive. Other examples include: XPath injection, HTML injections, CSS injection, PostScript injection and many more.

Several factors can contribute to the difficulty of avoiding structured output generation vulnerabilities:

- The structured output can be in a language that supports sublanguages with a significantly different syntactic structure. An important example of such a problematic case is HTML, that supports sublanguages such as JavaScript, CSS and SVG.
- The computation of the structured output can happen in different phases with outputs of one phase being stored and later retrieved as input for a later phase. Structured output generation vulnerabilities that go through multiple phases are sometimes referred to as *stored injection vulnerabilities*, or more generally as *higher-order injection vulnerabilities*. Examples include stored XSS and higher-order SQL injection.

Attack techniques for exploiting structured output generation vulnerabilities generally depend on the nature of the structured output language, but a wide range of attack techniques for exploiting SQL injection or script injection are known and documented.

The Web & Mobile Security Knowledge Area (Chapter 16) provides a more detailed discussion of such attack techniques.

15.1.3 Race Condition Vulnerabilities

When a program accesses resources (such as memory, files or databases) that it shares with other concurrent actors (other threads in the same process, or other processes), the program often makes assumptions about what these concurrent actors will do (or not do) to these shared resources.

Such assumptions can again be thought of as part of a specification of the program. This specification is no longer a contract between two sub-components of the program (a caller and a callee), but it is a contract between the actor executing the program and its environment (all concurrent actors), where the contract specifies the assumptions made on how the environment will interact with the program's resources. For instance, the specification can say that the program relies on exclusive access to a set of resources for a specific interval of its execution: only the actor executing the program will have access to the set of resources for the specified interval.

Violations of such a specification are *concurrency bugs*, also commonly referred to as *race conditions*, because a consequence of these bugs is that the behaviour of the program may depend on which concurrent actor accesses a resource first ('wins a race'). Concurrency, and the corresponding issues of getting programs correct in the presence of concurrency, is an important sub-area of computer science with importance well beyond the area of cybersecurity [1451].

But concurrency bugs can be security bugs, too. Concurrency bugs often introduce non-determinism: the behaviour of a program will depend on the exact timing or interleaving of the actions of all concurrent actors. In adversarial settings, where an attacker controls some of the concurrent actors, the attacker may have sufficient control on the timing of actions to influence the behaviour of the program such that a security objective is violated. A *race condition vulnerability* is a concurrency bug with such security consequences. A very common instance is the case where the program checks a condition on a resource, and then relies on that condition when using the resource. If an attacker can interleave his/her own actions to invalidate the condition between the check and the time of use, this is called a Time Of Check Time Of Use (TOCTOU) vulnerability.

Race condition vulnerabilities are relevant for many different types of software. Two important areas where they occur are:

- Race conditions on the file system: privileged programs (i.e., programs that run with more privileges than their callers, for instance, operating system services) often need to check some condition on a file, before performing an action on that file on behalf of a less privileged user. Failing to perform check and action atomically (such that no concurrent actor can intervene) is a race condition vulnerability: an attacker can invalidate the condition between the check and the action.
- Races on the session state in web applications: web servers are often multi-threaded for performance purposes, and consecutive HTTP requests may be handled by different threads. Hence, two HTTP requests belonging to the same HTTP session may access the session state concurrently. Failing to account for this is a race condition vulnerability that may lead to corruption of the session state.

15.1.4 API Vulnerabilities

An application programming interface, or API, is the interface through which one software component communicates with another component, such as a software library, operating system, web service, and so forth. Almost all software is programmed against one or more pre-existing APIs. An API comes with an (explicit or implicit) specification/contract of how it should be used and what services it offers, and just like the contracts we considered in previous subsections, violations of these contracts can often have significant consequences for security. If the client of the API violates the contract, the software system again enters an error-state, and the further behaviour of the software system will depend on implementation details of the API, and this may allow an attacker to break the security objective of the overall software system. This is essentially a generalisation of the idea of *implementation vulnerabilities as contract violations* from subsections 15.1.1, 15.1.2 and 15.1.3 to arbitrary API contracts.

Of course, some APIs are more security sensitive than others. A broad class of APIs that are security sensitive are APIs to libraries that implement security functionality like cryptography or access control logic. Generally speaking, a software system must use all the 'security components' that it relies on in a functionally correct way, or it is likely to violate a security objective. This is particularly challenging for cryptographic libraries: if a cryptographic library offers a flexible API, then *correct* use of that API (in the sense that a given security objective is achieved) is known to be hard. There is substantial empirical evidence [1452] that developers frequently make mistakes in the use of cryptographic APIs, thus introducing vulnerabilities.

An orthogonal concern to secure *use* is the secure *implementation* of the cryptographic API. Secure implementations of cryptography are covered in the Cryptography Knowledge Area (Chapter 10).

15.1.5 Side-channel Vulnerabilities

The execution of a program is ultimately a physical process, typically involving digital electronic circuitry that consumes power, emits electro-magnetic radiation, and takes time to execute to completion. It is common, however, in computer science to model the execution of programs abstractly, in terms of the execution of code on an abstract machine whose semantics is defined mathematically (with varying levels of rigour). In fact, it is common to model execution of programs at many different levels of abstraction, including, for instance, execution of assembly code on a specified Instruction Set Architecture (ISA), execution of Java bytecode on the Java Virtual Machine, or execution of Java source code according to the Java language specification. Each subsequent layer of abstraction is implemented in terms of a lower layer, but abstracts from some of the effects or behaviours of that lower layer. For instance, an ISA makes abstraction from some physical effects such as electro-magnetic radiation or power consumption, and the Java Virtual Machine abstracts from the details of memory management.

A *side-channel* is an information channel that communicates information about the execution of a software program by means of such effects from which the program's code abstracts. Some side-channels require physical access to the hardware executing the software program. Other side-channels, sometimes called *software-based side-channels* can be used from software running on the same hardware as the software program under attack.

Closely related to side-channels are *covert* channels. A covert channel is an information

channel where the attacker also controls the program that is leaking information through the side-channel, i.e., the attacker uses a side-channel to purposefully exfiltrate information.

Side-channels play an important role in the field of cryptography, where the abstraction gap between (1) the mathematical (or source code level) description of a cryptographic algorithm and (2) the physical implementation of that algorithm, has been shown to be relevant for security [1453]. It was demonstrated that, unless an implementation carefully guards against this, side-channels based on power consumption or execution time can easily leak the cryptographic key used during the execution of an encryption algorithm. This breaks the security objectives of encryption for an attacker model where the attacker can physically monitor the encryption process. Side-channel attacks against cryptographic implementations (and corresponding countermeasures) are discussed in the Cryptography Knowledge Area (Chapter 10).

But side-channels are broadly relevant to software security in general. Side-channels can be studied for any scenario where software is implemented in terms of a lower-layer abstraction, even if that lower-layer abstraction is itself not yet a physical implementation. An important example is the implementation of a processor's Instruction Set Architecture (ISA) in terms of a micro-architecture. The execution of assembly code written in the ISA will have effects on the micro-architectural state; for instance, an effect could be that some values are copied from main memory to a cache. The ISA makes abstraction of these effects, but under attacker models where the attacker can observe or influence these micro-architectural effects, they constitute a side-channel.

Side-channels, and in particular software-based side-channels, are most commonly a confidentiality threat: they leak information about the software's execution to an attacker monitoring effects at the lower abstraction layer. But side-channels can also constitute an integrity threat in case the attacker can modify the software's execution state by relying on lower layer effects. Such attacks are more commonly referred to as *fault injection* attacks. Physical fault-injection attacks can use voltage or clock glitching, extreme temperatures, or electromagnetic radiation to induce faults. Software-based fault-injection uses software to drive hardware components of the system outside their specification range with the objective of inducing faults in these components. A famous example is the Rowhammer attack that uses maliciously crafted memory access patterns to trigger an unintended interaction between high-density DRAM memory cells that causes memory bits to flip.

15.1.6 Discussion

15.1.6.1 Better connection with overall security objectives needs more complex specifications

We have categorised implementation vulnerabilities as violations of specific partial specifications of software components. However, the connection to the security objective of the overall software system is weak. It is perfectly possible that a software system has an implementation vulnerability, but that it is not exploitable to break a security objective of the system, for instance, because there are redundant countermeasures elsewhere in the system. Even more so, if a software system does not have any of the implementation vulnerabilities we discussed, it may still fail its security objective.

To have stronger assurance that the software system satisfies a security objective, one can formalise the security objective as a specification. During the design phase, on decomposition

of the system in sub-components, one should specify the behaviour of the sub-components such that they jointly imply the specification of the overall system. With such a design, the connection between an implementation vulnerability as a violation of a specification on the one hand, and the overall security objective of the system on the other, is much stronger.

It is important to note, however, that specifications would become more complex and more domain-specific in such a scenario. We discuss one illustration of additional complexity. For the vulnerability categories we discussed (memory management, structured output generation, race conditions and API vulnerabilities), the corresponding specifications express properties of single executions of the software: a given execution either satisfies or violates the specification, and the software has a vulnerability as soon as there exists an execution that violates the specification.

There are, however, software security objectives that cannot be expressed as properties of individual execution traces. A widely studied example of such a security objective is *information flow security*. A baseline specification of this security objective for deterministic sequential programs goes as follows: label the inputs and outputs of a program as either public or confidential, and then require that no *two* executions of the software with the same public inputs (but different confidential inputs) have different public outputs. The intuition for looking at pairs of executions is the following: it might be that the program does not leak confidential data directly but instead leaks some partial information about this data. If collected along multiple runs, the attacker can gather so much information that eventually relevant parts of the confidential original data are, in fact, leaked. The above specification effectively requires that confidential inputs can never influence public outputs in any way, and hence cannot leak even partial information. In a dual way, one can express integrity objectives by requiring that low-integrity inputs can not influence high-integrity outputs.

But an information flow specification is more complex than the specifications we considered in previous sections because one needs *two* executions to show a violation of the specification. *Information leak vulnerabilities* are violations of a (confidentiality-oriented) information flow policy. They can also be understood as violations of a specification, but this is now a specification that talks about multiple executions of the software system. This has profound consequences for the development of countermeasures to address these vulnerabilities [1381].

15.1.6.2 Side channel vulnerabilities are different

Side channel vulnerabilities are by definition not violations of a specification at the abstraction level of the software source code: they intrinsically use effects from which the source code abstracts. However, if one develops a model of the execution infrastructure of the software that is detailed enough to model side channel attacks, then side channel vulnerabilities can again be understood as violations of a partial specification. One can choose to locate the vulnerability in the execution infrastructure by providing a specification for the execution infrastructure that says that it should not introduce additional communication mechanisms. This is essentially what the theory of full abstraction [1454] requires. Alternatively, one can refine the model of the source code language to expose the effects used in particular side channel attacks, thus making it possible to express side-channel vulnerabilities at the source code level. Dealing with general software side-channel vulnerabilities is not yet well understood, and no generally applicable realistic countermeasures are known. One can, of course, isolate the execution, i.e., prevent concurrent executions on the same hardware, but that then contradicts other goals such as optimised hardware utilisation.

15.1.6.3 Vulnerabilities as faults

The classification of vulnerabilities by means of the specification they violate is useful for understanding relevant classes of vulnerabilities, but is not intended as a complete taxonomy: there are a very large number of partial specifications of software systems that contribute to achieving some security objective. Vulnerabilities can, however, be seen as an instance of the concept of *faults*, studied in the field of dependable computing, and a good taxonomy of faults has been developed in that field [1074].

15.2 PREVENTION OF VULNERABILITIES

[1455, 1456, 1457] [1458, c3]

Once a category of vulnerabilities is well understood, an important question is how the introduction of such vulnerabilities in software can be prevented or at least be made less likely. The most effective approaches eradicate categories of vulnerabilities by design of the programming language or API.

The general idea is the following. We have seen in Topic 15.1 that many categories of implementation vulnerabilities can be described as violations of a specification of some sub-component. Let us call an execution of the software system that violates this specification, an *erroneous execution*, or an execution with an error. From a security point of view, it is useful to distinguish between errors that cause the immediate termination of the execution (*trapped errors*), and errors that may go unnoticed for a while (*untrapped errors*) [1456]. Untrapped errors are particularly dangerous, because the further behaviour of the software system after an untrapped error can be arbitrary, and an attacker might be able to steer the software system to behaviour that violates a security objective. Hence, designing a language or API to avoid errors, and in particular untrapped errors, is a powerful approach to prevent the presence of vulnerabilities. For instance, languages like Java effectively make it impossible to introduce memory management vulnerabilities: a combination of static and dynamic checks ensures that no untrapped memory management errors can occur. This effectively protects against the attack techniques discussed in 15.1.1. It is, however, important to note that this does not prevent the presence of memory-management *bugs*: a program can still access an array out of bounds. But the bug is no longer a *vulnerability*, as execution is terminated immediately when such an access occurs. One could argue that the bug is still a vulnerability if one of the security objectives of the software system is *availability*, including the absence of unexpected program termination.

In cases where choice or redesign of the programming language or API itself is not an option, specific categories of vulnerabilities can be made less likely by imposing safe coding practices.

This topic provides an overview of these techniques that can prevent the introduction of vulnerabilities.

15.2.1 Language Design and Type Systems

A programming language can prevent categories of implementation vulnerabilities that can be described as violations of a specification by:

1. making it possible to express the specification within the language, and
2. ensuring that there can be no untrapped execution errors with respect to the expressed specification.

15.2.1.1 Memory management vulnerabilities

A programming language specification inherently includes a specification of all the memory allocation, access and deallocation features provided by that language. Hence, the specification of the memory management sub-component is always available. A programming language is called *memory-safe* if the language definition implies that there can be no untrapped memory management errors. Languages like C or C++ are not memory-safe because the language definition allows for implementations of the language that can have untrapped memory management errors, but even for such languages one can build specific *implementations* that are memory-safe (usually at the cost of performance).

A language can be made memory-safe through a combination of:

1. the careful selection of the features it supports: for instance, languages can choose to avoid mutable state, or can choose to avoid dynamic memory allocation, or can choose to avoid manual deallocation by relying on garbage collection,
2. imposing dynamic checks: for instance, imposing that every array access must be bounds-checked, and
3. imposing static checks, typically in the form of a static type system: for instance, object-field access can be guaranteed safe by means of a type system.

Programming languages vary widely in how they combine features, dynamic and static checks. Pure functional languages like Haskell avoid mutable memory and rely heavily on static checks and garbage collection. Dynamic languages like Python rely heavily on dynamic checks and garbage collection. Statically typed object-oriented languages like Java and C# sit between these two extremes. Innovative languages like SPARK (a subset of Ada) [1459] and Rust achieve memory safety without relying on garbage collection. Rust, for instance, uses a type system that allows the compiler to reason about pointers statically, thus enabling it to insert code to free memory at places where it is known to no longer be accessible. This comes at the expense of some decreased flexibility when it comes to structuring program code.

15.2.1.2 Structured output generation vulnerabilities

An important cause for structured output generation vulnerabilities is that the programmer leaves the intended structure of the output implicit, and computes the structured output by string manipulation. A programming language can help prevent such vulnerabilities by providing language features that allow the programmer to make the intended structure explicit, thus providing a specification. The language implementation can then ensure that no untrapped errors with respect to that specification are possible.

A first approach is to provide a type system that supports the description of structured data. This approach has been worked out rigorously for XML data: the programming language supports XML documents as first class values, and *regular expression types* [1460] support the description of the structure of XML documents using the standard regular expression operators. A type-correct program that outputs an XML document of a given type is guaranteed to generate XML output of the structure described by the type.

A second approach is to provide primitive language features for some of the common use cases of structured output generation. Language Integrated Query (LINQ) is an extension of the C# language with syntax for writing query expressions. By writing the query as an expression (as opposed to building a SQL query by concatenating strings), the intended structure of the query is explicit, and the LINQ provider that compiles the query to SQL can provide strong guarantees that the generated query has the intended structure.

15.2.1.3 Race condition vulnerabilities

Race condition vulnerabilities on heap allocated memory are often enabled by *aliasing*, the existence of multiple pointers to the same memory cell. If two concurrent threads both hold an alias to the same cell, there is the potential of a race condition on that cell. The existence of aliasing also leads to temporal memory-management vulnerabilities, when memory is deallocated through one alias but then accessed through another alias. The notion of *ownership* helps mitigate the complications that arise because of aliasing. The essence of the idea is that, while multiple aliases to a resource can exist, only one of these aliases is the *owner* of the resource, and some operations can only be performed through the owner. An *ownership regime* puts constraints on how aliases can be created, and what operations are allowed through these aliases. By doing so, an ownership regime can prevent race condition vulnerabilities, or it can support automatic memory management without a garbage collector. For instance, a simple ownership regime for heap allocated memory cells might impose the constraints that: (1) aliases can only be created if they are guaranteed to go out of scope before the owner does, (2) aliases can only be used for reading, and (3) the owner can write to a cell only if no aliases currently exist. This simple regime avoids data races: there can never be a concurrent read and write on the same cell. It also supports automatic memory management without garbage collection: a heap cell can be deallocated as soon as the owner goes out of scope. Of course, this simple regime is still quite restrictive, and a significant body of research exists on designing less restrictive ownership regimes that can still provide useful guarantees.

An ownership regime can be enforced by the programming language by means of a type system, and several research languages have done this with the objective of preventing data races or memory management vulnerabilities. The Rust programming language, a recent systems programming language, is the first mainstream language to incorporate an ownership type system.

15.2.1.4 Other vulnerabilities

Many other categories of vulnerabilities can, in principle, be addressed by means of programming language design and static type checking. There is, for instance, a wide body of research on language-based approaches to enforce information flow security [1297]. These approaches have until now mainly been integrated in research prototype languages. SPARK is an example of a real-world language that has implemented information flow analysis in the compiler. Language-based information flow security techniques have also had a profound influence on the static detection techniques for vulnerabilities (Topic 15.3).

15.2.2 API Design

The development of software not only relies on a programming language, it also relies on APIs, implemented by libraries or frameworks. Just like language design impacts the likelihood of introducing vulnerabilities, so does API design. The base principle is the same: the API should be designed to avoid execution errors (where now, execution errors are violations of the API specification), and in particular *untrapped* execution errors. It should be difficult for the programmer to violate an API contract, and if the contract is violated, that should be trapped, leading, for instance, to program termination or to well-defined error-handling behaviour.

Where the programming language itself does not prevent a certain category of vulnerabilities (e.g. C does not prevent memory-management vulnerabilities, Java does not prevent race conditions or structured output generation vulnerabilities), the likelihood of introducing these vulnerabilities can be reduced by offering a higher-level API:

- Several libraries providing less error-prone APIs for memory management in C or C++ have been proposed. These libraries offer fat pointers (where pointers maintain bounds information and check whether accesses are in bound), garbage collection (where manual deallocation is no longer required), or smart pointers (that support an ownership-regime to safely automate deallocation).
- Several libraries providing less error-prone APIs to do structured output generation for various types of structured output and for various programming languages have been proposed. Examples include Prepared Statement APIs that allow a programmer to separate the structure of a SQL statement from the user input that needs to be plugged into that structure, or library implementations of language integrated query, where query expressions are constructed using API calls instead of using language syntax.
- Several libraries providing less error-prone APIs to cryptography have been proposed. These libraries use simplification (at the cost of flexibility), secure defaults, better documentation and the implementation of more complete use-cases (for instance, include support for auxiliary tasks such as key storage) to make it less likely that a developer will make mistakes.

The use of assertions, contracts and defensive programming [1458, c3] is a general approach to construct software with high reliability, and it is a highly useful approach to avoid API vulnerabilities. Design by contract makes the contract of an API explicit by providing pre-conditions and post-conditions, and in defensive programming these preconditions will be checked, thus avoiding the occurrence of untrapped errors.

A programming language API also determines the interface between programs in the language and the surrounding system. For instance, JavaScript in a browser does not expose an API to

the local file system. As a consequence, JavaScript programs running in the browser can not possibly access the file system. Such less privileged APIs can be used to contain or *sandbox* untrusted code (see Section 15.4.3), but they can also prevent vulnerabilities. Object capability systems [1461] take this idea further by providing a language and API that supports structuring code such that each part of the code only has the privileges it really needs (thus supporting the *principle of least privilege*).

The design of cryptographic APIs that keep cryptographic key material in a separate protection domain, for instance in a Hardware Security Module (HSM) comes with its own challenges. Such APIs have a security objective themselves: the API to a HSM has the objective of keeping the encryption keys it uses confidential – it should not be possible to extract the key from the HSM. Research has shown [1013, c18] that maintaining such a security objective is extremely challenging. The HSM API has an *API-level vulnerability* if there is a sequence of API calls that extracts confidential keys from the HSM. Note that this is an API *design* defect as opposed to the implementation defects considered in Topic 1.

15.2.3 Coding Practices

The likelihood of introducing the various categories of vulnerabilities discussed in Topic 15.1 can be substantially reduced by adopting secure coding practices. Coding guidelines can also help against vulnerabilities of a more generic nature that can not be addressed by language or API design, such as, for instance, the guideline to not hard-code passwords. Secure coding practices can be formalised as collections of rules and recommendations that describe and illustrate good and bad code patterns.

A first approach to design such coding guidelines is heuristic and pragmatic: the programming community is solicited to provide candidate secure coding rules and recommendations based on experience in how things have gone wrong in the past. These proposed rules are vetted and discussed by the community until a consensus is reached that the rule is sufficiently appropriate to be included in a coding standard. Influential standards for general purpose software development include the SEI CERT coding standards for C [1457] and Java [1462].

For critical systems development, more rigorous and stricter coding standards have been developed. The MISRA guidelines [1463] have seen widespread recognition and adoption for development of critical systems in C. The SPARK subset of Ada [1459] was designed to support coding to enable formal verification of the absence of classes of vulnerabilities.

Rules can take many forms, including:

- the avoidance of dangerous language provided API functions (e.g., do not use the system() function in C),
- attempting to avoid undefined behaviour or untrapped execution errors (e.g., do not access freed memory in C),
- mitigations against certain vulnerabilities caused by the language runtime (e.g., not storing secrets in Java Strings, as the Java runtime can keep those Strings stored on the heap indefinitely), or,
- proactive, defensive rules that make it less likely to run into undefined behaviour (e.g., exclude user input from format strings).

Also, specific side-channel vulnerabilities can be addressed by coding rules, for instance

avoiding control flow or memory accesses that depend on secrets can prevent these secrets from leaking through cache-based or branch-predictor based side-channels.

When they are not enforced by a type system, ownership regimes for safely managing resources such as dynamically allocated memory can also be the basis for programming idioms and coding guidelines. For instance, the Resource Acquisition Is Initialisation (RAII) idiom, move semantics and smart pointers essentially support an ownership regime for C++, but without compiler enforced guarantees.

An important challenge with secure coding guidelines is that their number tends to grow over time, and hence programmers are likely to deviate from the secure practices codified in the guidelines. Hence, it is important to provide tool support to check compliance of software with the coding rules. Topic 15.3.1 discusses how static analysis tools can automatically detect violations against secure coding rules.

15.3 DETECTION OF VULNERABILITIES

[1449, 1464] [1458, c4]

For existing source code where full prevention of the introduction of a class of vulnerabilities was not possible, for instance, because the choice of programming language and/or APIs was determined by other factors, it is useful to apply techniques to *detect* the presence of vulnerabilities in the code during the development, testing and/or maintenance phase of the software.

Techniques to detect vulnerabilities must make trade-offs between the following two good properties that a detection technique can have:

- A detection technique is *sound* for a given category of vulnerabilities if it can correctly conclude that a given program has no vulnerabilities of that category. An unsound detection technique on the other hand may have *false negatives*, i.e., actual vulnerabilities that the detection technique fails to find.
- A detection technique is *complete* for a given category of vulnerabilities, if any vulnerability it finds is an actual vulnerability. An incomplete detection technique on the other hand may have *false positives*, i.e. it may detect issues that do not turn out to be actual vulnerabilities.

Trade-offs are necessary, because it follows from Rice's theorem that (for non-trivial categories of vulnerabilities) no detection technique can be both sound and complete.

Achieving soundness requires reasoning about *all* executions of a program (usually an infinite number). This is typically done by static checking of the program code while making suitable abstractions of the executions to make the analysis terminate.

Achieving completeness can be done by performing actual, concrete executions of a program that are witnesses to any vulnerability reported. This is typically done by dynamic detection where the analysis technique has to come up with concrete inputs for the program that trigger a vulnerability. A very common dynamic approach is *software testing* where the tester writes test cases with concrete inputs, and specific checks for the corresponding outputs.

In practice, detection tools can use a hybrid combination of static and dynamic analysis techniques to achieve a good trade-off between soundness and completeness.

It is important to note, however, that some detection techniques are heuristic in nature, and hence the notions of soundness and completeness are not precisely defined for them. For instance, heuristic techniques that detect violations of secure coding practices as described in 15.2.3 are checking compliance with informally defined rules and recommendations, and it is not always possible to unambiguously define the false positives or false negatives. Moreover, these approaches might highlight 'vulnerabilities' that are maybe not exploitable at this point in time, but should be fixed nonetheless because they are 'near misses', i.e., might become easily exploitable by future maintenance mistakes.

Static and dynamic program analysis techniques are widely studied in other areas of computer science. This Topic highlights the analysis techniques most relevant to software security.

Another important approach to detection of vulnerabilities is to perform manual code review and auditing. These techniques are covered in the Secure Software Lifecycle Knowledge Area (Chapter 17). When using tool-supported static detection, it makes sense to adjust such subsequent code review and other verification activities. For instance, if static detection is sound for a given category of vulnerabilities, then one might consider not to review or test for that category of vulnerabilities in later phases.

15.3.1 Static Detection

Static detection techniques analyse program code (either source code or binary code) to find vulnerabilities. Opposed to dynamic techniques, the static ones have the advantage that they can operate on incomplete code that is not (yet) executable, and that in a single analysis run they attempt to cover all possible program executions. Roughly speaking, one can distinguish two important classes of techniques, that differ in their main objective.

15.3.1.1 Heuristic static detection

First, there are static analysis techniques that detect violations of rules that are formal encodings of secure programming-practice heuristics. The static analysis technique builds a semantic model of the program, including, for instance, an abstract syntax tree , and abstractions of the data flow and control flow in the program. Based on this model, the technique can flag violations of simple syntactic rules such as, do not use this dangerous API function, or only use this API function with a constant string as first parameter.

An important indicator for the presence of vulnerabilities is the fact that (possibly malicious) program input can influence a value used in a risky operation (for instance, indexing into an array, or concatenating strings to create a SQL query). *Taint analysis* (sometimes also called *flow analysis*) is an analysis technique that determines whether values coming from program inputs (or more generally from designated *taint sources*) can influence values used in such a risky operation (or more generally, values flowing into a *restricted sink*). The same analysis can also be used to detect cases where confidential or sensitive information in the program flows to public output channels.

Many variants of static taint analysis exist. Important variations include (1) how much abstraction is made of the code, for instance, path-sensitive versus path-insensitive, or context-sensitive versus context-insensitive analysis, and (2) whether influences caused by the program control flow instead of program data flow are taken into account (often distinguished by using the terms *taint analysis* versus *information flow analysis*).

To reduce the number of false positives, a taint analysis can take into account *sanitisation* performed by the program. Tainted values that were processed by designated sanitisation functions (that are assumed to validate that the values are harmless for further processing) have their taint removed.

An important challenge is that taint analyses must be configured with the right sets of sources, sinks and sanitisers. In practice, such configurations currently often occur manually although some recent works have added tool assistance in which, for instance, machine learning is used to support security analysts in this task.

15.3.1.2 Sound static verification

Second, there are static analysis techniques that aim to be sound for well-defined categories of vulnerabilities (but usually in practice still make compromises and give up soundness to some extent). For categories of vulnerabilities that can be understood as specification or contract violations, the main challenge is to express this underlying specification formally. Once this is done, the large body of knowledge on static analysis and program verification developed in other areas of computer science can be used to check compliance with the specification. The three main relevant techniques are program verification, abstract interpretation and model checking.

Program verification uses a program logic to express program specifications, and relies on the programmer/verifier to provide an adequate abstraction of the program in the form of inductive loop invariants or function pre- and post-conditions to make it possible to construct a proof that covers all program executions. For imperative languages with dynamic memory allocation, separation logic [1332] is a program logic that can express absence of memory-management and race-condition vulnerabilities (for data races on memory cells), as well as compliance with programmer provided contracts on program APIs. Checking of compliance with a separation logic specification is typically not automatic: it is done by interactive program verification where program annotations are used to provide invariants, pre-conditions and post-conditions. However, if one is interested only in absence of memory management vulnerabilities, these annotations can sometimes be inferred, making the technique automatic. Also avoiding the use of certain language features (e.g., pointers), and adhering to a coding style amenable to verification can help making verification automatic.

Abstract interpretation is an automatic technique where abstraction is made from the concrete program by mapping the run-time values that the program manipulates to adequate finite abstract domains. For imperative programs that do not use dynamic allocation or recursion, abstract interpretation is a successful technique for proving the absence of memory management vulnerabilities automatically and efficiently.

Model checking is an automatic technique that exhaustively explores all reachable states of the program to check whether none of the states violates a given specification. Because of the state explosion problem, model checking can only exhaustively explore very small programs, and in practice techniques to bound the exploration need to be used, for instance, by bounding the number of times a program loop can be executed. Bounded model checking is no longer sound, but can still find many vulnerabilities.

Most practical implementations of these analysis techniques give up on soundness to some extent. In order to be both sound and terminating, a static analysis must *over-approximate* the possible behaviours of the program it analyses. Over-approximation leads to false positives. Real programming languages have features that are hard to over-approximate without leading

to an unacceptable number of false positives. Hence, practical implementations have to make engineering trade-offs, and will under-approximate some language features. This makes the implementation unsound, but more useful in the sense that it reduces the number of false positives. These engineering trade-offs are nicely summarised in the 'Soundiness Manifesto' [1465].

15.3.2 Dynamic Detection

Dynamic detection techniques execute a program and monitor the execution to detect vulnerabilities. Thus, if sufficiently efficient, they can also be used for just-in-time vulnerability mitigation (See Topic 4). There are two important and relatively independent aspects to dynamic detection: (1) how should one monitor an execution such that vulnerabilities are detected, and (2) how many and what program executions (i.e., for what input values) should one monitor?

15.3.2.1 Monitoring

For categories of vulnerabilities that can be understood as violations of a specified property of a single execution (See Topic 15.1.6), complete detection can be performed by monitoring for violations of that specification. For other categories of vulnerabilities, or when monitoring for violations of a specification is too expensive, approximative monitors can be defined.

Monitoring for memory-management vulnerabilities has been studied intensively. It is, in principle, possible to build complete monitors, but typically at a substantial cost in time and memory. Hence, existing tools explore various trade-offs in execution speed, memory use, and completeness. Modern C compilers include options to generate code to monitor for memory management vulnerabilities. In cases where a dynamic analysis is approximative, like a static analysis, it can also generate false positives or false negatives, despite the fact that it operates on a concrete execution trace.

For structured output generation vulnerabilities, a challenge is that the intended structure of the generated output is often implicit, and hence there is no explicit specification that can be monitored. Hence, monitoring relies on sensible heuristics. For instance, a monitor can use a fine-grained dynamic taint analysis [1464] to track the flow of untrusted input strings, and then flag a violation when untrusted input has an impact on the parse tree of generated output.

Assertions, pre-conditions and post-conditions as supported by the design-by-contract approach to software construction [1458, c3] can be compiled into the code to provide a monitor for API vulnerabilities at testing time, even if the cost of these compiled-in run-time checks can be too high to use them in production code.

Monitoring for race conditions is hard, but some approaches for monitoring data races on shared memory cells exist, for instance, by monitoring whether all shared memory accesses follow a consistent locking discipline.

15.3.2.2 Generating relevant executions

An important challenge for dynamic detection techniques is to generate executions of the program along paths that will lead to the discovery of new vulnerabilities. This problem is an instance of the general problem in software testing of systematically selecting appropriate inputs for a program under test [1458, c4]. These techniques are often described by the umbrella term *fuzz testing* or *fuzzing*, and can be classified as:

- *Black-box fuzzing*, where the generation of input values only depends on the input/output behaviour of the program being tested, and not on its internal structure. Many different variants of black-box fuzzing have been proposed, including (1) purely random testing, where input values are randomly sampled from the appropriate value domain, (2) model-based fuzzing, where a model of the expected format of input values (typically in the form of a grammar) is taken into account during generation of input values, and (3) mutation-based fuzzing, where the fuzzer is provided with one or more typical input values and it generates new input values by performing small mutations on the provided input values.
- *White-box fuzzing*, where the internal structure of the program is analysed to assist in the generation of appropriate input values. The main systematic white-box fuzzing technique is *dynamic symbolic execution*. Dynamic symbolic execution executes a program with concrete input values and builds at the same time a *path condition*, a logical expression that specifies the constraints on those input values that have to be fulfilled for the program to take this specific execution path. By solving for input values that do not satisfy the path condition of the current execution, the fuzzer can make sure that these input values will drive the program to a different execution path, thus improving coverage.

15.4 MITIGATING EXPLOITATION OF VULNERABILITIES

[1447, 1466]

Even with good techniques to prevent introduction of vulnerabilities in new code, or to detect vulnerabilities in existing code, there is bound to be a substantial amount of legacy code with vulnerabilities in active use for the foreseeable future. Hence, vulnerability prevention and detection techniques can be complemented with techniques that mitigate the exploitation of remaining vulnerabilities. Such mitigation techniques are typically implemented in the execution infrastructure, i.e., the hardware, operating system, loader or virtual machine, or else are inlined into the executable by the compiler (a so-called ‘inlined reference monitor’). An important objective for these techniques is to limit the impact on performance, and to maximise compatibility with legacy programs.

15.4.1 Runtime Detection of Attacks

Runtime monitoring of program execution is a powerful technique to detect attacks. In principle, program monitors to detect vulnerabilities during testing (discussed in 15.3.2 Dynamic Detection) could also be used at runtime to detect attacks. For instance, dynamic taint analysis combined with a dynamic check whether tainted data influenced the parse tree of generated output has also been proposed as a runtime mitigation technique for SQL injection attacks.

But there is an important difference in the performance requirements for monitors used during testing (discussed in Topic 15.3) and monitors used at runtime to mitigate attacks. For runtime detection of attacks, the challenge is to identify efficiently detectable violations of properties that are expected to hold for the execution trace of the program. A wide variety of techniques are used:

- Stack canaries detect violations of the integrity of activation records on the call stack, and hence detect some attacks that exploit memory management vulnerabilities to modify a return address.
- No Execute (NX) data memory detects attempts to direct the program counter to data memory instead of code memory and hence detects many direct code injection attacks.
- Control-Flow Integrity (CFI) is a class of techniques that monitors whether the runtime control flow of the program complies with some specification of the expected control flow, and hence detects many code-reuse attacks.

On detection of an attack, the runtime monitor must react appropriately, usually by terminating the program under attack. Termination is a good reaction to ensure that an attack can do no further damage, but it has of course a negative impact on availability properties.

15.4.2 Automated Software Diversity

Exploitation of vulnerabilities often relies on implementation details of the software under attack. For instance, exploitation of a memory management vulnerability usually relies on details of the memory layout of the program at runtime. A SQL injection attack can rely on details of the database to which the SQL query is being sent.

Hence, a generic countermeasure to make it harder to exploit vulnerabilities is to *diversify* these implementation details. This raises the bar for attacks in two ways. First, it is harder for an attacker to prepare and test his/her attack on an identical system. An attack that works against a web server installed on the attacker machine might fail against the same web server on the victim machine because of diversification. Second, it is harder to build attacks that will work against many systems at once. Instead of building an exploit once, and then using it against many systems, attackers now have to build customised exploits for each system they want to attack.

The most important realisation of this idea is Address Space Layout Randomization (ASLR), where the layout of code, stack and/or heap memory is randomised either at load or at runtime. Such randomisation can be *coarse-grained*, for instance, by just randomly relocating the base address of code, stack and heap segments, or *fine-grained* where addresses of individual functions in code memory, activation records in the stack, or objects in the heap are chosen randomly.

The research community has investigated many other ways of automatically creating diversity

at compilation time or installation time [1466], but such automatic diversification can also bring important challenges to software maintenance as bug reports can be harder to interpret, and software updates may also have to be diversified.

15.4.3 Limiting Privileges

The exploitation of a software vulnerability influences the behaviour of the software under attack such that some security objective is violated. By imposing general bounds on what the software can do, the damage potential of attacks can be substantially reduced.

Sandboxing is a security mechanism where software is executed within a controlled environment (the 'sandbox') and where a policy can be enforced on the resources that software in the sandbox can access. Sandboxing can be used to confine untrusted software, but it can also be used to mitigate the impact of exploitation on vulnerable software: after a successful exploit on the software, an attacker is still confined by the sandbox.

The generic idea of sandboxing can be instantiated using any of the isolation mechanisms that modern computer systems provide: the sandbox can be a virtual machine running under the supervision of a virtual-machine monitor, or it can be a process on which the operating system imposes an access control policy. In addition, several purpose-specific sandboxing mechanisms have been developed for specific classes of software, such as, for instance, *jails* that can sandbox network- and filesystem-access in virtual hosting environments. The Java Runtime Environment implements a sandboxing mechanism intended to contain untrusted Java code, or to isolate code from different stakeholders within the same Java Virtual Machine, but several significant vulnerabilities have been found in that sandboxing mechanism over the years [1467].

Compartimentalisation is a related but finer-grained security mechanism, where the software itself is divided in a number of *compartments* and where some bounds are enforced on the privileges of each of these compartments. This again requires some underlying mechanism to enforce these bounds. For instance, a compartimentalised browser could rely on operating system process access control to bound the privileges of its rendering engine by denying it file system access. Exploitation of a software vulnerability in the rendering engine is now mitigated to the extent that even after a successful exploit, the attacker is still blocked from accessing the file system. Very fine-grained forms of compartimentalisation can be achieved by *object-capability systems* [1461], where each application-level object can be a separate protection domain.

To mitigate side-channel vulnerabilities, one can isolate the vulnerable code, for instance, on a separate core or on separate hardware, such that the information leaking through the side channel is no longer observable for attackers.

15.4.4 Software Integrity Checking

Under the umbrella term *Trusted Computing*, a wide range of techniques have been developed to measure the state of a computer system, and to take appropriate actions if that state is deemed insecure. A representative technique is *Trusted Boot* where measurements are accumulated for each program that is executed. Any modification to the programs (for instance, because of a successful attack) will lead to a different measurement. One can then enforce that access to secret keys, for instance, is only possible from a state with a specified measurement.

Parno et al. [1468] give an excellent overview of this class of techniques.

CONCLUSIONS

Software implementation vulnerabilities come in many forms, and can be mitigated by a wide range of countermeasures. Table 15.1 summarises the relationship between the categories of vulnerabilities discussed in this chapter, and the relevant prevention, detection and mitigation techniques commonly used to counter them.

Vulnerability category	Prevention	Detection	Mitigation
Memory management vulnerabilities	memory-safe languages, fat/smart pointers, coding rules	many static and dynamic detection techniques	stack canaries, NX, CFI, ASLR, sandboxing
Structured output generation vulnerabilities	regular expression types, LINQ, Prepared Statements	taint analysis	runtime detection
Race condition vulnerabilities	ownership types, coding guidelines	static and dynamic detection	sandboxing
API vulnerabilities	contracts, usable APIs, defensive API implementations	runtime checking of pre- and post-conditions, static contract verification	compartimentalisation
Side channel vulnerabilities	coding guidelines	static detection	isolation

Table 15.1: Summary overview

Acknowledgments

The insightful and constructive comments and feedback from the reviewers and editor on earlier drafts have been extremely valuable, and have significantly improved the structure and contents of this chapter, as have the comments received during public review.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[1448]	[1450]	[1013]	[1455]	[1457]	[1458]	[1449]
15.1 Categories of Vulnerabilities							
15.1.1 Memory Management Vulnerabilities	c4,c5	c5					c6
15.1.2 Structured Output Generation Vulnerabilities	c10,c11	c17					c9
15.1.3 Race Condition Vulnerabilities	c7	c9					
15.1.4 API Vulnerabilities	c6	c9,c11					
15.1.5 Side-channel Vulnerabilities			c17				
15.2 Prevention of Vulnerabilities							
15.2.1 Language Design and Type Systems				c1			
15.2.2 API Design			c18			c3	
15.2.3 Coding Practices					*		
15.3 Detection of Vulnerabilities							
15.3.1 Static Detection							*
15.3.2 Dynamic Detection						c4	
15.4 Mitigating Exploitation of Vulnerabilities							
15.4.1 Runtime Detection of Attacks	c4						
15.4.2 Automated Software Diversity	c4						
15.4.3 Limiting Privileges	c7						

FURTHER READING

Building Secure Software [1469] and 24 Deadly Sins of Software Security [1470]

Building Secure Software was the first book focusing specifically on software security, and even if some of the technical content is somewhat dated by now, the book is still a solid introduction to the field and the guiding principles in the book have withstood the test of time.

24 Deadly Sins of Software Security is a more recent and updated book by mostly the same authors.

The Art of Software Security Assessment [1450]

Even if this is a book that is primarily targeted at software auditors, it is also a very useful resource for developers. It has clear and detailed descriptions of many classes of vulnerabilities, including platform-specific aspects.

Surreptitious Software [1471]

Software security in this chapter is about preventing, detecting and removing software implementation vulnerabilities. However, another sensible, and different, interpretation of the term is that it is about protecting the software code itself, for instance, against reverse engineering of the code, against extraction of secrets from the code, or against undesired tampering with the code before or during execution. Obfuscation, watermarking and tamperproofing are examples of techniques to protect software against such attacks. *Surreptitious Software* is a rigorous textbook about this notion of software security.

OWASP Resources

The Open Web Application Security Project (OWASP) is a not-for-profit, volunteer-driven organisation that organises events and offers a rich set of resources related to application security and software security. They offer practice-oriented guides on secure development and on security testing, as well as a collection of tools and awareness raising instruments. All these resources are publicly available at **https://www.owasp.org**.

Chapter 16
Web & Mobile Security

Sascha Fahl | Leibniz University Hannover

16.1 INTRODUCTION

The purpose of this Knowledge Area is to provide an overview of security mechanisms, attacks and defences in modern web and mobile ecosystems. This overview is intended for use in academic courses and to guide industry professionals interested in this area.

Web and mobile security have become the primary means through which many users interact with the Internet and computing systems. Hence, their impact on overall information security is significant due to the sheer prevalence of web and mobile applications (apps). Covering both web and mobile security, this Knowledge Area emphasises the intersection of their security mechanisms, vulnerabilities and mitigations. Both areas share a lot in common and have experienced a rapid evolution in the features and functionalities offered by their client side applications (apps). This phenomenon, sometimes called *appification*, is a driver in modern web and mobile ecosystems. Web and mobile client apps typically interact with server side application interfaces using web technologies. This second phenomenon, also sometimes called *webification*, equally affects both web and mobile ecosystems. In the 1990s, web and mobile security had a strong focus on server-side and infrastructure security. Web browsers were mostly used to render and display static websites without dynamic content. The focus on the server-side prevailed even with the rise of early scripting languages such as Perl and PHP. However, web content became more dynamic in the 2000s, and server-side security had to address injection attacks. Similarly to web browsers, early mobile devices had limited functionality and were mostly used to make calls or send SMS. Mobile security back then focused on access control, calls and SMS security.

The rise of modern web and mobile platforms brought notable changes. A significant amount of web application code is no longer executed on the server-side but runs in the browser. Web browser support for Java, Adobe Flash, JavaScript and browser plugins and extensions brought many new features to the client, which prompted a drastic change of the attack surface on the web. New types of attacks such as Cross-Site Scripting emerged and plugins proved to be vulnerable, e.g. Adobe Flash browser plugins are known for being an attractive target for attackers. In response to these new threats, browser vendors and website developers and operators took measures. For instance, Google Chrome disabled the Adobe Flash plugin by default in 2019 [1472] and new security best practices were developed [1473]. Similarly to web browsers, mobile devices became smarter and more feature-rich. Smartphones and tablets are equipped with sensors, including motion, GPS and cameras. They have extensive computing power, storage capacity and are connected to the Internet 24-7. Modern Android and iOS devices run full-blown operating systems and increasingly feature-rich and complex application frameworks. Mobile apps can request access to all the devices' resources and sensors using permission based access control, and process highly sensitive user information. Being powerful, feature-rich, and connected makes mobile clients promising and attractive targets for attackers.

Modern web and mobile ecosystems are the primary drivers for the rise of *appification* and the *"there is an app for everything"* motto sums up many of the technological and security developments in recent years. The appification trend resulted in millions of apps ranging from simple flashlight apps to online social network apps, from online banking apps to mobile and browser-based games. It also sparked the merging of technologies and security mechanisms used in web and mobile applications. Both ecosystems are typically client-server oriented. Web browsers and mobile apps communicate with back-end services often using web focused technologies. Communication is mostly based on the Hypertext Transfer Protocol (HTTP) and

its secure extension HTTPS. Both web-browsers and mobile applications tend to primarily exchange Hypertext Markup Language (HTML), JSON and XML documents and both make extensive use of the JavaScript programming language, on the server- and the client-side. *Webification* describes the conversion to these web technologies.

The sheer amount of applications in modern web and mobile ecosystems also impacted the software distribution model, which moved away from website downloads to centralised application stores, which allow developers to publish, advertise and distribute their software, and users to download new apps and app updates. The centralised software distribution had a positive impact on update frequencies and speed for both web and mobile.

This Knowledge Area focuses on the appification trend and an introduction to the core technologies of the *webification* phenomenon. Figure 16.1 provides an overview of the entities involved and their interactions.

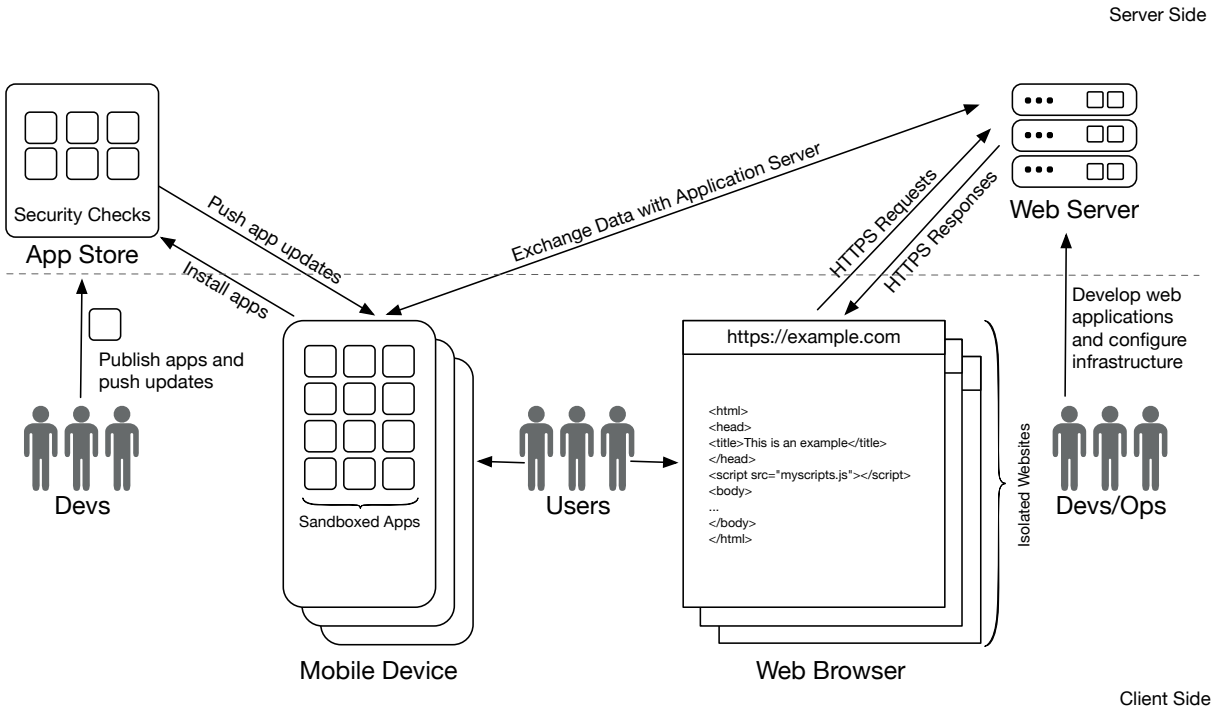


Figure 16.1: Web and Mobile Ecosystem

After introducing core technologies and concepts, we describe important security mechanisms and illustrate how they differ from non-web and non-mobile ecosystems. Software and content **isolation** are crucial security mechanisms and aim to protect apps and websites from malicious access. While isolation is understood in relation to traditional operating systems (cf. the Operating Systems & Virtualisation Knowledge Area (Chapter 11)), specifics for web and mobile platforms will be outlined.

Modern web and mobile platforms introduced new forms of access control based on **permission dialogues**. Whilst a more general discussion of access control is included in the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14), this Knowledge Area discusses web and mobile specifics. Web and mobile applications make extensive use of the HTTP and HTTPS protocols. Hence, we will discuss the Web Public-Key Infrastructure (PKI) and HTTPS extending the Transport Layer Security (TLS) section in the Network Security Knowledge Area (Section 19.3.2). Similarly, we will discuss web and mobile-specific authentication aspects, referring readers to the Authentication, Authorisation & Accountability

Knowledge Area (Chapter 14) for a more general discussion of authentication. Finally, we address **frequent software updates** as a crucial security measure. While software updates are equally important in traditional computer systems, the centralisation[1] of web and mobile ecosystems, introduces new challenges and opportunities.

The following sections focus on web and mobile-specific client and server-side security aspects. However, we will not address common software vulnerabilities (cf. the Software Security Knowledge Area (Chapter 15)) and operating system security (cf. Operating Systems & Virtualisation Knowledge Area (Chapter 11)) in general. Section 16.3 first covers phishing and clickjacking attacks and defenses. Both affect web and mobile clients and exploit human difficulties in correctly parsing URLs or identifying changes in the visual appearance of websites. As feature-rich web and mobile clients store sensitive data, we will then discuss client-side storage security issues and mitigations. Finally, Section 16.3 discusses physical attacks on mobile clients, including smudge attacks and shoulder surfing. Section 16.4 addresses server-side challenges, starting with an overview of frequent injection attacks. We discuss SQL and command injection attacks that allow malicious users to manipulate database queries to storage backends of web applications and commands that are executed. This is followed by a discussion of cross-site scripting and cross-site request forgery attacks and common server-side misconfigurations that might lead to vulnerable service backends.

Overall, the discussion of client- and server-side security challenges aims to serve as the underlining of the natural split between entities in web and mobile ecosystems. Additionally, the chosen aspects illustrate the difference between the web and mobile world from other ecosystems.

Due to its focus on the intersection of both web and mobile security, this Knowledge Area does not cover aspects that are unique to either web or mobile such as mobile device security, mobile network (i. e., 2G/3G/4G/5G) security (see Physical Layer and Telecommunications Security Knowledge Area (Chapter 22)), and mobile malware. Some of these aspects are discussed in the Hardware Security Knowledge Area (Chapter 20), the Malware & Attack Technologies Knowledge Area (Chapter 6) and the Network Security Knowledge Area (Chapter 19). We also do not discuss side-channel attacks; the concept and examples for side-channel security are given in the Hardware Security Knowledge Area (Chapter 20).

16.2 FUNDAMENTAL CONCEPTS AND APPROACHES

[1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482]

This section describes fundamental concepts and approaches of modern web and mobile platforms that affect security. The information presented in this section is intended to serve as a foundation to better understand the security challenges in the following sections. Similar to other software products and computer systems, mobile operating systems and applications and web browsers as well as web servers may contain exploitable bugs. General purpose software vulnerabilities are discussed in the Software Security Knowledge Area (Chapter 15).

[1]There are only a limited number of widely used web browsers and application stores.

16.2.1 Appification

Over the last ten years, the rise of mobile devices and ubiquitous Internet access have changed the way software is produced, distributed and consumed, altering how humans interact with computer devices and with software installed on the devices. While regular Internet browsers have been the dominant way of accessing content on the web in the pre-mobile era, the concept of *appification* significantly changed the way users access content online [1475]. Appification describes the phenomenon of moving away from a web-based platform to access most digital tools and media online with a web-browser through mobile applications with highly specialised, tiny feature sets. As mobile devices grew to become the primary interface for web access worldwide [1483], the number of apps rose enormously over the last decade. *"There is an app for everything"* became the mantra of appified software ecosystems, which produced numerous applications for all sorts of use cases and application areas. Many apps look like native local desktop or mobile applications. However, they are often (mobile) web applications that communicate with back end services, which then outsource computation and storage tasks to the client. The shift towards appification had a significant impact on web and mobile security creating more security challenges on the client-side. The rise of appification also impacted the developer landscape. In the pre-appification era, software development was mostly dominated by experienced developers. Due to the more extensive tool and framework support, the market entrance barrier is lower in appified ecosystems. This attracts more inexperienced developers, and has negative consequences for web and mobile security in general (cf. the Human Factors Knowledge Area (Chapter 4)).

The Rise of the Citizen Developer The appification trend attracts many non-professional software developers called citizen developers. Many of them do not have a software engineering education but make use of multiple simple APIs and tools available to build apps for different platforms. Oltrogge et al. [1484] found that the adoption of easy-to-use Online Application Generators (OAGs) to develop, distribute and maintain apps has a negative impact on application security. Generated apps tend to be vulnerable to reconfiguration and code injection attacks and rely on an insecure infrastructure.

16.2.2 Webification

Modern web and mobile platforms gave rise to another phenomenon. Many of the applications are not native applications written in compiled programming languages such as Java or Kotlin and C/C++ (e. g. for Android apps) or Objective-C and Swift (e. g. for iOS apps). Instead, they are based on web technologies including server-side Python, Ruby, Java or JavaScript scripts and client-side JavaScript. In addition to conventional web applications targeting regular web browsers, mobile web applications are more frequently built using these web technologies. In particular, mobile web applications make heavy use of the JavaScript language.

This section gives a brief introduction to the most essential technologies needed to explain vulnerabilities and mitigations later in the KA. We include Uniform Resource Locators (URLs), the Hypertext Transfer Protocol (HTTP), the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and the JavaScript programming language. For more detailed information, we suggest reading [1485].

16.2.2.1 Uniform Resource Locators

Uniform Resource Locators (URLs) [1476] are a core concept in the web. A URL is a well-formed and fully qualified text string that addresses and identifies a resource on a server. Address bars in modern browser User Interfaces (UIs) use the URLs to illustrate the remote address of a rendered document. A fully qualified absolute URL string consists of several segments and contains all the required information to access a particular resource. The syntax of an absolute URL is: **scheme://credentials@host:port/resourcepath?query_parameters#fragments**. Each segment has a particular meaning (cf. Table 16.1).

Segment	Optional	Description
scheme:	○	Indicates the protocol a web client should use to retrieve a resource. Common protocols in the web are **http:** and **https:**
//	○	Indicates a hierarchical URL as required by [1476]
credentials@	●	Can contain a username and password that might be needed to retrieve a resource from a remote server.
host	○	Specifies a case-insensitive DNS name *(e. g. cybok.org)*, a raw IPv4 *(e. g. 127.0.0.1)* or IPv6 address *(e. g. [0:0:0:0:0:0:0:1])* to indicate the location of the server hosting a resource.
:port	●	Describes a non-default network port number to connect to a remote server. Default ports are 80 for HTTP and 443 for HTTPS.
/resourcepath	○	Identifies the resource address on a remote server. The resource path format is built on top of Unix directory semantics.
?query_parameters	●	Passes non-hierarchical parameters to a remote resource, such as server-side script input parameters.
#fragment	●	Provides instructions for the browser. In practice, it is used to address an HTML anchor element for in-document navigation.

Table 16.1: URL segments.

16.2.2.2 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is the most widely used mechanism to exchange documents between servers and clients on the web. While HTTP is mostly used to transfer HTML documents, it can be used for any data. Although HTTP/2.0 [1486] is the newest protocol revision, the most widely supported protocol version is HTTP/1.1 [1474]. HTTP is a text-based protocol using TCP/IP. An HTTP client initiates a session by sending an HTTP request to an HTTP server. The server returns an HTTP response with the requested file.

The first line of a client request includes HTTP version information (e. g. `HTTP/1.1`). The remaining request header consists of zero or more `name:value` pairs. The pairs are separated by a new line. Common request headers are *User-Agent* – these include browser information, *Host* – the URL hostname, *Accept* – which carries all supported document types, `Content-Length` – the length of the entire request and *Cookie* – see Section 16.2.8. The request header is terminated with a single empty line. HTTP clients may pass any additional content to the server. Although the content can be of any type, clients commonly send HTML content to the server, e. g. to submit form data. The HTTP server responds to the request with a response header followed by the requested content. The response header contains the supported protocol version, a numerical status code, and an optional, human-readable status

message. The status notification is used to indicate request success (e. g. status `200`), error conditions (e. g. status `404` or `500`) or other exceptional events. Response headers might also contain *Cookie* headers – cf. Section 16.2.8. Additional response header lines are optional. The header ends with a single empty line followed by the actual content of the requested resource. Similar to the request content, the content may be of any type but is often an HTML document.

Although **cookies** were not part of the original HTTP RFC [1474], they are one of the most important protocol extensions. Cookies allow remote servers to store multiple *name=value* pairs in client storage. Servers can set cookies by sending a *Set-Cookie: name=value* response header and consume them by reading a client's *Cookie: name=value* request header. Cookies are a popular mechanism to maintain sessions between clients and servers and to authenticate users.

HTTP is request-response based and neatly fits unidirectional data transfer use cases. However, for better latency and more effective use of bandwidth, bidirectional network connections are needed. Bidirectional connections not only allow clients to pull data from the server, but also the server to push data to the client at any time. Therefore, the **WebSocket** protocol [1487] provides a mechanism on top of HTTP. WebSocket connections start with a regular HTTP request that includes an `Upgrade: WebSocket` header. After the WebSocket handshake is completed, both parties can send data at any time without having to run a new handshake.

16.2.2.3 Hypertext Markup Language

The Hypertext Markup Language (HTML) [1477] is the most widely used method to produce and consume documents on the web. The most recent version is HTML5. The HTML syntax is fairly straightforward: a hierarchical tree structure of tags, *name=value* tag parameters and text nodes form an HTML document. The Domain Object Model (DOM) defines the logical structure of an HTML document and rules how it is accessed and manipulated. However, competing web browser vendors introduced all sorts of custom features and modified the HTML language to their wishes. The many different and divergent browser implementations resulted in only a small portion of the websites on the Internet adhering to the HTML standard's syntax. Hence, implementations of HTML parsing modes and error recovery vary greatly between different browsers.

The HTML syntax comes with some constraints on what may be included in a parameter value or inside a text node. Some characters (e. g., angle brackets, single and double quotes and ampersands) make the blocks of the HTML markup. Whenever they are used for a different purpose, such as parts of substrings of a text, they need to be escaped. To avoid undesirable side effects, HTML provides an entity encoding scheme. However, the failure to properly apply the encoding to reserved characters when displaying user-controlled information may lead to severe web security flaws such as cross-site scripting (cf. Section 16.4).

16.2.2.4 Cascading Style Sheets

Cascading Style Sheets (CSS) [1488] are a consistent and flexible mechanism to manipulate the appearance of HTML documents. The primary goal of CSS was to provide a straightforward and simple text-based description language to supersede the many vendor-specific HTML tag parameters that lead to many inconsistencies. However, similar to divergent HTML parsing implementations, different browsers also implement different CSS parsing behavior. CSS allows HTML tags to be scaled, positioned or decorated without being limited by the original HTML markup constraints. Similar to HTML tag values, values inside CSS can be user-controlled or provided externally, which makes CSS crucial for web security.

16.2.2.5 JavaScript

JavaScript [1478] is a simple yet powerful object-oriented programming language for the web. It runs both client-side in web browsers and server-side as part of web applications. The language is meant to be interpreted at runtime and has a C-inspired syntax. JavaScript supports a classless object model, provides automatic garbage collection and weak and dynamic typing. Client-side JavaScript does not support I/O mechanisms out of the box. Instead, some limited predefined interfaces are provided by native code inside the browser. Server-side JavaScript (e. g., Node.js [1489]) supports a wide variety of I/O mechanisms, e. g., network and file access. The following discussion will focus on client JavaScript in web browsers. Every HTML document in a browser is given its JavaScript execution context. All scripts in a document context share the same sandbox (cf. Section 16.2.4). Inter-context communication between scripts is supported through browser-specific APIs. However, execution contexts are strictly isolated from each other in general. All JavaScript blocks in a context are executed individually and in a well-defined order. Script processing consists of three phases:

Parsing validates the script syntax and translates it to an intermediate binary representation for performance reasons. The code has no effect until parsing is completed. Blocks with syntax errors are ignored, and the next block is parsed.

Function Resolution registers all named, global functions the parser found in a block. All registered functions can be reached from the following code.

Execution runs all code statements outside of function blocks. However, exceptions may still lead to execution failures.

While JavaScript is a very powerful and elegant scripting language, it brings up new challenges and security issues such as Cross-Site Scripting vulnerabilities (cf. Section 16.4.1).

16.2.2.6 WebAssembly

WebAssembly (Wasm) [1490] is an efficient and fast binary instruction format and is supported by most modern browser vendors. It is a stack-based virtual machine language and mainly aims to execute at native speed on client machines. Code written in WebAssembly is memory safe and benefits from all security features provided by regular code associated with a website. WebAssembly code is sandboxed, enforces the same origin policy (cf. Section 16.2.4) and is limited to the resources provided by the corresponding website's permissions. Additionally, WebAssembly code can access JavaScript code running in the same origin container and provide its functionality to JavaScript code from the same origin.

16.2.2.7 WebViews

WebViews are a further trend in webification and mobile apps. They allow the easy integration of web content into mobile apps [1491]. Developers can integrate apps with HTML and JavaScript and benefit from portability advantages. WebViews run in the context of regular mobile apps and allow a rich two-way interaction with the hosted web content. Mobile apps can invoke JavaScript from within the web content, and monitor and intercept events in the web content. At the same time, specific JavaScript APIs allow WebView apps to interact with content and sensors outside the WebView context. The interaction of web content with native app content raises new security concerns and enables both *app-to-web* and *web-to-app* attacks [1492, 1493, 1494]. App-to-web attacks, allow malicious apps to inject JavaScript into hosted WebViews with the goal to exfiltrate sensitive information or trick WebViews into navigating to and presenting users with untrusted and potentially malicious websites. Web-to-app attacks inject untrusted web content into an app and leverage an app's JavaScript bridge to the underlying host app. The goal of a web-to-app attack is privilege escalation to the level of its hosting app's process.

Both the appification and webification phenomena led to a new way of software distribution. Instead of decentralised download sources, centralised application stores which are illustrated in the next section emerged.

16.2.3 Application Stores

Application stores are centralised digital distribution platforms that organise the management and distribution of software in many web and mobile ecosystems. Famous examples are the Chrome web store for extensions for the Chrome browser, Apple's AppStore for iOS applications, and Google Play for Android applications. Users can browse, download, rate and review mobile applications or browser plugins and extensions. Developers can upload their software to application stores that manage all of the software distribution challenges, including the provision of storage, bandwidth and parts of the advertisement and sales. Before publication, most application stores deploy application approval processes for testing reliability, adherence to store policies, and for security vetting [1495, 1496].

Most of the software available in ecosystems that have application stores is distributed through the stores. Only a few users side-load software (i. e. install software from other sources than the store). Application stores allow providers to control which applications are available in their stores, which allows them to ban particular applications. Whilst this can give rise to accusations of censorship, the deployment of security vetting techniques has helped to significantly reduce the amount of malicious software available in stores [1495] and to reduce the number of applications that suffer from vulnerabilities due to the misuse of security APIs by developers [404]. Deployed security vetting techniques include static and dynamic analysis applied to application binaries and running instances of applications. In addition to security vetting techniques, application stores require applications to be signed by developer or application store keys. In Android, application signing does not rely on the same public key infrastructures used on the web. Instead, developers are encouraged to use self-signed certificates and required to sign application updates with the same key to prevent malicious updates [1497]. The application signing procedure on iOS devices requires apps to be signed by Apple. Unsigned apps cannot be installed on iOS devices. Application stores not only allow developers and users centralised access to software publication, distribution and download, they also enable users to rate and review published applications. User rating and

reviews are intended to help other users make more informed download decisions, but they also have a direct connection to application security.

Impact of User Ratings and Reviews on Application Security Nguyen et al. [1498] conducted a large-scale analysis of user reviews for Android applications and their impact on security patches. They found that the presence of security- and privacy-related user reviews for applications are contributing factors to future security-related application updates.

16.2.4 Sandboxing

Both modern mobile and browser platforms make use of different sandboxing techniques to isolate applications and websites and their content from each other (cf. Operating Systems & Virtualisation Knowledge Area (Chapter 11)) [1499, 1500]. This also aims to protect the platform against malicious applications and sites. Major web browsers (e.g. Google Chrome [1501]) and mobile platforms (e.g. Android [1502]) implement isolation at an operating system process level. Each application or website runs in its own process [2]. By default, isolated processes cannot interact with each other and cannot share resources. In browsers, site isolation serves as a second line of defence as an extension to the *same-origin-policy* (cf. Section 16.2.4.2).

16.2.4.1 Application Isolation

Modern mobile platforms provide each application with their sandbox running in a dedicated process and their own file-system storage. Mobile platforms take advantage of underlying operating system process protection mechanisms for application resource identification and isolation. For example, application sandboxes in Android [1502] are set-up at kernel-level. Security is enforced through standard operating system facilities, including user and group IDs as well as security contexts. By default, sandboxing prevents applications from accessing each other and only allows limited access to operating system resources. To access protected app and operating system resources inter-app communication through controlled interfaces is required.

16.2.4.2 Content Isolation

Content isolation is one of the major security assurances in modern browsers. The main idea is to isolate documents based on their origin so that they cannot interfere with each other. The *Same-Origin-Policy (SOP)* [1503] was introduced in 1995 and affects JavaScript and its interaction with a document's DOM, network requests and local storage (e. g., cookies). The core idea behind SOP is that two separate JavaScript execution contexts are only allowed to manipulate a document's DOM if there is an exact match between the document host and the protocol, DNS name and port numbers[3]. Cross-origin manipulation requests are not allowed. Table 16.2 illustrates sample SOP validation results. Similar to JavaScript-DOM-interaction, the SOP limits the JavaScript `XMLHttpRequest` capabilities to only issue HTTP requests to the origin of the host document.

One major flaw of SOP is that it relies on DNS instead of IP addresses. Attackers who can intentionally change the IP address of a DNS entry can therefore circumvent SOP security

[2] Process-based site isolation is mostly used on desktop computers [1501].
[3] The protocol, DNS name and port number triple is called *origin*.

Originating document	Accessed document	Browser behaviour
https://www.cybok.org/**docs/**	https://www.cybok.org/**scripts/**	Access okay
https://www.cybok.org/	https://**books.**cybok.org/	Host mismatch
http://www.cybok.org/	**https**://www.cybok.org/	Protocol mismatch
https://www.cybok.org/	https://www.cybok.org**:10443**/	Port mismatch

Table 16.2: SOP validation examples.

guarantees.

Since code that enforces the *same-origin-policy* occasionally contains security bugs, modern browsers introduced a second line of defence: websites are rendered in their own processes that run in a sandbox. Sandboxing websites is meant to prevent attacks such as stealing cross-site cookies and saved passwords [1504].

Another additional layer of defence to enforce the same-origin policy and improve web application security is the Content Security Policy (CSP) mechanism [1505]. A CSP is primarily intended to prevent code injection attacks such as XSS (cf. Section 16.4.1), which exploit the browsers' trust of content that was sent by a web server. This allows malicious scripts to be executed in clients' browsers. CSP allows web developers and server operators to limit the number of origins that browsers should consider to be trusted sources of content – including executable code and media files. A CSP can be used so that servers can globally disable code execution on the client. To enable CSP, developers or operators can either configure a web server to send a `Content-Security-Policy` HTTP response header or add a HTML `<meta>` tag to a website. Compatible browsers will then only execute code or load media files from trusted origins.

Example: Content Security Policy Header The following CSP allows users of a web application to include images from any origin, but to restrict media data (audio or video media) to the trusted **trusted-media.com** domain. Additionally, scripts are restricted to the **trusted-scripts.com** origin that the web developer trusts:

```
Content-Security-Policy: default-src 'self'; img-src *; media-src
trusted-media.com; script-src trusted-scripts.com
```

16.2.5 Permission Dialog Based Access Control

Permission systems in modern mobile and web platforms enable protection of the privacy of their users and reduce the attack surface by controlling access to resources. The control of access to resources on a traditional computer system requires the accurate definition of all involved security principals and the protected resources in the system. Finally, an access control system requires a non-bypassable and trusted mechanism to evaluate access requests (the *reference monitor*) and sound security policies that define the appropriate course of action for all access requests. Based on the security policies, the reference monitor can decide whether it grants access or denies access (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)).

Modern mobile and web platforms deviate from conventional computer systems in multiple ways:

16.2.5.1 The Security Principals

Traditional computer systems are primarily multi-user systems with human users and processes running on their behalf. Modern mobile and web platforms extend conventional multi-user systems to also consider all involved developers that have their applications installed on the system as security principals.

16.2.5.2 The Reference Monitor

Typically, conventional computer systems implement access control as part of the Operating System (OS), e.g., the file system and network stack. User-level processes can then extend this OS functionality and implement their own access control mechanisms.

Like conventional computer systems, modern mobile and web platforms build on top of OS low-level access control mechanisms. Additionally, the extensive frameworks on top of which applications are developed and deployed, provide extended interfaces. Modern web and mobile platforms use Inter-Process Communication (IPC) for privilege separation and compartmentalisation between apps and between apps and the operating system instead of allowing direct access to resources. Access control mechanisms on calling processes are used to protect IPC interfaces.

16.2.5.3 The Security Policy

In conventional computer systems, a process can have different privilege levels. It can run as the superuser, as a system service, with user-level privileges or with guest privileges[4]. All processes that share the same privilege level have the same set of permissions and can access the same resources.

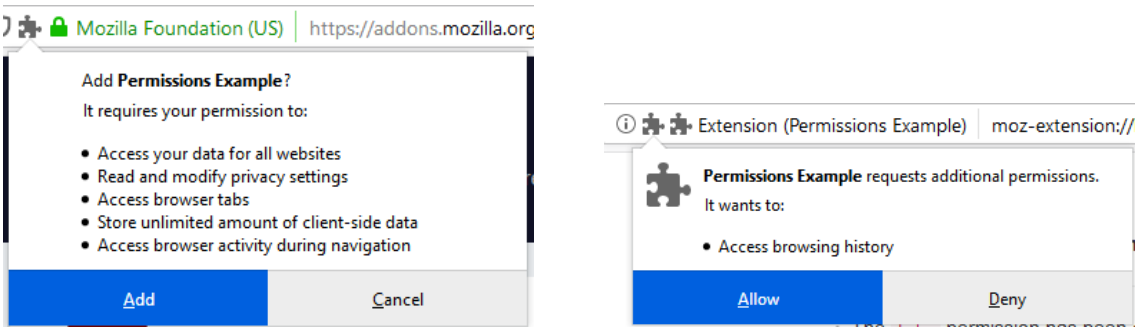
Modern mobile and web platforms make a clear distinction between system and third-party applications: access to security- and privacy-critical resources is only granted to designated processes and third-party applications have, by default, no access to critical resources. If such access is required, application developers must request permissions from a set commonly available to all third-party applications. Most permissions allow developers to use designated system processes as deputies to access protected sensitive resources. Those system processes serve as reference monitors and enforce access control policies.

16.2.5.4 Different Permission Approaches

Mobile and web platforms implement distinct permission approaches. First, platforms distinguish different privilege levels. A common distinction is two levels (e.g., as implemented on Android): normal (e.g., access to the Internet) and dangerous permissions (e.g., access to the camera or microphone). While application developers have to request both normal and dangerous permissions to grant their applications access to the respective resources, the levels differ for application users. Normal permissions are granted silently without any application user interaction. However, whenever applications require dangerous permissions, the underlying mobile or web platform presents users with permission dialogues. While earlier Android versions showed users a list of all the necessary permissions of an application at install time, modern mobile platforms and browsers present permission dialogues at run-time. A permission dialog usually is shown the first time an application requests access to the

[4]Depending on the system, more levels may be implemented.

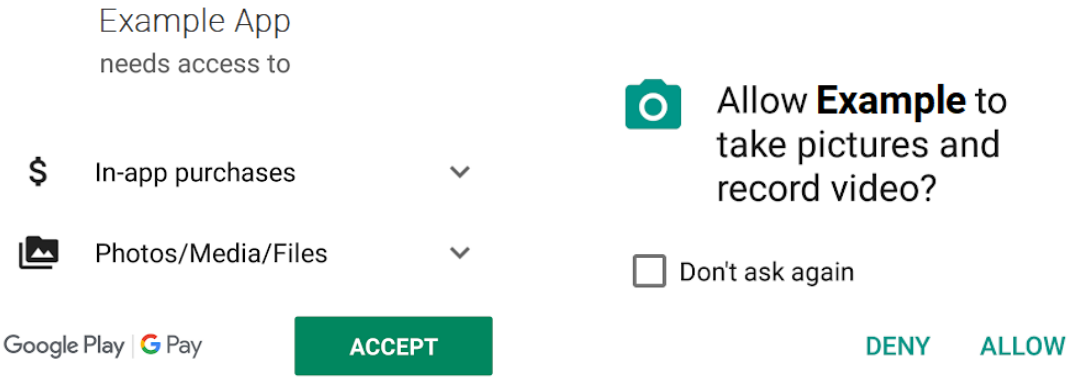
corresponding resource. Application users can then either grant or deny the application access to the resource. Modern permission-based access control systems allow greater flexibility and control for both developers and users.



(a) Install permissions

(b) Runtime permissions

Figure 16.2: Firefox Permission Dialogues



(a) Install permissions that can still be found on legacy apps

(b) Runtime permissions on modern apps

Figure 16.3: Android Permission Dialogues

Permission Dialogues: Attention, Comprehension and Behaviour While permission dialogues theoretically allow for greater flexibility and control, in practice they tend to have serious limitations. Porter Felt et al. found that Android applications developers tend to request more permissions for their applications than needed [1479]. Hence, applications request access to more resources than strictly necessary, which violates the least-privilege principle. Similarly to developers, end-users struggle with permission dialogues. Porter Felt et al. [1393] found that they often do not correctly understand permission dialogues and ignore them due to habituation (cf. the Human Factors Knowledge Area (Chapter 4)).

16.2.6 Web PKI and HTTPS

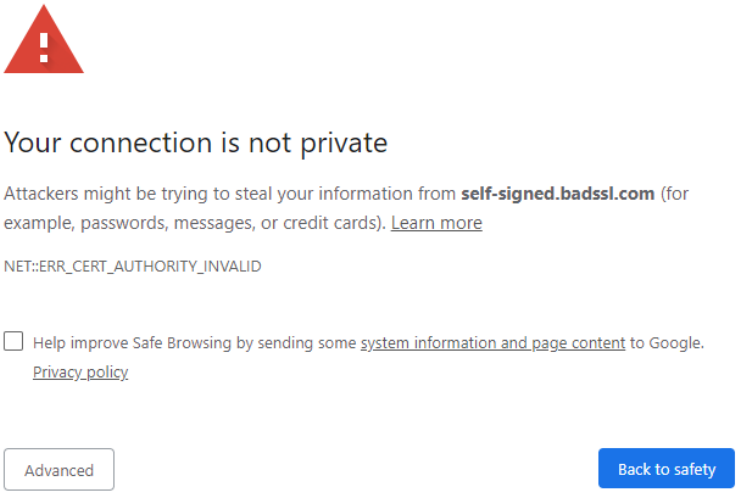
The web PKI and the HTTPS [1480, 1481] protocol play a central role in modern mobile and web platforms, both of which are based on client-server architectures. In the web, web servers or applications exchange information with browsers. On mobile platforms, apps exchange information with backend (web) servers. In both cases, HTTPS should always be used for secure network connections between clients and servers. To establish secure network connections, the web public key infrastructure is used. Using the web PKI and X.509 certificates, clients and servers can authenticate each other and exchange cryptographic key material for further encrypted information transport. This KA will not provide further details on how the authentication process and the key exchange procedures work in detail (cf. the Network Security Knowledge Area (Chapter 19)). Rather, it gives an overview of aspects specific to web and mobile platforms.

HTTPS is the most widely deployed secure network protocol on the web and mobile. It overlays HTTP on top of the TLS protocol to provide authentication of the server, and integrity and confidentiality for data in transit. While HTTPS offers mutual authentication of servers and clients based on X.509 certificates, the primary use is the authentication of the accessed server. Similar to TLS, HTTPS protects HTTP traffic against eavesdropping and tampering by preventing man-in-the-middle attacks. Since HTTPS encapsulates HTTP traffic, it protects URLs, HTTP header information including cookies and HTTP content against attackers. However, it does not encrypt the IP addresses and port numbers of clients and servers. While HTTPS can hide the information exchanged by clients and servers, it allows eavesdroppers to learn the top-level domains of the websites browsers that users visit, and to identify the backend servers that mobile apps communicate with.

Both web browsers and mobile apps authenticate HTTPS servers by verifying X.509 certificates signed by Certificate Authorities CAs. Browsers and mobile apps come with a list of pre-installed certificate authorities or rely on a list of pre-installed CAs in the host operating system. A pre-installed certificate authority list in modern browsers and on modern mobile platforms typically contains hundreds of CAs. To be trusted, an HTTPS server certificate needs to be signed by one pre-installed CA.[5]

Modern browsers present users with a warning message (e. g., see Figure 16.4) when the server certificate could not be validated. The warning messages are intended to indicate a man-in-the-middle attack. However, common reasons for warning messages are invalid certificates, certificates that were issued for a different hostname, network errors between the client and server and errors on the client such as misconfigured clocks [1506]. In most cases, browser users can click-through a warning message and visit a website even if the server certificate could not be validated [1507]. Browsers use coloured indicators in the address bar to display the security information for a website. Websites loaded via HTTP, websites loaded via HTTPS that load some of their content (e.g. CSS or JavaScript files) over an HTTP connection[6] and sites that use an invalid certificate but for which the user clicked through a warning are displayed as insecure. HTTPS websites with a valid certificate are displayed with a corresponding security indicator (e. g., see Figure 16.4). In contrast, users of mobile, non-browser apps cannot easily verify whether an application uses the secure HTTPS protocol with a valid certificate. No visual security indicators similar to those used in browsers are available. Instead, users have to trust application developers to take all the necessary security measures for HTTPS connections.

[5] See the Network Security Knowledge Area (Chapter 19) for details on the validation process.
[6] Called mixed content.



(a) Warning message for invalid certificate in Chrome

(b) Invalid certificate

(c) Valid and trusted certificate

(d) Extended validation certificate

Figure 16.4: Warning messages and security indicators in Chrome.

As of 2019, most of the popular websites support HTTPS, and the majority of connections from clients to servers in the web and mobile applications use HTTPS to protect their users against man-in-the-middle attacks. To further increase the adoption of HTTPS, server operators are encouraged to use HTTPS for all connections and deploy HTTP Strict Transport Security (HSTS) [1508]. Additionally, browser users can install extensions and plugins to rewrite insecure HTTP URLs to secure HTTPS URLs [1509] if possible, and mobile application frameworks make HTTPS the default network protocol for HTTP connections.

Using HTTPS does protect the content against attackers but does not preserve metadata (e. g., which websites a user visits). Please refer to the Privacy & Online Rights Knowledge Area (Chapter 5) for more information, including private browsing and the Tor network.

Rogue Certificate Authorities and Certificate Transparency The web PKI allows every trusted root certificate authority to issue certificates for any domain. While this allows website operators to freely choose a CA for their website, in the past some CAs have issued fraudulent certificates for malicious purposes. One of the most prominent examples is the DigiNotar CA, which in 2011 [1510] issued fraudulent certificates for multiple websites including Google's Gmail service. Nobody has been charged for the attack. However, DigiNotar went bankrupt in 2011. Certificate transparency [1511] was introduced to fight fraudulent certificate issuance. Certificate transparency provides a tamper proof data structure and monitors all certificate issuance processes of participating CAs. While it cannot prevent fraudulent certificate issuance, it improves the chances of detection. Clients can verify the correct operation of the certificate transparency providers and should only connect to websites that use X.509 certificates that include a signed certificate timestamp. Certificate transparency is supported by most major certificate authorities and browser vendors.

16.2.7 Authentication

Authentication in the web and on mobile platforms is an important security mechanism designed to enable human users to assert their identity to web applications, mobile devices or mobile apps. Authentication goes hand in hand with authorisation which describes the specification of access privileges to resources. The specified access privileges are later on used to grant or deny access to resources for authenticated users. This section will not give a detailed overview of authentication and authorisation concepts (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)) but will focus on authentication mechanisms and technologies relevant for web and mobile platforms.

16.2.7.1 HTTP Authentication

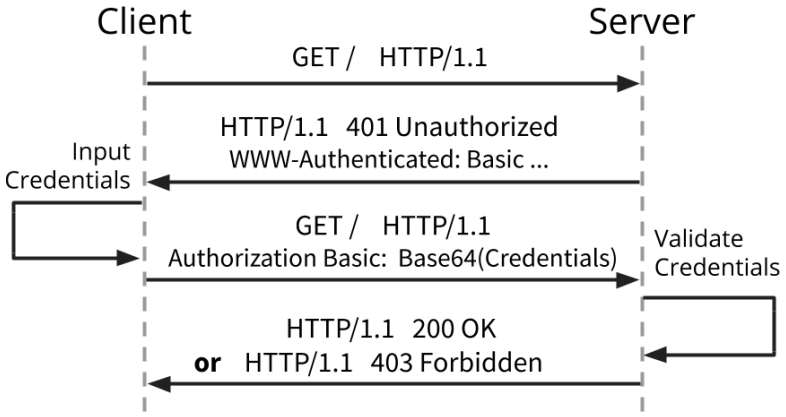


Figure 16.5: Basic HTTP Authentication exchange.

In the HTTP context, authentication generally refers to the concept of verifying the identity of a client to a server, e. g., by requiring the client to provide some pre-established secrets such as username and password with a request. This section highlights two widely used authentication methods on the web, *Basic HTTP authentication*, and the more frequently used *Form-based HTTP authentication*.

Basic HTTP authentication [1512] is a mechanism whose results are used to enforce access control to resources. It does not rely on session identifiers or cookie data. Nor does the Basic HTTP authentication scheme require the setup of dedicated login pages, as all major browsers provide an integrated login form. A server can trigger this authentication option by sending a response header containing the "`HTTP 401 Unauthorised`" status code and a "`WWW-Authenticate: Basic`" field. Credentials entered into this form by the client are combined with a ":" ("`Username:Password`"), Base64 encoded for transit ("`VXNlcm5hbWU6UGFzc3dvcmQK`"), and added as Authorisation header to the next request ("`Authorization: Basic VXNlcm5hbWU6UGFzc3dvcmQK`"). An example exchange between server and client is shown in Figure 16.5. The Basic authentication scheme is not secure, as the credentials are transmitted after a simple Base64 encoding, which is trivial to reverse. Hence, login credentials are transmitted in plain text across the network, which allows attackers or network observers to easily steal the credentials. Therefore, Basic HTTP authentication should not be used without additional enhancements that ensure confidentiality and integrity such as HTTPS.

Form-based HTTP authentication in which websites use a form to collect login credentials is a widely prevalent form of authentication in modern web and mobile applications. For this

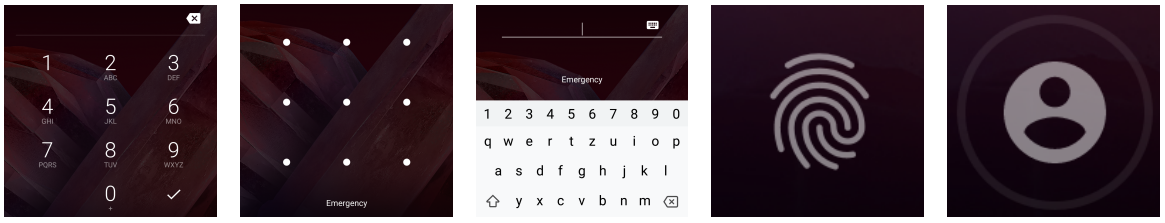
scheme, an unauthenticated client trying to access restricted content is shown an HTML-based web form that prompts for their credentials. The client then submits the entered credentials to the sever (e. g., in a POST request). The server validates the form data and authenticates the client on successful validation. Similar to Basic authentication, Form-based authentication exposes user credentials in plain text if not protected by HTTPS.

16.2.7.2 Mobile Device Authentication

Mobile devices deploy a variety of authentication mechanisms to unlock devices, grant users access, and protect their data from illegitimate access. The most common mechanisms for mobile device authentication are passwords, PINs, patterns and biometric features.

Users can use common alphanumeric passwords, including special characters. However, since mobile device authentication is a frequent task [1513], many users tend to unlock their mobile device using numerical PINs. Android devices also support unlock patterns (see Figure 16.6). Instead of choosing a password or PIN, users can pick an unlock pattern from a 3x3 grid.

Modern mobile devices allow users to authenticate using biometric features, including fingerprint and facial recognition. These authentication features rely on hardware security primitives, such as ARM's TrustZone (cf. the Human Factors Knowledge Area (Chapter 20)).



(a) PIN (b) Pattern (c) Password (d) Fingerprint Indicator (e) Facial Recognition Indicator

Figure 16.6: Android Device Unlocking

Android Unlock Patterns Similar to passwords (see Section 16.2.9) device unlock patterns suffer from multiple weaknesses. Uellenbeck et al. [367] conducted a study to investigate users' choices of 3×3 unlock patterns. They found empirical evidence that users tend to choose biased patterns, e. g., users typically started in the upper left corner and selected three-point long straight lines. Hence, similar to regular passwords (cf. the Human Factors Knowledge Area (Chapter 4)) the entropy of unlock patterns is rather low. In addition to users choosing weak unlock patterns, the mechanism is vulnerable to shoulder surfing attacks (see Section 16.3.3). As a countermeasure, De Luca et al. [1514] propose to use the back of a device to authenticate users.

16.2.8 Cookies

Web servers can associate stateful information with particular clients by using HTTP cookies [1515]. Cookie information (e.g., IDs of items added to the shopping cart in an online shop) is stored by the client. Cookies allow clients and servers to include their unique session identifiers in each HTTP request-response, avoiding the need for repeated authentication. Session cookies expire when the session is closed (e.g., by the client closing the browser) but persistent cookies only expire after a specific time.

Cookie-based authentication allows clients to re-establish sessions every time they send requests to the server with a valid cookie. Cookie-based session management is vulnerable to the hijacking of session identifiers [1516]. Hijackers who post valid session cookies can connect to the attacked server with the privileges of the authenticated victim.

Cookies can also be used to track users across multiple sessions by providers. This behaviour is generally jeopardising user privacy (cf. the Adversarial Behaviours Knowledge Area (Chapter 7) and the Privacy & Online Rights Knowledge Area (Chapter 5)).

16.2.9 Passwords and Alternatives

Passwords are the most widely deployed mechanism to let users authenticate to websites and mobile applications and protect their sensitive information against illegitimate access online. They are the dominant method for user authentication due to their low cost, deployability, convenience and good usability. However, the use of passwords for most online accounts harms account security [1482]. Since humans tend to struggle memorising many different complicated passwords, they often choose weak passwords and re-use the same password for multiple accounts. Weak passwords can easily be guessed by attackers offline or online. Re-used passwords amplify the severity of all password attacks. One compromised online account results in all other accounts protected with the same password as vulnerable. While password guidelines in the past frequently recommended the use of complex passwords, current guidelines state that requiring complex passwords actually weakens password security and advise against policies that include password complexity [1517, 1518]. These aspects are further discussed in the Human Factors Knowledge Area (Chapter 4).

Online service providers deploy various countermeasures to address security issues with weak passwords and password re-use:

16.2.9.1 Password Policies

Password policies are rule sets to encourage users to choose stronger passwords. Some password policies also address the memorability issue. To support stronger passwords, most rules address password length and composition, blacklists and the validity period of a password [1519, 1520].

16.2.9.2 Password Strength Meters

Password Strength Meters (PSMs) pursue the same goal as password policies and aim to encourage the choice of stronger passwords. PSMs typically provide visual feedback or assign passwords scores to express password strength (see Figure 16.7) [1521].

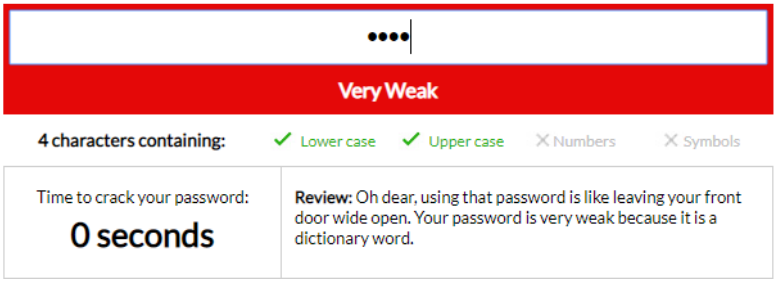


Figure 16.7: A password strength meter

However, addressing weak passwords and password re-use by deploying restrictive policies or PSMs has only a limited effect on overall password security [1522]. Hence, service providers can use extensions to simple passwords to increase authentication security.

16.2.9.3 Password Managers

Password managers can help users generate, store and retrieve strong passwords. Strong passwords are generated and stored using secure random number generators and secure encryption. They come as locally installable applications, online services or local hardware devices. While they can help users use more diverse and stronger passwords, their effect on overall password security is limited due to usability issues [1523]. For a more detailed discussion please refer to the Human Factors Knowledge Area (Chapter 4).

16.2.9.4 Multi-Factor Authentication

Instead of requiring only one factor (e. g., a password), multi-factor authentication systems require users to present multiple factors during the authentication process [1524]. Website passwords are often complemented with a second factor for two-factor authentication (2FA). Most commonly, the second factor typically makes use of a mobile device. So in addition to a password, users need to have their device at hand to receive a one-time token to authenticate successfully. The European Payment Services Directive 2 (PSD2) requires 2FA for all online payment services in web and mobile environments (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)).

16.2.9.5 WebAuthn

The WebAuthn (Web Authentication) [1525] web standard is a core component of the FIDO2 project (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)) and aims to provide a standardised interface for user authentication for web-based applications using public-key cryptography. WebAuthn is supported by most modern web-browsers and mobile operating systems. It can be used in single-factor or multi-factor authentication mode. In multi-factor authentication mode PINs, passcodes, swipe-patterns or biometrics are supported.

16.2.9.6 OAuth

While not an authentication mechanism itself (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)), Open Authorisation (OAuth) [1526] can be used for privacy-friendly authentication and authorisation for users against third-party web applications. OAuth uses secure tokens instead of requiring users to provide login credentials such as usernames and passwords. On behalf of their users, OAuth service providers provide access tokens that authorise specific account information to be shared with third-party applications. More recent successors of the OAuth protocol including OAuth 2 [1422] or OpenID Connect [1527] support federations (cf. the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)). Large providers of online services such as Google or Facebook can act as identity providers to authenticate users, thus helping users to reduce the number of login credentials and accounts. While such protocols aim to provide improved security, the correct and secure implementation of such complex protocols was shown to be error-prone and might allow malicious users to run impersonation attacks [1528].

16.2.10 Frequent Software Updates

Frequent software updates are a fundamental security measure and particularly crucial for web and mobile platforms. This section discusses the different components in the web and mobile ecosystems that require regular updates, the different update strategies, and their pros and cons. Traditionally, browser and mobile device updates required their users to install updates manually whenever new versions were available. Users had to keep an eye on software updates and were responsible for downloading and installing new releases. This approach was error-prone and resulted in many outdated and insecure deployed software components.

Most of the critical components on modern web and mobile platforms have short release cycles. Web browsers, including Google Chrome and Mozilla Firefox, implement auto-update features and frequently push new versions and security patches to their users.

Mobile platforms also provide automatic application updates for third-party apps. While this approach generally results in quicker updates and the timely distribution of security patches, automatic mobile application updates are only enabled by default for devices connected to WiFi. Devices connected to a cellular network (e. g., 3G/4G) do not benefit from automatic application updates by default. This update behaviour ensures most third-party application updates are installed on mobile devices within a week [1529]. Automatic third-party application updates work well on mobile devices. Mobile operating system update behaviour heavily depends on the platform. In particular, many non-Google Android devices suffer from outdated and insecure operating system versions.

Overall, modern web and mobile platforms recognised the disadvantages of non-automatic

software updates and now provide automatic or semi-automatic platform or application updates in most cases.

Outdated Third Party Libraries While frequent software updates are crucial in general, updates of third-party libraries is a particularly important security measure for software developers who need to patch their own code and distribute updates, while also tracking vulnerabilities in libraries they use and updating them for better security. Derr et al. [1530] conducted a measurement study of third-party library update frequencies in Android applications and found that a significant number of developers use outdated libraries, exposing their users to security issues in the affected third party libraries. Lauinger et al. [1531] conducted a similar study for JavaScript libraries in web applications and also found many websites that include outdated and vulnerable libraries.

16.3 CLIENT SIDE VULNERABILITIES AND MITIGATIONS

[1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544]

This section covers attacks and their countermeasures with a focus on the client. It discusses issues in both modern web browsers and mobile devices. The illustrated security issues highlight aspects that have dominated security discussions in recent years. We focus on attacks that exploit weaknesses in the interaction between users and web browsers and mobile apps. We then discuss challenges resulting from the trend of storing more and more information on the client instead of the server. Finally, we discuss physical attacks that do not focus on exploiting software or human vulnerabilities, but exploit weak points in mobile devices.

16.3.1 Phishing & Clickjacking

This section presents two prevalent issues that exploit user interface weaknesses of both web and mobile clients. Phishing and clickjacking rely on issues humans have with properly verifying URLs and the dynamic content of rendered HTML documents.

16.3.1.1 Phishing

Phishing attacks are fraudulent attacks that aim to steal sensitive information, including login credentials and credit card numbers from victims [1534]. Common types of phishing attacks use email, websites or mobile devices to deceive victims. Attackers disguise themselves as trustworthy parties and send fake emails, show fake websites or send fake SMS or instant messages. Fake websites may look authentic. Attackers can use successfully stolen login credentials or credit card numbers to impersonate victims and access important online accounts. Successful phishing attacks may result in identity theft or loss of money.

Attackers commonly forge websites that appear legitimate to trick users into believing they are interacting with the genuine website. To initiate a phishing attack, attackers plant manipulated links on users via email, a website or any other electronic communication. The manipulated link leads to a forged website that appears to belong to the genuine organisation behind the website in question. Attackers often spoof online social media, online banking or electronic payment provider websites. They trick victims into following manipulated links using misspelled URLs, subdomains or homograph attacks.

Example: Phishing URL In the following example URL:
https://paymentorganization.secure.server.com,
it appears that the URL points to the **secure.server** section of the **paymentorganization** website. However, in fact the link leads to the **paymentorganization.secure** section of the **server.com** website.

To make forged websites look even more authentic, some phishers alter a browser's address bar by replacing the original address bar with a picture of the legitimate URL or by replacing the original address bar with a new one. Address bar manipulation attacks require the use of JavaScript commands. Additionally, phishers leverage Internationalised Domain Name (IDN) homograph attacks [1545]. Such attacks exploit that users cannot easily distinguish different character encodings. For example, the letters "l" and "I" (capital i) are hard to distinguish, and by replacing the Latin characters "a" with the cyrilic character "a" in the **https://paypal.com** url, users can deceptively be redirected to a phished PayPal website. [1546]. Attacks that involve manipulated URLs and address bars are even harder to detect in mobile browsers since the address bar is not visible during regular browsing. Website phishing is one of the most frequent attacks. Most human users find it hard to to spot phishing URLs and websites [1532].

Therefore, common countermeasures are anti-phishing training and public awareness campaigns [1533] that try to sensitise users and teach them how to spot phishing URLs. Modern browsers deploy technical security measures, including blacklists and visual indicators that highlight the top-level domain of a URL, e.g. Google Chrome shows URLs using an encoding that exposes deceptive characters in IDN attacks [7].

Drive-by-download Attacks Drive-by-download attacks happen when users visit a website, click on a link or on an attachment in a phishing email or on a malicious popup window. While being a general problem in the web, drive-by-downloads play a particular role in phishing attacks. Instead of visiting a benign website, drive-by-download attacks download and install malware (cf. the Malware & Attack Technologies Knowledge Area (Chapter 6)) on a user's computer. Attackers need to fingerprint victim clients and exploit vulnerable software components on the client's computer to plant the malware. Detecting such attacks is an active research area and includes approaches such as anomaly or signature based malware detection [797].

16.3.1.2 Clickjacking

In a clickjacking attack, attackers manipulate the visual appearance of a website to trick users into clicking on a fake link, button, or image. Clickjacking is also known as a *user interface redress attack* and belongs to the class of confused deputy attacks [1535]. Attackers fool their victims using transparent or opaque layers over original websites. While victims believe they have clicked on the overlay element, the original website element is clicked on. Attackers can thus make their victims trigger arbitrary actions on the original website. The attack website uses an iFrame to load the target website and can make use of the absolute positioning features of iFrames for correct visual alignment. Thus, it is hard for victims to detect the attack elements over the original website. Clickjacking attacks are particularly dangerous when victims have already logged in to an online account and visit their account settings website. In those cases, an attacker can trick the victim into performing actions on a trusted site when the victim is already logged in. One of the most prominent clickjacking attacks hit the Adobe Flash plugin settings page [1536]. Attackers used invisible iFrames to trick their

[7] cf. **https://www.chromium.org/developers/design-documents/idn-in-google-chrome**

victims into changing the plugin's security settings and permitting the attackers to access the microphone and camera of their victims' machines.

Clickjacking attacks can be used to launch other attacks against websites and their users, including Cross-Site Request Forgery and Cross-Site Scripting attacks (see Section 16.4.1) [1535].

A clickjacking attack is not a programming mistake but a conceptual problem with JavaScript. Hence, detection and prevention are not trivial. Detecting and preventing clickjacking attacks can be done both server- and client-side. Web browser users can disable JavaScript and iFrames to prevent clickjacking attacks. However, since this would break many legitimate websites, different browser plugins (e. g., NoScript [1537]) allow the controlled execution of JavaScript scripts on behalf of the user. In order to contain the impact of clickjacking attacks, users should log out of online accounts when leaving a website, although this could be impractical. In order to prevent clickjacking attacks on the server-side, website developers need to make sure that a website is not frame-able, i. e. a website does not load if it is inside an iFrame. Websites can include JavaScript code to detect whether a website has been put into an iFrame and break out of the iFrame. This defence technique is called FrameBusting [1538]. However, since users might have disabled JavaScript, this method is not reliable. The recommended server-side defence mechanism is to set a proper HTTP response header. The `X-FRAME-OPTIONS` header can be set to `DENY`, which will prevent a website being loaded inside an iFrame.

Clickjacking attacks affect both desktop and mobile web browsers.

Phishing and Clickjacking on Mobile Devices Phishing and Clickjacking are not limited to browsers and the web. Mobile application users are susceptible to both attacks. Aonzo et al. [1547] find that it is possible to trick users into an end-to-end phishing attack that allows attackers to gain full UI control by abusing Android's Instant App feature and password managers to steal login credentials. Fratantonio et al. [1548] describe the Cloak & Dagger attack that allows a malicious application with only two permissions (cf. Section 16.2.5) to take control over the entire UI loop. The attack allows for advanced clickjacking, keylogging, stealthy phishing and silent phone unlocking.

16.3.2 Client Side Storage

Client-side storage refers to areas that a browser or operating system provides to websites or mobile applications to read and write information. Storage is local to the client and does not require server-side resources or an active Internet connection. At the same time, malicious users may manipulate stored information. Hence, client-side storage areas need to be protected from malicious access. This section describes common client-side storage areas and their protection mechanisms.

16.3.2.1 Client Side Storage in the Browser

Historically, client-side browser storage was only used to store cookie information (see Section 16.2.8). However, due to their simple design and limited capacity, cookies cannot be used to store large or complex amounts of information. With the rise of HTML5, more powerful and feature-rich alternatives for client-side storage in the browser exist. These include WebStorage [1539], which is similar to cookies and stores key-value pairs, and IndexedDB [1540], which serves as a database in the vein of noSQL databases and can be used to store documents, other files and binary blobs.

As mentioned, the primary security issue with client-side storage mechanisms is that malicious users can manipulate them. To guarantee integrity for sensitive information (e. g., session information), developers are advised to cryptographically sign the data stored on the client and verify it upon retrieval.

In addition to information integrity, a second important aspect of WebStorage and IndexedDB storage is that stored information is not automatically cleared after users leave a website. To store information in a session-like fashion, web application developers are advised to rely on the `sessionStorage` object of the WebStorage API [1542].

16.3.2.2 Client Side Storage in Mobile Applications

In mobile applications, handling client-side storage security also depends on the type of information and storage mechanism, e. g., private storage of an application or public storage such as an SD card. Most importantly, data should be digitally signed and verified (cf. the Cryptography Knowledge Area (Chapter 10)) for both browser and mobile client storage purposes. It is recommended that developers sign and encrypt sensitive information and apply proper user input sanitisation. This is particularly relevant for shared storage such as SD-cards that do not use secure access control mechanisms. Instead, proper access administration mechanisms are provided for storage areas that are private to an application.

Sensitive Information Leaks in Android Applications Enck et al. [410] investigated the security of 1,100 popular Android applications. Amongst other things, they found that a significant number of apps leaked sensitive user information to publicly readable storage locations such as log files and the SD card. Reardon et al. [1549] discovered that some sensitive information leaks are made intentionally to pass sensitive information to another, collaborating and malicious app.

16.3.3 Physical Attacks

Instead of attacking web or mobile applications' code, physical attacks aim to exploit bugs and weak points that result from using a device. We focus on two representative examples below.

16.3.3.1 Smudge attacks

In a smudge attack, an attacker tries to learn passwords, PINs or unlock patterns entered on a touchscreen device. The main problem with entering sensitive unlock information through a touchscreen is the oily smudges that users' fingers leave behind when unlocking a device. Using inexpensive cameras and image processing software, an attacker can recover the grease trails and infer unlock patterns, passwords, and PINs [1543]. To perform a smudge attack, an attacker needs a clear view of the target display.

16.3.3.2 Shoulder Surfing

Shoulder surfing is a physical attack where an attacker tries to obtain sensitive information such as passwords, PINs, unlock patterns, or credit card numbers [1544]. For a shoulder surfing attack, an attacker needs a clear view of the target display. The attacker can mount a shoulder surfing attack either directly by looking over the victim's shoulder or from a longer range by using dedicated tools such as cameras or telescopes. Shoulder surfing attacks are particularly dangerous for mobile device users when authenticating to the device or online services in public spaces such as trains, railways, and airports.

16.4 SERVER SIDE VULNERABILITIES AND MITIGATIONS

[1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559]

This section discusses server-side security. It provides details for common aspects of server security, including well-known vulnerabilities and mitigations. The section discusses root causes, illustrates examples, and explains mitigations. The aspects discussed below are central for the web and mobile environments and dominated many of the security discussions in this area in the past.

16.4.1 Injection Vulnerabilities

Injection attacks occur whenever applications suffer from insufficient user input validation so that attackers can insert code into the control flow of the application (cf. the Software Security Knowledge Area (Chapter 15)). Prevalent injection vulnerabilities for web and mobile applications are SQL and Shell injections [1485]. Due to inadequate sanitisation of user input, requests to a database or shell commands can be manipulated by an attacker. Such attacks can leak or modify information stored in the database or issue commands on a system in ways developers or operators have not intended. The main goal of injection attacks is to circumvent authentication and expose sensitive information such as login credentials, personally identifiable information, or valuable intellectual property of enterprises.

Injection vulnerabilities can be addressed by adequately sanitising attacker-controlled information and deploying proper access control policies. The goal of input sanitisation is to filter invalid and dangerous input. Additionally, strict access control policies can be implemented to prevent injected code from accessing or manipulating information [1550].

16.4.1.1 SQL-Injection

SQL-injection attacks refer to code injections into database queries issued to relational databases using the Structured Query Language (SQL). Many web and mobile applications allow users to enter information through forms or URL parameters. SQL injection occurs if such user input is not filtered correctly for escape characters and then used to build SQL statements. Enabling attackers to modify SQL statements can result in malicious access or manipulation of information stored in the database.

Example: SQL Injection attack The statement below illustrates the vulnerability.

```
vuln_statement = "'SELECT * FROM creditcards WHERE number = '" +
    user_input + ";'"
```

The intention of the statement is to retrieve credit card information for a given user input. An example for an expected input 123456789.
However, the statement above allows malicious values for the `user_input` variable. An attacker might provide ' OR '1'='1 as input which would render the following SQL statement:

```
vuln_statement = "'SELECT * FROM creditcards WHERE number = '' OR '1'='1';"
```

Instead of retrieving detailed credit card information only for one specific credit card number, the statement retrieves information for all credit cards stored in the database table. A potential web application with the above SQL injection vulnerability could leak sensitive credit card information for all users of the application.

The consequences of the above SQL injection vulnerability might be directly visible to the attacker if all credit card details are listed on a results page. However, the impact of an SQL injection can also be hidden and not visible to the attacker.

blind SQL injections [1551], do not display the results of the vulnerability directly to the attacker (e. g., because results are not listed on a website). However, the impact of an attack might still be visible through observing information as part of a true-false response of the database. Attackers might be able to determine the true-false response based on the web application response and the way the web site is displayed.

second order In contrast to the previous types of SQL injection attacks, second order attacks occur whenever user submitted input is stored in the database for later use. Other parts of the application then rely on the stored user input without escaping or filtering it properly.

One way to mitigate SQL injection attacks is with the use of prepared statements [1552, 1553]. Instead of embedding user input into raw SQL statements (see above), prepared statements use placeholder [8] variables to process user input. Placeholder variables are limited to store values of a given type and prohibit the input of arbitrary SQL code fragments. SQL injections attacks would result in invalid parameter values in most cases and not work as intended by

[8] Also called bind variables.

an attacker. Also, prepared statements are supported by many web application development frameworks at the coding level using Object Relational Mapping (ORM) interfaces. ORMs do not require developers to write SQL queries themselves but generate database statements from code. While prepared statements are an effective mitigation mechanism, a further straightforward way is to escape characters in user input that have a special meaning in SQL statements. However, this approach is error-prone, and many applications that apply some form of SQL escaping are still vulnerable to SQL injection attacks. The reasons for the mistakes are often incomplete lists of characters that require escaping. When escaping is used, developers should rely on functions provided by web application development frameworks (e. g. the `mysqli_real_escape_string()` function in PHP) instead of implementing their own escaping functionality.

16.4.1.2 Command Injections

This type of injection attack affects vulnerable applications that can be exploited to execute arbitrary commands on the host operating system of a web application [1560]. Similar to SQL injection attacks, command injections are mostly possible due to insufficient user input validation. Vulnerable commands usually run with the same privileges as the host application.

An example of a command injection attack is a web application that converts user-provided images using a vulnerable image command line program. Providing malicious input (e. g., a filename or a specially crafted support graphic that includes malicious code) might allow attackers to exploit insufficient input validation and extend the original command or run additional system commands.

A mitigation for command injection attacks is to construct the command strings, including all parameters in a safe way that does not allow attackers to exploit malicious string input. In addition to proper input validation due to escaping, following the principle of least-privilege and restricting the privileges of system commands and the calling application is recommended. The number of callable system commands should be limited by using string literals instead of raw user-supplied strings. In order to further increase security, regular code reviews are recommended, and vulnerability databases (e. g., the CVE [1554] database) should be monitored for new vulnerabilities. Finally, if possible, executing system commands should be avoided altogether. Instead, the use of API calls in the respective development framework is recommended.

16.4.1.3 User Uploaded Files

Files provided by users such as images or PDFs have to be handled with care. Malicious files trigger unwanted command execution on the host operating system of the server, overload the host system, trigger client-side attacks, or deface vulnerable applications [1485].

> **Example: Online Social Network** An example application could be an online social network that allows users to upload their avatar picture. Without proper mitigation techniques in place, the web application itself might be vulnerable. A malicious user could upload a `.php` file. Accessing that file might prompt the server to process it as an executable PHP file. This vulnerability would allow attackers to both execute code on the server with the permissions of the PHP process and also control the content served to other users of the application.

To prevent attacks through user-uploaded files, both meta-data including file names and the

actual content of user-uploaded files need to be restricted and filtered, e.g. looking for malware in uploaded files. Filenames and paths should be constructed using string literals instead of raw strings and proper mime-types for HTTP responses used whenever possible.

Files that are only available for download and should not be displayed inline in the browser, can be tagged with a `Content-Disposition` HTTP response header [1555]. Another successful mitigation for the above issue is to serve files from a different domain. If the domain is not a subdomain of the original domain, the SOP 16.2.4.2 prevents cookies and other critical information from being accessible to the malicious file. Additionally, JavaScript and HTML files are protected by the SOP as well.

16.4.1.4 Local File Inclusion

This type of vulnerability is a particular form of the above command injection or user-uploaded files vulnerabilities [1485]. For example, attackers can exploit a command injection, use a malformed path in a database or a manipulated filename. The file path resulting from one of these vulnerabilities can be crafted to point to a local file on the server, e. g., a `.htaccess` or the `/etc/shadow` file. A vulnerable web application might then access the maliciously crafted file path and instead of loading a benign file, read and send the content of the attacker-chosen file and e. g. leak login credentials in the `/etc/shadow` file.

In addition to sanitisation of file path parameters such as leading `/` and `..` in user input, the application of the least privilege principle is recommended. A web application should be executed with minimal privileges and so that it cannot access sensitive files.

16.4.1.5 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) [1556] attacks are injection vulnerabilities that allow attackers to inject malicious scripts (e. g., JavaScript) into benign websites. They can occur whenever malicious website users are able to submit client scripts to web applications that redistribute the malicious code to other end-users. Common examples of websites that are vulnerable to XSS attacks are message forums that receive user content and show it to other users. The primary root cause for XSS vulnerabilities is web applications that do not deploy effective input validation mechanisms. Untrusted and non-validated user-provided data might contain client-side scripts. Without proper user input validation, a malicious JavaScript previously provided by one user, might be distributed to other users and manipulate the website they are visiting or steal sensitive information. In an XSS attack, the client browser cannot detect the malicious code, since it is sent from the original remote host, i. e. *same-origin-policy* based security measures are ineffective. We distinguish two types of XSS attacks:

stored In a stored XSS attack the malicious script is permanently stored on the target server (e. g. in a database) and distributed to the victims whenever they request the stored script for example as part of a comment in a message forum. Stored XSS attacks are also called permanent or Type-I XSS.

reflected In a reflected XSS attack, the malicious script is not permanently stored on the target server, but reflected by the server to the victims. Malicious scripts in reflected attacks are distributed through different channels. A common way of delivering a malicious script is to craft a link to the target website. The link contains the script and clicking the link executes the malicious script in the website's script execution context. Reflected XSS attacks are also called non-permanent or Type-II XSS.

Preventing both types of XSS attacks requires rigorous user input validation and escaping by the server. The most effective means of input validation is a whitelist approach, which denies any input that is not explicitly allowed. For proper and secure entity encoding, the use of a security encoding library is recommended, since writing encoders is very difficult and code review in combination with the use of static code analysis tools is also valuable.

Since eliminating XSS vulnerabilities entirely due to user input sanitization is hard, different approaches are discussed in the literature. A promising approach is the randomisation of HTML tags and attributes. Web applications randomise their order so clients can distinguish between benign and trusted content and potentially untrusted malicious content. As long as an attacker does not know the randomisation mapping, clients can successfully distinguish trusted from untrusted scripts [1561].

16.4.1.6 Cross-Site Request Forgery

Cross Site Request Forgery (CSRF) [1557] attacks mislead victims into submitting malicious HTTP requests to remote servers. The malicious request is executed on behalf of the user and inherits their identity and permissions. CSRF attacks are so dangerous because most requests to remote servers include credentials and session information associated with a user's identity, including session cookies. Authenticated users are particularly attractive victims for attackers since it can be hard for remote servers to distinguish between benign and malicious requests as long as they are submitted from the victim's machine. CSRF attacks do not easily allow attackers to access the server response for the malicious request. Therefore, the main goal of a CSRF attack is to trick victims into submitting state-changing requests to remote servers. Attractive targets are requests that change the victim's credentials or purchase something.

Example: Online Banking In the following online banking scenario Alice wishes to transfer 50 EUR to Bob using an online banking website that is vulnerable to a CSRF attack. A benign request for an authenticated user Alice for the mentioned scenario could be similar to GET `https://myonlinebank.net/transaction?to=bob&value=50`. In a first step, an attacker can craft a malicious URL such as **https://myonlinebank.et/transaction?to=attacker&value=50** and replace the intended recipient of the transaction with the attacker's account. The second step for successful CSRF attack requires the attacker to trick Alice into sending the malicious request with her web browser, e. g. by sending a SPAM email containing the request which Alice subsequently clicks on. However, CSRF attacks are not limited to HTTP GET requests but also affect POST requests, e. g. by crafting malicious `<form>` tags.

Many misconceptions lead to ineffective countermeasures. CSRF attacks cannot be prevented by using secret cookies because all cookies are sent from a victim to the remote server. Also, the use of HTTPS is ineffective as long as the malicious request is sent from the victim, because the protocol does not matter and the use of POST requests for sensitive information is insufficient since attackers can craft malicious HTML forms with hidden fields. To effectively prevent CSRF attacks, it is recommended to include randomised tokens in sensitive requests, e. g., by adding them to the request headers. The tokens must be unique per session and generated with a secure random number generator to prevent attackers from predicting them. Servers must not accept requests from authenticated clients that do not include a valid token.

16.4.2 Server Side Misconfigurations & Vulnerable Components

A web application stack consists of multiple components, including web servers, web application frameworks, database servers, firewall systems, and load balancers and proxies. Overall, web application security highly depends on the security of each of the involved components.

A single insecure component is often enough to allow an attacker access to the web application and further escalate their attack from the inside. This is why deploying and maintaining a secure web application requires more than focusing on the code of the app itself. Every component of the web application stack needs to be configured securely and kept up to date (see Section 16.2.10).

The Heartbleed Vulnerability A famous example of a critical vulnerability that affected many web application stacks in 2014 is Heartbleed [1558]. Heartbleed was a vulnerability in the widely used OpenSSL library and caused web servers to leak information stored in the webservers' memory. This included TLS certificate information such as private keys, connection encryption details, and any data the user and server communicated, including passwords, usernames, and credit card information [1559]. To fix affected systems, administrators had to update their OpenSSL libraries as quickly as possible and ideally also revoke certificates and prompt users to change their passwords.

As previously discussed, the principle of least privilege can reduce a web application's attack surface tremendously. Proper firewall and load balancer configurations serve as examples:

16.4.2.1 Firewall

To protect a webserver, a firewall should be configured to only allow access from outside where access is needed. Access should be limited to ports like 80 and 443 for HTTP requests via the Internet and restricting system configuration ports for SSH and alike to the internal network (cf. the Network Security Knowledge Area (Chapter 19)).

16.4.2.2 Load Balancers

A load balancer is a widely deployed component in many web applications. Load balancers control HTTP traffic between servers and clients and provide additional access control for web application resources. They can be used to direct requests and responses to different web servers or ports, balance traffic load between multiple web servers and protect areas of a website with additional access control mechanisms. The most common approach for controlling access is the use of `.htaccess` files. They can restrict access to content on the original web server and instruct load balancers to require additional authentication.

Load balancers can also serve for rate limiting purposes. They can limit request size, allowed request methods and paths or define timeouts. The main use of rate-limiting is to reduce the potentially negative impact of denial of service attacks on a web server and prevent users from spamming systems, as well as restrict and prevent unexpected behavior.

Additionally, load balancers can be used to provide secure TLS connections for web applications. When managing TLS, load balancers serve as a network connection endpoint for the TLS encryption and either establish new TLS connections to the application service or connect to the web application server using plain HTTP. If the web application server is not hosted on the same machine, using plain network connections might leak information to the

internal network. However, if the web application server does not provide HTTPS itself, using a load balancer as a TLS endpoint increases security.

HTTPS Misconfigurations One cornerstone of web and mobile security is the correct and secure configuration of HTTPS on web servers. However, Holz et al. [1562] found that a significant number of popular websites deploy invalid certificates with incomplete certificate chains, issued for the wrong hostname or expired lifetime. In a similar study Fahl et al. [1563] confirmed these findings and also asked website operators for the reasons for deploying invalid certificates. Most operators were not aware of using an invalid certificate or used one on purpose because they did not trust the web PKI. Krombholz et al. [1564, 1565] conducted a set of studies and found that operators have difficulties with correctly configuring HTTPS, or they harbour misconceptions about the security features of HTTPS.

16.4.2.3 Databases

Similar to load balancers and firewalls, many web applications include databases to store user information permanently. Often, databases are operated as an additional service that is hosted on another server. The application server interacts with the database through libraries and APIs. It is important to prevent injection vulnerabilities on the server. Additionally, errors in the implementation of database libraries or coarse permissions required by the application can lead to vulnerabilities.

To reduce the attack vector, most database systems provide user management, to limit user privileges to create, read, delete or modify entries in tables and across databases. In this way one database per application can be created and particular users with read-only permissions can be used by the application server.

An important aspect of increasing database security is the decision on how to store data. Encrypting data before storage in the database can help. However, especially for passwords or other information that only needs to be compared for equality, hashing before storage can tremendously increase security. In the case of a data leak, the sensitive information remains unreadable. To store passwords securely, web and mobile app developers are recommended to use a secure hash function such as Argon2 [1566] or PBKDF2 [1567] in combination with a cryptographically strong credential-specific salt. A salt is a cryptographically strong fixed-length random value and needs to be newly generated for for each set of credentials [1568].

Password Leaks Developers tend to store plain passwords, credit card information or other sensitive information in databases instead of encrypting or hashing them (cf. the Human Factors Knowledge Area (Chapter 4)). Hence, many leaks of password databases or credit card information put users at risk [1569]. Modern browsers and password managers help users to avoid passwords that were part of a previous data breach [1570].

16.5 CONCLUSION

As we have shown, web and mobile security is a diverse and broad topic covering many areas. This Knowledge Area emphasised an intersectional approach by exploring security concepts and mechanisms that can be found in both the web and the mobile worlds. It therefore builds upon and extends the insights from other Knowledge Areas, in particular the Software Security Knowledge Area (Chapter 15), Network Security Knowledge Area (Chapter 19), Human Factors Knowledge Area (Chapter 4), Operating Systems & Virtualisation Knowledge Area (Chapter 11), Privacy & Online Rights Knowledge Area (Chapter 5), Authentication, Authorisation & Accountability Knowledge Area (Chapter 14) and the Physical Layer and Telecommunications Security Knowledge Area (Chapter 22).

We showed that due to the ubiquitous availability and use of web and mobile applications and devices, paying attention to their security issues is crucial for overall information security. We discussed web technologies that build the core of both web and mobile security, outlined their characteristics and illustrated how they are different from other ecosystems.

Later on, we split the discussion into client- and server-side aspects. In particular, this Knowledge Area has focused on attacks and defences that were prevalent in web and mobile clients and servers and that dominated discussions in recent years.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Section	References
16.2 Fundamental Concepts and Approaches	
16.2.1 Appification	[1475, 1484]
16.2.2 Webification	[1478, 1485, 1494]
16.2.3 Application Stores	[1495, 1496, 1498]
16.2.4 Sandboxing	[1499, 1500, 1501, 1502]
16.2.5 Permission Dialog Based Access Control	[1393, 1479]
16.2.6 Web PKI and HTTPS	[1480, 1481, 1506, 1511]
16.2.7 Authentication	[367, 1512]
16.2.8 Cookies	[1515]
16.2.9 Passwords and Alternatives	[1519, 1521, 1523]
16.2.10 Frequent Software Updates	[1530, 1531]
16.3 Client Side Vulnerabilities and Mitigations	
16.3.1 Phishing & Clickjacking	[1532, 1533, 1534, 1538]
16.3.2 Client Side Storage	[1539]
16.3.3 Physical Attacks	[1543, 1544]
16.4 Server Side Vulnerabilities and Mitigations	
16.4.1 Injection Vulnerabilities	[1550, 1551, 1552, 1556, 1557]
16.4.2 Server Side Misconfigurations & Vulnerable Components	[1562, 1564, 1565, 1570]

FURTHER READING

The following resources provide a deeper insight into web and mobile security as well as guidance and recommendations for preventing and handling the vulnerabilities presented and discussed above.

The OWASP Project & Wiki

The Open Web Application Security Project (OWASP) is an international not-for-profit charitable organisation providing practical information about application and web security. It funds many projects including surveys like the OWASP TOP 10, books, CTFs and a wiki containing in-depth descriptions, recommendations and checklists for vulnerabilities and security measurements. The core wiki can be found at **https://www.owasp.org/**.

Mozilla Developer Network

An all-encompassing resource provided by Mozilla covering open web standards, including security advice and cross platform behaviour for Javascript APIs, as well as a HTML and CSS specifications. It can be found at **https://developer.mozilla.org**

Android Developers

The official documentation for the Android development ecosystem, including security advice for client side storage, webviews, permissions, Android databases and network connections. It also includes information for outdated operating system versions and the Google Play Update process. Available at **https://developer.android.com**

Chapter 17
Secure Software Lifecycle

Laurie Williams | North Carolina State University

INTRODUCTION

The purpose of this Secure Software Lifecycle knowledge area is to provide an overview of software development processes for implementing secure software from the design of the software to the operational use of the software. This implementation may involve new coding as well as the incorporation of third party libraries and components. The goal of this overview is for use in academic courses in the area of software security; and to guide industry professionals who would like to use a secure software lifecycle.

The Software Security Knowledge Area (Chapter 15) provides a structured overview of secure software development and coding and the known categories of software implementation vulnerabilities and of techniques that can be used to prevent or detect such vulnerabilities or to mitigate their exploitation. By contrast, this Secure Software Lifecycle Knowledge Area focuses on the components of a comprehensive software development process to prevent and detect security defects and to respond in the event of an exploit.

This Knowledge Area will begin with a history of secure software lifecycle models. Section 2 provides examples of three prescriptive secure software lifecycle processes; the Microsoft Secure Development Lifecycle, Touchpoints, and SAFECode. Section 3 discusses how these processes can be adapted in six specific domains: agile/DevOps, mobile, cloud computing, internet of things, road vehicles, and ecommerce/payment card. Section 4 provides information on three frameworks for assessing an organisation's secure software lifecycle process.

CONTENT

17.1 MOTIVATION

[1469, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578]

Historically, and at times currently, organisations have focused their security strategies at the network system level, such as with firewalls, and have taken a *reactive* approach to software security, using an approach commonly referred to as 'penetrate and patch'. [1574] With this approach, security is assessed when the product is complete via penetration testing by attempting known attacks; or vulnerabilities are discovered post release when organisations are victims of an attack on deployed software. In either case, organisations then react by finding and fixing the vulnerability via a security patch. The following shortcomings are likely to be more prevalent with a predominantly reactive approach to cyber security:

- **Breaches are costly.** Based upon a study of 477 companies in 15 countries, in 2018 the Poneman Institute [1573] reported that a breach cost, on average, 7.9 million US dollars in the United States and 5.3 million US dollars in the Middle East. Breaches were the least expensive in India and Brazil, but these countries still spent an average of 1.8 million and 1.2 million US dollars per breach, respectively. Loss of reputation caused by a breach is difficult to quantify.
- **Attackers can find and exploit vulnerabilities without being noticed.** Based upon a study of 477 companies in 15 countries, in 2018 the Poneman Institute [1573] reported that the mean time to identify that a breach had occurred was 197 days, and the mean time to find and fix a vulnerability once the breach was detected was an additional 69 days.

- **Patches can introduce new vulnerabilities or other problems.** Vulnerability patches are considered urgent and can be rushed out, potentially introducing new problems to a system. For example, Microsoft's early patches for the Meltdown[1] chip flaw introduced an even more serious vulnerability in Windows 7[2]. The new vulnerability allowed attackers to read kernel memory much faster and to write their own memory, and could allow an attacker to access every user-level computing process running on a machine.
- **Patches often go unapplied by customers.** Users and system administrators may be reluctant to apply security patches. For example, the highly-publicised Heartbleed[3] vulnerability in OpenSSL allows attackers to easily and quietly exploit vulnerable systems, stealing passwords, cookies, private crypto-keys, and much more. The vulnerability was reported in April 2014; but in January 2017 a scan revealed 200,000 Internet-accessible devices remained unpatched [1575]. Once a vulnerability is publicly reported, attackers formulate a new mechanism to exploit the vulnerability with the knowledge that many organisations will not adopt the fix.

In 1998, McGraw [1574] advocated moving beyond the penetrate and patch approach based upon his work on a DARPA-funded research effort investigating the application of software engineering to the assessment of software vulnerabilities. He contended that *proactive* rigorous software analysis should play an increasingly important role in assessing and preventing vulnerabilities in applications based upon the well-known fact that security violations occur because of errors in software design and coding. In 2002, Viega and McGraw published the first book on developing secure programs, *Building Secure Software* [1469], with a focus on preventing the injection of vulnerabilities and reducing security risk through an integration of security into a software development process.

In the early 2000s, attackers became more aggressive, and Microsoft was a focus of this aggression with exposure of security weaknesses in their products, particularly the Internet Information Services (IIS). Gartner, a leading research and advisory company who seldom advises its clients to steer clear of specific software, advised companies to stop using IIS. In response to customer concerns and mounting bad press, the then Microsoft CEO, Bill Gates, sent the *Trustworthy Computing* memo [1571] to all employees on January 15, 2002. The memo was also widely circulated on the Internet. An excerpt of the memo defines Trustworthy Computing:

> 'Trustworthy Computing is the highest priority for all the work we are doing. We must lead the industry to a whole new level of Trustworthiness in computing ... Trustworthy Computing is computing that is as available, reliable and secure as electricity, water services and telephony'.

The Trustworthy Computing memo caused a shift in the company. Two weeks later, Microsoft announced the delay of the release of Windows .NET Server [1576] to ensure a proper security review (referred to as the Windows Security Push), as mandated by Microsoft's Trustworthy Computing initiative outlined in this memo. In 2003, Microsoft employees Howard and Le Blanc [1577] publicly published a second edition of a book on writing secure code to prevent vulnerabilities, to detect design flaws and implementation bugs, and to improve test code and documentation. The first edition had been required reading for all members of the Windows team during the Push.

[1] **https://meltdownattack.com/** Meltdown lets hackers get around a barrier between applications and computer memory to steal sensitive data.

[2] **https://www.cyberscoop.com/microsoft-meltdown-patches-windows-7-memory-management/**

[3] **http://heartbleed.com/**

During the ensuing years, Microsoft changed their development process to build secure products through a comprehensive overhaul of their development process from early planning through product end-of-life. Their products contained demonstrably fewer vulnerabilities [1577]. After internal use of the process, Microsoft codified and contributed their 13-stage internal development process, the Microsoft Security Development Lifecycle (SDL) to the community through its book entitled *The Security Development Lifecycle* [1572] in 2006. True to Gates' original intent, the Microsoft SDL provided the foundation for the information technology industry by providing the first documented comprehensive and prescriptive lifecycle. Also in 2006, McGraw published the first book on software security best practices [1578].

As discussed in the rest of this knowledge area, organisations have built upon the foundation set forth by Microsoft and by Viega and McGraw [1469, 1574].

17.2 PRESCRIPTIVE SECURE SOFTWARE LIFECYCLE PROCESSES

[8, 53, 58, 1469, 1572, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602]

Secure software lifecycle processes are *proactive* approaches to building security into a product, treating the 'disease' of poorly designed, insecure software at the source, rather than 'applying a band aid' to stop the symptoms through a *reactive* penetrate and patch approach. These processes work software security deeply into the full product development process and incorporate people and technology to tackle and prevent software security issues. This section will provide information on three prominent secure software lifecycle processes and then reflect on the commonalities between them in Table 17.1.

17.2.1 Secure Software Lifecycle Processes

Three exemplar prescriptive secure software lifecycle processes are summarised in this section. The processes are *prescriptive* in that they explicitly recommend software practices. The three processes were chosen because the practices of these processes are integrated and cover a broad spectrum of the lifecycle phases, from software requirements to release/deployment and software maintenance. Two of these processes were identified in a systematic mapping study [1591] on security approaches in software development lifecycles. As such, the practices span the prevention of security defects, the detection of security defects, and the mitigation of security defects once a product is in the field. The three were also chosen due to their maturity in terms of the number of years they have existed and in terms of their widespread acceptance in the industry. As will be discussed in Section 2.2, no 'best' secure software lifecycle process exists. Practitioners should consider incorporating practices from each of these processes into their own secure software process.

17.2.1.1 Microsoft Security Development Lifecycle (SDL)

As discussed in Section 17.1, Microsoft used an internal development process, the Security Development Lifecycle (SDL), to improve the security of their products. Howard and Lipner [1572] disseminated a snapshot of this process in 2006. Since that time, Microsoft has continued to evolve their SDL and to provide up-to-date resources to the community [1579], including an increased focus on compliance requirements that are being imposed on the industry.

Currently [1579], the Microsoft SDL contains 12 practices which are enumerated below. For each of the practices, techniques for implementing the practice may be mentioned though the SDL does not prescribe the specific technique.

1. **Provide Training**. A range of professionals, such as developers, service engineers, program managers, product managers, and project managers, participate in the development of secure products while still addressing business needs and delivering user value. Software developers and architects must understand technical approaches for preventing and detecting vulnerabilities. The entire development organisation should be cognisant of the attacker's perspective, goals, and techniques; and of the business implications of not building secure products.

   Often, the formal education of these professionals does not include cyber security. Additionally, attack vectors, security tools, secure programming languages, and experiences are constantly evolving, so knowledge and course material must be refreshed. Ongoing cyber security training is essential for software organisations.

2. **Define Security Requirements**. Security requirements should be defined during the initial design and planning phases. Factors that influence these requirements include the specific functional requirements of the system, the legal and industry compliance requirements, internal and external standards, previous security incidents, and known threats.

   Techniques have been developed for systematically developing security requirements. For example, Security Quality Requirements Engineering (SQUARE) [1592] is a nine-step process that helps organisations build security into the early stages of the production lifecycle. Abuse cases, as will be discussed in Section 2.1.2 bullet 5, are another means of specifying security requirements. van Lamsweerde extended the Keep All Objectives Satisfied (KAOS) framework for goal-based requirements specification language to include anti-models [1593]. An anti-model is constructed by addressing malicious obstacles (called anti-goals) set up by attackers to threaten a system's security goals. An obstacle negates existing goals of the system. Secure i* [1594] extends the i*-modeling framework with modeling and analysis of security trade-offs and aligns security requirements with other requirements.

   Security requirements must be continually updated to reflect changes in required functionality, standards, and the threat landscape.

3. **Define Metrics and Compliance Reporting**. Lord Kelvin is quoted as stating, 'If you can not measure it, you can not improve it'. The management team should understand and be held accountable for minimum acceptable levels of security using security metrics [1581]. A subset of these metrics may be set as Key Performance Indicators (KPIs) for management reporting. Defect tracking should clearly label security defects and security work items as such to allow for accurate prioritisation, risk management,

tracking, and reporting of security work. Additionally, products increasingly must comply with regulatory standards, such as the Payment Card Industry Data Security Standard (PCI DSS)[4], or the EU General Data Protection Regulation [105] (GDPR)[5], which may impose additional process steps and metrics for compliance, reporting, and audits.

4. **Perform Threat Modelling**. Through the use of threat modelling [58, 1588], teams consider, document and discuss the security implications of designs in the context of their planned operational environment and in a structured fashion. Teams should consider the motivations of their adversaries and the strengths and weaknesses of systems to defend against the associated threat scenarios. An approach is to consider the (1) the malicious and benevolent interactors with the system; (2) the design of the system and its components (i.e. processes and data stores), (3) the trust boundaries of the system; and (4) the data flow of the system within and across trust boundaries to/from its interactors. Threats can be enumerated using a systematic approach of considering each system component relative to the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) [1577] threats:

   (a) **Spoofing identity**. Spoofing threats allow an attacker to pose as another user or allow a rogue server to pose as a valid server.

   (b) **Tampering with data**. Data tampering threats involves malicious modification of data.

   (c) **Repudiation**. Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise.

   (d) **Information disclosure**. Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it.

   (e) **Denial of Service**. A Denial of Service (DoS) attack denies service to valid users by making the system unavailable or unusable.

   (f) **Elevation of privilege**. An unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the system.

   Threat modelling aids the team in enumerating threats, so that the system design can be fortified and security features can be selected. In addition to STRIDE, other models exist to formulate threat models, such as 12 methods[6], including attack trees [1595] which are conceptual diagrams of threats on systems and possible attacks to reach those threats. A closely-related practice to threat modelling is Architectural Risk Analysis, as will be discussed in Section 2.1.2 bullet 2.

   Games have been created to aid teams in collaboratively (and competitively) conducting threat modeling:

   (a) Elevation of Privilege[7]

   (b) Security Cards[8]

   (c) Protection Poker [1596]

[4] https://www.pcisecuritystandards.org/
[5] https://eugdpr.org/
[6] https://insights.sei.cmu.edu/sei_blog/2018/12/threat-modeling-12-available-methods.html
[7] https://www.usenix.org/conference/3gse14/summit-program/presentation/shostack
[8] https://securitycards.cs.washington.edu/

5. **Establish Design Requirements**. Design requirements guide the implementation of 'secure features' (i.e., features that are well engineered with respect to security). Additionally, the architecture and design must be resistant to known threats in the intended operational environment.

   The design of secure features involves abiding by the timeless security principles set forth by Saltzer and Schroeder [8] in 1975 and restated by Viega and McGraw [1469] in 2002. The eight Saltzer and Schroeder principles are:

   - **Economy of mechanism**. Keep the design of the system as simple and small as possible.
   - **Fail-safe defaults**. Base access decisions on permissions rather than exclusion; the default condition is lack of access and the protection scheme identifies conditions under which access is permitted. Design a security mechanism so that a failure will follow the same execution path as disallowing the operation.
   - **Complete mediation**. Every access to every object must be checked for authorisation.
   - **Open design**. The design should not depend upon the ignorance of attackers but rather on the possession of keys or passwords.
   - **Separation of privilege**. A protection mechanism that requires two keys to unlock is more robust than one that requires a single key when two or more decisions must be made before access should be granted.
   - **Least privilege**. Every program and every user should operate using the least set of privileges necessary to complete the job.
   - **Least common mechanism**. Minimise the amount of mechanisms common to more than one user and depended on by all users.
   - **Psychological acceptability**. The human interface should be designed for ease of use so that users routinely and automatically apply the mechanisms correctly and securely.

   Two other important secure design principles include the following:

   - **Defense in depth**. Provide multiple layers of security controls to provide redundancy in the event a security breach.
   - **Design for updating**. The software security must be designed for change, such as for security patches and security property changes.

   Design requirements also involve the selection of security features, such as cryptography, authentication and logging to reduce the risks identified through threat modelling. Teams also take actions to reduce the attack surface of their system design. The attack surface, a concept introduced by Howard [1582] in 2003, can be thought of as the sum of the points where attackers can try to enter data to or extract data from a system [1589, 1590].

   In 2014, the IEEE Center for Secure Design [1583] enumerated the top ten security design flaws and provided guidelines on techniques for avoiding them. These guidelines are as follows:

   (a) Earn or give, but never assume, trust.

(b) Use an authentication mechanism that cannot be bypassed or tampered with.

(c) Authorise after you authenticate.

(d) Strictly separate data and control instructions, and never process control instructions received from untrusted sources.

(e) Define an approach that ensures all data are explicitly validated.

(f) Use cryptography correctly.

(g) Identify sensitive data and how they should be handled.

(h) Always consider the users.

(i) Understand how integrating external components changes your attack surface.

(j) Be flexible when considering future changes to objects and actors.

6. **Define and Use Cryptography Standards**. The use of cryptography is an important design feature for a system to ensure security- and privacy-sensitive data is protected from unintended disclosure or alteration when it is transmitted or stored. However, an incorrect choice in the use of cryptography can render the intended protection weak or ineffective. Experts should be consulted in the use of clear encryption standards that provide specifics on every element of the encryption implementation and on the use of only properly vetted encryption libraries. Systems should be designed to allow the encryption libraries to be easily replaced, if needed, in the event the library is broken by an attacker, such as was done to the Data Encryption Standard (DES) through 'Deep Crack'[9], a brute force search of every possible key as designed by Paul Kocher, president of Cryptography Research.

7. **Manage the Security Risk of Using Third-Party Components**. The vast majority of software projects are built using proprietary and open-source third-party components. The Black Duck On-Demand audit services group [1584] conducted open-source audits on over 1,100 commercial applications and found open-source components in 95% of the applications with an average 257 components per application. Each of these components can have vulnerabilities upon adoption or in the future. An organisation should have an accurate inventory of third-party components [1598], continuously use a tool to scan for vulnerabilities in its components, and have a plan to respond when new vulnerabilities are discovered. Freely available and proprietary tools can be used to identify project component dependencies and to check if there are any known, publicly disclosed, vulnerabilities in these components.

8. **Use Approved Tools**. An organisation should publish a list of approved tools and their associated security checks and settings such as compiler/linker options and warnings. Engineers should use the latest version of these tools, such as compiler versions, and take advantage of new security analysis functionality and protections. Often, the resultant software must be backward compatible with previous versions.

9. **Perform Static Analysis Security Testing (SAST)**. SAST tools can be used for an automated security code review to find instances of insecure coding patterns and to help ensure that secure coding policies are being followed. SAST can be integrated into the commit and deployment pipeline as a check-in gate to identify vulnerabilities each time the software is built or packaged. For increased efficiency, SAST tools can integrate into

[9] https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html

the developer environment and be run by the developer during coding. Some SAST tools spot certain implementation bugs, such as the existence of unsafe or other banned functions and automatically replace with (or suggest) safer alternatives as the developer is actively coding. See also the Software Security Knowledge Area (Section 15.3.1).

10. **Perform Dynamic Analysis Security Testing (DAST)**. DAST performs run-time verification of compiled or packaged software to check functionality that is only apparent when all components are integrated and running. DAST often involves the use of a suite of pre-built attacks and malformed strings that can detect memory corruption, user privilege issues, injection attacks, and other critical security problems. DAST tools may employ *fuzzing*, an automated technique of inputting known invalid and unexpected test cases at an application, often in large volume. Similar to SAST, DAST can be run by the developer and/or integrated into the build and deployment pipeline as a check-in gate. DAST can be considered to be automated penetration testing. See also the Software Security Knowledge Area (Section 15.3.2).

11. **Perform Penetration Testing**. Manual penetration testing is black box testing of a running system to simulate the actions of an attacker. Penetration testing is often performed by skilled security professionals, who can be internal to an organisation or consultants, opportunistically simulating the actions of a hacker. The objective of a penetration test is to uncover any form of vulnerability - from small implementation bugs to major design flaws resulting from coding errors, system configuration faults, design flaws or other operational deployment weaknesses. Tests should attempt both unauthorised misuse of and access to target assets and violations of the assumptions. A widely-referenced resource for structuring penetration tests is the OWASP Top 10 Most Critical Web Application Security Risks[10]. As such, penetration testing can find the broadest variety of vulnerabilities, although usually less efficiently compared with SAST and DAST [1585]. Penetration testers can be referred to as white hat hackers or ethical hackers. In the penetration and patch model, penetration testing was the only line of security analysis prior to deploying a system.

12. **Establish a Standard Incident Response Process**. Despite a secure software lifecycle, organisations must be prepared for inevitable attacks. Organisations should proactively prepare an Incident Response Plan (IRP). The plan should include who to contact in case of a security emergency, establish the protocol for efficient vulnerability mitigation, for customer response and communication, and for the rapid deployment of a fix. The IRP should include plans for code inherited from other groups within the organisation and for third-party code. The IRP should be tested before it is needed. Lessons learned through responses to actual attack should be factored back into the SDL.

[10] **https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project**

17.2.1.2 Touchpoints

International software security consultant, Gary McGraw, provided seven Software Security Touchpoints [1578] by codifying extensive industrial experience with building secure products. McGraw uses the term *touchpoint* to refer to software security best practices which can be incorporated into a secure software lifecycle. McGraw differentiates vulnerabilities that are implementation bugs and those that are design flaws [1583]. *Implementation bugs* are localized errors, such as buffer overflow and input validation errors, in a single piece of code, making spotting and comprehension easier. *Design flaws* are systemic problems at the design level of the code, such as error-handling and recovery systems that fail in an insecure fashion or object-sharing systems that mistakenly include transitive trust issues [1578]. Kuhn et al. [1598] analysed the 2008 - 2016 vulnerability data from the US National Vulnerability Database (NVD)[11] and found that 67% of the vulnerabilities were implementation bugs. The seven touchpoints help to prevent and detect both bugs and flaws.

These seven touchpoints are described below and are provided in order of effectiveness based upon McGraw's experience with the utility of each practice over many years, hence prescriptive:

1. **Code Review (Tools)**.

   Code review is used to detect implementation bugs. Manual code review may be used, but requires that the auditors are knowledgeable about security vulnerabilities before they can rigorously examine the code. 'Code review with a tool' (a.k.a. the use of static analysis tools or SAST) has been shown to be effective and can be used by engineers that do not have expert security knowledge. For further discussion on static analysis, see Section 2.1.1 bullet 9.

2. **Architectural Risk Analysis**.

   Architectural Risk Analysis, which can also be referred to as threat modelling (see Section 2.1.1 bullet 4), is used to prevent and detect design flaws. Designers and architects provide a high-level view of the target system and documentation for assumptions, and identify possible attacks. Through architectural risk analysis, security analysts uncover and rank architectural and design flaws so mitigation can begin. For example, risk analysis may identify a possible attack type, such as the ability for data to be intercepted and read. This identification would prompt the designers to look at all their code's traffics flows to see if interception was a worry, and whether adequate protection (i.e. encryption) was in place. That review that the analysis prompted is what uncovers design flaws, such as sensitive data is transported in the clear.

   No system can be perfectly secure, so risk analysis must be used to prioritise security efforts and to link system-level concerns to probability and impact measures that matter to the business building the software. Risk exposure is computed by multiplying the probability of occurrence of an adverse event by the cost associated with that event [1599].

   McGraw proposes three basic steps for architectural risk analysis:

   - *Attack resistance analysis*. Attack resistance analysis uses a checklist/systematic approach of considering each system component relative to *known* threats, as is done in Microsoft threat modelling discussed in Section 2.1.1 bullet 4. Information

[11] http://nvd.nist.gov

about known attacks and attack patterns are used during the analysis, identifying risks in the architecture and understanding the viability of known attacks. Threat modelling with the incorporation of STRIDE-based attacks, as discussed in Section 2.1.1 bullet 4, is an example process for performing attack resistance analysis.

- *Ambiguity analysis*. Ambiguity analysis is used to capture the creative activity required to discover *new* risks. Ambiguity analysis requires two or more experienced analysts who carry out separate analysis activities in parallel on the same system. Through unifying the understanding of multiple analysis, disagreements between the analysts can uncover ambiguity, inconsistency and new flaws.
- *Weakness analysis*. Weakness analysis is focused on understanding risk related to security issues in other third-party components (see Section 2.1.1 bullet 7). The idea is to understand the assumptions being made about third-party software and what will happen when those assumptions fail.

Risk identification, ranking, and mitigation is a continuous process through the software lifecycle, beginning with the requirement phase.

3. **Penetration Testing**.

   Penetration testing can be guided by the outcome of architectural risk analysis (See Section 2.1.2 bullet 2). For further discussion on penetration testing, see Section 2.1.1, bullet 11.

4. **Risk-based Security Testing**.

   Security testing must encompass two strategies: (1) testing of security functionality with standard functional testing techniques; and (2) risk-based testing based upon attack patterns and architectural risk analysis results (see Section 2.1.2 bullet 2), and abuse cases (see Section 2.1.2 bullet 5). For web applications, testing of security functionality can be guided by the OWASP Application Security Verfication Standard (ASVS) Project[12] open standard for testing application technical security controls. ASVS also provides developers with a list of requirements for secure development.

   Guiding tests with knowledge of the software architecture and construction, common attacks, and the attacker's mindset is extremely important. Using the results of architectural risk analysis, the tester can properly focus on areas of code where an attack is likely to succeed.

   The difference between risk-based testing and penetration testing is the level of the approach and the timing of the testing. Penetration testing is done when the software is complete and installed in an operational environment. Penetration tests are outside-in, black box tests. Risk-based security testing can begin before the software is complete and even pre-integration, including the use of white box unit tests and stubs. The two are similar in that they both should be guided by risk analysis, abuse cases and functional security requirements.

5. **Abuse Cases**.

   This touchpoint codifies 'thinking like an attacker'. Use cases describe the desired system's behaviour by benevolent actors. Abuse cases [1586] describe the system's behaviour when under attack by a malicious actor. To develop abuse cases, an analyst

[12] **https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home**

enumerates the types of malicious actors who would be motivated to attack the system. For each bad actor, the analyst creates one or more abuse case(s) for the functionality the bad actor desires from the system. The analyst then considers the interaction between the use cases and the abuse cases to fortify the system. Consider an automobile example. An actor is the driver of the car, and this actor has a use case 'drive the car'. A malicious actor is a car thief whose abuse case is 'steal the car'. This abuse case *threatens* the use case. To prevent the theft, a new use case 'lock the car' can be added to *mitigate* the abuse case and fortify the system.

Human error is responsible for a large number of breaches. System analysts should also consider actions by benevolent users, such as being the victim of a phishing attack, that result in a security breach. These actions can be considered misuse cases [1587] and should be analysed similarly to abuse cases, considering what use case the misuse case threatens and the fortification to the system to mitigate the misuse case.

The attacks and mitigations identified by the abuse and misuse case analysis can be used as input into the security requirements (Section 2.1.1 bullet 2.); penetration testing (Section 2.1.1 bullet 11); and risk-based security testing (Section 2.1.2 bullet 4).

6. **Security Requirements**.

   For further discussion on security requirements, see Section 2.1.1 bullet 2.

7. **Security Operations**.

   Network security can integrate with software security to enhance the security posture. Inevitably, attacks will happen, regardless of the applications of the other touchpoints. Understanding attacker behaviour and the software that enabled a successful attack is an essential defensive technique. Knowledge gained by understanding attacks can be fed back into the six other touchpoints.

The seven touchpoints are intended to be cycled through multiple times as the software product evolves. The touchpoints are also process agnostic, meaning that the practices can be included in any software development process.

17.2.1.3 SAFECode

The Software Assurance Forum for Excellence in Code (SAFECode)[13] is a non-profit, global, industry-led organisation dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. The SAFECode mission is to promote best practices for developing and delivering more secure and reliable software, hardware and services. The SAFECode organisation publishes the 'Fundamental practices for secure software development: Essential elements of a secure development lifecycle program' [1600] guideline to foster the industry-wide adoption of fundamental secure development practices. The fundamental practices deal with assurance – the ability of the software to withstand attacks that attempt to exploit design or implementation errors. The eight fundamental practices outlined in their guideline are described below:

1. **Application Security Control Definition**. SAFECode uses the term Application Security Controls (ASC) to refer to security requirements (see Section 2.1.1 bullet 2). Similarly,

[13] **https://safecode.org/**

NIST 800-53 [53] uses the phrase *security control* to refer to security functionality and security assurance requirements.

The inputs to ASC include the following: secure design principles (see Section 2.1.3 bullet 3); secure coding practices; legal and industry requirements with which the application needs to comply (such as HIPAA, PCI, GDPR, or SCADA); internal policies and standards; incidents and other feedback; threats and risk. The development of ASC begins before the design phase and continues throughout the lifecycle to provide clear and actionable controls and to be responsive to changing business requirements and the ever-evolving threat environment.

2. **Design**. Software must incorporate security features to comply with internal security practices and external laws or regulations. Additionally, the software must resist known threats based upon the operational environment. (see Section 2.1.1 bullet 5.) Threat modelling (see Section 2.1.1 bullet 4), architectural reviews, and design reviews can be used to identify and address design flaws before their implementation into source code.

   The system design should incorporate an encryption strategy (see Section 2.1.1 bullet 6) to protect sensitive data from unintended disclosure or alteration while the data are at rest or in transit.

   The system design should use a standardised approach to identity and access management to perform authentication and authorisation. The standardisation provides consistency between components and clear guidance on how to verify the presence of the proper controls. Authenticating the identity of a principal (be it a human user, other service or logical component) and verifying the authorisation to perform an action are foundational controls of the system. Several access control schemes have been developed to support authorisation: mandatory, discretionary, role-based or attribute-based. Each of these has benefits and drawbacks and should be chosen based upon project characteristics.

   Log files provide the evidence needed in forensic analysis when a breach occurs to mitigate repudiation threats. In a well-designed application, system and security log files provide the ability to understand an application's behaviour and how it is used at any moment, and to distinguish benevolent user behaviour from malicious user behaviour. Because logging affects the available system resources, the logging system should be designed to capture the critical information while not capturing excess data. Policies and controls need to be established around storing, tamper prevention and monitoring log files. OWASP provides valuable resources on designing and implementing logging[14][15].

3. **Secure Coding Practices**. Unintended code-level vulnerabilities are introduced by programmer mistakes. These types of mistakes can be prevented and detected through the use of coding standards; selecting the most appropriate (and safe) languages, frameworks and libraries, including the use of their associated security features (see Section 2.1.1 bullet 8); using automated analysis tools (see Section 2.1.1 bullets 9 and 10); and manually reviewing the code.

   Organisations provide standards and guidelines for secure coding, for example:

   (a) OWASP Secure Coding Practices, Quick Reference Guide [16]

[14] **https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html**
[15] **https://www.owasp.org/images/e/e0/OWASP_Logging_Guide.pdf**
[16] **https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf**

(b) Oracle Secure Coding Guidelines for Java SE [17]

(c) Software Engineering Institute (SEI) CERT Secure Coding Standards [18]

Special care must also be given to handling unanticipated errors in a controlled and graceful way through generic error handlers or exception handlers that log the events. If the generic handlers are invoked, the application should be considered to be in an unsafe state such that further execution is no longer considered trusted.

4. **Manage Security Risk Inherent in the Use of Third-Party Components**. See Section 2.1.1 bullet 7.
5. **Testing and Validation**. See Section 2.1.1 bullets 9-11 and Section 2.1.2 bullets 1, 3 and 4.
6. **Manage Security Findings**. The first five practices produce artifacts that contain or generate findings related to the security of the product (or lack thereof). The findings in these artifacts should be tracked and actions should be taken to remediate vulnerabilities, such as is laid out in the Common Criteria (see Section 4.3) flaw remediation procedure [1601]. Alternatively, the team may consciously accept the security risk when the risk is determined to be acceptable. Acceptance of risk must be tracked, including a severity rating; a remediation plan, an expiration or a re-review deadline; and the area for re-review/validation.

   Clear definitions of severity are important to ensure that all participants have and communicate with a consistent understanding of a security issue and its potential impact. A possible starting point is mapping to the severity levels, attributes, and thresholds used by the Common Vulnerability Scoring System (CVSS)[19] such as 10–8.5 is critical, 8.4–7.0 is high, etc. The severity levels are used to prioritise mitigations based upon their complexity of exploitation and impact on the properties of a system.
7. **Vulnerability Response and Disclosure**. Even with following a secure software lifecycle, no product can be 'perfectly secure' because of the constantly changing threat landscapes. Vulnerabilities will be exploited and the software will eventually be compromised. An organisation must develop a vulnerability response and disclosure process to help drive the resolution of externally discovered vulnerabilities and to keep all stakeholders informed of progress. ISO provides industry-proven standards[20] for vulnerability disclosure and handling. To prevent vulnerabilities from re-occurring in new or updated products, the team should perform a root cause analysis and feed the lessons learned into the secure software lifecycle practices. For further discussion, see Sections 2.1.1 bullet 12 and 2.1.2 bullet 7.
8. **Planning the Implementation and Deployment of Secure Development**. A healthy and mature secure development lifecycle includes the above seven practices but also an integration of these practices into the business process and the entire organisation, including program management, stakeholder management, deployment planning, metrics and indicators, and a plan for continuous improvement. The culture, expertise and skill level of the organisation needs to be considered when planning to deploy a secure software lifecycle. Based upon past history, the organisation may respond better to a corporate mandate, to a bottom-up groundswell approach or to a series of pilot programs.

[17] https://www.oracle.com/technetwork/java/seccodeguide-139067.html
[18] https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
[19] https://www.first.org/cvss/
[20] https://www.iso.org/standard/45170.html and https://www.iso.org/standard/53231.html

Training will be needed (see Section 2.1.1 bullet 1). The specification of the organisation's secure software lifecycle including the roles and responsibilities should be documented. Plans for compliance and process health should be made (see Section 17.4).

17.2.2 Comparing the Secure Software Lifecycle Models

In 2009, De Win et al. [1602] compared CLASP, Microsoft's originally-documented SDL [1572], and Touchpoints (see Section 2.1.2)for the purpose of providing guidance on their commonalities and the specificity of the approach, and making suggestions for improvement. The authors mapped the 153 possible activities of each lifecycle model into six software development phases: education and awareness; project inception; analysis and requirements; architectural and detailed design; implementation and testing; and release, deployment and support. The activities took the practices in Sections 2.1.1–2.1.3 into much finer granularity. The authors indicated whether each model includes each of the 153 activities and provides guidance on the strengths and weaknesses of each model. The authors found no clear comprehensive 'winner' among the models, so practitioners could consider using guidelines for the desired fine-grained practices from all the models.

Table 17.1 places the the practices of Sections 2.1.1–2.1.3 into the six software development phases used by De Win et al. [1602]. Similar to prior work [1602], the models demonstrate strengths and weaknesses in terms of guidance for the six software development phases. No model can be considered perfect for all contexts. Security experts can customize a model for their organizations considering the spread of practices for the six software development phases.

	Microsoft SDL	Touchpoints	SAFECode
Education and awareness	• Provide training		• Planning the implementation and deployment of secure development
Project inception	• Define metrics and compliance reporting • Define and use cryptography standards • Use approved tools		• Planning the implementation and deployment of secure development
Analysis and requirements	• Define security requirements • Perform threat modelling	• Abuse cases • Security requirements	• Application security control definition
Architectural and detailed design	• Establish design requirements	• Architectural Risk Analysis	• Design
Implementation and testing	• Perform static analysis security testing (SAST) • Perform dynamic analysis security testing (DAST) • Perform penetration testing • Define and use cryptography standards • Manage the risk of using third-party components	• Code review (tools) • Penetration testing • Risk-based security testing	• Secure coding practices • Manage security risk inherent in the use of third-party components • Testing and validation
Release, deployment, and support	• Establish a standard incident response process	• Security operations	• Vulnerability response and disclosure

Table 17.1: Comparing the Software Security Lifecycle Models

17.3 ADAPTATIONS OF THE SECURE SOFTWARE LIFECYCLE

[1600, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611]

The secure software lifecycle models discussed in Section 17.2.1 can be integrated with any software development model and are domain agnostic. In this section, information on six adaptations to secure software lifecycle is provided.

17.3.1 Agile Software Development and DevOps

Agile and continuous software development methodologies are highly iterative, with new functionality being provided to customers frequently - potentially as quickly as multiple times per day or as 'slowly' as every two to four weeks.

Agile software development methodologies can be functional requirement-centric, with the functionality being expressed as *user stories*. SAFECode [1603] provides practical software security guidance to agile practitioners. This guidance includes a set of 36 recommended security-focused stories that can be handled similarly to the functionality-focused user stories. These stories are based upon common security issues such as those listed in the OWASP Top 10[21] Most Critical Web Application Security Risks. The stories are mapped to Common Weakness Enumerations (CWEs)[22] identifiers, as applicable. The security-focused stories are worded in a format similar to functionality stories (i.e., As a [stakeholder], I want to [new functionality] so that I can [goal]). For example, a security-focused story using this format is provided: *As Quality Assurance, I want to verify that all users have access to the specific resources they require which they are authorised to use*, that is mapped to CWE-862 and CWE-863. The security-focused stories are further broken down into manageable and concrete tasks that are owned by team roles, including architects, developers, testers and security experts, and are mapped to SAFECode Fundamental Practices [1600]. Finally, 17 operational security tasks were specified by SAFECode. These tasks are not directly tied to stories but are handled as continuous maintenance work (such as, *Continuously verify coverage of static code analysis tools*) or as an item requiring special attention (such as, *Configure bug tracking to track security vulnerabilities*).

With a DevOps approach to developing software, development and operations are tightly integrated to enable fast and continuous delivery of value to end users. Microsoft has published a DevOps secure software lifecycle model [1604] that includes activities for operations engineers to provide fast and early feedback to the team to build security into DevOps processes. The Secure DevOps model contains eight practices, including eight of the 12 practices in the Microsoft Security Development Lifecycle discussed in Section 2.1.1:

1. **Provide Training**. The training, as outlined in Section 2.1.1 bullet 1, must include the operations engineers. The training should encompass attack vectors made available through the deployment pipeline.
2. **Define Requirements**. See Section 2.1.1 bullet 2.

[21] **https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project**

[22] **https://cwe.mitre.org/**; CWE is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

3. **Define Metrics and Compliance Reporting**. See Section 2.1.1 bullet 3.
4. **Use Software Composition Analysis (SCA) and Governance**. When selecting both commercial and open-source third-party components, the team should understand the impact that a vulnerability in the component could have on the overall security of the system and consider performing a more thorough evaluation before using them. Software Composition Analysis (SCA) tools, such as WhiteSource[23] can assist with licensing exposure, provide an accurate inventory of components, and report any vulnerabilities with referenced components. See also Section 2.1.1 bullet 7.
5. **Perform Threat Modelling**. See Section 2.1.1 bullet 4. Threat modelling may be perceived as slowing down the rapid DevOps pace. However, products that are deployed rapidly under a DevOps deployment process should have a defined overall architecture within which the DevOps process makes changes and adds features. That architecture should be threat modeled, and when the team needs to change the architecture the threat model should also be updated. New features that do not have an architectural impact represent a null change to the threat model.
6. **Use Tools and Automation**. See Section 2.1.1 bullets 8, 9 and 10. The team should carefully select tools that can be integrated into the engineer's Integrated Development Environment (IDE) and workflow such that they cause minimal disruption. The goal of using these tools is to detect defects and vulnerabilities and not to overload engineers with too many tools or alien processes outside of their everyday engineering experience. The tools used as part of a secure DevOps workflow should adhere to the following principles:
   (a) Tools must be integrated into the Continuous Integration/Continuous Delivery (CI/CD) pipeline.
   (b) Tools must not require security expertise beyond what is imparted by the training.
   (c) Tools must avoid a high false-positive rate of reporting issues.
7. **Keep Credentials Safe**. Scanning for credentials and other sensitive content in source files is necessary during pre-commit to reduce the risk of propagating the sensitive information through the CI/CD process, such as through Infrastructure as Code or other deployment scripts. Tools, such as CredScan[24], can identify credential leaks, such as those in source code and configuration files. Some commonly found types of credentials include default passwords, hard-coded passwords, SQL connection strings and Certificates with private keys.
8. **Use Continuous Learning and Monitoring**. Rapidly-deployed systems often monitor the health of applications, infrastructure and networks through instrumentation to ensure the systems are behaving 'normally'. This monitoring can also help uncover security and performance issues which are departures from normal behaviour. Monitoring is also an essential part of supporting a defense-in-depth strategy and can reduce an organisation's Mean Time To Identify (MTTI) and Mean Time To Contain (MTTC) an attack.

[23] https://www.whitesourcesoftware.com/
[24] https://secdevtools.azurewebsites.net/helpcredscan.html

17.3.2 Mobile

Security concerns for mobile apps differ from traditional desktop software in some important ways, including local data storage, inter-app communication, proper usage of cryptographic APIs and secure network communication. The OWASP Mobile Security Project [1605] is a resource for developers and security teams to build and maintain secure mobile applications; see also the Web & Mobile Security Knowledge Area (Chapter 16).

Four resources are provided to aid in the secure software lifecycle of mobile applications:

1. **OWASP Mobile Application Security Verification Standard (MASVS) Security Requirements and Verification**. The MASVS defines a mobile app security model and lists generic security requirements for mobile apps. The MASVS can be used by architects, developers, testers, security professionals, and consumers to define and understand the qualities of a secure mobile app.
2. **Mobile Security Testing Guide (MSTG)**. The guide[25] is a comprehensive manual for mobile application security testing and reverse engineering for iOS and Android mobile security testers. The guide provides the following content:
   (a) *A general mobile application testing guide* that contains a mobile app security testing methodology and general vulnerability analysis techniques as they apply to mobile app security. The guide also contains additional technical test cases that are operating system independent, such as authentication and session management, network communications, and cryptography.
   (b) *Operating system-dependent testing guides* for mobile security testing on the Android and iOS platforms, including security basics; security test cases; reverse engineering techniques and prevention; and tampering techniques and prevention.
   (c) *Detailed test cases* that map to the requirements in the MASVS.
3. **Mobile App Security Checklist**. The checklist[26] is used for security assessments and contains links to the MSTG test case for each requirement.
4. **Mobile Threat Model**. The threat model [1606] provides a checklist of items that should be documented, reviewed and discussed when developing a mobile application. Five areas are considered in the threat model:
   (a) **Mobile Application Architecture**. The mobile application architecture describes device-specific features used by the application, wireless transmission protocols, data transmission medium, interaction with hardware components and other applications. The attack surface can be assessed through a mapping to the architecture.
   (b) **Mobile Data**. This section of the threat model defines the data the application stores, transmits and receives. The data flow diagrams should be reviewed to determine exactly how data are handled and managed by the application.
   (c) **Threat Agent Identification**. The threat agents are enumerated, including humans and automated programs.
   (d) **Methods of Attack**. The most common attacks utilised by threat agents are defined so that controls can be developed to mitigate attacks.

[25] https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide
[26] https://github.com/OWASP/owasp-mstg/tree/master/Checklists

(e) **Controls**. The controls to mitigate attacks are defined.

17.3.3 Cloud Computing

The emergence of cloud computing bring unique security risks and challenges. In conjunction with the Cloud Security Alliance (CSA)[27], SAFECode has provided a 'Practices for Secure Development of Cloud Applications' [1607] guideline as a supplement to the 'Fundamental Practices for Secure Software Development' guideline [1600] discussed in Section 17.2.1.3 - see also the Distributed Systems Security Knowledge Area (Chapter 12). The Cloud guideline provides additional secure development recommendations to address six threats unique to cloud computing and to identify specific security design and implementation practices in the context of these threats. These threats and associated practices are provided:

1. **Threat: Multitenancy**. Multitenancy allows multiple consumers or tenants to maintain a presence in a cloud service provider's environment, but in a manner where the computations, processes, and data (both at rest and in transit) of one tenant are isolated from and inaccessible to another tenant. Practices:

   (a) Model the application's interfaces in threat models. Ensure that the multitenancy threats, such as information disclosure and privilege elevation are modeled for each of these interfaces, and ensure that these threats are mitigated in the application code and/or configuration settings.

   (b) Use a 'separate schema' database design and tables for each tenant when building multitenant applications rather than relying on a 'TenantID' column in each table.

   (c) When developing applications that leverage a cloud service provider's Platform as a Service (PaaS) services, ensure common services are designed and deployed in a way that ensures that the tenant segregation is maintained.

2. **Tokenisation of Sensitive Data**. An organisation may not wish to generate and store intellectual property in a cloud environment not under its control. Tokenisation is a method of removing sensitive data from systems where they do not need to exist or disassociating the data from the context or the identity that makes them sensitive. The sensitive data are replaced with a token for those data. The token is later used to rejoin the sensitive data with other data in the cloud system. The sensitive data are encrypted and secured within an organisation's central system which can be protected with multiple layers of protection and appropriate redundancy for disaster recovery and business continuity. Practices:

   (a) When designing a cloud application, determine if the application needs to process sensitive data and if so, identify any organisational, government, or industry regulations that pertain to that type of sensitive data and assess their impact on the application design.

   (b) Consider implementing tokenisation to reduce or eliminate the amount of sensitive data that need to be processed and or stored in cloud environments.

   (c) Consider data masking, an approach that can be used in pre-production test and debug systems in which a representative data set is used, but does not need to have access to actual sensitive data. This approach allows the test and debug systems to be exempt from sensitive data protection requirements.

[27] **https://cloudsecurityalliance.org/**

3. **Trusted Compute Pools**. Trusted Compute Pools are either physical or logical groupings of compute resources/systems in a data centre that share a security posture. These systems provide measured verification of the boot and runtime infrastructure for measured launch and trust verification. The measurements are stored in a trusted location on the system (referred to as a Trusted Platform Module (TPM)) and verification occurs when an agent, service or application requests the trust quote from the TPM. Practices:
   (a) Ensure the platform for developing cloud applications provides trust measurement capabilities and the APIs and services necessary for your applications to both request and verify the measurements of the infrastructure they are running on.
   (b) Verify the trust measurements as either part of the initialisation of your application or as a separate function prior to launching the application.
   (c) Audit the trust of the environments your applications run on using attestation services or native attestation features from your infrastructure provider.
4. **Data Encryption and Key Management**. Encryption is the most pervasive means of protecting sensitive data both at rest and in transit. When encryption is used, both providers and tenants must ensure that the associated cryptographic key materials are properly generated, managed and stored. Practices:
   (a) When developing an application for the cloud, determine if cryptographic and key management capabilities need to be directly implemented in the application or if the application can leverage cryptographic and key management capabilities provided by the PaaS environment.
   (b) Make sure that appropriate key management capabilities are integrated into the application to ensure continued access to data encryption keys, particularly as the data move across cloud boundaries, such as enterprise to cloud or public to private cloud.
5. **Authentication and Identity Management**. As an authentication consumer, the application may need to authenticate itself to the PaaS to access interfaces and services provided by the PaaS. As an authentication provider, the application may need to authenticate the users of the application itself. Practices:
   (a) Cloud application developers should implement the authentication methods and credentials required for accessing PaaS interfaces and services.
   (b) Cloud application developers need to implement appropriate authentication methods for their environments (private, hybrid or public).
   (c) When developing cloud applications to be used by enterprise users, developers should consider supporting Single Sign On (SSO) solutions.
6. **Shared-Domain Issues**. Several cloud providers offer domains that developers can use to store user content, or for staging and testing their cloud applications. As such, these domains, which may be used by multiple vendors, are considered 'shared domains' when running client-side script (such as JavaScript) and from reading data. Practices:
   (a) Ensure that your cloud applications are using custom domains whenever the cloud provider's architecture allows you to do so.
   (b) Review your source code for any references to shared domains.

The European Union Agency for Cybersecurity (ENISA) [1609] conducted an in-depth and independent analysis of the information security benefits and key security risks of cloud computing. The analysis reports that the massive concentrations of resources and data in the cloud present a more attractive target to attackers, but cloud-based defences can be more robust, scalable and cost-effective.

17.3.4 Internet of Things (IoT)

The Internet of Things (IoT) is utilised in almost every aspect of our daily life, including the extension into industrial sectors and applications (i.e. Industrial IOT (IIoT)). IoT and IIoT constitute an area of rapid growth that presents unique security challenges. [From this point forth we include IIoT when we use IoT.] Some of these are considered in the Cyber-Physical Systems Security Knowledge Area (Chapter 21), but we consider specifically software lifecycle issues here. Devices must be securely provisioned, connectivity between these devices and the cloud must be secure, and data in storage and in transit must be protected. However, the devices are small, cheap, resource-constrained. Building security into each device may not be considered to be cost effective by its manufacturer, depending upon the value of the device and the importance of the data it collects. An IoT-based solution often has a large number of geographically-distributed devices. As a result of these technical challenges, trust concerns exist with the IoT, most of which currently have no resolution and are in need of research. However, the US National Institute of Standards and Technology (NIST) [1608] recommends four practices for the development of secure IoT-based systems.

1. **Use of Radio-Frequency Identification (RFID) tags**. Sensors and their data may be tampered with, deleted, dropped, or transmitted insecurely. Counterfeit 'things' exist in the marketplace. Unique identifiers can mitigate this problem by attaching Radio-Frequency Identification (RFID) tags to devices. Readers activate a tag, causing the device to broadcast radio waves within a bandwidth reserved for RFID usage by governments internationally. The radio waves transmit identifiers or codes that reference unique information associated with the device.
2. **Not using or allowing the use of default passwords or credentials**. IoT devices are often not developed to require users and administrators to change default passwords during system set up. Additionally, devices often lack intuitive user interfaces for changing credentials. Recommended practices are to require passwords to be changed or to design in intuitive interfaces. Alternatively, manufacturers can randomise passwords per device rather than having a small number of default passwords.
3. **Use of the Manufacturer Usage Description (MUD) specification**. The Manufacturer Usage Description (MUD)[28] specification allows manufacturers to specify authorised and expected user traffic patterns to reduce the threat surface of an IoT device by restricting communications to/from the device to sources and destinations intended by the manufacturer.
4. **Development of a Secure Upgrade Process**. In non-IoT systems, updates are usually delivered via a secure process in which the computer can authenticate the source pushing the patches and feature and configuration updates. IoT manufacturers have, generally, not established such a secure upgrade process, which enables attackers to conduct a man-in-the-middle push of their own malicious updates to the devices. The IoT

[28] https://tools.ietf.org/id/draft-ietf-opsawg-mud-22.html

Firmware Update Architecture [29] provides guidance on implementing a secure firmware update architecture including hard rules defining how device manufacturers should operate.

Additionally, the UK Department for Digital, Culture, Media, and Sport have provided the *Code of Practice for consumer IoT security*[30]. Included in the code of practice are 13 guidelines for improving the security of consumer IoT products and associated services. Two of the guidelines overlap with NIST bullets 2 and 4 above. The full list of guidelines include the following: (1) No default passwords; (2) Implement a vulnerability disclosure policy; (3) Keep software updated; (4) Securely store credentials and security-sensitive data; (5) Communicate securely (i.e. use encryption for sensitive data); (6) Minimise exposed attack surfaces; (7) Ensure software integrity (e.g. use of a secure boot); (8) Ensure that personal data is protected (i.e. in accordance with GDPR); (9) Make systems resilient to outages; (10) Monitor system telemetry data; (11) Make it easy for consumers to delete personal data; (12) Make installation and maintenance of devices easy; and (13) Validate input data. Finally, Microsoft has provided an Internet of Things security architecture.[31]

17.3.5 Road Vehicles

A hacker that compromises a connected road vehicle's braking or steering system could cause a passenger or driver to lose their lives. Attacks such as these have been demonstrated, beginning with the takeover of a Ford Escape and a Toyota Prius by white-hat hackers Charlie Miller and Chris Valasek in 2013[32]. Connected commercial vehicles are part of the critical infrastructure in complex global supply chains. In 2018, the number of reported attacks on connected vehicles shot up six times more than the number just three years earlier [1610], due to both the increase in connected vehicles and their increased attractiveness as a target of attackers [1611]. Broader issues with Cyber-Physical Systems are addressed in the Cyber-Physical Systems Security Knowledge Area (Chapter 21).

The US National Highway Traffic Safety Administration (HTSA) defines road vehicle cyber security as the protection of automotive electronic systems, communication networks, control algorithms, software, users and underlying data from malicious attacks, damage, unauthorised access or manipulation[33]. The HTSA provides four guidelines for the automotive industry for consideration in their secure software development lifecycle:

1. The team should follow a secure product development process based on a systems-engineering approach with the goal of designing systems free of unreasonable safety risks including those from potential cyber security threats and vulnerabilities.
2. The automotive industry should have a documented process for responding to incidents, vulnerabilities and exploits. This process should cover impact assessment, containment, recovery and remediation actions, the associated testing, and should include the creation of roles and responsibilities for doing so. The industry should also establish metrics to periodically assess the effectiveness of their response process.

[29] https://tools.ietf.org/id/draft-moran-suit-architecture-02.html
[30] https://www.gov.uk/government/publications/code-of-practice-for-consumer-iot-security/code-of-practice-for-consumer-iot-security
[31] https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-architecture
[32] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
[33] https://www.nhtsa.gov/crash-avoidance/automotive-cybersecurity#automotive-cybersecurity-overview

3. The automotive industry should document the details related to their cyber security process, including the results of risk assessment, penetration testing and organisations decisions related to cyber security. Essential documents, such as cyber security requirements, should follow a robust version control protocol.
4. These security requirements should be incorporated into the product's security requirements, as laid out in Section 2.1.1 bullet 2, Section 2.1.2 bullet 6, and Section 2.1.3 bullet 1.:
   (a) Limit developer/debugging access to production devices, such as through an open debugging port or through a serial console.
   (b) Keys (e.g., cryptographic) and passwords which can provide an unauthorised, elevated level of access to vehicle computing platforms should be protected from disclosure. Keys should not provide access to multiple vehicles.
   (c) Diagnostic features should be limited to a specific mode of vehicle operation which accomplishes the intended purpose of the associated feature. For example, a diagnostic operation which may disable a vehicle's individual brakes could be restricted to operating only at low speeds or not disabling all the brakes at the same time.
   (d) Encryption should be considered as a useful tool in preventing the unauthorised recovery and analysis of firmware.
   (e) Limit the ability to modify firmware and/or employ signing techniques to make it more challenging for malware to be installed on vehicles.
   (f) The use of network servers on vehicle ECUs should be limited to essential functionality, and services over these ports should be protected to prevent use by unauthorised parties.
   (g) Logical and physical isolation techniques should be used to separate processors, vehicle networks, and external connections as appropriate to limit and control pathways from external threat vectors to cyber-physical features of vehicles.
   (h) Sending safety signals as messages on common data buses should be avoided, but when used should employ a message authentication scheme to limit the possibility of message spoofing.
   (i) An immutable log of events sufficient to enable forensic analysis should be maintained and periodically scrutinised by qualified maintenance personnel to detect trends of cyber-attack.
   (j) Encryption methods should be employed in any IP-based operational communication between external servers and the vehicle, and should not accept invalid certificates.
   (k) Plan for and design-in features that could allow for changes in network routing rules to be quickly propagated and applied to one, a subset or all vehicles

The International Organization for Standardization (ISO)[34] and the Society for Automotive Engineering (SAE) International [35] are jointly developing an international Standard, *ISO 21434*

[34] **https://www.iso.org/standard/70918.html**
[35] **www.sae.org**

Road vehicles - cyber security engineering[36]. The standard will specify minimum requirements on security engineering processes and activities, and will define criteria for assessment. Explicitly, the goal is to provide a structured process to ensure cyber security is designed in upfront and integrated throughout the lifecycle process for both hardware and software.

The adoption of a secure software lifecycle in the automotive industry may be driven by legislation, such as through the US SPY Car Act[37] or China and Germany's Intelligent and Connected Vehicles (ICVs) initiative[38].

17.3.6 ECommerce/Payment Card Industry

The ability to steal large quantities of money makes the Payment Card Industry (PCI) an especially attractive target for attackers. In response, the PCI created the Security Standards Council, a global forum for the ongoing development, enhancement, storage, dissemination, and implementation of security standards for account data protection. The Security Standards Council established the Data Security Standard (PCI DSS), which must be upheld by any organisations that handle payment cards, including debit and credit cards. PCI DSS contains 12 requirements[39] that are a set of security controls that businesses are required to implement to protect credit card data. These specific requirements are incorporated into the product's security requirements, as laid out in Section 2.1.1 bullet 2, Section 2.1.2 bullet 6, and Section 2.1.3 bullet 1. The 12 requirements are as follows:

1. Install and maintain a firewall configuration to protect cardholder data.
2. Do not use vendor-supplied defaults for system passwords and other security parameters.
3. Protect stored cardholder data.
4. Encrypt transmission of cardholder data across open, public networks.
5. Use and regularly update antivirus software.
6. Develop and maintain secure systems and applications, including detecting and mitigating vulnerabilities and applying mitigating controls.
7. Restrict access to cardholder data by business need-to-know.
8. Assign a unique ID to each person with computer access.
9. Restrict physical access to cardholder data.
10. Track and monitor all access to network resources and cardholder data.
11. Regularly test security systems and processes.
12. Maintain a policy that addresses information security.

[36] **https://www.iso.org/standard/70918.html**
[37] **https://www.congress.gov/bill/115th-congress/senate-bill/680**
[38] **http://icv.sustainabletransport.org/**
[39] **https://searchsecurity.techtarget.com/definition/PCI-DSS-12-requirements**

17.4 ASSESSING THE SECURE SOFTWARE LIFECYCLE

[1612, 1613]

Organisations may wish to or be required to assess the maturity of their secure development lifecycle. Three assessment approaches are described in this section.

17.4.1 SAMM

The Software Assurance Maturity Model (SAMM)[40] is an open framework to help organisations formulate and implement a strategy for software security that is tailored to the specific risks facing the organisation. Resources are provided for the SAMM to enable an organisation to do the following:

1. Define and measure security-related activities within an organisation.
2. Evaluate their existing software security practices.
3. Build a balanced software security program in well-defined iterations.
4. Demonstrate improvements in a security assurance program.

Because each organisation utilises its own secure software process (i.e., its own unique combination of the practices laid out in Sections 2 and 3), the SAMM provides a framework to describe software security initiatives in a common way. The SAMM designers enumerated activities executed by organisations in support of their software security efforts. Some example activities include: build and maintain abuse case models per project; specify security requirements based upon known risks; and identify the software attack surface. These activities are categorised into one of 12 security practices. The 12 security practices are further grouped into one of four business functions. The business functions and security practices are as follows:

1. Business Function: Governance
   (a) Strategy and metrics
   (b) Policy and compliance
   (c) Education and guidance
2. Business Function: Construction
   (a) Threat assessment
   (b) Security requirements
   (c) Secure architecture
3. Business Function: Verification
   (a) Design review
   (b) Code review
   (c) Security testing

[40] **https://www.opensamm.org/** and **https://www.owasp.org/images/6/6f/SAMM_Core_V1-5_FINAL.pdf**

4. Business Function: Deployment
    (a) Vulnerability management
    (b) Environment hardening
    (c) Operational enablement

The SAMM assessments are conducted through self-assessments or by a consultant chosen by the organisation. Spreadsheets are provide by SAMM for scoring the assessment, providing information for the organisation on their current maturity level:

- 0: Implicit starting point representing the activities in the Practice being unfulfilled.
- 1: Initial understanding and ad hoc provision of the Security Practice.
- 2: Increase efficiency and/or effectiveness of the Security Practice.
- 3: Comprehensive mastery of the Security Practice at scale.

Assessments may be conducted periodically to measure improvements in an organisation's security assurance program.

17.4.2 BSIMM

Gary McGraw, Sammy Migues, and Brian Chess desired to create a descriptive model of the state-of-the-practice in secure software development lifecycle. As a result, they forked an early version of SAMM (see Section 4.1) to create the original structure of the Building Security In Maturity Model (BSIMM) [1612, 1613] in 2009. Since that time, the BSIMM has been used to structure a multi-year empirical study of the current state of software security initiatives in industry.

Because each organisation utilises its own secure software process (i.e., its own unique combination of the practices laid out in Sections 2 and 3), the BSIMM provides a framework to describe software security initiatives in a common way. Based upon their observations, the BSIMM designers enumerated 113 activities executed by organisations in support of their software security efforts. Some example activities include: build and publish security features; use automated tools along with a manual review; and integrate black-box security tools into the quality assurance process. Each activity is associated with a maturity level and is categorised into one of 12 practices. The 12 practices are further grouped into one of four domains. The domains and practices are as follows:

1. Domain: Governance
    (a) Strategy and metrics
    (b) Compliance and policy
    (c) Training
2. Domain: Intelligence
    (a) Attack models
    (b) Security features and design
    (c) Standards and requirements

3. Domain: Secure software development lifecycle touchpoints
   (a) Architecture analysis
   (b) Code review
   (c) Security testing
4. Domain: Deployment
   (a) Penetration testing
   (b) Software environment
   (c) Configuration management and vulnerability management

BSIMM assessments are conducted through in-person interviews by software security professionals at Cigital (now Synopsys) with security leaders in a firm. Via the interviews, the firm obtains a scorecard on which of the 113 software security activities the firm uses. After the firm completes the interviews, they are provided information comparing themselves with the other organisations that have been assessed. BSIMM assessments have been conducted since 2008. Annually, the overall results of the assessments from all firms are published, resulting in the BSIMM1 through BSIMM9 reports. Since the BSIMM study began in 2008, 167 firms have participated in BSIMM assessment, sometimes multiple times, comprising 389 distinct measurements. To ensure the continued relevance of the data reported, the BSIMM9 report excluded measurements older than 42 months and reported on 320 distinct measurements collected from 120 firms.

17.4.3 The Common Criteria

The purpose of this Common Criteria (CC)[41] is to provide a vehicle for international recognition of a secure information technology (IT) *product* (where the SAMM and BSIMM were assessments of a development process). The objective of the CC is for IT products that have earned a CC certificate from an authorised Certification/Validation Body (CB) to be procured or used with no need for further evaluation. The Common Criteria seek to provide grounds for confidence in the reliability of the judgments on which the original certificate was based by requiring that a CB issuing Common Criteria certificates should meet high and consistent standards. A developer of a new product range may provide guidelines for the secure development and configuration of that product. This guideline can be submitted as a Protection Profile (the pattern for similar products that follow on). Any other developer can add to or change this guideline. Products that earn certification in this product range use the protection profile as the delta against which they build.

Based upon the assessment of the CB, a product receives an Evaluation Assurance Level (EAL). A product or system must meet specific assurance requirements to achieve a particular EAL. Requirements involve design documentation, analysis and functional or penetration testing. The highest level provides the highest guarantee that the system's principal security features are reliably applied. The EAL indicates to what extent the product or system was tested:

- **EAL 1: Functionally tested**. Applies when security threats are not viewed as serious. The evaluation provides evidence that the system functions in a manner consistent with its

[41] https://www.commoncriteriaportal.org/ccra/index.cfm

documentation and that it provides useful protection against identified threats.

- **EAL 2: Structurally tested**. Applies when stakeholders require low-to-moderate independently-assured security but the complete development record is not readily available, such as with securing a legacy system.
- **EAL 3: Methodically tested and checked**. Applies when stakeholders require a moderate level of independently-assured security and a thorough investigation of the system and its development, without substantial re-engineering.
- **EAL 4: Methodically designed, tested and reviewed**. Applies when stakeholders require moderate-to-high independently-assured security in commodity products and are prepared to incur additional security-specific engineering costs.
- **EAL 5: Semi-formally designed and tested**. Applies when stakeholders require high, independently-assured security in a planned development and require a rigorous development approach that does not incur unreasonable costs from specialist security engineering techniques.
- **EAL 6: Semi-formally verified design and tested**. Applies when developing systems in high-risk situations where the value of the protected assets justifies additional costs.
- **EAL 7: Formally verified design and tested**. Applies when developing systems in extremely high-risk situations and when the high value of the assets justifies the higher costs.

The CC provides a set of security functional and security assurance requirements. These requirements, as appropriate, are incorporated into the product's security requirements, as laid out in Section 2.1.1 bullet 2, Section 2.1.2 bullet 6, and Section 2.1.3 bullet 1.

17.5 ADOPTING A SECURE SOFTWARE LIFECYCLE

[1612, 1613, 1614]

This knowledge area has provided a myriad of possible practices an organisation can include in its secure software lifecycle. Some of these practices, such as those discussed in Section 2, potentially apply to any product. Other practices are domain specific, such as those discussed in Section 3.

Organisations adopting new practices often like to learn from and adopt practices that are used by organisations similar to themselves [1614]. When choosing which security practices to include in a secure software lifecycle, organisations can consider looking at the latest BSIMM [1612, 1613] results which provide updated information on the adoption of practices in the industry.

DISCUSSION

[1615]

This chapter has provided an overview of of three prominent and prescriptive secure software lifecycle processes and six adaptations of these processes that can be applied in a specified domain. However, the cybersecurity landscape in terms of threats, vulnerabilities, tools, and practices is ever evolving. For example, a practice has has not be been mentioned in any of these nine processes is the use of a bug bounty program for the identification and resolution of vulnerabilities. With a bug bounty program, organisations compensate individuals and/or researchers for finding and reporting vulnerabilities. These individuals are external to the organisation producing the software and may work independently or through a bug bounty organisation, such as HackerOne[42].

While the majority of this knowledge area focuses on technical practices, the successful adoption of these practices involves organisational and cultural changes in an organisation. The organisation, starting from executive leadership, must support the extra training, resources, and steps needed to use a secure development lifecycle. Additionally, every developer must uphold his or her responsibility to take part in such a process.

A team and an organisation need to choose the appropriate software security practices to develop a customised secure software lifecycle based upon team and technology characteristics and upon the security risk of the product.

While this chapter has provided practices for developing secure products, information insecurity is often due to economic disincentives [1615] which drives software organizations to choose the rapid deployment and release of functionality over the production of secure products. As a result, increasingly governments and industry groups are imposing cyber security standards on organisations as a matter of legal compliance or as a condition for being considered as a vendor. Compliance requirements may lead to faster adoption of a secure development lifecycle. However, this compliance-driven adoption may divert efforts away from the real security issues by driving an over-focus on compliance requirements rather than on the pragmatic prevention and detection of the most risky security concerns.

[42] **https://www.hackerone.com**

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[1572]	[1469]	[1577]	[1600]
17.1 Motivation	c1	c1	c1	
17.2 Prescriptive Secure Software Lifecycle Processes				
17.2.1 Secure Software Lifecycle Processes	c2	c2	c2	c2
17.2.2 Comparing the Secure Software Lifecycle Models				
17.3 Adaptations of the Secure Software Lifecycle				
17.3.1 Agile Software Development and DevOps				c3
17.3.2 Mobile				
17.3.3 Cloud Computing				
17.3.4 Internet of Things (IoT)				
17.3.5 Road Vehicles				
17.3.6 ECommerce/Payment Card Industry				
17.4 Assessing the Secure Software Lifecycle				
17.5 Adopting a Secure Software Lifecycle				

FURTHER READING

Building Secure Software: How to Avoid Security Problems the Right Way [1469]

This book introduces the term software security as an engineering discipline for building security into a product. This book provides essential lessons and expert techniques for security professionals who understand the role of software in security problems and for software developers who want to build secure code. The book also discusses risk assessment, developing security tests, and plugging security holes before software is shipped.

Writing Secure Code, Second Edition. [1577]

The first edition of this book was internally published in Microsoft and was required reading for all members of the Windows team during the Windows Security Push. The second edition was made publicly available in the 2003 book and provides secure coding techniques to prevent vulnerabilities, to detect design flaws and implementation bugs, and to improve test code and documentation.

Software Security: Building Security In [1578]

This book discusses seven software securing best practices, called touchpoints. It also provides information on software security fundamentals and contexts for a software security program in an enterprise.

The Security Development Lifecycle (Original Book) [1572]

This seminal book provides the foundation for the other processes laid out in this knowledge area, and was customised over the years by other organisations, such as Cisco [43]. The book lays out 13 stages for integrating practices into a software development lifecycle such that the product is more secure. This book is out of print, but is avaialble as a free download[44].

The Security Development Lifecycle (Current Microsoft Resources) [1579]

The Microsoft SDL are practices that are used internally to build secure products and services, and address security compliance requirements by introducing security practices throughout every phase of the development process. This webpage is a continuously-updated version of the seminal book [1572] based on Microsoft's growing experience with new scenarios such as the cloud, the Internet of Things (IoT) and Artificial Intelligence (AI).

Software Security Engineering: A Guide for Project Managers [1592]

This book is a management guide for selecting from among sound software development practices that have been shown to increase the security and dependability of a software product, both during development and subsequently during its operation. Additionally, this book discusses governance and the need for a dynamic risk management approach for identifying priorities throughout the product lifecycle.

Cyber Security Engineering: A Practical Approach for Systems and Software Assurance [1616]

This book provides a tutorial on the best practices for building software systems that exhibit superior operational security, and for considering security throughout your full system development and acquisition lifecycles. This book provides seven core principles of software assurance, and shows how to apply them coherently and systematically. This book addresses important topics, including the use of standards, engineering security requirements for acquiring COTS software, applying DevOps, analysing malware to anticipate future vulnerabilities, and planning ongoing improvements.

[43] https://www.cisco.com/c/en/us/about/trust-center/technology-built-in-security.html#~stickynav=2
[44] https://blogs.msdn.microsoft.com/microsoft_press/2016/04/19/free-ebook-the-security-development-lifecycle/

SAFECode's Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program, Third Edition [1600]

Eight practices for secure development are provided based upon the experiences of member companies of the SAFECode organisation.

OWASP's Secure Software Development Lifecycle Project (S-SDLC) [1580]

Based upon a committee of industry participants, the Secure-Software Development Lifecycle Project (S-SDLC) defines a standard Secure Software Development Life Cycle and provides resources to help developers know what should be considered or best practices at each phase of a development lifecycle (e.g., Design Phase/Coding Phase/Maintain Phase/etc.) The committee of industry participants are members of the Open Web Application Security Project (OWASP)[45], an international not-for-profit organisation focused on improving the security of web application software. The earliest secure software lifecycle contributions from OWASP were referred to as the Comprehensive, Lightweight Application Security Process (CLASP).

Security controls

Government and standards organizations have provided security controls to be integrated in a secure software or systems lifecyle:

1. The Trustworthy Software Foundation [46] provides the the Trustworthy Software Framework (TSFr) [47] a collection of good practice, existing guidance and relevant standards across the five main facets of trustworthiness: Safety; Reliability; Availability; Resilience; and Security. The purpose of the TSFr is to provide a minimum set of controls such that, when applied, all software (irrespective of implementation constraints) can be specified, realised and used in a trustworthy manner.
2. The US National Institute of Standards and Technology (NIST) has authored the Systems Security Engineering Cyber Resiliency Considerations for the Engineering [1617] framework (NIST SP 800-160). This Framework provides resources on cybersecurity Knowledge, Skills and Abilitiess (KSAs), and tasks for a number of work roles for achieving the identified cyber resiliency outcomes based on a systems engineering perspective on system life cycle processes.
3. The Software Engineering Institute (SEI) has collaborated with professional organisations, industry partners and institutions of higher learning to develop freely-available curricula and educational materials. Included in these materials are resources for a software assurance program[48] to train professionals to build security and correct functionality into software and systems.
4. The UK National Cyber Security Centre (NCSC)[49] provide resources for secure software development:

[45] https://www.owasp.org/
[46] https://tsfdn.org
[47] https://tsfdn.org/ts-framework/
[48] https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/index.cfm
[49] https://www.ncsc.gov.uk/

(a) Application development[50]: recommendations for the secure development, procurement, and deployment of generic and platform-specific applications.

(b) Secure development and deployment guidance[51]: more recommendations for the secure development, procurement, and deployment of generic and platform-specific applications.

(c) The leaky pipe of secure coding[52]: a discussion of how security can be woven more seamlessly into the development process, particularly by developers who are not security experts.

Training materials

Training materials are freely-available on the Internet. Some sites include the following:

1. The Trustworthy Software Foundation provides a resource library [53] of awareness materials and guidance targeted for those who teach trustworthy software principles, those who seek to learn about Trustworthy Software and those who want to ensure that the software they use is trustworthy. The resources available include a mixture of documents, videos, animations and case studies.
2. The US National Institute of Standards and Technology (NIST) has created the NICE Cyber security Workforce Framework [1618]. This Framework provides resources on cyber security Knowledge, Skills and Abilitiess (KSAs), and tasks for a number of work roles.
3. The Software Engineering Institute (SEI) has collaborated with professional organisations, industry partners and institutions of higher learning to develop freely-available curricula and educational materials. Included in these materials are resources for a software assurance program[54] to train professionals to build security and correct functionality into software and systems.
4. SAFECode offers free software security training courses delivered via on-demand webcasts[55].

[50] https://www.ncsc.gov.uk/collection/application-development
[51] https://www.ncsc.gov.uk/collection/developers-collection
[52] https://www.ncsc.gov.uk/blog-post/leaky-pipe-secure-coding
[53] https://tsfdn.org/resource-library/
[54] https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/index.cfm
[55] https://safecode.org/training/

V Infrastructure Security

Chapter 18
Applied Cryptography

Kenneth G. Paterson | ETH Zürich

INTRODUCTION

This document provides a broad introduction to the field of cryptography, focusing on applied aspects of the subject. It complements the Cryptography Knowledge Area (Chapter 10) which focuses on formal aspects of cryptography (including definitions and proofs) and on describing the core cryptographic primitives. That said, formal aspects are highly relevant when considering applied cryptography. As we shall see, they are increasingly important when it comes to providing security assurance for real-world deployments of cryptography.

The overall presentation assumes a basic knowledge of either first-year undergraduate mathematics, or that found in a discrete mathematics course of an undergraduate Computer Science degree. Good cryptography textbooks that cover the required material include [963, 1619, 1620].

We begin by informally laying out the key themes that we will explore in the remainder of the document.

Cryptography is a Mongrel

Cryptography draws from a number of fields including mathematics, theoretical computer science and software and hardware engineering. For example, the security of many public key algorithms depends on the hardness of mathematical problems which come from number theory, a venerable branch of mathematics. At the same time, to securely and efficiently implement such algorithms across a variety of computing platforms requires a solid understanding of the engineering aspects. To make these algorithms safely usable by practitioners, one should also draw on usability and Application Programming Interface (API) design. This broad base has several consequences. Firstly, almost no-one understands all aspects of the field perfectly (including the present author). Secondly this creates gaps — between theory and practice, between design and implementation (typically in the form of a cryptographic library, a collection of algorithm and protocol implementations in a specific programming language) and between implementations and their eventual use by potentially non-expert developers. Thirdly, these gaps lead to security vulnerabilities. In fact, it is rare that standardised, widely-deployed cryptographic algorithms directly fail when they are properly used. It is more common that cryptography fails for indirect reasons — through unintentional misuse of a library API by a developer, on account of bad key management, because of improper combination of basic cryptographic algorithms in a more complex system, or due to some form of side-channel leakage. All of these topics will be discussed in more detail.

Cryptography $\neq$ Encryption

In the popular imagination cryptography equates to encryption; a cryptographic mechanism providing confidentiality services. In reality, cryptography goes far beyond this to provide an underpinning technology for building security services more broadly. Thus, secure communications protocols like Transport Layer Security (TLS) rely on both encryption mechanisms (Authenticated Encryption, AE) and integrity mechanisms (e.g. digital signature schemes) to achieve their security goals. In fact, in its most recent incarnation (version 1.3), TLS relies exclusively on Diffie-Hellman key exchange to establish the keying material that it consumes, whereas earlier versions allowed the use of public key encryption for this task. We will discuss TLS more extensively in Section 18.5; here the point is that, already in the literally classic application of cryptography, encryption is only one of many techniques used.

Moreover, since the boom in public research in cryptography starting in the late 1970s, researchers have been incredibly fecund in inventing new types of cryptography to solve seemingly impossible tasks. Whilst many of these new cryptographic gadgets were initially of purely theoretical interest, the combination of Moore's law and the growth of technologies such as cloud computing has made some of them increasingly important in practice. Researchers have developed some of these primitives to the point where they are efficient enough to be used in large-scale applications. Some examples include the use of zero-knowledge proofs in anonymous cryptocurrencies, the use of Multi-Party Computation (MPC) techniques to enable computations on sensitive data in environments where parties are mutually untrusting, and the (to date, limited) use of Fully Homomorphic Encryption (FHE) for privacy-preserving machine learning.

Cryptography is Both Magical and Not Magical

Cryptography can seem magical in what it can achieve. Consider the millionaire's problem: two millionaires want to find out who is richer, without either telling the other how much they are worth. This seems impossible, but it can be done in a reasonably efficient manner and under mild assumptions. But there is no such thing as cryptographic "fairy dust" that can be sprinkled on a bad (i.e. insecure) system to make it good (i.e. secure). Rather, at the outset of a system design activity, cryptography should be thought of as being something that will work in concert with other security building blocks to build a secure system (or more pessimistically, a system in which certain risks have been mitigated). In this sense, cryptography makes systems stronger by making certain attack vectors infeasible or just uneconomic to attack. Consider again the secure communications protocol TLS. When configured properly, TLS offers end-to-end security for pairs of communicating devices, providing strong assurances concerning the confidentiality and integrity of messages (and more). However, it says nothing about the security of the endpoints where the messages are generated and stored. Moreover, TLS does not prevent traffic analysis attacks, based on analysing the number, size and direction of flow of TLS-encrypted messages. So the use of cryptography here reduces the "attack surface" of the communication system, but does not eliminate all possible attacks on the system, nor does it aim to do so.

Developing the systemic view further, it is unfortunate that cryptography is generally brittle and can fail spectacularly rather than gracefully. This can be because of a breakthrough in cryptanalysis, in the sense of breaking one of the core cryptographic algorithms in use. For example, there could be an unexpected public advance in algorithms for integer factorisation that renders our current choices of key-sizes for certain algorithms totally insecure. This seems unlikely — the last significant advance at an algorithmic level here was in the early 1990s with the invention of the Number Field Sieve. So more likely this is because the cryptography is provided in a way that makes it easy for non-experts to make mistakes, or the realisation of a new attack vector enabling an attacker to bypass the cryptographic mechanism in use.

It is also an unfortunate fact that, in general, cryptography is non-composable, in the sense that a system composed of cryptographic components that are individually secure (according to some suitable formal definitions for each component) might itself fail to be secure in the intended sense. A simple example arises in the context of so-called *generic composition* of symmetric encryption and MAC algorithms to build an overall encryption scheme that offers both confidentiality and integrity. Here, the "E&M" scheme obtained by encrypting the plaintext and, in parallel, applying a MAC to the plaintext, fails to offer even a basic level of confidentiality. This is because the MAC algorithm, being deterministic, will leak plaintext

equality across multiple encryptions. This example, while simple, is not artificial: the SSH protocol historically used such an E&M scheme and only avoided the security failure due to the inclusion of a per-message sequence number as part of the plaintext (this sequence number was also needed to achieve other security properties of the SSH secure channel). This example generalises, in the sense that even small and seemingly trivial details can have a large effect on security: in cryptography, every bit matters.

In view of the above observations, applied cryptography is properly concerned with a broader sweep of topics than just the low-level cryptographic algorithms. Of course these are still crucial and we will cover them briefly. However, applied cryptography is also about the integration of cryptography into systems and development processes, the thorny topic of key management and even the interaction of cryptography with social processes, practices and relations. We will touch on all of these aspects.

Cryptography is Political

Like many other technologies, cryptography can be used for good or ill. It is used by human rights campaigners to securely organise their protests using messaging apps like Telegram and Signal [1621]. It is used by individuals who wish to maintain their privacy against the incursions of tech companies. It enables whistle-blowers to securely communicate with journalists when disclosing documents establishing company or governmental wrong-doing (see Privacy & Online Rights Knowledge Area (Section 5.4)). But it can also be used by terrorists to plan attacks or by child-abusers to share illegal content. Meanwhile cryptocurrencies can be used by drug dealers to launder money [1622] and as a vehicle for extracting ransom payments.[1]

These examples are chosen to highlight that cryptography, historically the preserve of governments and their militaries, is now in everybody's hands — or more accurately, on everybody's phone. This is despite intensive, expensive efforts over decades on the part of governments to regulate the use of cryptography and the distribution of cryptographic technology through export controls. Indeed, such laws continue to exist, and violations of them can produce severe negative consequences so practitioners should be cautious to research applicable regulation (see Law & Regulation Knowledge Area (Section 3.11.3) for further discussion of this topic).

But the cryptographic genie has escaped the bottle and is not going back in. Indeed, cryptographic software of reasonable quality is now so widespread that attempts to prevent its use or to introduce government-mandated back-doors are rendered irrelevant for anyone with a modicum of skill. This is to say nothing as to whether it is even possible to securely engineer cryptographic systems that support exceptional access for a limited set of authorised parties, something which experts doubt, see for example [1624]. Broadly, these efforts at control and the reaction to them by individual researchers, as well as companies, are colloquially known as The Crypto Wars. Sometimes, these are enumerated, though it is arguable that the First Crypto War never ended, but simply took on another, less-intense, less-visible form, as became apparent from the Snowden revelations [1625].

[1] On the other hand, a 2019 RAND report [1623] concluded there is little evidence for use of cryptocurrencies by terrorist groups.

The Cryptographic Triumvirate

A helpful classification of cryptographic applications arises from considering what is happening to the data. The classical cryptographic applications relate to *data in transit*, i.e. secure communications. Cryptography can be applied to build secure data storage systems, in which case we talk about *data at rest*. In fact these two application domains are quite close in terms of the techniques they use. This is because, to a first-order approximation, one can regard a secure storage system as a kind of communications channel in time. Finally, in the era of cloud computing, outsourced storage and privacy-preserving computation, we have seen the emergence of cryptography for *data under computation*. This refers to a broad set of techniques enabling computations to be done on data that is encrypted — imagine outsourcing an entire database to an untrusted server in such a way that database operations (insert, delete, search queries, etc) can still be carried out without leaking anything about those operations — or the data itself — to the server. Keeping in mind this triumvirate — data in transit, data at rest, data under computation — can be useful when understanding what to expect in terms of the security, performance and maturity of systems using cryptography. In short, systems providing security for data in transit and data at rest are more mature, are performant at scale and tend to be standardised. By contrast, systems providing security for data under computation are largely in an emergent phase.

This classification focuses on data and therefore fails to capture some important applications of cryptography such as user authentication[2] and attestation.[3]

Organisation

Having laid out the landscape of Applied Cryptography, we now turn to a more detailed consideration of sub-topics. The next section is concerned with cryptographic algorithms and schemes — the building blocks of cryptography. It also discusses protocols, which typically combine multiple algorithms into a more complex system. In Section 18.2 we discuss implementation aspects of cryptography, addressing what happens when we try to turn a mathematical description of a cryptographic algorithm into running code. Cryptography simply translates the problem of securing data into that of securing and managing cryptographic keys, following Wheeler's aphorism that every problem in computer science can be solved by another level of indirection. We address the topic of key management in Section 18.3. Section 18.4 covers a selection of issues that may arise for non-expert consumers of cryptography, while Section 18.5 discusses a few core cryptographic applications as a means of showing how the different cryptographic threads come together in specific cases. Finally, Section 18.6 looks to the future of applied cryptography and conveys closing thoughts.

[2]Here, for example, FIDO is developing open specifications of interfaces for authenticating users to web-based applications and services using public key cryptography.

[3]This concept refers to methods by which a hardware platform can provide security guarantees to third parties about how it will execute code. It is addressed in the Hardware Security Knowledge Area (Chapter 20).

18.1 ALGORITHMS, SCHEMES AND PROTOCOLS

[963, 1619, 1620]

18.1.1 Basic Concepts

In this subsection, we provide a brief summary of some basic concepts in cryptography. A more detailed and formal introduction to this material can be found in Cryptography Knowledge Area (Chapter 10).

Cryptographic algorithms are at the core of cryptography. There are many different classes of algorithm and many examples within each class. Moreover, it is common to group algorithms together to form cryptographic *primitives* or *schemes*. For example, a *Public Key Encryption (PKE) scheme* consists of a collection of three algorithms: a *key generation* algorithm, an *encryption* algorithm and a corresponding *decryption* algorithm.

Unfortunately, there is no general agreement on the terminology and the meanings of the terms *algorithm*, *scheme*, *primitive* and even *protocol* overlap and are sometimes used interchangeably. We will reserve the term *algorithm* for an individual algorithm (in the computer science sense — a well-defined procedure with specific inputs and outputs, possibly randomised). We will use *scheme* to refer to a collection of algorithms providing some functionality (e.g. as above, a PKE scheme) and *protocol* for interactive systems in which two or more parties exchange messages.[4] Such protocols are usually built by combining different cryptographic schemes, themselves composed of multiple algorithms.

Most cryptographic algorithms are *keyed* (the main exception are hash functions). This means that they have a special input, called a key, which controls the operation of the algorithm. The manner in which the keying is performed leads to the fundamental distinction between symmetric and asymmetric schemes. In a symmetric scheme the same key is used for two operations (e.g. encryption and decryption) and the confidentiality of this key is paramount for the security of the data which it protects. In an asymmetric scheme, different keys are used for different purposes (e.g. in a PKE scheme, a public key is used for encryption and a corresponding private key is used for decryption, with the *key pair* of public and private keys being output by the PKE scheme's key generation algorithm). The usual requirement is that the private key remains confidential to the party who runs the key generation algorithm, while the public key (as the name suggests) can be made public and widely distributed.

We now turn to discussion of the most important (from the perspective of applied cryptography) cryptographic primitives and schemes. Our treatment is necessarily informal and incomplete. Any good textbook will provide missing details. Martin's book [1620] provides an accessible and mostly non-mathematical treatment of cryptography. Smart's book [963] is aimed at Computer Science undergraduates with some background in mathematics. The text by Boneh and Shoup [1619] is more advanced and targets graduate students.

[4] Alternatively, a protocol is a distributed algorithm.

18.1.2 Hash functions

A hash function is usually an unkeyed function H which takes as input bit-strings of variable length and produces short outputs of some fixed length, n bits say.

A crucial security property is that it should be hard to find collisions for a hash function, that is, pairs of inputs (m_0, m_1) resulting in the same hash value, i.e. such that $H(m_0) = H(m_1)$ (such collisions must exist because the output space is much smaller than the input space). If the output size is n bits, then there is a generic attack based on the birthday paradox that will find collisions with effort about $2^{n/2}$ hash function evaluations.[5]

Other important security properties are pre-image resistance and second pre-image resistance. Informally, pre-image resistance says that it is hard, given an output from a hash function h, to find an input m such that $H(m) = h$. Second pre-image resistance says that, given an m, it is difficult to find $m' \neq m$ such that $H(m) = H(m')$.

Hash functions are often modelled as random functions in formal security analyses, leading to the Random Oracle Model. Of course, a given hash function is a *fixed* function and not a random one, so this is just a heuristic, albeit a very useful one when performing formal security analyses of cryptographic schemes making use of hash functions.

SHA-1 with $n = 160$ is a widely-used hash function. However, collisions for SHA-1 can be found using much less than 2^{80} effort, so it is now considered unsuitable for applications requiring collision-resistance. Other common designs include SHA-256 (with $n = 256$ and still considered very secure) and SHA-3 (with variable length output and based on different design principles from the earlier SHA families). The SHA families are defined in a series of NIST standards [954, 1626].

18.1.3 Block ciphers

A block cipher is a function taking as input a symmetric key K of k bits and a plaintext P with n bits and producing as output a ciphertext C also of n bits. For each choice of K, the resulting function, often written $E_K(\cdot)$, is a permutation mapping n-bit strings to n-bit strings. Since $E_K(\cdot)$ is a permutation, it has an inverse, which we denote $D_K(\cdot)$. We require that both $E_K(\cdot)$ and $D_K(\cdot)$ be fast to compute.

Many security properties can be defined for block ciphers, but the most useful one for formal security analysis demands that the block cipher be a Pseudo-Random Permutation (PRP). Informally this means, if K is chosen uniformly at random, then no efficient adversary can tell the difference between outputs of the block cipher and the outputs of a permutation selected uniformly at random from the set of all permutations of n bits.

It is notable that block cipher designers do not typically target such a goal, but rather resistance to a range of standard attacks. One such attack is exhaustive key search: given a few known plaintext/ciphertext pairs for the given key, try each possible K to test if it correctly maps the plaintexts to the ciphertexts. This *generic* attack means that a block cipher's key length k must be big enough to make the attack infeasible. The Data Encryption Standard (DES) had

[5] The birthday paradox is a generalisation of the initially surprising observation that in a group of 23 randomly selected people there is a 50-50 chance of two people sharing a birthday; more generally, if we make $\sqrt{N}$ selections uniformly at random from a set of N objects, then there is a constant probability of two of the selections being the same.

$k = 56$ which was considered by experts already too short when the algorithm was introduced by the US government in the mid 1970s.

The Advanced Encryption Standard (AES) [1627] is now the most-widely used block cipher. The AES was the result of a design competition run by the US government agency NIST. It has a 128-bit block ($n = 128$) and its key-length k is either 128, 192, or 256, precluding exhaustive key search. Fast implementation of AES is supported by hardware instructions on many Central Processing Unit (CPU) models. Fast and secure implementation of AES is challenging in environments where an attacker may share memory resources with the victim, for example a cache. Still, with its widespread support and lack of known security vulnerabilities, it is rarely the case that any block cipher other than AES is needed. One exception to this rule is constrained computing environments.

Except in very limited circumstances, block ciphers should not be used directly for encrypting data. Rather, they are used in *modes of operation* [1628]. Modes are discussed further below under Authenticated Encryption schemes.

18.1.4 Stream ciphers

Stream ciphers are algorithms that can encrypt a stream of bits (as opposed to a block of n bits in the case of block ciphers) under the control of a k-bit key K. Most stream ciphers use the key to generate a key-stream and then combine it in a bit-by-bit fashion with the plaintext using an XOR operation, to produce the ciphertext stream.

Keystream reuse is fatal to security, since the XOR of two ciphertexts created using the same keystream reveals the XOR of the plaintexts, from which recovering the individual plaintexts becomes possible given enough plaintext redundancy [1629]. To avoid this, stream ciphers usually employ an Initialisation Vector (IV) along with the key; the idea is that each choice of IV should produce an independent keystream. IVs need not be kept secret and so can be sent along with the ciphertext or derived by sender and receiver from context.

The main security requirement for a stream cipher is that, for a random choice of key K, and each choice of IV, it should be difficult for an adversary to distinguish the resulting keystream from a truly random bit-string of the same length.

It is easy to build a stream cipher from a block cipher by using a *mode of operation*; these are discussed in Section 18.1.6.4. Dedicated stream ciphers suitable for implementation in hardware have traditionally relied on combining simpler but insecure components such as Linear Feedback Shift Registers. The A5/1 and A5/2 stream ciphers once widely used in mobile telecommunications systems are of this type. The RC4 stream cipher was well-suited for implementation in software and has a very simple description making it attractive to developers. RC4 became widely used in IEEE wireless communications systems (WEP, WPA) and in Secure Socket Layer/Transport Layer Security (SSL/TLS). It is now considered obsolete because of a variety of security vulnerabilities that it presents in these applications.

18.1.5 Message Authentication Code (MAC) schemes

MAC schemes are used to provide authentication and integrity services. They are keyed. A MAC scheme consists of three algorithms. The first is called KeyGen for key generation. It is randomised and usually consists of selecting a key K at random from the set of bit-strings of some fixed length k. The second algorithm is called Tag. Given as input K and a message m encoded as a bit-string, Tag produces a MAC tag τ, usually a short string of some fixed length t. The third algorithm is called `Verify`. This algorithm is used to verify the validity of a message/MAC tag combination (m, τ) under the key K. So `Verify` takes as input triples (K, m, τ) and produces a binary output, with "1" indicating validity of the input triple.

The main security property required of a MAC scheme is that it should be hard for an adversary to come up with a new pair (m, τ) which `Verify` accepts with key K, even when the adversary has already seen many pairs $(m_1, \tau_1), (m_2, \tau_2), \ldots$ produced by Tag with the same key K for messages of its choice $m_1, m_2, \ldots$. The formal security notion is called Strong Unforgeability under Chosen Message Attack (SUF-CMA for short).

SUF-CMA MAC schemes can be used to provide data origin authentication and data integrity services. Suppose two parties Alice and Bob hold a shared key K; then no party other than those two can come up with a correct MAC tag τ for a message m. So if Alice attaches MAC tags τ to her messages before sending them to Bob, then Bob, after running `Verify` and checking that it accepts, can be sure the messages only came from Alice (or maybe himself, depending on how restrictive he is in using the key) and were not modified by an attacker.

MAC schemes can be built from hash functions and block ciphers. HMAC [1630] is a popular MAC scheme which, roughly speaking, implements its Tag algorithm by making two passes of a hash function applied to the message m prefixed with the key K. The security analysis of HMAC requires quite complex assumptions on the underlying hash function [1619, Section 8.7]. MAC schemes can also be built from so-called *universal hash functions* in combination with a block cipher. Security then relies only on the assumption that the block cipher is a PRP. Since universal hash functions can be very fast, this can lead to very efficient MACs. A widely used MAC of this type is GMAC, though it is usually used only as part of a more complex algorithm called AES-GCM that is discussed below.

18.1.6 Authenticated Encryption (AE) schemes

An Authenticated Encryption (AE) scheme [1631] is a symmetric scheme which transforms plaintexts into ciphertexts and which simultaneously offers confidentiality and integrity properties for the plaintext data. After a long period of debate, AE has emerged as a powerful and broadly applicable primitive for performing symmetric encryption. In most cases where symmetric encryption is needed, AE is the right tool for the job.

An AE scheme consists of three algorithms: KeyGen, Enc and Dec. The first of these is responsible for key generation. It is randomised and usually consists of selecting a key K at random from the set of bit-strings of some fixed length k. Algorithm Enc performs encryption. It takes as input a key K and a plaintext M to be encrypted. Practical AE schemes allow M to be a bit-string of variable length. In the *nonce-based* setting, Enc has an additional input called the *nonce*, denoted N, and usually selected from a set of bit-strings of some fixed size n. In this setting, Enc is deterministic (i.e. it needs no internal randomness). In the *randomised* setting, Enc is a randomised algorithm and does not take a nonce input. The third algorithm in an AE scheme is the decryption algorithm, Dec. It takes as input a key K, a

ciphertext string C and, in the nonce-based setting, a nonce N. It returns either a plaintext M or an error message indicating that decryption failed. Correctness of a nonce-based AE scheme demands that, for all keys K, all plaintexts M and all nonces N, if running Enc on input (K, M, N) results in ciphertext C, then running Dec on input (K, C, N) results in plaintext M. Informally, correctness means that, for a given key, decryption "undoes" encryption.

18.1.6.1 AE Security

Security for nonce-based AE is defined in terms of the combination of a confidentiality property and an integrity property.

Confidentiality for AE says, roughly, that an adversary learns nothing that it does not already know from encryptions of messages. Slightly more formally, we consider an adversary that has access to a *left-or-right (LoR) encryption oracle*. This oracle takes as input a pair of equal-length plaintexts (M_0, M_1) and a nonce N selected by the adversary; internally the oracle maintains a randomly generated key K and a randomly sampled bit b. It selects message M_b, encrypts it using Enc on input (K, M_b, N) and returns the resulting ciphertext C to the adversary. The adversary's task is to make an estimate of the bit b, given repeated access to the oracle while, in tandem, performing any other computations it likes. The adversary is considered successful if, at the end of its attack, it outputs a bit b' such that $b' = b$. An AE scheme is said to be IND-CPA secure ("indistinguishability under chosen plaintext attack") if no adversary, consuming reasonable resources (quantified in terms of the computational resources it uses, the number of queries it makes and sometimes the bit-length of those queries) is able to succeed with probability significantly greater than $1/2$. An adversary can always just make a complete guess b' for the bit b and will succeed half of the time; hence we penalise the adversary by demanding it do significantly better than this trivial guessing attack. The intuition behind IND-CPA security is that an adversary, even with perfect control over which pairs of messages get encrypted, cannot tell from the ciphertext which one of the pair — the left message or the right message — gets encrypted each time. So the ciphertexts do not leak anything about the messages, except perhaps their lengths.[6]

The integrity property says, roughly, that an adversary cannot create new ciphertexts that decrypt to plaintexts, instead of producing decryption failures. Slightly more formally, we give an adversary an encryption oracle; this oracle internally maintains a randomly generated key K and on input (M, N) from the adversary, runs Enc on input (K, M, N) and returns the resulting ciphertext C to the adversary. We also give the adversary a decryption oracle, to which it can send inputs (C, N). In response to such inputs, the oracle runs Dec on input (K, C, N) (for the same key K used in the encryption oracle) and gives the resulting output to the adversary — this output could be a message or an error message. The adversary against integrity is considered to be successful if it at some point sends an input (C, N) to its decryption oracle which results in an output that is a plaintext M and not an error message. Here, we require that (C, N) be such that C is *not* a ciphertext output by the encryption oracle when it was given some input (M, N) during the attack — otherwise the adversary can win trivially. Under this restriction, for the adversary to win, the pair (C, N) must be something new that the adversary could not have trivially obtained from its encryption oracle. In this sense, C is a ciphertext forgery for some valid plaintext, which the adversary may not even know before it sends (C, N) for decryption. An AE scheme is said to be INT-CTXT secure (it has "integrity of ciphertexts")

[6] But note that hiding the length of data can be a more complicated task than hiding the data itself. Moreover, just knowing the length of data can lead to significant attacks. Such attacks, more broadly covered under side-channel attacks, are discussed in Section 18.2.3.

if no adversary, consuming reasonable resources (quantified as before) is able to succeed with probability significantly greater than 0.

Similar security notions can be developed for the randomised setting.

An AE scheme is said to be secure (or AE secure) if it is both IND-CPA and INT-CTXT secure. This combination of security properties is very strong. It implies, for example, IND-CCA security, a traditional security notion which extends IND-CPA security by also equipping the adversary with a decryption capability, capturing the capabilities that a practical attacker may have. It also implies a weaker "integrity of plaintexts" notion, which roughly states that it should be hard for the adversary to create a ciphertext that encrypts a new plaintext.

18.1.6.2 Nonces in AE

It is a requirement in the IND-CPA security definition that the nonces used by the adversary be *unique* across all calls to its LoR encryption oracle. Such an adversary is called a *nonce respecting adversary*. In practice, it is usually the responsibility of the application using an AE scheme to ensure that this condition is met across all invocations of the Enc algorithm for a given key K. Note that the nonces do not need to be random. Indeed choosing them randomly may result in nonce collisions, depending on the quality of the random bit source used, the size of the nonce space and the number of encryptions performed. For example, the nonces could be invoked using a stateful counter. The core motivation behind the nonce-based setting for AE is that it is easier for a cryptographic implementation to maintain state across all uses of a single key than it is to securely generate the random bits needed to ensure security in the randomised setting. This is debatable and nonce repetitions have been observed in practice [1632]. For some AE schemes such as the widely deployed AES-GCM scheme, the security consequences of accidental nonce repetition are severe, e.g. total loss of integrity and/or partial loss of confidentiality. For this reason, misuse-resistant AE schemes have been developed. These are designed to fail more gracefully under nonce repetitions, revealing less information in this situation than a standard AE scheme might. They are generally more computationally expensive than standard AE schemes. AES-GCM-SIV [1633] is an example of such a scheme.

18.1.6.3 AE Variants

Many variants of the basic AE formulation and corresponding security notions have been developed. As an important example, Authenticated Encryption with Associated Data (AEAD) [1634] refers to an AE extension in which an additional data field, the Associated Data (AD), is cryptographically bound to the ciphertext and is integrity protected (but not made confidential). This reflects common use cases. For example, we have a packet header that we wish to integrity protect but which is needed in the clear to deliver data, and a packet body that we wish to both integrity protect and make confidential. Even the basic AE security notion can be strengthened by requiring that ciphertexts be indistinguishable from random bits or by considering security in the multi-user setting, where the adversary interacts with multiple AE instantiations under different, random keys and tries to break any one of them. The latter notion is important when considering large-scale deployments of AE schemes. The two separate notions, IND-CPA and INT-CTXT, can also be combined into a single notion [1635].

18.1.6.4 Constructing AE Schemes

Secure AE (and AEAD) schemes can be constructed *generically* from simpler encryption schemes offering only IND-CPA security and SUF-CMA secure MAC schemes. There are three basic approaches: Encrypt-then-MAC (EtM), Encrypt-and-MAC (E&M) and MAC-then-Encrypt (MtE). Of these, the security of EtM is the easiest to analyse and provides the most robust combination, because it runs into fewest problems in implementations. Both MtE and E&M have been heavily used in widely-deployed secure communications protocols such as SSL/TLS and Secure Shell (SSH) with occasionally disastrous consequences [1636, 1637, 1638]. Broad discussions of generic composition can be found in [1639] (in the randomised setting) and [1640] (more generally).

This generic approach then leaves the question of how to obtain an encryption scheme offering IND-CPA security. This is easily achieved by using a block cipher in a suitable mode of operation [1628], for example, counter (CTR) mode or CBC mode. Such a mode takes a block cipher and turns it into a more general encryption algorithm capable of encrypting messages of variable length, whereas a block cipher can only encrypt messages of length n bits for some fixed n. The IND-CPA security of the mode can then be proved based on the assumption that the used block cipher is a PRP. Indeed, the nonce-based AEAD scheme AES-GCM [1628] can be seen as resulting from a generic EtM construction applied using AES in a specific nonce-based version of CTR mode and an SUF-CMA MAC constructed from a universal hash function based on finite field arithmetic. AES-GCM is currently used in about 90% of all TLS connections on the web. It has excellent performance on commodity CPUs) from Intel and AMD because of their hardware support for the AES operations and for the finite field operations required by the MAC. A second popular AEAD scheme, ChaCha20-Poly1305 [1641], arises in a similar way from different underlying building blocks. The CAESAR competition[7] was a multi-year effort to produce a portfolio of AEAD schemes for three different use cases: lightweight applications, high-performance applications and defence in depth (essentially, misuse-resistant AE).

18.1.7 Public Key Encryption Schemes and Key Encapsulation Mechanisms

A Public Key Encryption (PKE) scheme consists of three algorithms: KeyGen, Enc and Dec. The first of these is responsible for key generation. It is randomised and outputs key pairs (sk, pk) where sk denotes a private key (often called the secret key) and pk denotes a public key. The algorithm Enc performs encryption. It takes as input the public key pk and a plaintext M to be encrypted and returns a ciphertext C. In order to attain desirable security notions (introduced shortly), Enc is usually randomised. The plaintext M comes from some set of possible plaintexts that the scheme can handle. Usually this set is limited and dictated by the mathematics from which the PKE scheme is constructed. This limitation is circumvented using *hybrid encryption*, which combines PKE and symmetric encryption to allow a more flexible set of plaintexts. The third algorithm in a PKE scheme is the decryption algorithm, Dec. It takes as input the private key sk and a ciphertext C. It returns either a plaintext M or an error message indicating that decryption failed. Correctness of a PKE scheme requires that, for all key pairs (sk, pk) and all plaintexts M, if running Enc on input (pk, M) results in ciphertext C, then running Dec on input (sk, C) results in plaintext M. Informally, correctness means that, for a given key, decryption "undoes" encryption. Notice here the fundamental asymmetry in the use of keys in a PKE scheme: pk is used during encryption and sk during

[7] See **https://competitions.cr.yp.to/caesar-submissions.html**.

decryption.

18.1.7.1 PKE Security

There are many flavours of security for PKE. We focus on just one, which is sufficient for many applications, and provide a brief discussion of some others.

Recall the definition of IND-CPA and IND-CCA security for AE schemes from Section 18.1.6. Analogous notions can be defined for PKE. In the IND-CPA setting for PKE, we generate a key pair (sk, pk) by running KeyGen and give the public key pk to an adversary (since public keys are meant to be public!). The adversary then has access to an LoR encryption oracle which, on input a pair of equal-length messages (M_0, M_1), performs encryption of M_b under the public key pk, i.e. runs the randomised algorithm Enc on input (pk, M_b), to get a ciphertext C which is then returned to the adversary. The adversary's task is to make an estimate of the bit b, given repeated access to the oracle while, in tandem, performing any other computations it likes. The adversary is considered successful if, at the end of its attack, it outputs a bit b' such that $b' = b$. A PKE scheme is said to be IND-CPA secure ("indistinguishability under chosen plaintext attack") if no adversary, consuming reasonable resources (quantified in terms of the computational resources it uses and the number of queries it makes) is able to succeed with probability significantly greater than $1/2$. The intuition behind IND-CPA security for PKE is the same as that for AE: even with perfect control over which pairs of messages get encrypted, an adversary cannot tell from the ciphertext which one of the pairs is encrypted each time.

Note that in order to be IND-CPA secure, a PKE scheme must have a randomised encryption algorithm (if Enc was deterministic, then an adversary that first makes an encryption query on the pair (M_0, M_1) with $M_0 \neq M_1$ and then an encryption query on (M_0, M_0) could easily break the IND-CPA notion). If a PKE scheme is IND-CPA secure, then it must be computationally difficult to recover pk from sk, since, if this were possible, then an adversary could first recover sk and then decrypt one of the returned ciphertexts C and thereby find the bit b.

IND-CCA security for PKE is defined by extending the IND-CPA notion to also equip the adversary with a decryption oracle. The adversary can submit (almost) arbitrary bit-strings to this oracle. The oracle responds by running the decryption algorithm and returning the resulting plaintext or error message to the adversary. To prevent trivial wins for the adversary and therefore avoid a vacuous security definition, we have to restrict the adversary to not make decryption oracle queries for any of the outputs obtained from its encryption oracle queries.

We do not generally consider integrity notions for PKE schemes. This is because, given the public key pk, an adversary can easily create ciphertexts of its own, so no simple concept of "ciphertext integrity" would make sense for PKE. Integrity in the public key setting, if required, usually comes from the application of digital signatures, as discussed in Section 18.1.9. Digital signatures and PKE can be combined in a cryptographic primitive called *signcryption*. This can be a useful primitive in some use-cases, e.g. secure messaging (see Section 18.5.2).

In some applications, such as anonymous communications or anonymous cryptocurrencies, anonymity of PKE plays a role. Roughly speaking, this says that a PKE ciphertext should not leak anything about the public key pk that was used to create it. This is an orthogonal property to IND-CPA/IND-CCA security. A related concept is robustness for PKE, which informally says that a ciphertext generated under one public key pk should not decrypt correctly under the private key sk' corresponding to a second public key pk'. Such a property, and stronger variants of it, are needed to ensure that *trial decryption* of anonymous ciphertexts does not produce unexpected results [1642].

18.1.7.2 Key Encapsulation Mechanisms

A Key Encapsulation Mechanism (KEM) is a cryptographic scheme that simplifies the design and use of PKE. Whereas a PKE scheme can encrypt arbitrary messages, a KEM is limited to encrypting symmetric keys. One can then build a PKE scheme from a KEM and an AE scheme (called a Data Encapsulation Mechanism, DEM, in this context): first use the KEM to encrypt a symmetric key K, then use K in the AE scheme to encrypt the desired message; ciphertexts now consist of two components: the encrypted key and the encrypted message.

We can define IND-CPA and IND-CCA security notions for KEMs. These are simpler to work with than the corresponding notions for PKE and this simplifies security analysis (i.e. generating and checking formal proofs). Moreover, there is a *composition theorem* for KEMs which says that if one takes an IND-CCA secure KEM and combines it with an AE-secure DEM (AE scheme) as above, then one gets an IND-CCA secure PKE scheme. As well as simplifying design and analysis, this KEM-DEM or hybrid viewpoint on PKE reflects how PKE is used in practice. Because PKE has generally slow algorithms and has large ciphertext overhead (compared to symmetric alternatives like AE), we do not use it directly to encrypt messages. Instead, we use a KEM to encrypt a short symmetric key and then use that key to encrypt our bulk messages.

18.1.7.3 Some common PKE schemes and KEMs

Perhaps the most famous PKE scheme is the RSA scheme. In its *textbook* form, the public key consists of a pair (e, N) where N is a product $p \cdot q$ of two large primes and the private key consists of a value d such that $de = 1 \bmod (p-1)(q-1)$. Encryption of a message M, seen as a large integer modulo N, sets $C = M^e \bmod N$. On account of the mathematical relationship between d and e, it can be shown that then $M = C^d \bmod N$. So encryption is done by "raising to the power of e mod N" and decryption is done by "raising to the power of d mod N". These operations can be carried out efficiently using the square-and-multiply algorithm and its variants. Decryption can be accelerated by working separately modulo p and q and then combining the results using the Chinese Remainder Theorem (CRT).

The security of RSA, informally, depends on the hardness of the Integer Factorisation Problem (IFP): if an adversary can recover p and q from N, then it can recompute d from e, p and q using the extended Euclidean algorithm. But what we really want is a converse result: if an adversary can break RSA, then it should be possible to use that adversary in a black-box manner as a subroutine to create an algorithm that factors the modulus N or solves some other presumed-to-be-hard problem.

This textbook version of RSA is completely insecure and must not be used in practice. Notice, for example, that it is not randomised, so it certainly cannot be IND-CPA secure. Instead RSA must be used as the basis for constructing more secure alternatives. This is usually done by performing a keyless encoding step, represented as a function $\mu(\cdot)$, on the message before applying the RSA transform. Thus we have $C = \mu(M)^e \bmod N$. Decryption then involves applying the reverse transform and decoding.

In order to achieve modern security notions, $\mu(\cdot)$ must be randomised. One popular encoding scheme, called PKCS#1 v1.5 and specified in [1643], became very common in applications due to its relatively early standardisation and its ease of implementation. Unfortunately, RSA with PKCS#1 v1.5 encoding does not achieve IND-CCA security, as demonstrated by the famous Bleichenbacher attack [1644]. Despite now being more than 20 years old, variants of the Bleichenbacher attack still regularly affect cryptographic deployments, see for example [1645].

A better encoding scheme is provided in PKCS#1 v2.1 (also specified in [1643]), but it has not fully displaced the earlier variant. RSA with PKCS#1 v2.1 encoding — also called RSA-OAEP — can be proven to yield an IND-CCA secure PKE scheme but the best security proof we have [1646] is unsatisfactory in various technical respects: its proof is not tight and requires making a strong assumption about the hardness of a certain computational problem. Improved variants with better security analyses do exist in the literature but have not found their way into use.

One can easily build an IND-CCA secure KEM from the RSA primitive, as follows: select M at random from $\{0, 1, \ldots, N-1\}$, set $C = M^e \bmod N$ (as in textbook RSA) and define $K = H(M)$ to be the encrypted symmetric key. Here H is a cryptographic hash function (e.g. SHA-256). This scheme can be proven secure by modelling H as a random oracle, under the assumption that the *RSA inversion problem* is hard. The RSA inversion problem is, informally, given e, M and $M^e \bmod N$ for a random M, to recover M. The RSA inversion problem is not harder than the IFP, since any algorithm to solve IFP can be used to construct an algorithm that solves the RSA inversion problem. But the RSA inversion problem could be easier than the IFP and it is an open problem to fully decide this question.

Note that RSA encryption is gradually being displaced in applications by schemes using Elliptic Curve Cryptography (ECC) because of its superior performance and smaller key-sizes for a given target security level. See Section 18.1.13 for further discussion.

We will discuss another class of PKE schemes, based on the Discrete Logarithm Problem (DLP) after discussing Diffie-Hellman Key Exchange.

18.1.8 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange (DHKE) is a fundamental tool in cryptography and was introduced by Diffie and Hellman in their seminal paper on Public Key Cryptography [1647]. DHKE allows two parties to set up a shared key based only on a public exchange of messages. First the two parties (let us follow convention and call them Alice and Bob) agree on some common parameters: a group G of prime order q and a generator g of that group. Typically, these are agreed during the exchange of messages or are pre-agreed; ideally, they are standardised so that well-vetted parameters of well understood cryptographic strength are used. Alice then chooses a value x uniformly at random from $\{0, 1, \ldots, q-1\}$, computes g^x using the group operation in G and sends g^x to Bob. Similarly Bob chooses a value y uniformly at random from $\{0, 1, \ldots, q-1\}$, computes g^y and sends it to Alice. Now Bob can compute $(g^x)^y = g^{xy}$ while Alice can compute $(g^y)^x = g^{yx} = g^{xy}$. Thus the value g^{xy} is a common value that both Alice and Bob can compute.

An adversary Eve who eavesdrops on the communications between Alice and Bob can see g^x and g^y and can be assumed to know the public parameters g, q and a description of the group G. This adversary is then faced with the problem of computing g^{xy} from the triple (g, g^x, g^y). Informally, this is known as the Computational Diffie-Hellman Problem (CDHP). One way for Eve to proceed is to try to compute x from g^x and then follow Alice's computation. Computing x from g and g^x is known as the Discrete Logarithm Problem (DLP). The CHDP is not harder than the DLP (since an algorithm to solve the latter can be used to solve the former) but the exact relationship is not known.

The traditional setting for DHKE is to select large primes p and q such that q divides $p-1$ and then take g to be an element of multiplicative order q modulo p; such a g generates a group of

order q. By choosing q and p of an appropriate size, we can make the DLP, and presumably the CDHP, hard enough to attain a desired security level (see further discussion in Section 18.1.13). An alternative that has largely now displaced this traditional "finite field Diffie-Hellman" setting in applications is to use as G the group of points on an elliptic curve over a finite field. This allows more efficient implementation and smaller key sizes at high security levels, but comes with additional implementation pitfalls.

The raw DHKE protocol as described directly above is rarely used in practice, because it is vulnerable to active Man-in-the-Middle (MitM) attacks, in which the adversary replaces the values g^x and g^y exchanged in the protocol with values for which it knows the discrete logarithms. However, the core idea of doing "multiplication in the exponent" is used repeatedly in cryptographic protocols. MitM attacks are generally prevented by adding some form of authentication (via MACs or digital signatures) to the protocol. This leads to the notion of Authenticated Key Exchange (AKE) protocols — see [1648] for a comprehensive treatment.

It is important also for Alice and Bob to use trusted parameters, or to verify the cryptographic strength of the parameters that they receive, in DHKE. This can be a complex undertaking even in the traditional setting, since a robust primality test is needed [1649, 1650]. In the elliptic curve setting, it is not reasonable to expect the verification to be done by protocol participants and we use one of a small number of standardised curves. It is also important for the respective parties to check that the received values g^y and g^x do lie in the expected group, otherwise the protocol may be subject to attacks such as the small sub-group attack. These checks may be computationally costly.

18.1.8.1 From Diffie-Hellman to ElGamal

It is easy to build a KEM from the DHKE primitive. We simply set the KeyGen algorithm to output a key pair $(sk, pk) = (y, g^y)$ (where y is generated as in DHKE), while `Encrypt` selects x as in DHKE and then simply outputs the group element g^x as the ciphertext. Finally, `Decrypt` takes as input a group element h and outputs KDF(h^y) where KDF$(\cdot)$ denotes a suitable key derivation function (as covered in Section 18.3.2). So the symmetric key encapsulated by group element g^x is KDF(g^{xy}).

From this KEM, using the standard KEM-DEM construction, we obtain a variant of the ElGamal encryption scheme [1651] called the Diffie-Hellman Integrated Encryption Scheme (DHIES) and analysed in [1652]. In the elliptic curve setting, the scheme is known as ECIES. It is a particularly neat PKE scheme with compact ciphertexts and strong security properties. It avoids many of implementation issues associated with standardised variants of RSA.

18.1.9 Digital Signatures

Digital signatures schemes are used to provide authentication, integrity and non-repudiation services. They are the asymmetric analogue of MACs. A digital signature scheme consists of three algorithms: KeyGen, `Sign` and `Verify`. The first, KeyGen, is responsible for key generation. It is randomised and outputs key pairs (sk, vk) where sk denotes a private key (often called the secret key or signing key) and vk denotes a verification key. The second algorithm `Sign` takes as input sk and a message m encoded as a bit-string and produces a signature σ, usually a short string of some fixed length t. The third algorithm is called `Verify`. This algorithm is used to verify the validity of a message/signature combination (m, σ) under the key vk. So `Verify` takes as input triples (vk, m, σ) and produces a binary output, with "1"

indicating validity of the input triple.

The main security property required of a digital signature scheme is that it should be hard for an adversary to come up with a new pair (m, σ) which `Verify` accepts with key vk, even when the adversary has already seen many pairs $(m_1, \sigma_1), (m_2, \sigma_2), \ldots$ produced by `Sign` with the matching signing key sk for messages of its choice $m_1, m_2, \ldots$. As for MACs, the formal security notion is called Strong Unforgeability under Chosen Message Attack (SUF-CMA for short).

SUF-CMA digital signature schemes can be used to provide data origin authentication and data integrity services as per MACs. In addition, they can offer *non-repudiation*: if Alice is known to be associated with a key pair (sk, vk) and provided sk has not been compromised, then Alice cannot deny having created a valid message/signature pair (m, σ) which `Verify` accepts. In practice, making this non-repudiation property meaningfully binding (e.g. for it to have legal force) is difficult. See further discussion in Law & Regulation Knowledge Area (Section 3.10.4).

A common pitfall is to assume that a signature σ must bind a message m and a verification key vk; that is, if `Verify` accepts on input (vk, m, σ), then this implies that σ must have been produced using the corresponding signing key sk on message m. In fact, the SUF-CMA security definition does not imply this, as it only refers to security under a single key pair (sk, vk) but does not rule out the possibility that, given as input a valid triple (vk, m, σ), an adversary can concoct an alternative key pair (sk', vk') such that `Verify` also accepts on input (vk', m, σ). This gap leads to Duplicate Signature Key Selection (DSKS) attacks, see [1653] for a detailed treatment. Such attacks can lead to serious vulnerabilities when using digital signatures in more complex protocols.

Signature schemes can be built from the same kinds of mathematics as PKE schemes. For example, the textbook RSA signature scheme signs a message M by computing $\sigma = H(m)^d \bmod N$ where H is a cryptographic hash function. Here d is the signing key and the verification key is a pair $(e, N = pq)$ such that $de = 1 \bmod (p-1)(q-1)$. Then verification of a purported signature σ on a message m involves checking the equality $\sigma^e = H(m) \bmod N$. The similarity to RSA encryption, wherein signing uses the same operation of "raising to the power d mod N" as does decryption in textbook RSA, is not coincidental. It is also a source of much confusion, since in the case of general signature schemes, signing is not related to any PKE decryption operation. A variation of textbook RSA in which $H(m)$ is replaced by a randomised encoding of the hashed message according to the PKCS#1 version 1.5 encoding scheme for signatures [1643] is widely used in practice but has significant security shortcomings and lacks a formal security proof. The RSA-PSS encoding scheme, another randomised variant, is also specified in [1643]; it does permit a formal security proof of security [1654].

The DSA scheme [1655] works in the finite field Diffie-Hellman setting, while ECDSA translates DSA to the elliptic curve setting. Notably, despite their standardised form and widespread use, neither DSA nor ECDSA enjoys a fully satisfactory UF-CMA security proof. The EdDSA signature scheme [1656] is a variant of ECDSA that arguably enjoys better implementation security than ECDSA. In particular, ECDSA has a randomised signing algorithm and its security fails spectacularly (allowing recovery of the signing key) if the same random value is used to sign two different messages; ECDSA is also vulnerable to attacks exploiting partial leakage of the random values used during signing. By contrast, EdDSA is a deterministic scheme and avoids the worst of these failure modes. EdDSA, being more closely based on Schnorr signatures than ECDSA, also enjoys security proofs based on assumptions that are milder than those needed for proving the security of ECDSA.

Interestingly, signature schemes can be built from symmetric primitives, specifically hash functions. The original idea goes back to Lamport [1657]: to be able to sign a single bit message, commit in the verification key to two values $h_0 = H(M_0)$ and $h_1 = H(M_1)$, where M_0 encodes a zero-bit and M_1 encodes a one-bit. So the verification key is (h_0, h_1) and the signing key is (M_0, M_1). Now to sign a single bit b, the signer simply outputs M_b; the verification algorithm checks the relation $h_b = H(M_b)$, outputting "1" if this holds. The original Lamport scheme is one-time use only and only signs a single-bit message. But many enhancements have been made to it over the years bringing it to a practically usable state. A specific hash-based signature scheme SPHINCS+ is an "alternate candidate" in the NIST PQC process for selecting post-quantum secure schemes, see Section 18.1.16 for further discussion.

Many forms of signature scheme with advanced security properties have been researched and sometimes find their way into use, especially in privacy-oriented applications. For example, *blind* signatures [1658] allow a party to obtain signatures on messages without the signer knowing which message is being signed. Blind signature schemes can be used as a building block in electronic cash and electronic voting schemes. As a second example, *group* signatures allow one of many parties to sign messages in such a way that the verifier cannot tell exactly which party produced the signature; meanwhile a group manager can "break" this anonymity property. Group signatures have been used in the Trusted Computing Group's Direct Anonymous Attestation protocol [1396] to enable remote authentication of Trusted Platform Modules whilst preserving privacy. A third example is provided by *ring* signatures [1659], which have functionality similar to group signatures but without the opening capability possessed by a group manager. The cryptocurrency Monero has used ring signatures to provide anonymity.

18.1.10 Cryptographic Diversity

In the preceding subsections we have focused on the main algorithms and schemes that will be encountered in today's deployed cryptographic systems. However, a brief glance at the literature will reveal that this is just the tip of the iceberg in terms of what cryptographic ideas have been researched. But the vast majority of these ideas are not standardised, or do not have readily available implementations of production quality, or are not fully fleshed out in a way that a software engineer could easily implement them. There is a long road from cryptography as presented in most research papers and cryptography as it needs to be packaged for easy deployment.

18.1.11 The Adversary

So far, we have referred to the adversary as an abstract entity. In practice there are adversaries with different motivations, levels of skill and available computational resources. In applied cryptography we generally want to design cryptographic components that are usable across a wide variety of applications, so we should be conservative in how we model the adversary, assuming it is capable of marshalling significant resources.

At the same time, for the most part, we cannot achieve *unconditional security* in practical systems, that is security against adversaries with unbounded computational power. This is because such systems consume large amounts of key material that cannot be established using public key methods. So we have to place some limits on the adversary's computational capabilities. A typical objective is to make the adversary expend work comparable to an exhaustive key search for a block cipher like AES with 128-bit keys, in which case we speak of

a "128-bit security level".[8] Such a computational feat seems still well beyond the horizon of even state security agencies. It's naturally hard to estimate their capabilities but, for comparison, the number of hash computations carried out in global Bitcoin mining currently stands at around 2^{67} per second and has the electricity consumption of a small sovereign state. At that rate, and assuming the cost of computing a hash is the same as that of testing an AES key, an exhaustive key search would still require 2^{36}, or about 10^{11}, years.[9] If we are even more conservative, or want a very large security margin over a long period of time (during which large-scale quantum computers may become available), or are concerned about concrete security in the context of multi-target attacks, then aiming for 192-bit or 256-bit security may be attractive.

We should also be conservative in rejecting algorithms and schemes that have known weaknesses, even if seemingly minor. It is a truism that attacks in cryptography only get stronger with time, either due to computational advances or the introduction of new cryptanalytic ideas. This conservatism is in tension with the fact that replacing one cryptographic scheme with another can be costly and time-consuming, unless *cryptographic agility* is built into our system (see Section 18.1.14 for further discussion). We are often encumbered with legacy cryptographic systems that cannot be easily updated, or where the system owners do not see the value in doing so until a practical break is exhibited.

18.1.12 The Role of Formal Security Definitions and Proofs

We have described in the preceding subsections, in an informal manner, syntax and security definitions for the main cryptographic schemes. These informal definitions are backed by fully formal ones, see for example [1619, 1660]. The value of such definitions are manifold. Syntax and correctness definitions enable one to be precise about what behaviour to expect from a cryptographic scheme and to build schemes out of simpler components in a precise way. Security definitions allow different schemes to be compared and to check whether application requirements will be met. Strong security definitions can rule out many classes of practical attack. A security proof — showing that a given scheme satisfies a given formal security definition under certain assumptions — then gives assurance as to the soundness of a scheme's design and makes it clear what assumptions security rests on.

Indeed, formal security analysis in Applied Cryptography has reached a maturity level where there should be no need to ever deploy a cryptographic scheme or protocol that does *not* come with a clear syntax and have a rigorous security proof under clearly stated assumptions. Unfortunately, this rule is still often ignored in practice. This does leave opportunities for *post hoc* analysis by researchers, either to find flaws or to provide positive security results. In an ideal world, one would first gather all the requirements for any new scheme or protocol, then design the system and simultaneously develop security proofs for it. In reality, one is often trapped in a design-release-break-fix cycle. An intermediate approach of design-break-fix-release was used for TLS 1.3. Further discussion comparing these different models of cryptographic development can be found in [1661].

[8] It is difficult to be precise about concrete security levels. Issues arise because of different cost models for computation, e.g. counting the unit of cost as being an AES operation versus a single CPU) instruction; on some platforms, an AES round operation can be performed via a single CPU instruction! One also needs to account for the use of CPUs, GPUs and special purpose hardware for cryptanalysis, such as might be within the reach of a well-funded security agency.

[9] It is worth noting that the sun is expected to consume the earth in a super nova in about 10^{10} years, assuming it is not swallowed by a black hole first.

Notice also that such proofs are not unconditional, unlike most proofs in mathematics. A typical proof shows that a given scheme or protocol satisfies a particular security definition under some assumptions. Such proofs are often stated in a reductive fashion (i.e. as with reductions from complexity theory): given any adversary in the form of an arbitrary algorithm that can break the scheme according to the security definition, the proof shows that the adversary A can be used as a subroutine in building an algorithm B that can break one of the components of the scheme (e.g. find a collision in a hash function) or in building a different algorithm C that can break some underlying hardness assumption (e.g. solve the IFP for moduli n with distribution given by the KeyGen algorithm of a PKE scheme). For Applied Cryptography, *concrete* reductions are to be preferred. In our example, these are ones in which in which we eschew statements describing B or C as simply being "polynomial time" but in which the resources (computation, storage, etc) consumed by the adversary A (and its advantage in breaking the scheme) are carefully related to those of algorithms B and C. Furthermore, it is preferable that proofs should be *tight*. That is, we would like to have proofs showing a close relationship between the resources consumed by and advantage of adversary A on the one hand, and the resources consumed by and advantages of algorithms B and C constructed from A on the other. The result of having a tight proof is that the scheme's security can be meaningfully related to that of its underlying components. This is not always achieved, resulting in proofs that may be technically vacuous.

For complex cryptographic schemes and protocols, the security statements can end up being difficult to interpret, as they may involve many terms and each term may relate to a different component in a non-trivial way. Such security statements typically arise from proofs with many hand-written steps that can hide errors or be difficult for humans to verify. Typically though, such proofs are modularised in a sequence of steps that can be individually checked and updated if found to be in error. A popular approach called "game hopping" or "sequences of games", as formalised in [1662, 1663] in two slightly different ways, lends itself to the generation of such proofs. An alternative approach to taming the complexity of proofs comes from the use of formal and automated analysis methods, see Formal Methods for Security Knowledge Area (Chapter 13) for an extensive treatment.

The proofs are usually for mathematically tractable pseudo-code descriptions of the schemes, not for the schemes as implemented in some high-level programming language and certainly not for schemes as implemented in a machine language. So there is a significant gap in terms of what artefacts the proofs actually make statements about. Researchers have had some success in developing tools that can prove the security of running code and some code of this type has been deployed in practice; for a good overview, see [1664]. Furthermore, a security proof only gives guarantees concerning the success of attacks that lie within the scope of the model and says nothing about what happens beyond that. For example, an adversary operating in the real world may have greater capabilities than are provided to it in the security model used for proving security. We shall return to these issues in the next section on cryptographic implementation.

A sustained critique of the provable security approach has been mounted by Koblitz and Menezes in their "Another look at . . ." series of papers, see [1665] for a retrospective. This critique has not always been welcomed by the theoretical cryptography research community, but any serious field should be able to sustain, reflect on and adapt to such critique. In our view, the work of Koblitz and Menezes has helped to bridge the gap between theory and practice in cryptography, since it has helped the community to understand and begin to address the limitations of its formal foundations.

18.1.13 Key Sizes

We have discussed why aiming for the 128-bit security level makes sense — it provides a sufficient margin of security in almost every conceivable circumstance, at least within the realm of conventional computing. Another reason is that except in specific application domains such as constrained environments, it is efficiently achievable. In other words, there is no good reason *not* to aim this high.

Algorithms and their keys do have finite lifetimes. Advances in cryptanalysis may render once secure algorithms or key sizes insecure. Moreover, the longer an individual key is in use, the more likely it is to become compromised. These issues for individual keys are discussed in more detail in Section 18.3. Changing algorithms and key sizes can be inconvenient and costly. This provides arguments in favour of making conservative choices in the first place.

The target security level of 128 bits brings efficiency considerations into play, especially for asymmetric algorithms. For example, it is estimated that forcing a direct attack on the IFP to break RSA cost 2^{128} effort would require the use of a 3072-bit modulus [1666, Table 2].[10] This is because of the sub-exponential complexity of the best known algorithm for solving the IFP (the Number Field Sieve). Such a modulus size is large enough to significantly slow down the basic RSA operations (in the general case, the complexity of modular exponentiation grows with the cube of the modulus bit length). The same is true for finite-field DLP-based cryptosystems (e.g. Diffie-Hellman and ElGamal). By contrast, because only square-root speed-ups exist for the ECDLP, we can escape with much smaller parameters when targeting 128-bit security for ECC: a curve over a 256-bit prime field suffices. So, for the standard 128-bit security level, ECC-based schemes become more efficient than IFP-based or finite field DLP-based ones. The contrast is even more stark if one targets a 256-bit security level: there 15360-bit RSA public keys are recommended by NIST [1666, Table 2], while the required size for ECC only moves up to 512 bits.

These considerations explain the recent rise in popularity of ECC and may lead to the slow death of RSA-based cryptosystems. The US National Security Agency (NSA) has recommended organisations who have not already done so to not make a significant expenditure to transition from RSA to ECC, but to wait for post-quantum algorithms (i.e. algorithms that aim to be secure against large-scale quantum computers) that should result from the on-going NIST standardisation effort.[11]

18.1.14 Cryptographic Agility

Occasionally it is necessary in some system or protocol to exchange one algorithm for another in the same class. One reason might be that the original algorithm is broken. The history of hash functions provides notable examples, with once secure hash functions like MD5 now being considered trivially insecure. Another reason might be that a more efficient alternative becomes available — consider the switch from RSA-based algorithms to ECC-based ones discussed in Section 18.1.13. A third reason is the introduction of a technology shift — for example, a precautionary shift to post-quantum algorithms as a hedge against the development of large-scale quantum computers.

This exchange is made easier if the system or protocol is cryptographically agile — that is,

[10] Other estimates are available, see summary at **https://www.keylength.com/en/**, but all estimates are in the same ballpark.

[11] See **https://apps.nsa.gov/iad/programs/iad-initiatives/cnsa-suite.cfm**.

if it has an in-built capability to switch one algorithm for another and/or from one version to another. This facility is enabled in secure communications protocols like IPsec, SSH and SSL/TLS through cipher suite and version negotiation: the algorithms that will be used and the protocol version are negotiated between the participating parties during the protocol execution itself. Adding this facility to an already complex protocol may introduce security vulnerabilities, since the negotiation mechanisms themselves may become a point of weakness. An example of this is downgrade attacks in the context of SSL/TLS, which have exploited the co-existence of different protocol versions [1667] as well as support for deliberately weakened "EXPORT" cipher suites [438, 1668]. Cryptographic agility may also induce software bloat as there is an incipient temptation to add everyone's favourite algorithm.

At the opposite end of the spectrum from cryptographic agility lies systems (and their designers) that are *cryptographically opinionated*, that is, where a single set of algorithms is selected and hard-coded. WireGuard [1669] is an example of such a protocol: it has no facility to change algorithms and not even a protocol version field.

There is a middle-way: support cryptographic agility where possible, but with tight control over which algorithms and legacy versions are supported.

For more information on cryptographic agility, especially in the post-quantum setting, we recommend [1670].

18.1.15 Development of Standardised Cryptography

Standardisation plays an important role in cryptography. Standards provide a suite of carefully vetted primitives, higher-level protocols and cryptographic best practices that can be used by non-expert developers. They can also help to ensure interoperability, not only by providing complete specifications of algorithms but also by helping to identify a smaller set of algorithms on which implementers can focus.

There are multiple standardisation bodies producing cryptographic standards. Most prominent are ISO/IEC, the US National Institute for Standards and Technology (NIST) and the Internet Engineering Task Force (IETF). ISO/IEC and NIST cryptographic standards tend to focus (though not exclusively) on lower-level primitives, while IETF works more at the protocol level.

The three bodies work in quite different ways.

ISO/IEC uses a closed model, where country representatives come together to produce standards through a multi-stage drafting and voting process. ISO/IEC working groups can and do invite external experts to attend their meetings and provide input.

NIST employees directly write some of their standards, with open calls for comment from the wider community. NIST also runs cryptographic competitions in which external teams of researchers compete to meet a design specification. The AES and SHA-3 algorithms were produced in this way. Although NIST is a US federal government body, its standards tend to become *de facto* international standards. Its competitions are also entered by teams from all around the world and the winners are frequently not from the US.

The IETF model is completely open. Anyone can join an IETF mailing list and join a technical discussion or, given financial resources, attend IETF meetings where its standards are developed. For a document to become an IETF standard (technically, a "Request for Comments" or RFC), the key requirement is "rough consensus and running code". Multiple levels of review and consensus-building are involved before a draft document becomes an RFC. The Internet

Research Task Force (IRTF) is a sister-organisation to the IETF and its Crypto Forum Research Group (CFRG)[12] acts as a repository of expertise on which the IETF can draw. CFRG also produces its own RFCs.

Standards bodies are not perfect. Too many bodies — and standards produced by them — can lead to cryptographic proliferation, which makes inter-operation harder to achieve.They can also lead to subtle incompatibilities between different versions of the same algorithms. Even completely open standards bodies may fail to gather input from the right set of stakeholders. Standards bodies can act prematurely and standardise a version of a scheme that is later found to be deficient in some way, or where improved options only emerge later. Once the standard is set, in the absence of serious attacks, there may be little incentive to change it. The history of PKE schemes based on RSA illustrates this well (see Section 18.1.7.3): RSA with PKCS#1 v1.5 encoding has led to many security issues and the introduction of attack-specific work-arounds; RSA with PKCS#1 v2.1 encoding (RSA-OAEP) has been standardised for many years but has not become widely used; meanwhile even better ways of turning RSA into a KEM or a PKE have been discovered but have not become mainstream.

Standards bodies are also subject to “regulatory capture”, whereby groups representing specific national or commercial interests have the potential to influence the work of a standards body. For example, NSA had a role in the design of the DES algorithm [1671, pp. 232-233], and, on another occasion, supplied the overall design of the Dual_EC_DBRG pseudorandom generator that was specified in a NIST standard [1672], along with certain critical parameters [1673, p. 17]. In such contexts, transparency as to the role of any national or commercial stakeholders is key. For instance, NIST have reviewed their cryptographic standardisation process [1673] to increase transparency and decrease reliance on external organisations.

Other standards bodies relevant for cryptography include ETSI (which is active in post-quantum cryptographic standardisation, as discussed immediately below) and IEEE (which developed an early series of standards for Public Key Cryptography, IEEE P1363).

18.1.16 Post-quantum Cryptography

Large-scale quantum computers, if they could be built, would severely threaten RSA-based and discrete-log-based cryptographic algorithms in both finite field and elliptic curve settings. This includes almost all of the Public Key Cryptography in use today. This is because of Shor’s algorithm [1674], a quantum algorithm that leads to polynomial-time algorithms for both IFP and DLP in both finite field and the elliptic curve settings. This stands in strong contrast to the situation with the best known classical, non-quantum, algorithms for these problems which are super-polynomial, but sub-exponential for IFP and the DLP in finite fields (and fully exponential for the DLP in the elliptic curve setting). Quantum computers also have some consequences for symmetric algorithms due to Grover’s algorithm [1675], which in theory provides a quadratic speed-up for exhaustive key search. However, the impact there is less substantial — as a rule of thumb, doubling the key size is sufficient to thwart quantum attacks.

Post-quantum cryptography (PQC) refers to the study of cryptographic algorithms and schemes that are plausibly secure against large-scale quantum algorithms. Such algorithms and schemes are still classical: they are designed to run on classical computers. We use the term “plausibly” here, since the field of quantum algorithms is still young and it is hard to anticipate future developments. Note that PQC is sometimes referred to as quantum-safe,

[12] See **https://irtf.org/cfrg**.

quantum resistant, or quantum-immune cryptography. PQC has been an active but niche research field for many years.

In late 2016, in response to projected progress in scaling quantum computing and recognising the long transition times needed for introducing new cryptographic schemes, NIST launched a process to define a suite of post-quantum schemes.[13] The focus of the NIST process is on KEMs and digital signature schemes, since the threat quantum computing poses for symmetric schemes is relatively weaker than it is for public key schemes. At the time of writing in mid 2021, the process (actually a competition) has entered its third round and a set of finalist schemes has been selected, alongside a set of alternate, or back-up schemes. The NIST process should result in new NIST standards in the mid 2020s.

It will be a significant challenge to integrate the new schemes into widely-used protocols and systems in a standardised way. This is because the NIST finalists have quite different (and usually worse) performance profiles, in terms of key sizes, ciphertext or signature size and computation requirements, from existing public key schemes. Work is underway to address this challenge in IETF and ETSI and some deployment experiments have been carried out for the TLS protocol by Google and CloudFlare.[14] It is likely that post-quantum schemes will initially be deployed in *hybrid* modes alongside classical public key algorithms, to mitigate against immaturity of implementation and security analysis. The recent deployment experiments used hybrid modes.

18.1.17 Quantum Key Distribution

PQC should be distinguished from quantum cryptography, which attempts to harness quantum effects to build secure cryptographic schemes. The most mature branch of quantum cryptography is Quantum Key Distribution (QKD). A QKD protocol typically uses polarised photons to transmit information from one party to another, in such a way that an eavesdropper trying to intercept the signals will disturb them in a detectable way. The two parties can engage in a resolution protocol over a classical, authenticated channel that leads to them agreeing on keying material about which even a computationally unbounded adversary has minimal information.

QKD is often marketed as offering unconditional security assuming only the correctness of the known laws of physics. In terms of commercial deployment, QKD faces severe challenges. The main one is that it does not solve a problem that we cannot solve satisfactorily using other means. Moreover, those means are already commoditised. Subsidiary issues are: the need for expensive special-purpose hardware, the need for an authentic channel (if we have such a channel, then we could use it to distribute public keys instead), limitations on range that stand in opposition to standard end-to-end security requirements, limitations on the rate at which secure keys can be established (leading to hybrid QKD/classical systems, thereby obviating any unconditional security guarantees), and the fact that theoretical guarantees of unconditional security are hard to translate into practice.

[13] See **https://csrc.nist.gov/projects/post-quantum-cryptography**.
[14] See **https://blog.cloudflare.com/the-tls-post-quantum-experiment**.

18.1.18 From Schemes to Protocols

In this section, we have focused on low-level cryptographic algorithms and schemes. In real systems, these are combined to build more complex systems. Often, the result is a collection of interactive algorithms, commonly called a cryptographic protocol. An overall cryptographic system may use a number of sub-systems and protocols. We will look briefly at a few specific systems in Section 18.5. Here, we restrict ourselves to general comments about such systems.

First, to reiterate from Section 18.1.12, even complex cryptographic protocols and systems, and their security properties, are amenable to rigorous definitions and analysis. The approach is to analyse the security of such systems in terms of simpler, easier to analyse security properties of their components. We have seen a simple example of this in our treatment of AE Section 18.1.6 in, where a generic composition approach allows the AE(AD) security of the EtM composition to be established based on the IND-CPA security of its "E" component and the SUF-CMA security of its "M" component.

This process could be carried to the next level. Consider, for example, building a unidirectional secure channel protocol, assuming a suitable symmetric key is already in place at the sender and receiver. First we need a definition of what functionality and security such a protocol should provide. For example, we could demand integrity and confidentiality of plaintexts sent and that an adversary that tries to reorder, drop, or replay ciphertexts can be detected. Suitable formal definitions capturing these requirements can be found in [1676].

Then we can try to realise the unidirectional secure channel protocol from a nonce-based AEAD scheme and prove that its security follows from (or can be reduced to) the standard security definition for AEAD security. Here, a candidate construction is to make the sender stateful, by having it maintain a counter for the number of plaintexts encrypted. That counter is encoded as the nonce for the AEAD scheme. The receiver maintains an independent copy of the counter, using it as the nonce when performing decryption. Intuitively, confidentiality and integrity for individual ciphertexts in the secure channel follows immediately from the AEAD security definition. Meanwhile, an adversary tampering with the order of ciphertexts will lead to the receiver using the wrong counter value when decrypting, which leads to an error by the integrity properties of the AEAD scheme. These ideas can and should be formalised.

We can go even further and build a bidirectional secure channel protocol from a unidirectional one. Here, additional considerations arise from the possibility of reflection attacks and whether the channel should preserve the joint ordering of ciphertexts in both directions. We can also consider secure channel protocols in which the protocol recovers from accidental packet losses or reordering arising from the underlying network transport, or where such errors are fatal and lead to termination of the protocol. We can try to remove the assumption of having pre-established symmetric keys by bringing a key exchange component into play as a separate or integrated sub-protocol.

Again, all these aspects can be formally defined and analysed. However, the challenges in dealing with the complexity inherent in such systems should already be apparent, especially when the models and proofs are all hand-generated. For this reason, it is common to make some kind of "cryptographic core" of a system the focus of analysis and to abstract away many details. For example, a typical analysis will completely ignore all key management aspects, including PKI (which we discuss in Section 18.3.8). Instead, it is simply assumed that all keys are authentic and where they need to be, as we did in the example above. However, these details are relevant to the overall security of the system. So too much abstraction brings the risk of missing important facets or making assumptions that are not warranted in practice.

It is then important to be clear about what is — and is not — being formally specified and analysed.

An alternative approach to taming complexity is to use mechanised tools, letting a computer do the heavy lifting. However, the currently available tools are quite difficult to use and require human intervention and, more often than not, input from the tool designer. One of the more successful approaches here is to use a symbolic model of the cryptographic primitives rather than a computational one as we have been considering so far. This provides a level of abstraction that enables more complex protocols to be considered, but which misses some of the subtleties of the computational approach. Symbolic approaches to formal analysis are covered in more detail in Formal Methods for Security Knowledge Area (Chapter 13).

18.2 CRYPTOGRAPHIC IMPLEMENTATION

[405, 1453, 1637, 1677]

So far, we have focused on describing cryptographic algorithms and schemes in terms of abstract algorithms or in mathematical terms. In research papers, new schemes are usually presented as pseudo-code. Of course, this all needs to be translated into actual code (or hardware) for practical use. In this section, we briefly discuss some of the considerations that arise in this process.

18.2.1 Cryptographic Libraries

From the perspective of the developer, cryptography is usually consumed via a cryptographic library, that is, a collection of algorithms and schemes accessed through an API. Many different cryptographic libraries are available, in many different programming languages, though 'C' and Java libraries are most common. Different libraries have different licence restrictions, though many are available under non-restrictive "open source" licences of various kinds.

Some libraries (e.g. OpenSSL[15] or BouncyCastle[16]) are richly featured, supporting many different cryptographic schemes and processes and can be used across a wide range of applications. Others are much more restrictive and designed to support only certain use cases. In part, this reflects the taste of the libraries' authors, but also age and available development resources.

Some libraries are better maintained than others. For example, prior to the Heartbleed vulnerability (discussed in Section 18.3.5), OpenSSL had fallen into a state of some ossification and disrepair. Consequently, Google and OpenBSD separately decided to "fork" OpenSSL, that is to create entirely separate development branches of the library, resulting in the BoringSSL and LibreSSL libraries. Heartbleed also resulted in a broader realisation of how important OpenSSL was to the whole Internet ecosystem. A sequence of reforms of the OpenSSL project followed and today the project is in much better shape, with a larger team of core developers, better funding and more active development.

Some cryptographic libraries are developed by professional software engineers with considerable experience in avoiding some of the pitfalls we discuss in this section and elsewhere. Others are not. In many cases, the developers are working on a volunteer basis; most of

[15] https://www.openssl.org/
[16] https://www.bouncycastle.org/

OpenSSL's code development is done in this way. As part of its support model, a cryptographic library should have a clear process for notifying its maintainers of bugs and security vulnerabilities. The library's developers should commit to address these in a timely manner.

18.2.2 API Design for Cryptographic Libraries

The API that a cryptographic library presents to its consumers is critical. There is a delicate balance to be struck between flexibility (allowing developers to use the library in a wide variety of ways, thereby making it more useful) and security (restricting the API in an effort to prevent developers from using the library in insecure ways). Consider the problem of providing an API for symmetric encryption. Should the library allow direct access to a raw block cipher capability? Possibly, since some developers may need that functionality at some point. But also perhaps not — since it's probable that an inexperienced developer will use the block cipher in ECB mode to perform bulk encryption, with predictably insecure results.[17] This simple example is not an isolated one. It could be replaced, for example, with one involving nonces in an API for AEAD, or one involving the selection of parameters for a primality test.

Green and Smith [405] present ten principles for API design for cryptographic libraries. Their principles are derived empirically from their analysis of an *ad hoc* collection of examples and from interviews with developers. More systematic approaches, relying on standard methodologies from social science, have followed, see [1678] for an illustrative example of this line of work.

Green and Smith observe that developers' mistakes affect many users, so it makes sense to focus on them and not the actual end users, who are typically the target of usable security research. They point out that cryptographic libraries appear to be uniquely prone to misuse by developers, with even subtle misuse leading to catastrophic security failures. They also argue that as the use of cryptography in applications becomes more common, so cryptographic libraries are increasingly used by developers without cryptographic expertise.

Green and Smith's ten principles can be summarised as follows:

1. Integrate cryptographic functionality into standard APIs; that is, hide cryptography from developers where possible.
2. Make APIs sufficiently powerful to satisfy both security and non-security requirements. The argument here is that developers ultimately don't have a security goal in mind and satisfying their actual requirements, ascertained through interviewing them, will encourage them to use an API rather than writing their own cryptographic code.
3. Design APIs that are easy to learn without cryptographic expertise.
4. Don't break the developer's paradigm (or mental model) of what the API should look like.
5. Design APIs that are easy to use even without reading the documentation (since developers will not read it!).
6. Create APIs that are hard to misuse — visible errors should result from incorrect usage.
7. APIs should have safe and unambiguous defaults.
8. APIs should have testing modes, because otherwise developers will hack the API to turn off security during development to ease testing, but then may fail to properly remove

[17] See **https://blog.filippo.io/the-ecb-penguin/** for a vivid illustration of the limitations of ECB mode.

their hacks. An issue here is that the resulting code could be released with the testing mode still enabled, but one would hope that regular software assurance would detect this before release.

9. Code that uses the API should be easy to read and maintain. For example, iteration counts for password hashing should not be set by a developer via the API, but instead internally in the library. One issue here is that the internal defaults may be overkill and hurt performance in some use cases. This relates to the tension between flexibility and security.
10. The API should assist with or handle end user interaction, rather than leave the entire burden of this to the developer using the API. Here, error messages are highlighted as a particular concern by Green and Smith: the API and the library documentation should help developers understand what failure modes the library has, what the security consequences of these are and how the resulting errors should be handled by the calling code.

For additional references and discussion, see Human Factors Knowledge Area (Section 4.6.2).

18.2.3 Implementation Challenges

Having discussed libraries and their APIs, we now turn to challenges arising in securely implementing the algorithms and schemes within these libraries.

The main problem is to translate a purely mathematical or pseudo-code description of a scheme (the typical unit of analysis in formal security proofs arising in research papers) into running code on a real computer in such a way that the abstraction level involved in the security analysis is still properly respected by the running code. Put another way, the challenge is to ensure there are no mechanisms through which sensitive information can leak that are not already anticipated and eliminated by the security analysis. There are multiple ways in which such leakage can arise. We consider a representative selection here.

18.2.3.1 Length Side Channels

As we noted in Section 18.1.6, the usual security goal of an AEAD scheme does not guarantee that the length of plaintexts will be hidden. Indeed, AEAD schemes like AES-GCM make it trivial to read off the plaintext length from the ciphertext length. However, it is clear that length leakage can be fatal to security. Consider a simplistic secure trading system where a user issues only two commands, “BUY” or “SELL”, with these commands being encoded in simple ASCII and sent over a network under the protection of AES-GCM encryption. An adversary sitting on the network who can intercept the encrypted communications can trivially infer what commands a user is issuing, just by looking at ciphertext lengths (the ciphertexts for “SELL” will be one byte longer than those for “BUY”). More generally, attacks based on *traffic analysis* and on the analysis of metadata associated with encrypted data can result in significant information leaking to an adversary.

18.2.3.2 Timing Side Channels

The amount of time that it takes to execute the cryptographic code may leak information about the internal processing steps of the algorithm. This may in turn leak sensitive information, e.g. information about keys. The first public demonstration of this problem was made by Kocher [1453] with the attacker having direct access to timing information. Later it was shown that such attacks were even feasible remotely, i.e. could be carried out by an attacker located at a different network location from the target, with timing information being polluted by network noise [1679].

Consider for example a naive elliptic curve scalar multiplication routine which is optimised to ignore leading zeros in the most significant bits of the scalar. Here we imagine the scalar multiplication performing doubling and adding operations on points, with the operations being determined by the bits of the scalar from most significant to least significant. If the adversary can somehow time the execution of the scalar multiplication routine, it can detect cases where the code finishes early and infer which scalars have some number of most significant bits equal to zero. Depending on how the routine is used, this may provide enough side channel information to enable a key to be recovered. This is the case, for example, for the ECDSA scheme, where even partial leakage of random values can be exploited. Recent systematic studies in this specific setting [1680, 1681] show that timing attacks are still pertinent today.

18.2.3.3 Error Side Channels

Errors arising during cryptographic processing can also leak information about internal processing steps. Padding oracle attacks on CBC mode encryption, originally introduced in [1636], provide a classic and persistent example of this phenomenon. CBC mode uses a block cipher to encrypt plaintext data that is a multiple of the block cipher's block length. But in applications, we typically want to encrypt data of arbitrary length. This implies that data needs to be padded to a block boundary of the block cipher before it can be encrypted by CBC mode. Vaudenay observed that, during decryption, this padding needs to be removed, but the padding may be invalidly formatted and the decryption code may produce an error message in this case. If the adversary can somehow observe the error message, then it can infer something about the padding's validity. By carefully constructing ciphertexts and observing errors arising during their decryption, an adversary can mount a plaintext recovery attack via this error side channel.

In practice, the error messages may themselves be encrypted, but then revealed via a secondary side channel, e.g. a timing side channel (since an implementation might abort further processing once a padding error is encountered). For examples of this in the context of SSL/TLS and which illustrate the difficulty of removing this class of side channel, see [1637, 1682].

18.2.3.4 Attacks Arising from Shared Resources

The cryptographic code may not be running in perfect isolation from potential adversaries. In particular, in modern CPUs), there is a memory cache hierarchy in which the same fast memory is shared between different processes, with each process potentially overwriting portions of the cache used by other processes. For example, in a cloud computing scenario, many different users' processes may be running in parallel on the same underlying hardware, even if they are separated by security techniques like virtualisation. So an attacker, running in a separate process in the CPU, could selectively flush portions of the cache and then, after the victim process has run some critical code, observe by timing its own cache accesses, whether that part of the cache has been accessed by the victim process or not. If the victim process has a pattern of memory access that is key-dependent, then this may indirectly leak information about the victim's key. This particular attack is known as a Flush+Reload attack and was introduced in [1683]; several other forms of cache-based attack are known. The possibility of such attacks was first introduced in [1684]; later such attacks were shown to be problematic for AES in particular [1685].[18] In the last few years, researchers have had a field day developing cache-based and related micro-architectural attacks against cryptographic implementations. These attacks arise in general from designers of modern CPUs making architectural compromises in search of speed.

18.2.3.5 Implementation Weaknesses

More prosaically, cryptographic keys may be improperly deleted after use, or accidentally written to backup media. Plaintext may be improperly released to a calling application before its integrity has been verified. This can occur in certain constructions where MAC verification is done after decryption and also in streaming applications where only a limited-size buffer is available for holding decrypted data.

18.2.3.6 Attacks Arising from Composition

A system making use of multiple cryptographic components may inadvertently leak sensitive information through incorrect composition of those components. So we have leakage at a system level rather than directly from the individual cryptographic components. Consider the case of Zcash,[19] an anonymous cryptocurrency. Zcash uses a combination of zero-knowledge proofs, a PKE scheme and a commitment scheme in its transaction format. The PKE scheme is used as an outer layer and is anonymous, so the identity of the intended recipient is shielded. How then should a Zcash client decide if a transaction is intended for it? It has to perform a trial decryption using its private key; if this fails, no further processing is carried out. Otherwise, if decryption succeeds, then further cryptographic processing is done (e.g. the commitment is checked). This creates a potentially observable difference in behaviour that breaks the intended anonymity properties of Zcash [1686]. The PKE scheme used may be IND-CCA secure and anonymous, but these atomic security properties do not suffice if the overall system's behaviour leaks the critical information.

[18] See also **https://cr.yp.to/antiforgery/cachetiming-20050414.pdf** for contemporaneous but unpublished work.
[19] See **https://z.cash/**.

18.2.3.7 Hardware Side Channels

Cryptography is often implemented directly in hardware. For example, hardware acceleration of cryptographic functions was once common, both in low-cost environments such as payment cards and in higher-end applications, such as server-side SSL/TLS operations. Today, Internet-of-Things (IoT) deployments may use hardware components to implement expensive cryptographic functions. Hardware-based cryptography can also be found in Trusted Platform Modules (TPMs) as specified by the Trusted Computing Group and in systems like Intel Software Guard eXtensions (SGX) and ARM Trustzone. As noted in Section 18.1.3, modern CPUs) have instructions to enable high performance implementation of important cryptographic algorithms like AES.

There are additional sources of leakage in hardware implementations of cryptography. For example, an attacker against a smartcard might be able to observe how much power the smartcard draws while carrying out its cryptographic operations at a fine-grained time resolution and this might reveal the type of operation being carried out at each moment in time. To give a more specific example, in an implementation of RSA decryption using a basic "square and multiply" approach, the two possible operations for each private key bit — either square or square & multiply — could consume different amounts of power and thus the private key can be read off bit-by-bit from a power trace. The electromagnetic emissions from a hardware implementation might also leak sensitive information. Even sonic side channels are possible. For example the first working QKD prototype [1687] reportedly had such a side channel, since an observer could listen to the optical components physically moving and thereby learn which polarisation was being used for each signal being sent. This highlights just one of the many challenges in achieving unconditional security according to the laws of physics.

For a fuller discussion of hardware side channels, we refer the reader to Hardware Security Knowledge Area (Chapter 20).

18.2.3.8 Fault Attacks

Hardware implementations may also be vulnerable to fault or glitch attacks, where an error is introduced into cryptographic computations at a precise moment resulting in leakage of sensitive data (typically keys) via the output of the computation. The first such attack focused on implementations of RSA using the CRT [1688]. A more recent incarnation of this form of attack called Rowhammer targets the induction of faults in memory locations where keys are stored by repeatedly writing to adjacent locations [989].

18.2.4 Defences

General techniques for defending against cryptographic implementation vulnerabilities (as opposed to weaknesses in the algorithms and schemes themselves) come from the fields of software and hardware security and are well-summarised in [1689, 1690]. Indeed, it can be argued that conventional software security may be more important for cryptographic code than for other forms of code. For hardware, blinding, masking, threshold techniques and physical shielding are commonly used protections. For software, common techniques include formal specification and verification of software and hardware designs, static and dynamic analysis of code, fuzzing, information flow analysis, the use of domain-specific languages for generating cryptographic code and the use of strong typing to model and enforce security properties. Most of the software techniques are currently supported only by experimental

tools and are not at present widely deployed in production environments. Additionally, the objects they analyse — and therefore the protections they offer — only extend so far, down to code at Instruction Set Architecture level at best.

Length side channels can be closed by padding plaintexts to one of a set of predetermined sizes before encryption and by adding cover or dummy traffic. Secure communications protocols like SSL/TLS and IPsec have features supporting such operations, but these features are not widely used in practice.

A set of coding practices aim to achieve what is loosely called *Constant-Time Cryptography*. The core idea is to remove, through careful programming, any correlation between the values of sensitive data such as keys or plaintexts, and variables that can be observed by an adversary such as execution time. This entails avoiding, amongst other things, key-dependent memory accesses, key-dependent branching and certain low-level instructions whose running time is operand-dependent. It may also require writing high-level code in particular ways so as to prevent the compiler from optimising away constant-time protections.[20] Writing constant-time code for existing algorithms is non-trivial. In some cases, cryptographic designers have taken it into account from the beginning when designing their algorithms. For example, Bernstein's ChaCha20 algorithm[21] does so, while using certain coordinate systems makes it easier to achieve constant-time implementation of elliptic curve algorithms [1691].

18.2.5 Random Bit Generation

Cryptography relies on randomness in a crucial way. Most obviously, random bits are needed for symmetric keys, and for more complex key generation algorithms in the public key setting. But to achieve standard security notions such as IND-CPA security, PKE schemes need to have a randomised encryption algorithm. Fortunately, in the nonce-based AE setting, we can avoid the need for randomness during encryption. Some signature schemes have a randomised signing algorithm. This is the case for RSA PSS, DSA and ECDSA, for example.[22]

A failure to supply suitable randomness to such algorithms can have disastrous consequences. We already remarked on this in the context of DSA and ECDSA in Section 18.1.9. We will discuss examples for asymmetric key pair generation in Section 18.3.4.

So our cryptographic algorithms need to have access to "strong" random bit sources. To be generally applicable, such a source should offer a plentiful supply of bits that are independent and uniformly distributed, such that the adversary has no information about them. Specific algorithms may reveal their random bits, but general usage requires that they remain hidden.

In an ideal world, every computing device would be equipped with a True Random Bit Generator (TRBG)[23] whose output is hidden from potential adversaries. In practice, this has proven to be very difficult to achieve. Intel and AMD CPUs) do offer access to the post-processed output of a TRBG via the RDRAND instruction. However, the designs of these TRBGs are not fully open.

In the absence of a TRBG, common practice is for the operating system to gather data from weak, local entropy sources such as keyboard timings, disk access times, process IDs and packet arrival times, to mix this data together in a so-called *entropy pool* and then to extract pseudo-random bits from the pool as needed using a suitable cryptographic function (a

[20] An introduction to the paradigm can be found at **https://www.bearssl.org/constanttime.html**.
[21] See **https://cr.yp.to/chacha.html**.
[22] But signature schemes can always be derandomised by using a standard method, see [1692].
[23] Also called a True Random Number Generator (TRNG).

Pseudo-Random Number Generator, PRNG, using a seed derived from the entropy pool). Designs of this type are standardised by NIST in [1672]. They are also used in most operating systems but with a variety of *ad hoc* and hard-to-analyse constructions. Mature formal security models and constructions for random bit generators do exist, see [1677] for a survey. But this is yet another instance where practice initially got ahead of theory, then useful theory was developed, and now practice is yet to fully catch up again.

It is challenging to estimate how much true randomness can be gathered from the aforementioned weak entropy sources. In some computing environments, such as embedded systems, some or all of the sources may be absent, leading to slow filling of the entropy pool after a reboot — leaving a "boot time entropy hole" [1693, 1694]. A related issue arises in Virtual Machine (VM) environments, where repeated random bits may arise if they are extracted from the Operating System too soon after a VM image is reset [1695].

There has been a long-running debate on whether such random bit generators should be *blocking* or *non-blocking*: if the OS keeps a running estimate of how much true entropy remains in the pool as output is consumed, then should the generator block further output being taken if the entropy estimate falls below a certain threshold? The short answer is no, if we believe we are using a cryptographically-secure PRNG to generate the output, provided the entropy pool is properly initialised with enough entropy after boot. This is because we should trust our PRNG to do a good job in generating output that is computationally indistinguishable from random, even if not truly random. Some modern operating systems now offer an interface to a random bit generator of this "non-blocking-if-properly-seeded" type.

18.3 KEY MANAGEMENT

[1620, 1666, 1696, 1697]

Cryptographic schemes shift the problem of securing data to that of securing and managing keys. Therefore no treatment of applied cryptography can ignore the topic of key management. As explained by Martin [1620, Chapter 10], cryptographic keys are in the end just data, albeit of a special and particularly sensitive kind. So key management must necessarily involve all the usual processes involved in Information Security management, including technical controls, process controls and environmental controls.

An introductory treatment of key management can be found in the aforementioned [1620, Chapter 10]. A more detailed approach can be found in the NIST three part series [1666, 1696, 1697].

18.3.1 The Key Life-cycle

Keys should be regarded as having a life-cycle, from creation all the way to destruction.

Keys first need to be generated, which may require cryptographically secure sources of randomness, or even true random sources, in order to ensure keys have sufficient entropy to prevent enumeration attacks.

Keys may then need to be securely distributed to where they will be used. For example, a key may be generated as part of a smartcard personalisation process and injected into a smartcard from the personalisation management system through a physically secure channel; or a symmetric key for protecting a communication session may be established at a client

and server in a complex cryptographic protocol, perhaps with one party choosing the session key and then making use of PKE to transport it to the other party.

Keys may also be derived from other keys using suitable cryptographic algorithms known as *Key Derivation Functions*.

Keys need to be stored securely until they are needed. We discuss some of the main key storage options in more detail in the sequel.

Then keys are actually used to protect data in some way. It may be necessary to impose limits on how much data the keys are used to protect, due to intrinsic limitations of the cryptographic scheme in which they are being used. Keys may then need to be changed or updated. For example, the TLS specification in its latest version, TLS 1.3 [1480], contains recommendations about how much data each AEAD key in the protocol can be used to protect. These are set by analysing the security bounds for the employed AEAD schemes. TLS also features a key update sub-protocol enabling new keys to be established within a secure connection.

Keys may need to be revoked if they are discovered to have been compromised. The revocation status of keys must then be communicated to parties relying on those keys in a timely and reliable manner.

Keys may also need to be archived — put into long-term, secure storage — enabling the data they protect to be retrieved when needed. This may involve encrypting the keys under other keys, which themselves require management. Finally, keys should be securely deleted at the end of their lifetime. This may involve physical destruction of storage media, or carefully overwriting keys.

Given the complexity in the key life-cycle, it should be apparent that the key life-cycle and its attendant processes need to be carefully considered and documented as part of the design process for any system making use of cryptography.

We have already hinted that keys in general need to remain secret in order to be useful (public keys are an exception; as we discuss below, the requirement for public keys is that they be securely bound to identity of the key owner and their function). Keys can leak in many ways — through the key generation procedure due to poor randomness, whilst being transported to the place where they will be needed, through compromise of the storage system on which they reside, through side-channel attacks while in use, or because they are not properly deleted once exhausted. So it may be profitable for attackers to directly target keys and their management rather than the algorithms making use of them when trying to break a cryptographic system.

Additionally, it is good practice that keys come with what Martin [1620] calls *assurance of purpose* — which party (or parties) can use the key, for which purposes and with what limits. Certain storage formats — for example, digital certificates — encode this information along with the keys. This relates to the *principle of key separation* which states that a given key should only ever be used for one purpose (or in one cryptographic algorithm). This principle is perhaps more often broken than observed and has led to vulnerabilities in deployed systems, see, for example [1667, 1698].

18.3.2 Key Derivation

Key derivation refers to the process of creating multiple keys from a single key. The main property required is that exposure of any of the derived keys should not compromise the security of any of the others, nor the root key from which they are derived. This is impossible in an information theoretic sense, since given enough derived keys, the root key must be determinable through an exhaustive search. But it can be assured under suitable computational assumptions. For example, suppose we have a Pseudo-Random Function (PRF) F which takes as input key K and "message" input m; then outputs $F(K, m_i)$ on distinct inputs $m_1, m_2, \ldots$ will appear to be random values to an adversary and even giving the adversary many input/output pairs $(m_i, F(K, m_i))$ will not help it in determining K nor any further input/output pairs. Thus a pseudo-random function can be securely used as a Key Derivation Function (KDF) with root key K. We refer then to the m_i as *labels* for key derivation.

In many situations, the root key K may itself not be of a suitable length for use in a PRF or come from some non-uniform distribution. For example, the key may be a group element resulting from a Diffie-Hellman key exchange. In this case, one should first apply an entropy extraction step to make a suitable key K, then apply a PRF. One may also desire a key derivation function with variable length output (different functions may require keys of different sizes) or variable size label inputs. So the general requirements on a KDF go beyond what a simple PRF can offer. HKDF is one general-purpose KDF that uses the HMAC algorithm as a variable-input PRF. It is defined in [1699]. As well as key and label inputs and variable length output, it features an optional non-secret *salt* input which strengthens security across multiple, independent invocations of the algorithm. In legacy or constrained systems, one can find many *ad hoc* constructions for KDFs, using for example block ciphers or hash functions.

Using a KDF allows for key diversification and makes it easier to comply with the principle of key separation. For example, one can use a single symmetric key to derive separate encryption and MAC keys to be used in an EtM construction. Modern AE schemes avoid this need, effectively performing key derivation internally. In the extreme, one might use a KDF in combination with a specific label to derive a fresh key for each and every application of a cryptographic algorithm, a process known in the financial cryptography context as *unique key per transaction*. One can also choose to derive further keys from a derived key, creating a *key hierarchy* or *tree of keys* of arbitrary (but usually bounded) depth. Such key hierarchies are commonly seen in banking systems. For example, a bank may have a payment-system-wide master secret, from which individual card secrets are derived; in turn, per transaction keys are derived from the card secrets. Of course, in such a system, protection of the master secret — the key to the kingdom! — is paramount. Generally specialised hardware is used for storing and operating with such keys.

18.3.3 Password-Based Key Derivation

A very common practice, arising from the inevitable involvement of humans in cryptographic systems, is to derive cryptographic keys from passwords (or passphrases). Humans cannot remember passwords that have the high entropy required to prevent exhaustive searches, so special-purpose KDFs should be used in such applications, called password-based KDFs (PBKDFs). The idea of these is to deliberately slow-down the KDF to limit the speed at which exhaustive searches can be carried out by an attacker (standard KDFs do not need to be slowed in this way, since they should only operate on high-entropy inputs). Memory-hard PBKDFs such as scrypt (defined in [1700] and analysed in [1701]) or Argon2 (the winner

of a password hashing design competition[24]), which are designed to force an attacker to expend significant memory resources when carrying out exhaustive searches, should be used in preference to older designs. These PBKDFs have adjustable hardness parameters and permit salting, important for preventing pre-computation attacks on the collections of hashed passwords that are typically stored in authentication databases. It is worth keeping in mind that commercial password cracking services exist. These cater for many different formats and make extensive use of Graphical Processing Units (GPUs). They are impressively fast, rendering human-memorable passwords on the verge of being obsolete and forcing the adoption of either very long passwords or entirely different authentication methods.

An alternative to using a password directly to derive a key is to use the password as an authentication mechanism in a key exchange protocol, leading to the concept of Password Authenticated Key Exchange (PAKE). When designed well, a PAKE can limit an attacker to making a single password guess in each execution of the protocol. PAKE has not seen widespread adoption yet, but is currently undergoing standardisation in the IRTF.[25].

18.3.4 Key Generation

The main requirement for symmetric keys that are being generated from scratch is that they should be chosen close to uniformly at random from the set of all bit-strings of the appropriate key length. This requires access to good sources of random bits at the time when the key is selected. These bits may come from true random sources (e.g. from noise in electronic circuits or from quantum effects) or from an operating-system supplied source of pseudo-random bits, which may in turn be seeded and refreshed using random bits gathered from the local environment. This topic is discussed in more detail in Section 18.2. An alternative is to use a Physically Unclonable Function (PUF) to generate keying material from the intrinsic properties of a piece of hardware. Depending on what properties are measured, significant post-processing may be needed to correct for errors and to produce output with good randomness properties. An overview of some of the challenges arising for PUFs can be found in [1702].

For asymmetric algorithms there may be additional requirements, for example the key pairs may require special algebraic structure or need to lie in certain numerical ranges. However, the KeyGen algorithms for such schemes should handle this internally and themselves only require access to a standard random bit source.

There are (in)famous cases where key generation processes were not appropriately randomised [1693, 1703] (see also the Debian incident[26]). Another challenge is that in some cases keys need to be generated in constrained environments, e.g. on smartcards that will be used as personal identity cards, where generating the keys “off-card” and then injecting them into the card would not meet the security requirements of the application. There may then be a temptation to over-optimise the key generation process. This can result in significant security vulnerabilities [1704].

[24] See **https://www.password-hashing.net/**.
[25] Details at **https://github.com/cfrg/pake-selection**.
[26] See **https://www.debian.org/security/2008/dsa-1571**.

18.3.5 Key Storage

Once keys are generated, they typically need to be stored. An exception is where the keys can be generated on the fly from a human-memorable password, as discussed in Section 18.3.3.)

In general, the storage medium needs to be appropriately secured to prevent the keys becoming available to adversaries. In many cases, the only security for stored keys comes from that provided by the local operating system's access control mechanisms, coupled with hardware-enforced memory partitioning for different processes. Such security mechanisms can be bypassed by *local* attackers using low-level means that exploit the presence of shared resources between different executing processes, e.g. a shared memory cache. These are discussed in Section 18.2. This is a particular issue in multi-tenant computing environments, e.g. cloud computing, where a customer has little control over the processes running on the same CPU) as its own. But it is also an issue in single-tenant computing environments when one considers the possibility of malicious third-party software. An acute challenge arises for "crypto in the browser", where users must rely on the browser to enforce separation and non-interference between code running in different browser tabs, and where the code running in one tab may be loaded from a malicious website "`evil.example.com`" that is trying to extract cryptographic secrets from code running another tab "`your-bank.example.com`".

The Heartbleed incident [1559] was so damaging precisely because it enabled a fully *remote* attacker (i.e. one not running on the same machine as the victim process but merely able to connect over a network to that machine) to read out portions of a TLS server's memory, potentially including the server's private key. It shows that any amount of strong cryptography can easily be undermined thanks to a simple software vulnerability. In the case of Heartbleed, this was a buffer over-read in the OpenSSL implementation of the TLS/DTLS Heartbeat protocol.

Developers who store cryptographic keys in memory (or in software) often use software obfuscation techniques to try to make it harder to identify and extract the keys. Correspondingly, there are de-obfuscation tools which try to reverse this process and there is a long-running arms race in this domain. The research topic of white-box cryptography [1705] attempts to formalise this game of attack and defence.

The alternative to in memory storage of keys is to rely on special purpose secure storage for keys. Devices for this exist at all scales and price points. For example, a smartcard may cost a few cents to manufacture, but can be well-protected against passive and invasive attacks that attempt to recover the keys that it stores. A well-designed smartcard will have an interface which carefully limits how an external card reader communicating with the card can interact with it. In particular, there will not be any commands that a reader can send to the card that would allow its keys to directly leave the card. Rather, a reader will only be able to send data to the card and have it cryptographically transformed locally using the stored keys, then receive the results of the cryptographic computations. Thus, the reader will be able to interact with the secrets on the card only via a cryptographic API. Smartcards are typically bandwidth- and computation-limited.

At the other end of the scale are Hardware Security Modules (HSMs), which may cost many thousands of US dollars, offer much greater capabilities and be tested to a high level of security using industry-standard evaluation methodologies (e.g. the NIST FIPS 140 series). HSMs are frequently used in the financial sector, and are also now offered in cloud environments, see Section 18.4. As with smartcards, an HSM offers key storage and key usage functions via a carefully controlled API. The API provided by an HSM can be used to extend the security that

the HSM offers to keys that it directly stores to a larger collection of keys. Consider the simple case of using the HSM to store a Key Encryption Key (KEK), such that the HSM's API allows that key to be used internally for authorised encryption and decryption functions. Then the HSM can be used to wrap and, when needed, unwrap many Data Encryption Keys (DEKs) using a single KEK that is stored inside the HSM. Here wrapping means encrypting and unwrapping means decrypting. Assuming the used encryption mechanism is strong, the wrapped DEKs can be stored in general, unprotected memory. Further details on HSMs can be found in the Hardware Security Knowledge Area (Chapter 20).

TPMs also provide hardware-backed key storage. Technologies aiming to provide secure execution environments, such as Intel SGX and ARM Trustzone, enable secure storage of keys but also possess much more general capabilities. On mobile devices, the Android and iOS operating systems offer similar key storage features through Android Keystore and iOS Secure Enclave — both of these are essentially mini-HSMs.

18.3.6 Key Transportation

Depending on system design, keys may be generated at one place but need to be transported to another place where they will subsequently be used. Traditionally, secure couriers were used to physically transport keys stored on paper tape or magnetic media. Of course, this is costly and time-consuming. A related method, used often in low-value consumer applications (e.g. domestic wireless routers), is to simply print the key on the back of the device. Then security is reduced to that of the physical security of the device.

An alternative approach, widely used in the mobile telecommunications and consumer finance sector, is to inject symmetric keys into low-cost security modules (e.g. a Subscriber Identity Module, SIM, card in the case of mobile telephones or a bank card in the case of finance) and distribute those to customers. At the same time, copies of the symmetric keys may be kept *en masse* at a centralised location. In the case of mobile telecommunications, this is at a logical network component called the Authentication Centre (AuC); the symmetric keys are then used to perform authentication of SIM cards to the network and as the basis for deriving session keys for encrypting voice and data on the wireless portion of the network. In the financial setting, the symmetric keys injected into cards are typically already derived from master keys using a KDF, with the master keys being held in an HSM.

If a key is already in place, then that key can be used to transport fresh keys. Here, the idea is to sparingly use an expensive but very secure algorithm to transport the keys. With the advent of Public Key Cryptography, the problem of key transportation can be reduced to the problem of establishing an authentic copy of the public key of the receiver at the sender. As we discuss in Section 18.3.8, this is still a significant problem. Earlier versions of the TLS protocol used precisely this mechanism (with RSA encryption using PKCS#1 v1.5 padding) to securely transport a master key from a client in possession of a server's public key to that server. In TLS, various session keys are derived from this master key (using an *ad hoc* KDF construction based on iteration of MD5 and SHA-1).

18.3.7 Refreshing Keys and Forward Security

Consider a scenario where a symmetric key is being used to protect communications between two parties A and B, e.g. by using the key in an AEAD scheme as was described in Section 18.1.18. Suppose that key is compromised by some attack — for example, a side-channel attack on the AEAD scheme. Then the security of *all* the data encrypted under that key is potentially lost. This motivates the idea of regularly refreshing keys. Another reason to do this, beyond key compromise, is that the formal security analysis of the AEAD scheme will indicate that any given key can only be used to encrypt a certain amount of data before the security guarantees are no longer meaningful.

Several mechanisms exist to refresh symmetric keys. A simple technique is to derive, using a KDF, a new symmetric key from the existing symmetric key at regular intervals, thereby creating a chain of keys. The idea is that if the key is compromised at some point in time, then it should still be hard to recover all previous keys (but of course possible to compute forward using the KDF to find all future keys). This property, where the security of old keys remains after the compromise of a current key, is somewhat perversely known as *forward security*. In our example it follows from standard security properties of a KDF. In practice, hashing is often improperly used in place of a KDF, leading to the notion of a *hash chain*. This approach does not require direct interaction between the different users of a given symmetric key (i.e. A and B in our example). But it does need a form of synchronisation so that the two parties know which version of the key to use.

Another idea is to use a PKE scheme (more properly a KEM) to regularly transport a fresh symmetric key from one party to the other, e.g. from A to B under the protection of B's public key and then use that *session* key in the AEAD scheme (as discussed above was done in earlier versions of TLS). This creates fresh session keys whenever needed. Now to fully evaluate the system under key compromises, we should consider the effect of compromise of B's PKE private key too. This takes us back to square one: if the attacker can intercept the PKE ciphertexts as they are sent, store them and later learn B's PKE private key by some means (e.g. by factorising the modulus in the case of RSA, or simply as a result of an order made by a lawful authority), then it can recover all the session keys that were ever transported and thereby decrypt all the AEAD ciphertexts. In short, if the capabilities of the adversary include PKE private key compromise, then the use of PKE does not achieve forward security.

To improve the situation, we can make use of *ephemeral* Diffie-Hellman key exchange, in combination with appropriate authentication to prevent active MiTM attacks, to set up fresh session keys. Here, *ephemeral* refers to the Diffie-Hellman values used being freshly generated by both parties in each key exchange instance. We might use digital signatures for the authentication, as TLS 1.3 does. Now what happens in the event of compromise of the equivalent of the KEM private key? This is the signing key of the digital signature scheme. We see, informally, that an active MiTM attacker who knows the signing key could spoof the authentication and fool the honest protocol participants into agreeing on fresh session keys with the attacker rather than each other. This cannot be prevented. However, a compromise at some point in time of the signing key has no effect on the security of previously established, old session keys. If the authentication was sound at the time of the exchange, so that active attacks were not possible, then an adversary has to break a Diffie-Hellman key exchange in order to learn a previously established session key. Moreover breaking one Diffie-Hellman key exchange does not affect the security of the others.[27] In comparison to the use of PKE

[27] Except in the case where the discrete logarithm algorithm used allows reuse of computation when solving multiple DLPs. This is the case for the best known finite field algorithms, see [438] for the significant real-world

to transport session keys, Diffie-Hellman key exchange achieves strictly stronger forward security properties.

Complementing forward security is the notion of backward security, aka post compromise, security [1706]. This refers to the security of keys established *after* a key compromise has occurred. Using Diffie-Hellman key exchange can help here too, to establish fresh session keys, but only in the event that the adversary is restricted to being passive for at least one run of the Diffie-Hellman protocol.

18.3.8 Managing Public Keys and Public Key Infrastructure

The main security requirement for a public key is that it be authentically bound to the identity of the party who is in legitimate possession of the corresponding private key. Otherwise impersonation attacks become possible. For example, a party A might encrypt a session key to what it thinks is the public key of B, but the session key can be recovered by some other party C. Or a party A might receive a digital signature on some message M purporting to be generated by B using its private key, but where the signature was actually generated by C; here A would verify the signature using C's (public) verification key thinking it was using B's key.

A second requirement is that parties who rely on the soundness of the binding between public key and identity are also able to gain assurance that the binding is still valid (i.e. has not expired or been revoked).

These two requirements have many implications. Meeting them leads to the introduction of additional infrastructure and management functions. Conventionally, this collection of components is called a *Public Key Infrastructure (PKI)*.

18.3.8.1 Binding Public Keys and Identities via Certificates

A suitable mechanism to bind public keys and identities — and possibly other information — is needed. In early proposals for deploying Public Key Cryptography, it was proposed that this could take the form of a trusted bulletin board, where all the public keys and corresponding identities are simply listed. But this of course requires trust in the provider of the bulletin board service.

A non-scalable and inflexible solution, but one that is commonly used in mobile applications and IoT deployments, is to hard-code the required public key into the software of the party that needs to use the public key. Here, security rests on the inability of an adversary to change the public key by over-writing it in a local copy of the software, substituting it during a software update, changing it in the code repository of the software provider, or by other means.

Another solution is to use *public key digital certificates* (or just certificates for short). These are data objects in which the data includes identity, public key, algorithm type, issuance and expiry dates, key usage restrictions and potentially other fields. In addition, the certificate contains a digital signature, over all the other fields, of some Trusted Third Party (TTP) who attests to the correctness of the information. This TTP is known as a Certification Authority (CA). The most commonly used format for digital certificates is X.509 version 3 [1707].

The use of certificates moves the problem of verifying the binding implied by the digital signature in the certificate to the authentic distribution of the CA's public key. In practice, the

security issues this can introduce, but not for the elliptic curve setting.

problem may be deferred several times via a *certificate chain*, with each TTP's public key in the chain being attested to via a certificate issued by a higher authority. Ultimately, this chain is terminated at the highest level by a root certificate that is self-signed by a root CA. That is, the root certificate contains the public verification key of the root CA and a signature that is created using the matching private signing key. A party wishing to make use of a user-level public key (called a relying party) must now verify a chain of certificates back to the root and also have means of assuring that the root public key is valid. This last step is usually solved by an out of band distribution of the root CA's public key. Root CAs may also cross-sign each other's root certificates.

As an important and visible example of a PKI, consider the *Web PKI*. Web browser vendors embed a list of the public keys of a few hundred different root CAs in their software and update the list from time to time via their software update mechanisms, which in turn may rely for its security on a separate PKI. Website owners pay to obtain certificates binding their sites' URLs to their public keys from subordinate CAs. Then, when running the TLS protocol for secure communications between a web browser and a website, the website's server sends a certificate chain to the web browser client. The chain provides the web browser with a copy of the server's public key (in the lowest certificate from the chain, the leaf or end-entity certificate) as well as a means of verifying the binding between the web site name in the form of its URL, and that public key. The operations and conventions of the Web PKI are managed by the CA/Browser Forum.[28]

18.3.8.2 Reliance on Naming, CA Operations and Time

In addition to needing a suitable binding mechanism, there must be a stable, controlled naming mechanism for parties. Moreover, parties need to have means of proving to CAs that they own a specific identity and CAs need to check such assertions. Equally, CAs need to be trusted to only issue certificates to the correct parties. This aspect of PKI intersects heavily with legal and regulatory aspects of Information Security and is covered in more detail in Law & Regulation Knowledge Area (Section 3.10.3).

For the Web PKI, there have been numerous incidents where CAs were found to have mis-issued certificates, either because they were hacked (e.g. DigiNotar[29]), because of poor control over the issuance process (e.g. TurkTrust[30]), or because they were under the control of governments who wished to gain surveillance capabilities over their citizens. This can lead to significant commercial impacts for affected CAs: in DigiNotar's case, the company went bankrupt. In other cases, CAs were found to not be properly protecting their private signing keys, leaving them vulnerable to hacking.[31] In response to a growing number of such incidents, Google launched the *Certificate Transparency (CT)* effort. CT provides an open framework for monitoring and auditing certificates; it makes use of multiple, independent public logs in an attempt to record all the certificates issued by browser-trusted CAs. The protocols and data formats underlying CT are specified in [1708].[32]

Relying parties (i.e. parties verifying certificates and then using the embedded public keys) need access to reliable time sources to be sure that the certificate's lifetime, as encoded in

[28] See **https://cabforum.org/**.
[29] See **https://en.wikipedia.org/wiki/DigiNotar**.
[30] See **https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/**.
[31] See for example the case of CNNIC, **https://techcrunch.com/2015/04/01/google-cnnic/**.
[32] See also **https://certificate.transparency.dev/** for the project homepage.

the certificate, is still valid. Otherwise, an attacker could send an expired certificate for which it has compromised the corresponding private key to a relying party and get the relying party to use the certificate's public key. This requirement can be difficult to fulfill in low-cost or constrained environments, e.g. IoT applications.

18.3.8.3 Reliance on Certificate Status Information

Relying parties verifying certificates also need access to reliable, timely sources of information about the status of certificates — whether the certificate is still valid or has been revoked for some security or operational reason. This can be done by regularly sending lists of revoked certificates to relying parties (known as Certificate Revocation Lists, CRLs), or having the relying parties perform a real-time status check with the issuing CA before using the public key using the Online Certificate Status Protocol, OCSP [1709]. The former approach is more private for relying parties, since the check can be done locally, but implies the existence of a window of exposure for relying parties between the time of revocation and the time of CRL distribution. The latter approach provides more timely information but implies that large CAs issuing many certificates need to provide significant bandwidth and computation to serve the online requests.

In the web context, OCSP has become the dominant method for checking revocation status of certificates. OCSP's bandwidth issue is ameliorated by the practice of *OCSP stapling*, wherein a web server providing a certificate regularly performs its own OCSP check and includes the certified response from its CA along with its certificate. In an effort to further improve user privacy, in 2020, Mozilla experimentally deployed[33] an approach called CRLite developed in [1710] in their Firefox browser. CRLite uses CT logs and other sources of information to create timely and compact CRLs for regular distribution to web browsers.

18.3.8.4 Reliance on Correct Software and Unbroken Cryptography

The software at a relying party that validates certificate chains needs to work properly. This is non-trivial, given the complexity of the X.509 data structures involved, the use of complex encoding languages and the need to accurately translate security policy into running code. There have been numerous failures. A prominent and entirely avoidable example is Apple's "goto fail" from 2014. Here a repeated line of code[34] for error handling in Apple's certificate verification code in its SSL/TLS implementation caused all certificate checking to be bypassed. This made it trivial to spoof a web server's public key in a fake certificate to clients running Apple's code. This resulted in a total bypass of the server authentication in Apple's SSL/TLS implementation, undermining all security guarantees of the protocol.[35]

The certificate industry has been slow to react to advances in the cryptanalysis of algorithms and slow to add support for new signature schemes. The story of SHA-1 and its gradual removal from the Web PKI is a prime example. This relates to the discussion of cryptographic agility in Section 18.1.14. The first cracks in SHA-1 appeared in 2005 [1711]. Already at this point, cryptographers taking their standard conservative approach, recommended that SHA-1 be deprecated in applications requiring collision resistance. From 2005 onwards, the crypt-analysis of SHA-1 was refined and improved. Finally in 2017, the first public collisions for

[33] See **https://blog.mozilla.org/security/2020/01/09/crlite-part-1-all-web-pki-revocations-compressed/**.
[34] The offending line of code was literally "`goto fail`".
[35] See **https://dwheeler.com/essays/apple-goto-fail.html** for a detailed write-up of the incident and its implications for Apple's software development processes.

SHA-1 were exhibited [1712]. This was followed in 2019 by a chosen-prefix collision attack that directly threatened the application of SHA-1 in certificates [1713]. However, despite the direction of travel having been clear for more than a decade, it took until 2017 before the major web browsers finally stopped accepting SHA-1 in Web PKI certificates. Today, SHA-1 certificates are still to be found in payment systems and elsewhere. The organisations running these systems are inherently change-averse because they have to manage complex systems that must continue to work across algorithm and technology changes. In short, these organisations are not cryptographically agile, as discussed in Section 18.1.14.

18.3.8.5 Other Approaches to Managing Public Keys

The *web of trust* is an alternative to hierarchical PKIs in which the users in a system vouch for the authenticity of one another's public keys by essentially cross-certifying each other's keys. It was once popular in the PGP community but did not catch on elsewhere. Such a system poses significant usability challenges for ordinary users [347].

Identity-Based Cryptography (IBC) [1714] offers a technically appealing alternative to traditional Public Key Cryptography in which users' private keys are derived directly from their identities by a TTP called the Trusted Authority (TA) in possession of a master private key. The benefit is that there is no need to distribute public keys; a relying party now needs only to know an identity and have an authentic copy of the TA's public key. The down-side for many application domains is that trust in the TA is paramount, since it has the capability to forge users' signatures and decrypt ciphertexts intended for them through holding the master private key. On the other hand, IBC's built-in key escrow property may be useful in corporate security applications. *Certificateless Cryptography* [1715] tries to strike a balance between traditional PKI and IBC. These and other related concepts have sparked a lot of scientific endeavour, but little deployment to date.

18.4 CONSUMING CRYPTOGRAPHY

The content of this section is based on the author's personal experience.

18.4.1 The Challenges of Consuming Cryptography

Numerous people, including those who are well educated in computer science and mathematics, can and do regularly make fundamental errors when attempting to devise novel cryptographic schemes or to implement existing mechanisms.

Perhaps one reason for this is that many people receive informal exposure to cryptography through popular culture, where it is often shown in a simplified way, for example using pen and paper ciphers or watching a password being revealed character-by-character through some unspecified search process. The subject also has basic psychological appeal — after all, who does not love receiving secret messages?

The issue shows up in several different ways. Least dangerous, the author has seen that cryptographic conferences and journals regularly receive submissions from people who are not aware of the advanced state-of-the-art. A classic trope is papers on encryption algorithms for digital images, where usually some low-level manipulation of pixel data is involved and security rests on taking a standard benchmark picture from the image processing community

and showing that it is visually scrambled by the algorithm. (Of course, image data is ultimately represented by bits and standard cryptographic algorithms operate on those bits.) Other topics common in such papers are chaos-based cryptography and combining multiple schemes (RSA, ElGamal, etc) to make a stronger one. The activity of generating such papers is a waste of time for the authors and reviewers alike, while it misleads students involved in writing the papers about the true nature of cryptography as a research topic.

This author has seen multiple examples where complete outsiders to the field have been persuaded to invest in cryptographic technologies which either defy the laws of information theory or which fall to the “kitchen sink” fallacy of cryptographic design — push the data through enough complicated steps and it must be secure. Another classic design error is for an inventor to fall under the spell of the “large keys” fallacy: if an algorithm has a very large key space, then surely it must be secure? Certainly a large enough key space is necessary for security, but it is far from sufficient. A third fallacy is that of “friendly cryptanalysis”: the inventor has tried to break the new algorithm themselves, so it must be secure. There is no substitute for independent analysis.

Usually these technologies are invented by outsiders to the field. They may have received encouragement from someone who is a consumer of cryptography but not themselves an expert or someone who is too polite to deliver a merciful blow. Significant effort may be required to dissuade the original inventors and their backers from taking the technology further. A sometimes useful argument to deploy in such cases is that, while the inventor’s idea may or may not be secure, we already have available standardised, carefully-vetted, widely-deployed, low-cost solutions to the problem and so it will be hard to commercialise the invention in a heavily commoditised area.

Another set of issues arise when software developers, perhaps with the best of intentions and under release deadline pressure, “roll their own crypto”. Maybe having taken an introductory course in Information Security or Cryptography at Bachelor’s level, they have accrued enough knowledge not to try to make their own low-level algorithms and they know they can use an API to a cryptographic library to get access to basic encryption and signing functions. However, with today’s cryptographic libraries, it is easy to accidentally misuse the API and end up with an insecure system. Likely the developer wants to do something more complex than simply encrypting some plaintext data, but instead needs to plug together a collection of cryptographic primitives to do something more complex. This can lead to the “kitchen sink” fallacy at the system level. Then there is the question of how the developer’s code should deal with key management — recall that cryptographic schemes only shift the problem of securing data to that of securing keys. Unfortunately, key management is rarely taught to Bachelor’s students as a first class issue and this author has seen that basic issues like hard-coded keys are still found on a regular basis in deployed cryptographic software.

18.4.2 Addressing the Challenges

How can individuals and businesses acting as consumers of cryptography avoid these problems? Some general advice follows.

There is no free lunch in cryptography. If someone without a track record or proper credentials comes bearing a new technological breakthrough, or just a new algorithm: beware. Consumers can look for independent analyses by reputable parties — there are companies and individuals who can provide meaningful cryptographic review. Any reputable consultant can supply a list of recent consulting engagements and contact details for references. Consumers can also check for scientific publications in reputable academic venues that support any new technology; very few crank ideas survive peer review, and consumers should look for clear indications of publication quality. Consumers can learn to detect cryptographic snake-oil, for example, by looking for instances of the kitchen sink, large keys and friendly cryptanalysis fallacies.[36]

Developers should not roll their own cryptographic algorithms. They should rely on vetted, standardised algorithms already available in packaged forms through cryptographic libraries. Ideally, they should not roll their own higher-level cryptographic systems and protocols, but rely on existing design patterns and standards. As just one example, there is usually no need to build a new, secure, application-layer communications protocol when SSL/TLS support is ubiquitous. Developers who are developing cloud-based solutions (an extremely common use-case) should make use of cryptographic services from cloud providers — an emerging approach called "Cryptography-as-a-Service" (CaaS). This is a natural extension of the "Software-as-a-Service" paradigm, but commonly extends it to include key management services. A key feature in commercial CaaS offerings is "HSM-as-a-service", allowing service users to avoid the cost and expertise needed to maintain on-premise HSMs.[37]

When these options are not possible, developers should seek expert advice (and not Crypto Stack Exchange [1716]). Very large organisations should have an in-house cryptography development team who vet all uses of cryptography before they go into production, ideally with involvement from the design stage. There is at least one large software company that today runs detection tools across its entire codebase to find instances where cryptography is being misused, and sends email alerts to the cryptography development team.

Smaller organisations for whom cryptography is a core technology should employ applied cryptographers with proven credentials in the field. Alternatively such medium-sized organisations — and the smallest companies — should cultivate relationships with trusted external partners who can provide cryptographic consulting services. The cost of such services is relatively small compared to the potential damage to reputation and shareholder value in the event of a major security incident arising from the improper use of cryptography.

[36] A more extensive list of snake-oil indicators can be found at: **http://www.interhack.net/people/cmcurtin/snake-oil-faq.html**.

[37] Without wishing to favour any particular vendors, interested readers can learn more about the typical features of such services at **https://aws.amazon.com/cloudhsm/** or **https://www.entrust.com/digital-security/certificate-solutions/products/pki/managed-services/entrust-cryptography-as-a-service**.

18.4.3 Making Cryptography Invisible

Ultimately, the best kind of cryptography is that which is as invisible as possible to its end users. Five years ago web browsers displayed locks of different colours to indicate whether SSL/TLS connections were secure or not, but users could always click-through to reach the website regardless. Today, web browsers like Google Chrome and Mozilla Firefox are more aggressive in how they handle SSL/TLS security failures and in some cases simply do not allow a connection to be made to the website.[38] There is a continuous tension here between protecting users and enabling functionality that users demand. For example, at the time of writing, users can still freely visit websites that do not offer SSL/TLS connections in both Chrome and Firefox, receiving a "not secure" warning in the browser bar; this behaviour may change in the future as more and more websites switch to supporting SSL/TLS.

Meanwhile, for more than 25 years, in most jurisdictions, mobile telephone calls have been encrypted between the mobile device and the base-station (or beyond) in order to prevent eavesdropping on the broadcast communications medium. However, even in the latest 5G systems, the encryption is optional [1717, Annex D] and can be switched off by the network operator. However, very few mobile telephones actually display any information to the user about whether encryption is enabled or not, so it is debatable to what extent users are really protected.

By contrast, secure messaging services like Signal offer end-to-end security by default and with no possibility of turning these security features off. Signal's servers are not able to read users' messages — as a January 2021 advert for Signal pointedly stated, "we know nothing". Cryptographic security and user privacy are core to the company's business model. So much so that if a significant cryptographic flaw were to be found in Signal's design, then, at the very least, it would lead to a significant loss of customers and possibly even lead to the company's downfall. Yet there is very little in-app security messaging — there is no equivalent of a browser lock that users are asked to interpret, for example.

18.5 APPLIED CRYPTOGRAPHY IN ACTION

[444, 1480, 1718]

Having explored the landscape of applied cryptography from the bottom up, we now take a brief look at three different application areas, using them to draw out some of the key themes of the KA.

[38] Readers interested in learning more about what different browsers do and do not allow can visit **https://badssl.com/** or **http://example.com**.

18.5.1 Transport Layer Security

The Transport Layer Security (TLS) protocol has already been mentioned several times in passing. The protocol has a long history, arriving at TLS 1.3 [1480], completed in 2018. It is a complex protocol with several interrelated sub-protocols: the TLS Handshake Protocol uses (usually) asymmetric cryptography to establish session keys; these are then used in the TLS Record Protocol in conjunction with symmetric techniques to provide confidentiality and integrity for streams of application data; meanwhile the TLS Alert Protocol can be used to transport management information and error alerts. TLS is now used to protect roughly 95% of all HTTP traffic.[39] In this application domain, TLS relies heavily on the Web PKI for server authentication.

TLS 1.3 represents a multi-year effort by a coalition of industry-based engineers and academics working under the aegis of the IETF to build a general-purpose secure communications protocol. The main drivers for starting work on TLS 1.3 were, firstly, to improve the security of the protocol by updating the basic design and removing outdated cryptographic primitives (e.g. switching the MtE construction for AEAD only; removing RSA key transport in favour of DHKE to improve forward security) and, secondly, to improve the performance of the protocol by reducing the number of communication round trips needed to establish a secure channel. Specifically, in earlier versions of TLS, two round trips were needed, while in TLS 1.3 only one is needed in most cases and even a zero round trip mode is supported in TLS 1.3 when keys have been established in a previous session.

The design process for TLS 1.3 is described and contrasted with the process followed for previous versions in [1661]. A key difference is the extensive reliance on formal security analysis during the design process. This resulted in many research papers being published along the way, amongst which we highlight [1719, 1720, 1721]. Some of the formal analysis was able to detect potential security flaws in earlier versions of the protocol, thereby influencing the protocol design in crucial ways — for example, the use of HKDF in a consistent, hierarchical manner to process keying material from different stages of the protocol. The protocol was also modified to make it more amenable to formal security analysis. It is an almost complete success story in how academia and industry can work together. Only a few criticisms can be levelled at the process and the final design:

- Not all of the security and functionality requirements were elicited at the outset, which meant many design changes along the way, with attendant challenges for researchers doing formal analysis. However, such an iterative approach seems to be unavoidable in a complex protocol designed to serve many use cases.
- The formal analyses missed a few corner cases, especially in the situation where authentication is based on pre-shared symmetric keys. An attack in one such corner case was subsequently discovered [1722].
- The zero round trip mode of TLS 1.3 has attractive latency properties but achieves these at the expense of forward security. Defending against replay attacks in this mode is difficult in general and likely impossible in distributed server settings when interactions with typical web clients are taken into account. Recent research showing how to add forward security to the zero round trip mode of TLS 1.3 can be found in [1723, 1724, 1725, 1726].

[39] See **https://transparencyreport.google.com/https/overview?hl=en** for Google data supporting this statistic.

18.5.2 Secure Messaging

On the surface, secure messaging seems to have a very similar set of requirements to TLS: we have pairs of communicating parties who wish to securely exchange messages. However, there are significant differences, leading to different cryptographic solutions. First, secure messaging systems are asynchronous, meaning that the communicating parties cannot easily run an equivalent of the TLS Handshake Protocol to establish symmetric keys. Second, there is no infrastructure analogous to the Web PKI that can be leveraged to provide authentication. Third, group communication, rather than simply pairwise communication, is an important use case. Fourth, there is usually a bespoke server that is used to relay messages between the pairs of communicating parties but which should not have access to the messages themselves.

We discuss three different secure messaging systems: Apple's iMessage, Signal (used in WhatsApp) and Telegram. We focus on the two-party case for brevity.

18.5.2.1 Apple iMessage

Apple's iMessage system historically used an *ad hoc* signcryption scheme that was shown to have significant vulnerabilities in [1727]. This was despite signcryption being a well-known primitive with well-established models, generic constructions from PKE and digital signatures and security proofs in the academic literature. Conjecturally, the designers of Apple's scheme were constrained by the functions available in their cryptographic library. The Apple system relied fully on trust in Apple's servers to distribute authentic copies of users' public keys — a PKI by fiat. The system was designed to be end-to-end secure, meaning that without active impersonation via key substitution, Apple could not read users' messages. It did not enjoy any forward-security properties, however: once a user's private decryption key was known, all messages intended for that user could be read. Note that Apple's iMessage implementation is not open source; the above description is based on the reverse engineering carried out in [1727] and so may no longer be accurate.

18.5.2.2 Signal

The Signal design, which is used in both Signal and WhatsApp, takes a slightly different approach in the two-party case. It uses a kind of asynchronous DHKE approach called *ratcheting*. At a high level, every time Alice sends user Bob a new message, she also includes a Diffie-Hellman (DH) value and updates her symmetric key to one derived from that DH value and the DH value she most recently received from Bob. On receipt, Bob combines the incoming DH value with the one he previously sent to make a new symmetric key on his side. This key is called a chaining key.

For each message that Alice sends to Bob without receiving a reply from Bob, she derives two new keys from the current chaining key by applying a KDF (based on HKDF) to it; one key is used as the next chaining key, the other is used to encrypt the current message. This is also called ratcheting by the Signal designers and the combination of ratcheting applied to both DH values and symmetric keys is called *double ratcheting*.[40] This mechanism provides forward security for Signal messages, despite its asynchronous nature. It also provides post compromise security. The use of ratcheting, however, entails problems with synchronisation:

[40] See **https://signal.org/docs/specifications/doubleratchet/** for a concise overview of the process.

if a message is lost between Alice and Bob, then their keys will end up in different states. This is solved by keeping caches of recent chaining keys.

For symmetric encryption, Signal uses a simple generic AE construction based on EtM relying on CBC mode using AES with 256 bit keys for the “E” component and HMAC with SHA-256 for the “M” component. This is a conservative and well-understood design.

Authentication in Signal is ultimately the same as in iMessage: it depends on trust in the server. The idea is that users register a collection of DH values at the server; these are fetched by other users and used to establish initial chaining keys. However, a malicious server could replace these values and thereby mount a MitM attack. The use of human-readable key fingerprints provides mitigation against this attack.

A formal security analysis of the double ratcheting process used by Signal can be found in [444]. Note that, in order to tame complexity, this analysis does not treat the composition of the double ratchet with the symmetric encryption component. The Signal design has spurred a spate of recent research into the question of what is the best possible security one can achieve in two-party messaging protocols and how that security interacts with the synchronisation issues.

18.5.2.3 Telegram

A third design is that used by Telegram.[41] It is notable for the way it combines various cryptographic primitives (RSA, finite field DHKE, a hash-based key derivation function, a hash-based MAC and a non-standard encryption mode called IGE). Moreover, it does not have proper key separation: keys used to protect messages from Alice to Bob share many overlapping bits with keys used in the opposite direction; moreover those key bits are taken directly from “raw” DHKE values. These features present significant barriers to formal analysis and violate cryptographic best practices. Furthermore, Telegram does not universally feature end-to-end encryption; rather it has two modes, one of which is end-to-end secure, the other of which provides secure communications only from each client to the server. The latter seems to be much more commonly used in practice, but is of course subject to interception. This is concerning, given that Telegram is frequently used by higher-risk users in undemocratic countries.

18.5.3 Contact Tracing à la DP-3T

The DP-3T project[42] was formed by a group of academic researchers in response to the COVID-19 pandemic, with the core aim of rapidly developing automated contact tracing technology based on mobile phones and Bluetooth Low Energy beacons. A central objective of DP-3T was to enable automated contact tracing in a privacy-preserving way, so without using location data and without storing lists of contacts at a central server. DP-3T’s approach directly influenced the Google-Apple Exposure Notification (GAEN)[43] system that forms the basis for dozens of national contact tracing apps around the world, including (after a false start) the UK system. An overview of the DP-3T proposal is provided in [1718].

The DP-3T design uses cryptography at its heart. Each phone generates a symmetric key and uses this as the root of a chain of keys, one key per day. Each day key in the chain is then used

[41] See **https://telegram.org/**.
[42] https://github.com/DP-3T/documents
[43] See **https://www.google.com/covid19/exposurenotifications/** and **https://covid19.apple.com/contacttracing**.

to generate, using a pseudo-random generator built from AES in CTR mode, a sequence of 96 short pseudo-random strings called *beacons*. At each 15-minute time interval during the day, the next beacon from the sequence is selected and broadcast using BLE. Other phones in the vicinity pick up and record the beacon-carrying BLE signals and store them in a log along with metadata (time of day, received signal strength). Notice that the beacons are indistinguishable from random strings, under the assumption that AES is a good block cipher.

When a user of the system receives a positive COVID-19 test, they instruct their phone to upload the recent day keys to a central server, possibly along with sent signal strength information.[44] All phones in the system regularly poll the server for the latest sets of day keys, use them to regenerate beacons and look in their local logs to test if at some point they came into range of a phone carried by a person later found to be infected. Using sent and received signal strength information in combination with the number and closeness (over time) of matching beacons, the phone can compute a risk score. If the score is above a threshold, then the phone user can be instructed to get a COVID-19 test themselves. Setting the threshold in practice to balance false positives against false negatives is a delicate exercise made more difficult by the fact that BLE permits only inaccurate range estimation.

Notice that the central server in DP-3T stores only day keys released from phones by infected parties. The central server is not capable of computing which users were in proximity to which other users, nor even the identity of users who uploaded keys (though this information will become visible to the health authority because of the issuance of authorisation codes). All the comparison of beacons and risk computations are carried out on users' phones. One can contend that a fully centralised system could provide more detailed epidemiological information — some epidemiologists unsurprisingly made this argument. On the other hand, the strict purpose of the DP-3T system was to enable automated contact tracing, not to provide an epidemiological research tool. A more detailed privacy analysis of DP-3T can be found in [1718].

The DP-3T design was produced, analysed, prototyped and deployed under test conditions all in the space of a few weeks. After adoption and adaptation by Google and Apple, it made its way into national contact tracing apps within a few months. Given the pace of development, simplicity of the core design was key. Only "off the shelf" cryptographic techniques available in standard cryptographic libraries could be used. Given the likely scaling properties of the system (possibly tens of millions of users per country) and the constraints of BLE message sizes, using Public Key Cryptography was not an option; only symmetric techniques could be countenanced. Many follow-up research papers have proposed enhanced designs using more complex cryptographic techniques. The DP-3T team did not have this luxury and instead stayed resolutely pragmatic in designing a system that balances privacy, functionality and ease of deployment, and that resists repurposing.

[44] To prevent spurious uploads, the day keys can only be uploaded after entering an authorisation code issued by the local health authority into the app.

18.6 THE FUTURE OF APPLIED CRYPTOGRAPHY

The first 2000 years of applied cryptography were mostly about securing data in transit, starting from the Caesar cipher and ending with today's mass deployment of TLS, secure messaging and privacy-preserving contact tracing systems. Today, cryptography is also heavily used to protect data at rest, the second element in our cryptographic triumvirate from the KA's introduction. These two application domains will continue to develop and become more pervasive in our everyday lives — consider for example the rise of the Internet of Things and its inherent need for communications security.

We anticipate significant developments in the following directions:

- The debate around lawful access to encrypted data will continue. However, we reiterate that in the face of determined but not even particularly sophisticated users, this is a lost battle. For example, it seems unlikely that state security agencies can break the cryptography that is used in Signal today. Instead, they will have to continue to bypass cryptographic protections through exploitation of vulnerabilities in end systems. Legal frameworks to enable this exist in many countries. Of course, governments could pass legislation requiring developers to include mechanisms to enable lawful access, or could pressure vendors into removing secure messaging applications from app stores.
- We expect that the current focus on cryptocurrencies and blockchains will result in a core set of useful cryptographic technologies. For example, anonymous cryptocurrencies have already been an important vehicle for forcing innovation in and maturation of zero-knowledge proofs.
- The third element in our cryptographic triumvirate was cryptography for data under computation. This area is undergoing rapid technical development and there is a healthy bloom of start-up companies taking ideas like FHE and MPC to market. A good overview of the status for "applied MPC" can be found in [463], while [1728] provides insights into deployment challenges specific to FHE. The idea of being able to securely outsource one's data to third party providers and allow them to perform computations on it (as FHE does) is very alluring. However, some 15 years after its invention, FHE still incurs something like a 10^6—10^8 times overhead compared to computing on plaintext data. This limits its application to all but the most sensitive data at small scales. Meanwhile, the core applications become ever more data- and computation-hungry, so remain out of reach of FHE for now. FHE, MPC and related techniques also face a significant challenge from trusted execution technologies like SGX. Approaches like SGX still rely on cryptography for attesting to the correct execution of code in secure enclaves, but can effectively emulate FHE functionality without such large overheads. On the other hand, SGX and related technologies have themselves been the subject of a series of security vulnerabilities so may offer less protection in practice than cryptographic approaches.
- One area where computing on encrypted data may have a medium-term commercial future is in specialised applications such as searching over encrypted data and more generally, database encryption. Current state-of-the-art solutions here trade efficiency (in terms of computation and bandwidth overheads) for a certain amount of leakage. Quantifying the amount of leakage and its impact on security is a challenging research problem that must be solved before these approaches can become widely adopted.[45]

[45] But see **https://docs.mongodb.com/manual/core/security-client-side-encryption/** for details of MongoDB's use of deterministic symmetric encryption to enable searchable field-level encryption.

- Another growth area for applied cryptography is in privacy-preserving techniques for data-mining and data aggregation. Google's privacy-preserving advertising framework [1729] provides one prominent example. Another is the Prio system [461] that allows privacy-preserving collection of telemetry data from web browsers. Prio has been experimentally deployed in Mozilla's Firefox browser.[46]
- Electronic voting (e-voting) has long been touted as an application area for cryptography. There is a large scientific literature on the problem. However, the use of e-voting in local and national elections has proved problematic, with confidence-sapping security vulnerabilities having been found in voting software and hardware. For example, a recent Swiss attempt to develop e-voting was temporarily abandoned after severe flaws were found in some of the cryptographic protocols used in the system during a semi-open system audit [1730]. The Estonian experience has been much more positive, with a system built on Estonia's electronic identity cards having been in regular use (and having seen regular upgrades) since 2005. Key aspects of the Estonian success are openness, usability and the population's broad acceptance of and comfort with online activity.
- We may see a shift in how cryptography gets researched, developed and then deployed. The traditional model is the long road from research to real-world use. Ideas like MPC have been travelling down this road for decades. Out of sheer necessity, systems like DP-3T have travelled down the road much more quickly. A second model arises when practice gets ahead of theory and new theory is eventually developed to analyse what is being done in practice; often this leads to a situation where the practice could be improved by following the new theory, but the improvements are slow in coming because of the drag of legacy code and the difficulty of upgrading systems in operation. Sometimes a good attack is needed to stimulate change. A third model is represented by TLS 1.3: academia and industry working together to develop a complex protocol over a period of years.
- Cryptography involves a particular style of thinking. It involves quantifying over all adversaries in security proofs (and not just considering particular adversarial strategies), being conservative in one's assumptions, and rejecting systems even if they only have "certificational flaws". Such adversarial thinking should be more broadly applied in security research. Attacks on machine learning systems are a good place where this cross-over is already bearing fruit.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

Topics	Cites
18.1 Algorithms, Schemes and Protocols	[963, 1619, 1620]
18.2 Cryptographic Implementation	[405, 1453, 1637, 1677]
18.3 Key Management	[1620, 1666, 1696, 1697]
18.4 Consuming Cryptography	
18.5 Applied Cryptography in Action	[444, 1480, 1718]
18.6 The Future of Applied Cryptography	

[46] See https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

Chapter 19
Network Security

Christian Rossow | CISPA Helmholtz Center for Information Security
Sanjay Jha | University of New South Wales

INTRODUCTION

The ubiquity of networking allows us to connect all sorts of devices and gain unprecedented access to a whole range of applications and services anytime, anywhere. However, our heavy reliance on networking technology also makes it an attractive target for malicious users who are willing to compromise the security of our communications and/or cause disruption to services that are critical for our day-to-day survival in a connected world. In this chapter, we will explain the challenges associated with securing a network under a variety of attacks for a number of networking technologies and widely used security protocols, along with emerging security challenges and solutions. This chapter aims to provide the necessary background in order to understand other knowledge areas, in particular the Security Operations & Incident Management Knowledge Area (Chapter 8) which takes a more holistic view of security and deals with operational aspects. An understanding of the basic networking protocol stack and popular network protocols is assumed. Standard networking text books explain the fundamentals of the layered Internet Protocol suite [1731, 1732].

This chapter is organized as follows. In Section 19.1, we lay out the foundations of this chapter and define security goals in networked systems. As part of this, we also outline attackers and their capabilities that threaten these goals. In Section 19.2, we describe six typical networking scenarios that nicely illustrate why security in networking is important, and achieving it can be non-trivial. We then discuss the security of the various networking protocols in Section 19.3, structured by the layered architecture of the Internet protocol stack. In Section 19.4, we present and discuss several orthogonal network security tools such as firewalls, monitoring and Software Defined Networking (SDN). We complete this chapter with a discussion on how to combine the presented mechanisms in Section 19.5.

CONTENT

19.1 SECURITY GOALS AND ATTACKER MODELS

[1731, c8] [1733, c1] [1734, c1] [1735, c6] [1732, c8]

We want and need secure networks. But what does *secure* actually mean? In this chapter, we define the fundamentals of common security goals in networking. Furthermore, we discuss capabilities, positions and powers of attackers that aim to threaten these security goals.

19.1.1 Security Goals in Networked Systems

When designing networks securely, we aim for several orthogonal *security goals* [1733]. The most commonly used security goals are summarized in the CIA triad: confidentiality, integrity and availability. *Confidentiality* ensures that untrusted parties cannot leak or infer sensitive information from communication. For example, in a confidential email communication system, (i) only the sender and recipient of an email can understand the email content, not anyone on the communication path (e.g., routers or email providers), and (ii) no one else ought to learn that email was sent from the sender to the recipient. *Integrity* ensures that untrusted parties cannot alter information without the recipient noticing. Sticking to our email example, integrity guarantees that any in-flight modification to an email (e.g., during its submission, or on its way between email providers) will be discovered as such by the recipient. Finally,

availability ensures that data and services should be accessible by their designated users all the time. In our email scenario, a Denial of Service (DoS) attacker may aim to threaten the availability of email servers in order to prevent or delay email communication.

Next to the CIA triad, there are more subtle security goals, not all of which apply in each and every application scenario. *Authenticity* is ensured if the recipient can reliably attribute the origin of communication to the sender. For example, an email is authentic if the recipient can ensure that the claimed sender actually sent this email. *Non-repudiation* extends authenticity such that we can prove authenticity to arbitrary third parties, i.e., allowing for public verification. In our email scenario, non-repudiation allows the email recipient to prove to anyone else that a given email stems from a given sender. *Anonymity* means that communication cannot be traced back to its sender (sender anonymity) and/or recipient (recipient anonymity). For example, if an attacker sends a spoofed email that cannot be reliably traced back to its actual sender (e.g., the correct personal identity of the attacker), it is anonymous. There are further privacy-related guarantees such as *unlinkability* that go beyond the scope of this chapter and are defined in the Privacy & Online Rights Knowledge Area (Chapter 5).

To achieve security goals, we will heavily rely on cryptographic techniques such as public and symmetric keys for encryption and signing, block and stream ciphers, hashing, and digital signature, as described in the Cryptography Knowledge Area (Chapter 10) and the Applied Cryptography Knowledge Area (Chapter 10). Before showing how we can use these techniques for secure networking, though, we will discuss attacker models that identify capabilities of possible attackers of a networked system.

19.1.2 Attacker Models

Attacker models are vital to understand the security guarantees of a given networked system. They define the capabilities of attackers and determine their access to the network.

Often, the Dolev-Yao [1230] attacker model is used for a formal analysis of security protocols in the research literature. The Dolev-Yao model assumes that an attacker has complete control over the entire network, and concurrent executions of the protocol between the same set of two or more parties can take place. The Dolev-Yao model describes the worst possible attacker: an attacker that sees all network communication, allowing the attacker to read any message, prevent or delay delivery of any message, duplicate any message, or otherwise synthesise any message for which the attacker has the relevant cryptographic keys (if any).

Depending on the context, real attackers may have less power. For example, we can distinguish between *active* and *passive* attackers. Active attackers, like Dolev-Yao, can manipulate packets. In contrast, passive attackers (*eavesdroppers*) can observe but not alter network communication. For example, an eavesdropper could capture network traffic using packet sniffing tools in order to extract confidential information such as passwords, credit card details and many other types of sensitive information from unprotected communication. But even if communication is encrypted, attackers may be able to leverage communication patterns statistics to infer sensitive communication content (“traffic analysis”, see Section 19.3.1.6).

We furthermore distinguish between *on-path* and *off-path* attackers. The prime example for an on-path attacker is a person-in-the-middle (PITM), where an attacker is placed between two communication parties. In contrast, off-path attackers can neither see nor directly manipulate the communication between parties. Still, off-path attackers can cause severe harm. For example, an off-path attacker could spoof TCP packets aiming to maliciously terminate a

suspected TCP connection between two parties. Similarly, off-path attackers could abuse high-bandwidth links and forge Internet Protocol (IP) headers (IP spoofing, see Section 19.3.2.4) to launch powerful and anonymous Denial of Service (DoS) attacks. Using forged routing protocol message, off-path attackers may even try to become on-path attackers.

In addition, the position of attackers heavily influences their power. Clearly, a single Internet user has less power than an entire rogue Internet Service Provider (ISP). The single user can leverage their relatively small bandwidth to launch attacks, while an ISP can generally also sniff on and alter communication, abuse much larger bandwidths, and correlate traffic patterns. Then again, as soon as attackers aggregate the power of many single users/devices (e.g., in form of a botnet), their overall power amplifies. Attackers could also be in control of certain Internet services, routers, or any combination thereof. We also distinguish between insider and outsider attackers, which are either inside or outside of a trusted domain, respectively.

Overall, we model (i) where attackers can be positioned, (ii) who they are, and (iii) which capabilities they have. Unfortunately, in strong adversarial settings, security guarantees diminish way too easily. For example, strong anonymity may not hold against state actors who can (theoretically) control major parts of the Internet, such as Tier-1. Similarly, availability is hard to maintain for spontaneous and widely distributed DoS incidents.

19.2 NETWORKING APPLICATIONS

[1731, c1] [1732, c1]

We now turn to a selection of popular concrete networking applications that help us illustrate why achieving security is far from trivial. This chapter is less about providing solutions—which are discussed in the subsequent two chapters—but more about outlining how security challenges differ by application and context. To this end, we will introduce various networking applications, starting with typical local networks and the Internet, and continuing with similarly ubiquitous architectures such as bus networks (e.g., for cyber-physical system), fully-distributed networks, wireless networks, and finally, Software Defined Networking (SDN).

19.2.1 Local Area Networks (LANs)

Arguably, a Local Area Network (LAN) is the most intuitive and by far predominant type of network with the (seemingly) lowest security requirements. Local networks connect systems within an internal environment, shaping billions of home networks and corporate networks alike. A typical fallacy of LAN operators is to blindly trust their network. However, the more clients (can possibly) connect to a LAN, the harder it is to secure a LAN environment. In fact, the clients themselves may undermine the assumed default security of a LAN. For example, without further protection, existing clients can find and access unauthorized services on a LAN, and possibly exfiltrate sensitive information out of band. This becomes especially problematic if the clients are no longer under full control of the network/system operators. A prime example is the recent rise of the Bring Your Own Device (BYOD) principle, which allows employees to integrate their personal untrusted devices into corporate networks—in the worst case, even creating network bridges to the outside world. Similarly, attackers may be able to add malicious clients, such as by gaining physical access to network plugs. Finally, attackers may impersonate other LAN participants by stealing their loose identity such as publicly-known hardware addresses (e.g., cloning MAC addresses in Ethernet).

Consequently, even though a LAN is conceptually simple, we still have uncertainties regarding LAN security: Can we control which devices become part of a network to exclude untrusted clients and/or device configurations? (Sections 19.3.4.1 and 19.4.5) Can we monitor their actions to identify attackers and hold them accountable? (Section 19.4.3) Can we partition larger local networks into multiple isolated partitions to mitigate potential damage? (Section 19.3.4.5)

19.2.2 Connected Networks and the Internet

Securing communication becomes significantly more challenging when connecting local networks. The most typical scenario is connecting a LAN to the Internet, yet also LAN-to-LAN communication (e.g., two factories part of a joint corporate network using a Virtual Private Network (VPN)) is common. With such connected networks, we suddenly face an insecure end-to-end channel between the networks which is no longer in control of blindly trusted network operators. When communicating over the Internet, the traffic will pass several Autonomous Systems (ASs), each of which can in principle eavesdrop and manipulate the communication. Worse, senders typically have little to no control over the communication paths. In fact, even ASs cannot fully trust all paths in their routing table. Without further precautions, any other (misbehaving) AS on the Internet can reroute a target's network traffic by hijacking Internet routes. For example, a state actor interested in sniffing on the communication sent to a particular network can send malicious route announcements to place itself as PITM between all Internet client and the target network. Finally, especially corporate networks may choose to "include" third-party services into their network, such as data centers for off-site storage or clouds for off-site computations. Suddenly, external networks are integrated into corporate networks, and organizations may send sensitive data to servers outside of their control.

This leads to several questions: Can two parties securely communicate over an insecure channel, i.e., with guaranteed confidentiality and integrity? (Section 19.3.2.1) How should we securely connect networks, e.g., in a corporate setting? (Section 19.3.3.1) Can we ensure that the underlying routing is trusted, or at least detect attacks? (Section 19.3.3.3)

19.2.3 Bus Networks

Cyber-physical systems regularly use bus networks to communicate. For example, industrial control systems may use Modbus [1736] to exchange status information and steer cyber-physical devices. Home automation networks (e.g., Konnex Bus (KNX) [1737]) allow to sense inputs in houses (temperature, brightness, activity) which in turn govern actuators (e.g., heating/cooling, light, doors). Vehicular networks (e.g., Controller Area Network (CAN) [1738]) connect dozens if not hundreds of components that frequently communicate, e.g., sensors such as a radar to control actuators such as the engine or brakes. All these bus networks typically also form local networks, yet with subtle differences to the aforementioned LANs. Frequently, bus clients require real-time guarantees for the arrival and processing time of data. Clearly, even safety is affected if the signal of a car's brake pedal arrives "too late" at its destination (the brake system). This need for real-time guarantees are often in direct conflict to simple security demands such as authenticity that may add "expensive" computations. Furthermore, by design, bus networks introduce a shared communication medium, which allows clients to both, sniff and manipulate communication. On top, many of the bus protocols were designed without paying particular attention to security, partially because they predate security best practices we know today. Finally, bus clients can have limited computing

resources, having only limited capability to perform expensive security operations.

We summarize our discussion with a set of questions on bus security: Can we retrospectively add security to unprotected bus networks without breaking compatibility? Is it possible to gain security without violating real-time guarantees? (Section 19.3.4.7) Which additional mechanisms can we use to reduce the general openness of bus networks? (Section 19.3.4.5)

19.2.4 Wireless Networks

When moving from wired to wireless networks, we do not necessarily face fundamentally new threats, but increase the likelihood of certain attacks to occur. In particular, wireless networks (e.g., Wireless LAN) are prone to eavesdropping due to the broadcast nature of their media. Similarly, while simple physical access control may still partially work to protect a cable-connected LAN network, it fails terribly for wireless networks that—by construction—have no clear boundary for potential intruders. For example, walls around a house do not stop signals from and to wireless home networks, and without further protection, attackers can easily join and sniff on networks. In wireless settings, we thus have to pay particular attention to both access control and secure communication (including securing against traffic analysis). As it turns out, though, defining secure wireless standards is far from trivial, and several wireless and cellular standards have documented highly-critical vulnerabilities or design flaws.

To better understand their deficiencies, we will study the following questions: How can we enforce access control in wireless networks, while not sacrificing usability? How can we prevent eavesdropping attacks in wireless communication? (Section 19.3.4.6)

19.2.5 Fully-Distributed Networks: DHTs and Unstructured P2P Networks

Centralized client-server architectures have clear disadvantages from an operational point of view. They have a single point of failure, and require huge data centers in order to scale to many clients. Furthermore, centralized architectures have a central entity that can control or deliberately abandon them. Fully-distributed networks provide scalability and resilience by design. Not surprisingly, several popular networks chose to follow such distributed designs, including cryptocurrencies such as Bitcoin or file sharing networks. Decentralized network can be roughly grouped in two types. Distributed Hash Tables (DHTs) such as Kademlia [1069] and Freenet [569] have become the de-facto standard for *structured* Peer to Peer (P2P) networks, and offer efficient and scalable message routing to millions of peers. In *unstructured* P2P networks, peers exchange data hop-by-hop using gossip protocols. Either way, P2P networks are designed to *welcome* new peers at all times. This lack of peer authentication leads to severe security challenges. A single entity may be able to aggressively overpopulate a network if no precautions are taken. DHTs further risk that attackers can potentially disrupt routing towards certain data items. Similarly, gossip networks risk that single entities flood the network with malicious data. Finally, due to the open nature of the networks, we also face privacy concerns, as network participants can infer which data items a peer retrieves.

All in all, fully-distributed networks raise fundamentally different security concerns: Can we reliably route data (or data requests) to their target? Can we authenticate peers even in distributed networks? What are the security implications of storing data in and retrieving data from a publicly accessible network? (Section 19.3.1.5)

19.2.6 Software-Defined Networking and Network Function Virtualisation

Software Defined Networking (SDN) is our final use case. SDN is strictly speaking not a networking application, but more of a technology to enable for dynamic and efficient network configuration. Yet it concerns similarly many security implications as other applications do. SDN aims to ease network management by decoupling packet forwarding (data plane) and packet routing (control plane). This separation and the underlying flow handling have enabled for drastic improvements from a network management perspective, especially in highly-dynamic environments such as data centers. The concept of Network Functions Virtualisation (NFV) complements SDN, and allows to virtualize network node functions such as load balancers or firewalls.

We will revisit SDN by discussing the following questions: How can SDN help in network designs and better monitoring? Can NFV help securing networks by virtualizing security functions? Are there new, SDN-specific threats? (Section 19.4.4)

19.3 NETWORK PROTOCOLS AND THEIR SECURITY

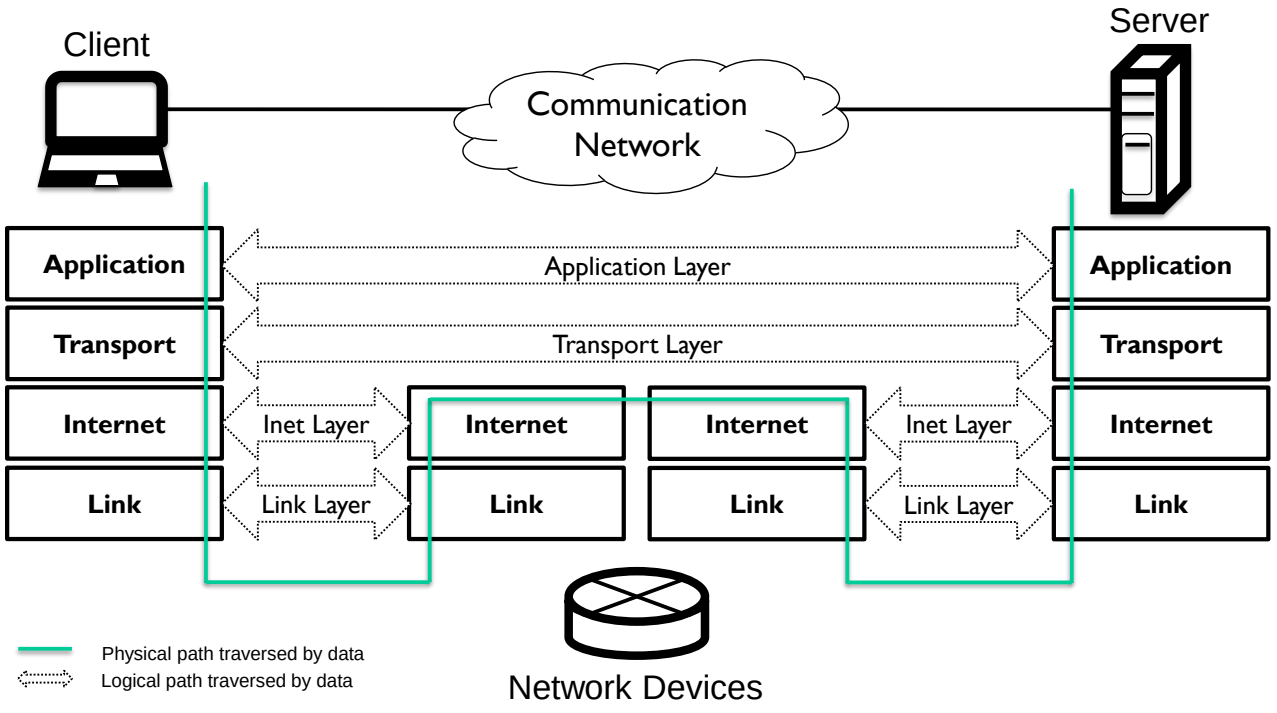


Figure 19.1: Four-Layer Internet Protocol Stack (a.k.a. *TCP/IP stack*).

After having introduced several networking applications, we now turn to the security of networking protocols. To guide this discussion, we stick to a layered architecture that categorizes protocols and applications. Indeed, a complex system such as distributed applications running over a range of networking technologies is best understood when viewed as layered architecture. Figure 19.1 shows the 4-layer Internet protocol suite and the interaction between the various layers. For each layer, we know several network protocols—some of which are quite generic, and others that are tailored for certain network architectures. The Internet is

the predominant architecture today, and nicely maps to the TCP/IP model. It uses the Internet Protocol (IP) (and others) at the Internet layer, and UDP/TCP (and others) at the transport layer—hence, the Internet protocol suite is also known as the *TCP/IP stack*.

Other networking architectures such as automotive networks use completely different sets of protocols. Not always it is possible to directly map their protocol to the layered architecture of the TCP/IP model. Consequently, more fine-grained abstractions such as the ISO/OSI model extend this layered architecture. For example, the ISO/OSI model splits the link layer into two parts, namely the data link layer (node-to-node data transfer) and the physical layer (physical transmission and reception of raw data). The ISO/OSI model defines a network layer instead of an Internet layer, which is more inclusive to networks that are not connected to the Internet. Finally, ISO/OSI defines two layers below the application layer (presentation and session).

The vast majority of topics covered in this chapter do not need the full complexity of the ISO/OSI model. In the following, we therefore describe the security issues and according countermeasures at each layer of the TCP/IP model. We thereby follow a top-down approach, starting with application-layer protocols, and slowly going down to the lower layers until the link layer. Whenever possible, we abstract from the protocol specifics, as many discussed network security principles can be generically applied to other protocols.

19.3.1 Security at the Application Layer

[1731, c8] [1734, c6,c15,c19–c22] [1732, c8]

We first seek to answer how application protocols can be secured, or how application-layer protocols can be leveraged to achieve certain security guarantees.

19.3.1.1 Email and Messaging Security

As a first example of an application-layer security protocol, we will look at secure email. Given its age, *the* protocol for exchanging emails, Simple Mail Transfer Protocol (SMTP), was not designed with security in mind. Still, businesses use email even now. Communication parties typically want to prevent others from reading (confidentiality) or altering (integrity) their emails. Furthermore, they want to verify the sender's identity when reading an email (authenticity). Schemes like Pretty Good Privacy (PGP) and Secure Multipurpose Internet Mail Extensions (SMIME) provide such end-to-end security for email communication. Their basic idea is that each email user has their own private/public key pair–see the Cryptography Knowledge Area (Chapter 10) for the cryptographic details, and Section 19.3.2.2 for a discussion how this key material can be shared. The sender signs the hash of a message using the sender's private key, and sends the hash along with the (email) message to the recipient. The recipient can then validate the email's signature using the sender's public key. Checking this signature allows for an integrity check and authentication at the same time, as only the sender knows their private key. Furthermore, this scheme provides non-repudiation as it can be publicly proved that the hash (i.e., the message) was signed by the sender's private key. To gain confidentiality, the sender encrypts the email before submission using "hybrid encryption". That is, the sender creates a fresh symmetric key used for message encryption, which is significantly faster than using asymmetric cryptography. The sender then shares this symmetric key with the recipient, encrypted under the recipient's public key.

This very same scheme can be applied to other client-to-client communication. For example, instant messengers (e.g., WhatsApp, Threema or Signal) or video conference systems can use

this general principle to achieve the same end-to-end guarantees. One remaining challenge for such strong guarantees to hold is that user identities (actually, their corresponding key material) have to be reliably validated [1739]. The Applied Cryptography Knowledge Area (Chapter 18) has more details.

Not all email users leverage such client-to-client security schemes, though. Both PGP and SMIME have usability challenges (e.g., key distribution, difficulty of indexed searches, etc.) that hamper wide adoption [347]. To address this issue, we can secure mail protocols (SMTP, but also Internet Message Access Protocol (IMAP) and Post Office Protocol (POP)) with the help of TLS (see Section 19.3.2.1). By wrapping them in TLS, we at least achieve hop-by-hop security, e.g., between client and their mail submission server or between mail servers during email transfer. Consequently, we can protect email submission, retrieval and transport from on-path adversaries. However, even though communication is protected hop-by-hop, curious mail server operators can see emails in plain. Only end-to-end security schemes like PGP/SMIME protect against untrusted mail server operators.

There are other challenges to secure email, such as phishing and spam detection, which are described in depth in the Adversarial Behaviours Knowledge Area (Chapter 7).

19.3.1.2 Hyper Text Transfer Protocol Secure (HTTPS)

The most prominent application-layer protocol, the Hypertext Transfer Protocol (HTTP), was designed without any security considerations. Yet, the popularity of HTTP and its unprecedented adoption for e-commerce imposed strict security requirements on HTTP later on. Its secure counterpart HTTPS wraps HTTP using a security protocol at the transport layer (TLS, see Section 19.3.2.1), which can be used to provide confidentiality and integrity for the entire HTTP communication—including URL, content, forms and cookies. Furthermore, HTTPS allows clients to implicitly authenticate web servers using certificates. HTTPS is described in much greater detail in the Web & Mobile Security Knowledge Area (Chapter 16).

19.3.1.3 DNS Security

In its primary use case, the Domain Name System (DNS) translates host names to their corresponding IP addresses. A hierarchy of authoritative name servers (NSs) maintain this mapping. Resolving NSs (*resolvers*) iteratively look up domain names on behalf of clients. In such an iterative lookup, the resolver would first query the root NSs, which then redirect the resolver to NSs lower in the DNS hierarchy, until the resolver contacts an NS that is authoritative for the queried domain. For example, in a lookup for a domain `sub.example.com`, a root NS would redirect the resolver to a NS that is authoritative for the `.com` zone, which in turn tell the resolver to contact the NS authoritative for `*.example.com`. To speed up these lookups, resolvers cache DNS records according to a lifetime determined by their authoritative NSs. To minimize privacy leaks towards NSs in the upper hierarchy, resolvers can minimize query names such that NSs higher up in the hierarchy do not learn the fully-qualified query name [1740].

Unfortunately, multiple attacks aim to abuse the lack of authentication in plain text DNS. A PITM attacker can impersonate a resolver, return bogus DNS records and divert traffic to a malicious server, thus allowing them to collect user passwords and other credentials. In a DNS cache poisoning attack, adversaries aim to implant bogus name records, thus diverting a user's traffic towards the target domain to attacker-controlled servers. Learning from these attacks, the IETF introduced the DNS Security Extensions (DNSSEC). DNSSEC allows authoritative

name servers to sign DNS records using their private key. The authenticity of the DNS records can be verified by a requester using the corresponding public key. In addition, a digital signature provides integrity for the response data. The overall deployment of DNSSEC at the top-level domain root name servers—a fundamental requirement to deploy DNSSEC at lower levels in the near future—steadily increases [1741].

DNSSEC explicitly does not aim to provide confidentiality, i.e., DNS records are still communicated unencrypted. DNS over TLS (DoT) and DNS over HTTPS (DoH) address this problem. They provide end-to-end security between the DNS client and its chosen resolver, by tunneling DNS via secure channels, namely TLS (see Section 19.3.2.1) or HTTPS (see Section 19.3.1.2), respectively. More and more popular Web browsers (e.g., Chrome, Firefox) enable DoH by default, using selected resolvers preconfigured by the browser vendors. This resulted in a massive centralization of DNS traffic towards just a few resolvers received. Such a centralization puts resolvers in a quite unique position with the power of linking individual clients (by IP addresses) to their lookups. Oblivious DNS Over HTTPS (ODoH) addresses this issue by adding trusted proxies between DNS clients and their chosen resolvers [1742].

Irrespective of these security protocols, resolvers are in a unique situation to monitor name resolutions of their clients. Resolver operators can leverage this in order to protect clients by offering some sort of blocklist of known “misbehaving” domains which have a bad reputation. Such DNS filtering has the potential to mitigate cyber threats, e.g., by blocking phishing domains or command & control domains of known malware variants.

Finally, DNS is prone to Distributed Denial of Service (DDoS) attacks [1743]. DNS authoritative servers can be targeted by NXDOMAIN attacks, in which an IP-spoofing client looks up many unassigned subdomains of a target domain at public (open) resolvers. Subdomains are typically chosen at random and are therefore not cached, hence sometimes referred to as *random subdomain attack*. Consequently, the resolvers have to forward the lookups and hence flood the target authoritative name server. In another type of DDoS attack, DNS servers (both resolvers and authoritatives) are regularly abused for amplification DDoS attacks, in which they reflect IP-spoofed DNS requests with significantly larger responses [1744]. Reducing the number of publicly-reachable open DNS resolvers [1745] and DNS rate limiting can mitigate these problems.

19.3.1.4 Network Time Protocol (NTP) Security

The Network Time Protocol (NTP) is used to synchronise devices (hosts, server, routers etc.) to within a few milliseconds of Coordinated Universal Time (UTC). NTP clients request times from NTP servers, taking into account round-trip times of this communication. In principle, NTP servers use a hierarchical security model implementing digital signatures and other standard application-layer security mechanisms to prevent transport-layer attacks such as replay or PITM. However, these security mechanisms are rarely enforced, easing attacks that shift the time of target system [1746] in both, on-path and off-path attacks. In fact, such time shifting attacks may have severe consequences, as they, e.g., allow attackers to use outdated certificates or force cache flushes. The large number of NTP clients that rely on just a few NTP servers on the Internet to obtain their time are especially prone to this attack [1747]. To counter this threat, network operators should install local NTP servers that use and compare multiple trusted NTP server peers. Alternatively, hosts can use NTP client implementations that offer provably secure time crowdsourcing algorithms [1748].

19.3.1.5 Distributed Hash Table (DHT) Security

There are two main threats to DHTs: (i) *Eclipse* and (ii) *Sybil* attacks. An Eclipse attacker aims to poison routing tables to isolate target nodes from other, benign overlay peers. Redundancy helps best against Eclipse attacks. For example, systems like Kademlia foresee storage and routing redundancy, which mitigate some low-profile attacks against DHTs. In the extreme, DHT implementations can use dedicated routing tables with verified entries [1749]. Central authorities—which lower the degree of distribution of DHTs, though—can solve the underlying root problem and assign stable node identifiers [1088].

In a Sybil attack, an adversary introduces malicious nodes with self-chosen identifiers to subvert DHT protocol redundancy [1084]. To prevent such Sybils, one can limit the number of nodes per entity, e.g., based on IP addresses [1749]—yet causing collateral damage to nodes sharing this entity (e.g., multiple peers behind a NAT gateway). Others suggested to use peer location as identifier validation mechanisms, which, however, prevents that nodes can relocate [1750]. Computational puzzles can slow down the pace at which attackers can inject malicious peers [1751], but are ineffective against distributed botnet attacks. Finally, reputation systems enable peers to learn trust profiles of their neighbors [1752], which ideally discredit malicious nodes [1752].

Unfortunately, all these countermeasures either restrict the generality of DHTs, or introduce a centralized component. Therefore, most defenses have not fully evolved from academia into practice. A more complete treatment of DHT security is provided by Urdaneta and van Steen [1088] and in the Distributed Systems Security Knowledge Area (Chapter 12).

19.3.1.6 Anonymous and Censorship-Free Communication

Anonymity in communication is a double-edged sword. On the one hand, we want to hold malicious communication parties accountable. On the other hand, other scenarios may indeed warrant for anonymous communication. For example, democratic minds may want to enable journalists to communicate even in suppressing regimes.

Anonymity can best be achieved when mixing communication with others and rerouting it over multiple parties. Onion routing represents the *de facto* standard of such Anonymous Communication Networks (ACNs). *Tor* is the most popular implementation of this general idea, in which parties communicate via (typically three) onion routers in a multi-layer encrypted overlay network [1753]. A Tor client first selects (typically three) onion routers, the entry node, middle node(s), and exit node. The client then establishes an end-to-end secure channel (using TLS) to the entry node, which the client then uses to create another end-to-end protected between stream to the middle node. Finally, the client uses this client-to-middle channel to initiate an end-to-end protected stream to the exit node. The resulting path is called a circuit. Tor clients communicate over their circuit(s) to achieve sender anonymity. Any data passed via this circuit is encrypted three times, and each node is responsible for decrypting one layer.

Onion routing provides quite strong security guarantees. First of all, the entry and middle node cannot decrypt the communication passed over the circuit, only the exit node can. Furthermore, none of the proxies can infer *both* communication endpoints. In fact, only the entry node knows the client, and only the exit node knows the server. This also means that servers connected via onion routing networks do not learn the true addresses of their clients.

This general concept can be expanded to achieve recipient anonymity, i.e., protect the identity of servers. For example, Tor allows for so-called *onion services* which can only be contacted

by Tor clients that know the server identity (a hash over their public key). As onion services receive data via Tor circuits and can never be contacted directly, their identity remains hidden.

While Tor gives strong anonymity guarantees, it is not fully immune against deanonymisation. In particular, traffic analysis and active traffic delay may help to infer the communication partners, especially if entry and exit node collaborate. In fact, it is widely accepted that powerful adversaries can link communication partners by correlating traffic entering and leaving the Tor network [515, 1754]. Furthermore, patterns such as inter-arrival times or cumulative packet sizes were found sufficient to attribute encrypted communication to a particular website [1755]. Consequently, attackers may be able to predict parts of the communication content even though communication is encrypted and padded. As a response, researchers explored countermeasures such as constant rate sending or more efficient variants of it [1756].

Orthogonal to ACNs, censorship-resistant networks aim to prevent that attackers can suppress communication. The typical methodology here is to blend blocklisted communication into allowed traffic. For example, decoy routing uses on-path routers to extract covert (blocklisted) information from an overt (allowed) channel and redirects this hidden traffic to the true destination [581]. Similarly, domain fronting leverages allowed TLS endpoints to forward a covert stream—hidden in an allowed TLS stream—to the actual endpoint [578]. Having said this, nation state adversaries have the power to turn off major parts (or even all) of the communication to radically subvert these schemes at the expense of large collateral damage.

19.3.2 Security at the Transport Layer

[1731, c8] [1733, c4,c6] [1732, c8]

In this subsection, we discuss the security on the transport layer which sits below the application layer in the protocol stack.

19.3.2.1 TLS (Transport Layer Security)

Application-layer protocols rely on the transport layer to provide confidentiality, integrity and authentication mechanisms. These capabilities are provided by a shim layer *between* the application and transport layers, called the Transport Layer Security (TLS). In this section, our discussions will be hugely simplified to just cover the basics of the TLS protocol. For a more detailed discussion, including the history of TLS and past vulnerabilities, see Applied Cryptography Knowledge Area (Section 18.5).

We discuss the most recent and popular TLS versions 1.2 and 1.3, with a particular focus on their handshakes. Irrespective of the TLS version, the handshake takes care of cryptographic details that application-layer protocols otherwise would have to deal with themselves: authenticating each other, agreeing on cryptographic cipher suites, and deriving key material.

The handshakes differ between the two TLS versions, as shown in Figure 19.2. We start discussing TLS 1.2, as shown on the left-hand side of the figure. First, client and server negotiate which TLS version and cipher suites to use in order to guarantee compatibility even among heterogeneous communication partners. Second, server and client exchange certificates to authenticate each other, whereas client authentication is optional (and for brevity, omitted in Figure 19.2). Certificates contain communication partner identifiers such as domain names for web servers, and include their vetted public keys (see Section 19.3.2.2 for details). Third, the communication partners derive a symmetric key that can be used to

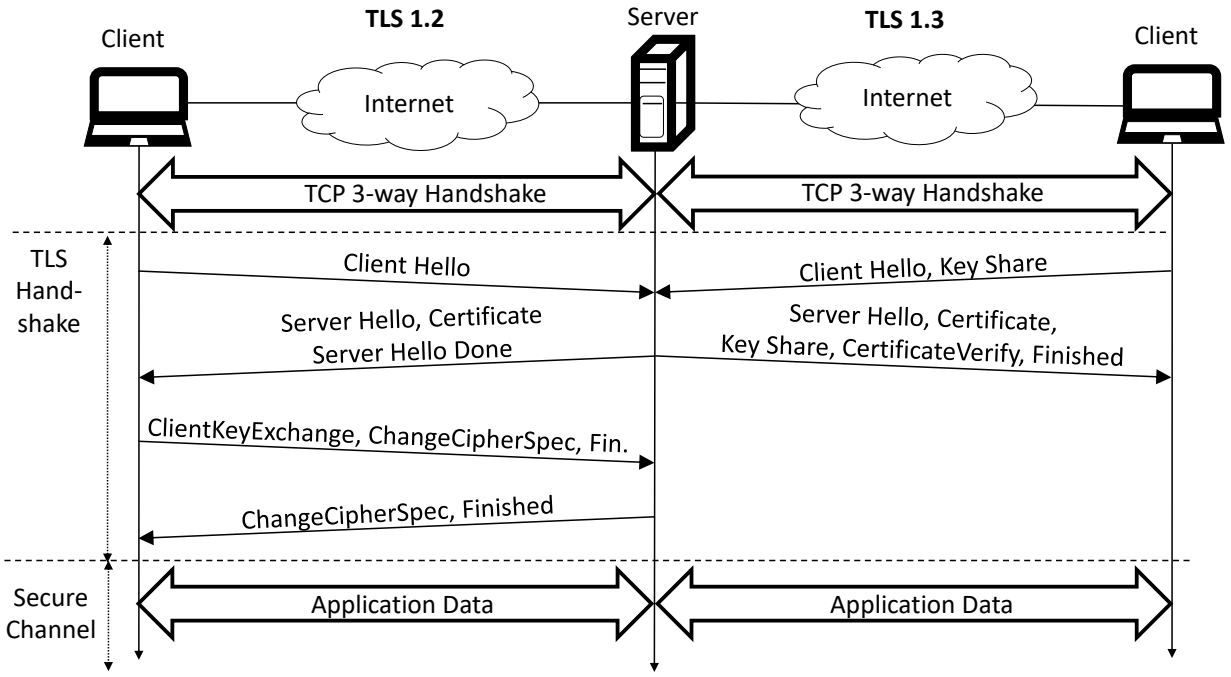


Figure 19.2: TLS Handshake: Comparison between TLS 1.2 (on the left) and TLS 1.3 (on the right), excluding the optional steps for client authentication.

secure the data transfer. To derive a key, the client can encrypt a freshly generated symmetric key under the server's public (e.g., RSA) key. Alternatively, the partners can derive a key using a Diffie-Hellman Key Exchange (DHKE). The DHKE provides TLS with perfect forward secrecy that prevents attackers from decrypting communication even if the server's private key leaks. As a final step, the handshake then validates the integrity of the handshake session. From now on, as part of the data transfer phase, TLS partners use the derived key material to encrypt and authenticate the subsequent communication.

TLS 1.3, as shown on the right of Figure 19.2, designs this handshake more efficiently. Without sacrificing security guarantees, TLS 1.3 reduces the number of round-trip times to one (1-RTT). TLS 1.3 no longer supports RSA-based key exchanges in favor of DHKE. The client therefore guesses the chosen key agreement protocol (e.g., DHKE) and sends its key share right away in the first step. The server would then respond with the chosen protocol, its key share, certificate and a signature over the handshake (in a *CertificateVerify* message). If the client was connected to the server before, TLS 1.3 even supports a handshake without additional round-trip time (0-RTT)—at the expense of weakening forward secrecy and replay prevention. Finally, as Formal Methods for Security Knowledge Area (Chapter 13) explores, TLS 1.3 has the additional benefit that it is formally verified to be secure [1719, 1757].

We now briefly discuss how TLS successfully secures against common network attacks. First, consider an eavesdropper that wants to obtain secrets from captured TLS-protected traffic. As the user data is encrypted, no secrets can be inferred. Second, in an IP spoofing attack, attackers may try any of the TLS partners to accept bogus data. However, to inject data, attackers lack the secret key to inject encrypted content. Third, also data cannot be altered, as TLS protects data integrity using authenticated encryption or message authentication codes. Finally, even a strong PITM attack is prevented by the help of certificates that authenticate the parties—unless the PITM attacker can issue certificates that the TLS partners trust, as discussed next. The TLS protocol also guarantees that payload arrives at the application in

order, detects dropped and modified content, and also effectively prevents replay attacks that resend the same encrypted traffic to duplicate payload. Having said this, TLS does not prevent attackers from delaying parts or all of the communication.

19.3.2.2 Public Key Infrastructure

So far we have simply assumed that communication partners can reliably obtain trustworthy public keys from each other. However, in presence of active on-path attackers, how can one trust public keys exchanged via an insecure channel? The fundamental "problem" is that, conceptually, everyone can create public/private key pairs. Public-Key Infrastructure (PKI) provides a solution for managing *trustworthy* public keys (and, implicitly, their private key counterparts). Government agencies or standard organisations appoint registrars who issue and keep track of so-called *certificates* on behalf of entities (individuals, servers, routers etc).

Assume a user wants to obtain a trusted certificate and the corresponding key material. To this end, the user first generates a public/private key pair on their own hardware. The private key is never shared with anyone. The public key becomes part of a certificate signing request (CSR) that the user sends to a registration authority. Before this authority signs the certificate as requested, the user has to prove their identity (e.g., possession of a domain name for an HTTPS certificate, or personal identifiers for an S/MIME certificate) to registrars. The registrar's signature prevents forgery, as anyone can now verify the certificate using the (publicly known or similarly verifiable) registrar's public key. The resulting certificate contains the user's identity and public key, as well as CA information and a period of certificate validity. Its format and PKI management specifications are specified in RFC 1422 and the ITU-X.509 standard.

The existing PKI model has faced several challenges, as evidenced by cases where CAs have issued certificates in error, or under coercion, or through their own infrastructure being attacked. As a response, CAs publish a list of revoked/withdrawn certificates, which can be queried using the Online Certificate Status Protocol (OCSP) as defined in RFC 6960, or is piggy-backed ("stapled") in TLS handshakes. To avoid wrong (but validated) certificates being issued, browsers temporarily started "pinning" them. However, this practice that was quickly abandoned and deprecated in major browsers, as it turned out to be prone to human errors (in case of key theft or key loss). Instead, big players such as Google or Cloudflare started collecting any observed and valid certificates in public immutable logs. TLS client such as browsers can then opt to refuse non-logged certificates. This scheme, known as Certificate Transparency (CT) [1758], forces attackers publishing their rogue certificates. Consequently, certificate owners can notice whether malicious parties started abusing their identifies (e.g., domains).

The web of trust is an alternative, decentralized PKI scheme where users can create a community of trusted parties by mutually signing certificates without needing a registrar. The PGP scheme we discussed in Section 19.3.1.1 and its prominent implementation GNU Privacy Guard (GPG) is a good example, in which users certify each others' key authenticity.

A more detailed PKI discussion is part of the Applied Cryptography Knowledge Area (Section 18.3.8).

19.3.2.3 TCP Security

TLS does a great deal in protecting the TCP payloads and prevents session hijacks and packet injection. Yet what about the security of TCP headers of TLS connections or other, non-TLS connections? In fact, attackers could try launching TCP reset attacks that aim to maliciously tear down a target TCP connection. To this end, they guess or bruteforce valid sequence numbers, and then spoof TCP segments with the RST flag being set. If the spoofed sequence numbers hit the sliding window, the receiving party will terminate the connection. There are mainly two orthogonal solutions to this problem deployed in practice: (i) TCP/IP stacks have to ensure strong randomness for (initial) sequence number generation. (ii) Deny RST segments with sequence numbers that fall in the middle of the sliding window. Conceptually, these defenses are ineffective against on-path attackers that can reliably manipulate TCP segments (e.g., dropping payload and setting the RST flag). Having said this, Weaver et al. [1759] show that race conditions allow for detecting RST attacks launched by off-path attackers even if they can infer the correct sequence number.

A *SYN Flooding attacker* keeps sending TCP SYN segments and forces a server to allocate resources for half-opened TCP connections. When servers limit the number of half-opened connections, benign clients can no longer establish TCP connections to the server. To mitigate this session exhaustion, servers can delete a random half-opened session whenever a new session needs to be created—potentially deleting benign sessions, though. A defence known as SYN Cookies has been implemented by operating systems as a more systematic response to SYN floods [RFC4987]. When enabled, the server does not half open a connection right away on receiving a TCP connection request. It selects an Initial Sequence Number (ISN) using a hash function over source and destination IP addresses, port numbers of the SYN segment, a timestamp with a resolution of 64 seconds, as well as a secret number only known to the server. The server then sends the client this ISN in the SYN/ACK message. If the request is from a legitimate sender, the server receives an ACK message with an acknowledgment number which is ISN plus 1. To verify if an ACK is from a benign sender, the server thus again computes the SYN cookie using the above-mentioned data, and checks if the acknowledge number in the ACK segment minus one corresponds to the SYN cookie. If so, the server opens a TCP connection, and only then starts using resources. A DoS attacker would have to waste resources themselves and reveal the true sending IP address to learn the correct ISN, hence, leveling the fairness of resource consumption.

19.3.2.4 UDP Security

What TLS is for TCP, Datagram TLS (DTLS) is for UDP. Yet again there are additional security considerations for UDP that we briefly discuss next. In contrast to its big brother TCP, UDP is designed such that application-layer protocols have to handle key mechanisms themselves (or tolerate their absence), including reordering, reliable transport, or identifier recognition.

Furthermore, being a connection-less protocol, UDP endpoints do not implicitly verify each others' IP address before communication starts. Consequently, if not handled at the application layer, UDP protocols are prone to IP spoofing attacks. We already showcased the consequences of this at the example of DNS spoofing. In general, to protect against this threat, any UDP-based application protocol must gauge the security impact of IP spoofing.

Reflective DDoS attacks are a particular subclass of IP spoofing attacks. Here, attackers send IP packets in which the source IP address corresponds to a DDoS target. If the immediate recipients (called *reflectors*) reply to such packets, their answers overload the victim with

undesired replies. We mentioned this threat already in the context of DNS (Section 19.3.1.3). The general vulnerability boils down to the lack of IP address validation in UDP. Consequently, several other UDP-based protocols are similarly vulnerable to reflection [1744]. Reflection attacks turn into amplification attacks, if the responses are significantly larger than the requests, which effectively amplifies the attack bandwidth. Unless application-level protocols validate addresses, or enforce authentication, reflection for UDP-based protocols will remain possible. If protocol changes would break compatibility, implementations are advised to rate limit the frequency in which clients can trigger high-amplification responses. Alternative, non-mandatory instances of amplifying services can be taken offline [1745].

19.3.2.5 QUIC

QUIC is a new transport-level protocol that saw rapid deployment by popular Web browsers. QUIC offers faster communication using UDP instead of HTTP over TCP. QUIC was originally designed by Google, and was then standardized by the IETF in 2021 [1760]. Its main goal is increasing communication performance using multiplexed connections. Being a relatively new protocol, in contrast to other protocols, QUIC was designed to be secure. Technically, QUIC uses most of the concepts described in TLS 1.3, but replaces the TLS Record Layer with its own format. This way, QUIC cannot only encrypt payload, but also most of the header data. QUIC, being UDP-based, "replaces" the TCP three-way handshake by its own handshake, which integrates the TLS handshake. This eliminates any round-trip time overhead of TLS. With reference to Figure 19.2 (page 657), QUIC integrates the only two TLS 1.3 handshake messages in its own handshake. When serving certificates and additional data during the handshake, QUIC servers run the risk of being abused for amplification attacks (cf. Section 19.3.2.4), as server responses are significantly larger than initial client requests. To mitigate this problem, QUIC servers verify addresses during the handshake, and must not exceed certain amplification prior to verifying addresses (the current IETF standard draft defines a factor of three).

19.3.3 Security at the Internet Layer

[1731, c8] [1733, c5,c9] [1734, c17] [1732, c8]

Although application-layer and transport-layer security help to provide end-to-end security, there is also merit in adding security mechanisms to the network layer. First, higher-layer security mechanisms do not necessarily protect an organisation's internal network links from malicious traffic. If and when malicious traffic is detected at the end hosts, it is too late, as the bandwidth has already been consumed. The second major issue is that the higher-layer security mechanisms described earlier (e.g., TLS) do not conceal or protect IP headers. This makes the IP addresses of the communicating end hosts visible to eavesdroppers and even modifiable to PITM attackers.

19.3.3.1 IPv4 Security

IP Spoofing: IP spoofing, as discussed for UDP and DNS (sections 19.3.2.4 and 19.3.1.3, respectively), finds its root in the Internet Protocol (IP) and affects both IPv4 and IPv6. In principle, malicious clients can freely choose to send traffic with any arbitrary IP address. Thankfully, most providers perform egress filtering and discard traffic from IP addresses outside of their domain [1761]. Furthermore, Unicast Reverse Path Forwarding (uRPF) enables on-path routers to drop traffic from IP addresses that they would have expected entering on other interfaces [1761].

Fragmentation Attacks: IPv4 has to fragment packets that do not fit the network's Maximum Transmission Unit (MTU). While fragmentation is trivial, defragmentation is not so, and has led to severe security problems in the past. For example, a Teardrop attack abuses the fact that operating systems may try to retain huge amounts of payload when trying to reassemble highly-overlapping fragments of a synthetic TCP segment. Fragmentation also eases DNS cache poisoning attacks in that attackers need to bruteforce a reduced search space by attacking only the non-starting fragments [1762]. Finally, fragmentation may assist attackers in evading simple payload matches by scattering payload over multiple fragments.

VPNs and IPsec: Many organisations prefer their traffic to be fully encrypted as it leaves their network. For example, they may want to connect several islands of private networks owned by an organisation via the Internet. Also, employers and employees want a flexible work environment where people can work from home, or connect from a hotel room or an airport lounge without compromising their security. If only individual, otherwise-internal web hosts need to made available, administrators can deploy web proxies that tunnel traffic (sometimes referred to as *WebVPN*). In contrast, a full-fledged Virtual Private Network (VPN) connects two or more otherwise-separated networks, and not just individual hosts.

There are plenty of security protocols that enable for VPNs, such as Point-to-Point Tunneling Protocol (PPTP) (deprecated), TLS (used by, e.g., OpenVPN [1763]), or Secure Socket Tunneling Protocol (SSTP). We will illustrate the general VPN concept at the example of the Internet Protocol Security (IPsec) protocol suite. Figure 19.4 shows that an employee working from home accesses a server at work, the VPN client in their host encapsulates IPv4 datagrams into IPsec and encrypts IPv4 payload containing TCP or UDP segments, or other control messages. The corporate gateway detects the IPsec datagram, decrypts it and decapsulates it back to the IPv4 datagram before forwarding it to the server. Every response from the server is also encrypted by the gateway. IPsec also provides data integrity, origin authentication and replay attack prevention. These guarantees depend on the chosen IPSec protocol, though. Only the recommended and widely-deployed Encapsulation Security Payload (ESP) protocol (part of IPSec) provides these guarantees, including confidentiality and origin authentication. In contrast, the less popular Authentication Header (AH) protocol just provides integrity. Similarly, several tunneling protocols such as Generic Routing Encapsulation (GRE), Layer 2 Tunneling Protocol (L2TP) or Multiprotocol Label Switching (MPLS) do not provide CIA guarantees. Should those be required in untrusted networks, e.g., due to GRE's multi-protocol or multi-casting functionality, it is advisable to use them in combination with IPSec.

The entire set of modes/configurations/standards provided by IPsec is extensive [1764]. Here, we only briefly introduce that IPsec supports two modes of operation: *tunnel mode* and *transport mode*, as compared in Figure 19.3. In transport mode, only the IP payload—not the original IP header—is protected. The tunnel mode represent a viable alternative if the edge devices (routers/gateways) of two networks are IPsec aware. Then, the rest of the servers/hosts need not worry about IPsec. The edge devices encapsulate every IP packet

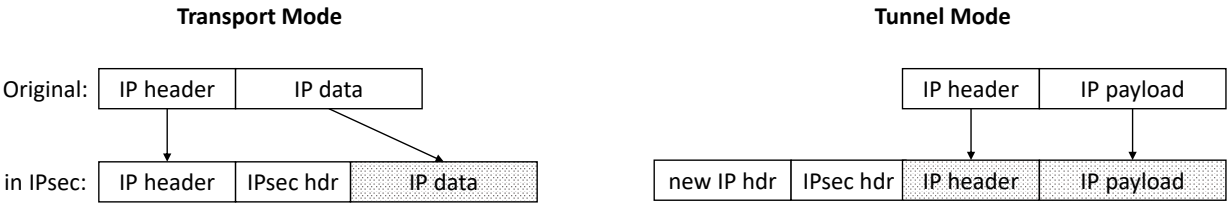


Figure 19.3: Comparison between IPsec transport mode and tunnel mode. All parts of the original packets that are shaded in gray are protected in the respective mode.

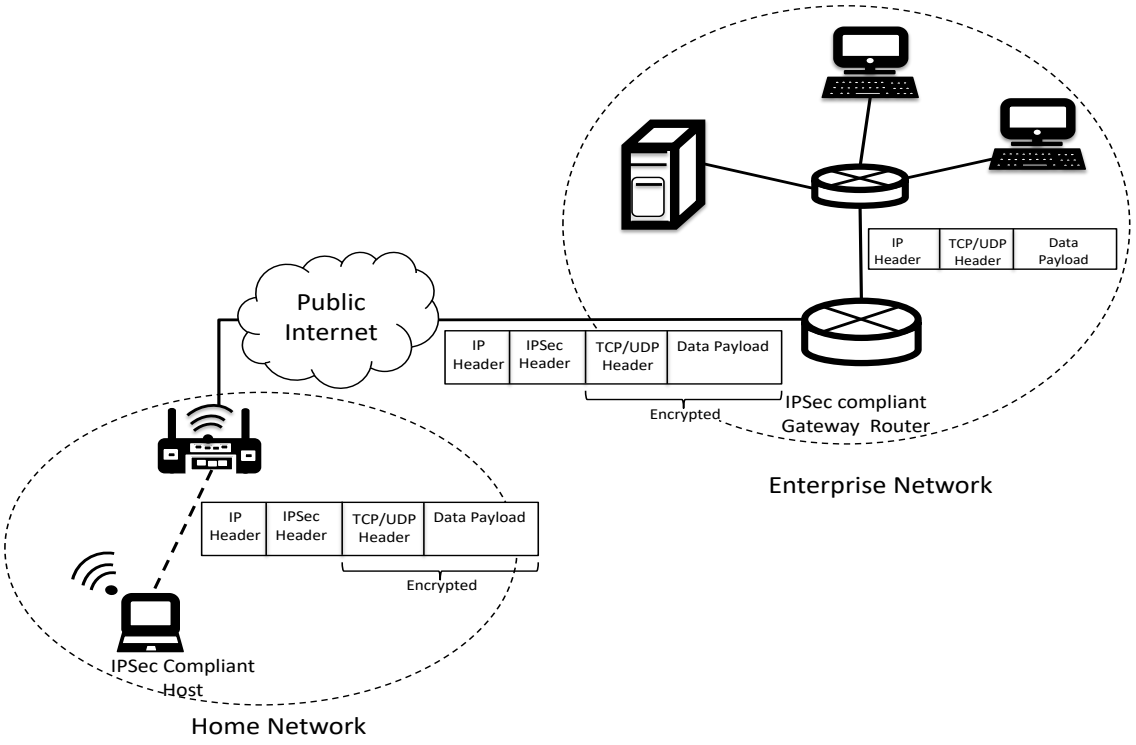


Figure 19.4: IPsec client-server interaction in transport mode (no protection of IP headers).

including the header. This virtually creates a secure tunnel between the two edge devices. The receiving edge device then decapsulates the IPv4 datagram and forwards within its network using standard IP forwarding. Tunnel mode simplifies key negotiation, as two edge devices can handle connections on behalf of all hosts in their respective networks. An additional advantage is that also IP headers (including source/destination address) gets encrypted.

When a large number of endpoints use IPsec, manually distributing the IPsec keys becomes challenging. RFC 7296 [1765] defines the Internet Key Exchange protocol (IKEv2). Readers will observe a similarity between TLS (Section 19.3.2) and IKE, in that IKE also requires an initial handshake process to negotiate cryptographic algorithms and other values such as nonces and exhange identities and certificates. We will skip the details of a complex two-phase protocol exchange which results in the establishment of a quantity called SKEYSEED. These SKEYSEEDs are used to generate the keys used during a session (Security Associations (SAs)). IKEv2 uses the Internet Security Association and Key Management Protocol (ISAKMP) [1766], which defines the procedures for authenticating the communicating peer, creation and management of SAs, and the key generation techniques.

NAT: Due to the shortage of IPv4 address space, Network Address Translation (NAT) was designed so that private IP addresses could be mapped onto an externally routable IP address by the NAT device [1731]. For an outgoing IP packet, the NAT device changes the private source IP address to a public IP address of the outgoing link. This has implicit, yet unintentional security benefits. First, NAT obfuscates the internal IP address from the outside world. To a potential attacker, the packets appear to be coming from the NAT device, not the real host behind the NAT device. Second, unless loopholes are opened via port forwarding or via UPnP, NAT gateways such as home routers prevent attackers from reaching internal hosts.

19.3.3.2 IPv6 Security

Although conceptually very similar from a security perspective, IPv6 brings a few advantages over IPv4. For example, IPv6's 128-bit address space slows down port scans, as opposed to IPv4, where the entire 32-bit address space can be scanned in less than an hour [1767]. Similarly, IPv6 comes with built-in encryption in form of IPsec. IPsec that was initially mandated in the early IPv6 standard. Yet nowadays, due to implementation difficulties, IPsec remains a recommendation only. Furthermore, in contrast to IPv4, IPv6 has no options in its header—these were used for attacks/exploits in IPv4.

The community has debated many years over the potential security pitfalls with IPv6. As a quite drastic change, the huge address space in IPv6 obsoletes NATing within the IPv6 world, including all its implicit security benefits. In particular, NAT requires state tracking, which devices often couple with a stateful firewall (which we will discuss in Section 19.4.1) that brings additional security. Furthermore, NAT hides the true IP addresses and therefore complicates IP-based tracking—providing some weak form of anonymity. Having said this, experts argue that these perceived advantages also come with lots of complexity and disadvantages (e.g., single point of failure), and that eliminating NAT by no means implies that Internet-connected devices no longer have firewalls [1768]. Furthermore, having large networks to choose addresses from, IPv6 may allow to rotate IP addresses more frequently to complicate address-based tracking. Summarizing this debate, as long as we do not drop firewalls, and are careful with IP address assignment policies, IPv6 does not weaken security.

Finally, another important aspect to consider is that we are still in a steady transition from IPv4 to IPv6. Hence, many devices feature a so-called *dual stack*, i.e., IPv4 and IPv6 connectivity. This naturally asks for protecting both network accesses simultaneously.

19.3.3.3 Routing Security

IPv4/IPv6 assume that Internet routers reliably forward packets from source to destination. Unfortunately, a network can easily be disrupted if either the routers themselves are compromised or they accept spurious routing exchange messages from malicious actors. We will discuss these threats in the following, distinguishing between internal and external routing.

Within an Autonomous System (AS): Interior Gateway Protocols (IGPs) are used for exchanging routing information within an Autonomous System (AS). Two such protocols, Routing Information Protocol (RIPv2) and Open Shortest Path First (OSPFv2), are in widespread use with ASs for IPv4 networks. The newer RIPng and OSPFv3 versions support IPv6. These protocols support no security by default but can be configured to support either plain text-based authentication or MD5-based authentication. Authentication can avoid several kinds of attacks such as bogus route insertion or modifying and adding a rogue neighbour. Older routing protocols, including RIPv1 or Cisco's proprietary Interior Gateway Routing Protocol

(IGRP)—unlike its more secure successor, the Enhanced Interior Gateway Routing Protocol (EIGRP)—do not offer any kind of authentication, and hence, should be used with care.

Across ASs: The Internet uses a hierarchical system where each AS exchanges routing information with other ASs using the Border Gateway Protocol (BGP) [1769, 1770]. BGP is a path vector routing protocol. We distinguish between External BGP used across ASs, and Internal BGP that is used to propagate routes *within* an AS. From now on, when referring to BGP, we talk about External BGP, as it comes with the most interesting security challenges. In BGP, ASs advertise their IP prefixes (IP address ranges of size /24 or larger) to peers, upstreams and customers [1731]. BGP routers append their AS information before forwarding these prefixes to their neighbors. Effectively, this creates a list of ASs that have to be passed to reach the prefix, commonly referred to as the *AS path*.

High-impact attacks in the past have highlighted the security weakness in BGP due to its lack of integrity and authentication [1771]. In particular, in a *BGP prefix hijacking attack* [1772], a malicious router could advertise an IP prefix, saying that the best route to a service is through its network. Once the traffic starts to flow through its network, it can drop (DoS, censorship), sniff on (eavesdrop) or redirect traffic in order to overload an unsuspecting AS. As a countermeasure, the Resource Public Key Infrastructure (RPKI) [1773], as operated by the five Regional Internet Registrys (RIRs), maps IP prefixes to ASs in so-called Route Origin Authorization (ROA). When neighbors receive announcements, RPKI allows them to discard BGP announcements that are not backed by an ROA or are more specific than allowed by the ROA. This process, called Route Origin Validation (ROV), enables to drop advertisements in which the AS that owns the advertised prefix is not on the advertised path.

RPKI *cannot* detect bogus advertisements where the owning AS is on path, but a malicious AS aims to reroute the target's AS traffic as an intermediary. BGPsec partially addresses this remaining security concern [1774]. Two neighbouring routers can use IPsec mechanisms for point-to-point security to exchange updates. Furthermore, BGPsec enables routers to verify the incremental updates of an announced AS path. That is, they can verify which on-path AS has added itself to the AS path, preventing bogus paths that include a malicious AS that lacks the according cryptographic secrets. However, BGPsec entails large overheads, such as verifying a larger number of signatures on booting, and splitting up bulk announcements into many smaller ones. Furthermore, BGPsec only adds security if all systems on the AS path support it. Hence, not many routers deploy BGPsec yet, fueled by the lack of short-term benefits [1775]—and it is likely to take years until it will find wide adoption, if ever.

Despite the fact that BGP prefix hijacks are a decade-old problem, fixing them retroactively remains one of the great unsolved challenges in network security. In fact, one camp argues that the BGP design is inherently flawed [1776], and entire (yet not widely deployed) Internet redesigns such as SCION [1777] indeed provide much stronger guarantees. Others did not give up yet, and hope to further strengthen the trust in AS paths by the help of ongoing initiatives such as Autonomous System Provider Authorization (ASPA) [1778].

19.3.3.4 ICMP Security

Internet Control Message Protocol (ICMP) is a supportive protocol mainly used for exchanging status or error information. Unfortunately, it introduced several orthogonal security risks, most of which are no longer present but still worth mentioning. Most notably, there many documented cases in which ICMP was an enabler for Denial of Service (DoS) attacks. The *Ping of Death* abused a malformed ICMP packet that triggered a software bug in earlier versions of the Windows operating system, typically leading to a system crash at the packet recipient. In an ICMP flood, an attacker sends massive amounts of ICMP packets to swamp a target network/system. Such floods can be further amplified in so-called *smurf attacks*, in which an attacker sends IP-spoofed ICMP ping messages to the broadcast address of an IP network. If the ICMP messages are relayed to all network participants using the (spoofed) address of the target system as source, the target receives ping responses from all active devices. Smurf attacks can be mitigated by dropping ICMP packets from outside of the network, or by dropping ICMP messages destined to broadcast addresses.

But also outside of the DoS context, ICMP is worth considering from a security perspective. Insider attackers can abuse ICMP as covert channel to leak sensitive data unless ICMP is closely monitored or forbidden. ICMP reachability tests allow attackers to perform network reconnaissance during network scans (see also Section 19.4.3). Many network operators thus balance pros and cons of ICMP in their networks, often deciding to drop external ICMP messages using a firewall (see also Section 19.4.1).

19.3.4 Security on the Link Layer

[1731, c8] [1733, c7] [1732, c8]

In this section, we are confining our attention to the security of the link layer. We mostly focus on the logical part of the link layer. The physical part is addressed in the Physical Layer and Telecommunications Security Knowledge Area (Chapter 22).

19.3.4.1 Port-based Network Access Control (IEEE 802.1X)

The IEEE 802.1X is a port-based authentication for securing both wired and wireless networks. Before a user can access a network at the link layer, it must authenticate the switch or access point (AP) they are attempting to connect to, either physically or via a wireless channel. As with most standards bodies, this group has its own jargon. Figure 19.5 shows a typical 802.1X setup. A user is called a supplicant and a switch or AP is called an authenticator. Supplicant software is typically available on various OS platforms or it can also be provided by chipset vendors. A supplicant (client) wishing to access a network must use the Extensible Authentication Protocol (EAP) to connect to the Authentication Server (AuthS) via an authenticator. The EAP is an end-to-end (client to authentication server) protocol. When a new client (supplicant) is connected to an authenticator, the port on the authenticator is set to the 'unauthorised' state, allowing only 802.1X traffic. Other higher layer traffic, such as TCP/UDP is blocked. The authenticator sends out the EAP-Request identity to the supplicant. The supplicant responds with the EAP-response packet, which is forwarded to the AS, and typically proves the supplicant possesses its credentials. After successful verification, the authenticator unblocks the port to let higher layer traffic through. When the supplicant logs off, the EAP-logoff to the authenticator sets the port to block all non-EAP traffic.

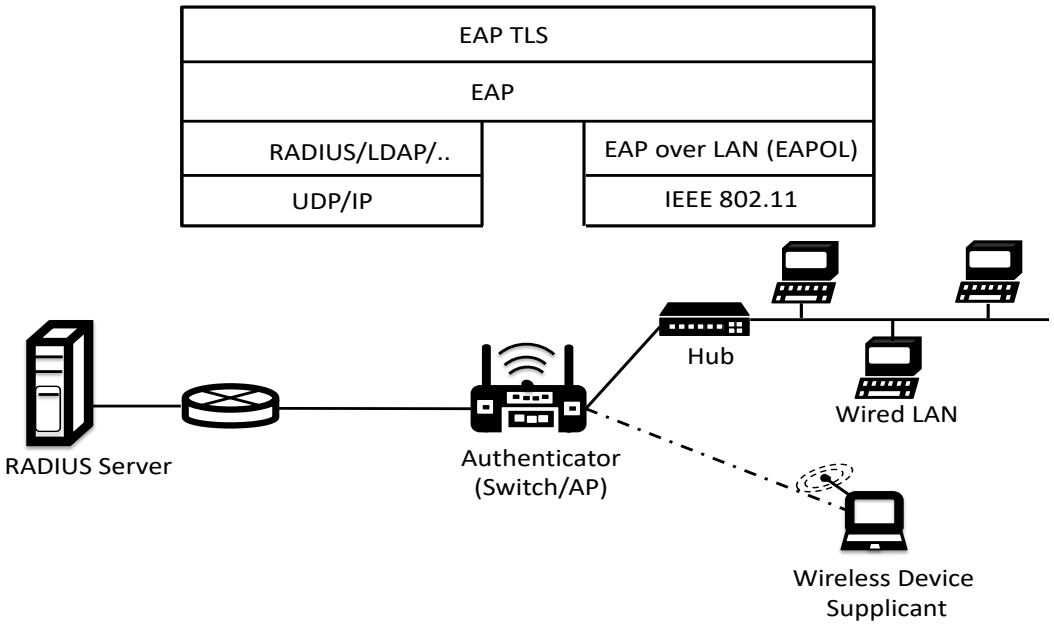


Figure 19.5: Extensible Authentication Protocol (EAP)

There are a couple of pitfalls when deploying EAP and choosing the wrong mode of operation. Certain EAP modes are prone to PITM attacks, especially in a wireless setting. It therefore is advised to use any sort of TLS-based EAP variant, such as EAP-TLS [1779] or the Protected Extensible Authentication Protocol (PEAP). Similarly, dictionary attacks can weaken the security guarantees of certain EAP modes (e.g., EAP-MD5) that should be avoided.

19.3.4.2 WAN Link-Layer Security

What IEEE 802.1X is for local networks, protocols like Point-to-Point Protocol (PPP), its sibling PPP over Ethernet (PPPoE), or High-Level Data Link Control (HDLC) are for Wide Area Networks (WANs). They offer clients means to connect to the Internet by the help of their ISPs. PPP(oE) is the most widely used protocol in this context, used by billions of broadcast devices worldwide. Although optional in its standard, in practice, ISPs usually mandate client authentication to hold unauthorized users off. Popular examples of such authentication protocols within PPP are Password Authentication Protocol (PAP), Challenge Handshake Authentication Protocol (CHAP), or any of the authentication protocols supported by EAP. Usage of PAP is discouraged, as it transmits client credentials in plain. Instead, CHAP uses a reasonbly secure challenge-response authentication, which is however susceptible to offline bruteforce attacks against recorded authentication sessions that contain weak credentials.

19.3.4.3 Attacks On Ethernet Switches

Ethernet switches maintain forwarding table entries in a Content Addressable Memory (CAM). As a switch learns about a new destination host, the switch includes this host's address and its physical port in the CAM. For all future communications, this table entry is looked up to forward a frame to the correct physical port. MAC spoofing allow attackers to manipulate this mapping by forging their Message Authentication Code (MAC) addresses when sending traffic. For example, to poison the forwarding table, an attacker crafts frames with random addresses to populate an entire CAM. If successful, switches have to flood all the incoming data frames to all the outgoing ports, as they can no longer enter new address-to-port mappings. This makes the data available to the attacker attached to any of the switch ports.

Such MAC spoofing attacks can also be more targeted. Assume attackers want to steal traffic destined to one particular target host only, instead of seeing all traffic. The attacker then copies the target's MAC address. This way, the attacker may implicitly rewrite the target's entry in the switch forwarding table. If so, the switch will falsely forward frames to the attacking host that were actually destined for the target host.

Mitigating these MAC spoofing attacks requires authenticating the MAC addresses before populating the forwarding table entry. For example, IEEE 802.1X (see Section 19.3.4.1) mitigates such attack, vetting hosts before they can connect. Furthermore, switches may limit the number of MAC addresses per interface or enforce MAC bindings, as described next.

19.3.4.4 Address Resolution Protocol (ARP) / Neighbor Discovery Protocol (NDP)

The Address Resolution Protocol (ARP) translates IPv4 addresses to link layer addresses (e.g., MAC addresses in Ethernet). ARP spoofing is similar to MAC spoofing, yet does not (only) target the switch's address mappings. Instead, ARP spoofing target the IP-to-MAC address mappings of all network participants (possibly including the switch) in the same segment. To this end, ARP spoofing attackers send fake ARP messages over a LAN. For example, they can broadcast crafted ARP requests and hope participants learn wrong IP-to-MAC mappings on the fly, or reply with forged replies to ARP request. Either way, attackers aim to (re-)bind the target's IP address to their own MAC address. If successful, attackers will receive data that were intended for the target's IP address. ARP spoofing is particularly popular for session hijacking and PITM attacks. Similar attacks are possible for the Reverse Address Resolution Protocol (RARP), which—by now rarely used—allows hosts to discover their IP address. To mitigate ARP spoofing, switches employ (or learn) a trusted database of static IP-to-MAC address mappings, and refuse to relay any ARP traffic that contradicts these trusted entries. Alternatively, network administrators can spot ARP anomalies [1780], e.g., searching for suspicious cases in which one IP address maps to multiple MAC addresses.

What ARP is for IPv4, the Neighbor Discovery Protocol (NDP) is for IPv6. NDP is based on ICMPv6 and is more feature-rich than ARP. Conceptually, NDP underlies the same spoofing risks as ARP, though, and requires the same countermeasures. Furthermore, there is one more caveat due to automatic IPv6 address assignment. In IPv6's most basic (yet common) IP address autoconfiguration scheme, layer-3 addresses are derived directly from layer-2 addresses without any need for address resolution. Knowledge of the MAC address may allow attackers to infer information about the host/servers which can be handy when launching attacks, or to track devices even if they change network prefixes. Using hash function for address generation is recommended as a mitigation technique. Further, RFC 4982 extends IPv6 by allowing for a Cryptographically Generated Address (CGA) where an address is bound

to a public signature key. Orthogonal to this, RFC 7217 proposes to have stable addresses within a network prefix, and change them when clients switch networks to avoid cross-network tracking.

19.3.4.5 Network Segmentation

MAC spoofing and ARP spoofing nicely illustrate how fragile security on the network layer is. Consequently, network architects aim to split their physical network into several smaller networks—a practice known as *network segmentation*. Highly-critical environments such as sensitive military or control networks use *physical* segmentation. As the required change of cables and wires is quite expensive, *virtual* network segmentation has become a popular alternative. Virtual LANs (VLANs) are the *de facto* standard for virtual segmentation on Ethernet networks. VLANs can split sensitive (e.g., internal servers) from less sensitive (guest WiFi) network segments. VLANs enforce that routers can see and react upon traffic between segments, and limit the harm attackers can do to the entire LAN. It is important to note that network segmentation (e.g., via VLANs) does not necessarily require VPNs to bridge the networks. If all network segments are local, a router that is part of multiple subnetworks can connect them, ideally augmented with secure firewall policies (cf. Section 19.4.1) that control inter-network communication at the IP layer.

VLAN hopping attacks allow an attacking host on a VLAN to gain access to resources on other VLANs that would normally be restricted. There are two primary methods of VLAN hopping: switch spoofing and double tagging. In a switch spoofing attack, an attacking host impersonates a trunking switch responding to the tagging and trunking protocols (e.g., IEEE 802.1Q or Dynamic Trunking Protocol) typically used in a VLAN environment. The attacker now succeeds in accessing traffic for multiple VLANs. Vendors mitigate these attacks by proper switch configuration. For example, the ports are assigned a trunking role explicitly and the others are configured as access ports only. Also, any automatic trunk negotiation protocol can be disabled. In a double tagging attack, an attacker succeeds in sending its frame to more than one VLAN by inserting two VLAN tags to a frame it transmits. However, this attack does not allow them to receive a response. Again, vendors provide recommended configuration methods to deal with these possible attacks. A comprehensive survey of Ethernet attacks and defence can be found in [1781].

Organizations like hosting providers that heavily virtualize services quickly reach the limitation of having a maximum of little less than 4096 VLANs when trying to isolating their services. Virtual eXtensible LAN (VXLAN) tackles this limitation by introducing an encapsulation scheme for multi-tenant environments [1782]. Unlike VLANs, which work on the link layer, VXLANs strictly speaking operate at the network layer to emulate link-layer networks. VXLAN allows creating up to ≈16M virtually separated networks, which can additionally be combined with VLAN functionality. Like VLANs, VXLANs also do not aim to provide confidentiality or integrity in general. Instead, they are means to segment networks. Worse, however, being on the network layer, VXLAN packets can traverse the Internet, and may allow attackers to inject spoofed VXLAN packets into “remote” networks. Thus, care has to be taken, e.g., by ingress filtering at the network edge to drop VXLAN packets that carry a valid VXLANs endpoint IP address.

19.3.4.6 Wireless Security

Wireless LAN are more vulnerable to security risks due to the broadcast nature of media, which simplifies eavesdropping. There have been several failed attempts to add integrity and confidentiality to WLANs communication. First, the Wired Equivalent Privacy (WEP) protocol used a symmetric key encryption method where the host shares a key with an Access Point (AP) out of band. WEP had several design flaws. First, a 24-bit IV introduced a weakness in that ≈16 million unique IVs can be exhausted in high-speed links in less than 2 hours. Given that IVs are sent in plaintext, an eavesdropper can easily detect this reuse and mount a known plaintext attack. Furthermore, using RC4 allowed for the Fluhrer, Martin and Shamir (FMS) attacks, in which an attacker can recover the key in an RC4 encrypted stream by capturing a large number of messages in that stream [1783, 1784]. Furthermore, WEP's linear CRC was great for detecting random link errors, but failed to reliably reveal malicious message modifications.

An interim standard called the Wi-Fi Protected Access (WPA) was quickly developed for backward hardware compatibility, while WPA2 was being worked out. WPA uses the Temporal Key Integrity Protocol (TKIP) but maintains RC4 for compatibility. The Pre-Shared Key (PSK), also known as WPA-Personal, is similar to the WEP-Key. However, the PSK is used differently, a nonce, and PSK are hashed to generate a temporal key. Following this, a cryptographic mixing function is used to combine this temporal key, the Temporal MAC (TMAC), and the sequence counter resulting in one key for encryption (128 bits) and another key for integrity (64 bits). As a consequence, every packet is encrypted with a unique encryption key to avoid FMS-style attacks. Also, the WPA extends the WEP IV to 48 bits. Several new fields include a new Frame Check Sequence (FCS) field, a CRC-32 checksum for error correction and a hash function for a proper integrity check. Due to compromises it made with respect to backwards compatibility, the WPA has had its own share of attacks, though [1785].

The Wifi alliance then standardized WPA2 in 2004. WPA2 relies on more powerful hardware supporting a 128-bit AES Counter Mode with the Cipher Block Chaining Message Authentication Code Protocol (CCMP), obsoleting RC4. It also provides an improved 4-way handshake and temporary key generation method (which does not feature forward secrecy, though). While implementations of this handshake were shown insecure [1786], the general handshake methodology was formally verified and is still believed to be secure [1787, 1788].

In 2018, a new WPA3 standard was accepted to make a gradual transition and eventually replace the WPA2. WPA3 overcomes the lack of perfect forward secrecy in WPA and WPA2. The PSK is replaced with a new key distribution called the Simultaneous Authentication of Equals (SAE) based on the IETF Dragonfly key exchange. The WPA3-Personal mode uses a 128-bit encryption, whereas the WPA3-Enterprise uses 192-bit encryption.

The discussion so far assumed that there is a shared secret between WLAN users and APs from which session keys can be derived. In fact, enterprise settings usually handle WLAN access control using perform strong authentication such as 802.1X (Section 19.3.4.1). Ideally, WLAN users have their own client certificates that provide much stronger security than any reasonably user-friendly password. In contrast, for openly accessible networks such as at airports or restaurants, there may neither be PSKs nor certificates. Consequently, the lack of strong encryption would leave communication unprotected. Opportunistic Wireless Encryption (OWE) tackles this open problem [1789]. Instead of using a PSK during the WPA2/3 four-way handshake, the client and AP use a pairwise secret derived from an initial DHKE.

19.3.4.7 Bus Security

Bus networks follow a special topology in that all nodes are directly connected to a shared medium (the bus). Securing a bus is inherently complex, especially if we assume that an insider attacker is connected to the bus. In order to illustrate this, we will focus on the Controller Area Network (CAN) standard, which despite its age is still quite commonly used in cars today. CAN nicely reveals many issues that can arise on bus networks in general. CAN connects so-called Electrical Control Units (ECUs), such as a car's wheel, break pedal or the radio. CAN is a real-time protocol designed to give priority to more urgent ECUs (e.g., brake pedal) over less pressing ones (e.g., multimedia control). Sadly, CAN suffers from severe security vulnerabilities. They become especially problematic if ECUs are or turn malicious (e.g., after compromise). First, CAN does not authenticate messages, i.e., any compromised ECU (e.g., multimedia system) can easily spoof messages of critical components (e.g., wheel speed sensor). Second, compromised bus components can receive and invalidate all messages of any arbitrary other ECUs on the same bus. For example, a compromised ECU could suppress the signals sent by an activated brake pedal. Finally, and a little less concerning than the previous examples, CAN is unencrypted, providing no confidentiality against sniffing.

A radical protocol change could solve all these problems. In fact, there are new standards like AUTomotive Open System ARchitecture (AUTOSAR) [1790] that provide improved security principles. Yet, as always, such radical changes take a long time in practice, as they break compatibility of existing devices. Also, devices have a years-long development cycle and usage time. Vendors are aware of these issues and aim to mitigate the problem by segmenting critical components from less critical ones (segmentation in general is discussed in Section 19.3.4.5). While certainly a vital step, as it physically disconnects more complex and vulnerable devices such as multimedia systems from safety-critical devices, this only reduces and not entirely eliminates the attack surface. A star topology would solve many of these issues, as the medium is no longer shared and address spoofing could be validated by a central entity. Yet star topologies incur significant additional physical cables, and thus, higher costs, manufacturing complexity, and weight. Academia explored several approaches to add message authenticity to CAN and to prevent spoofing on CAN without breaking backwards-compatibility [1791, 1792]. None of them found wide deployment in practice yet, though, possibly due to costs and the need to adapt ECUs. Alternative approaches aim to detect spoofed messages by learning and modeling the per-ECU voltage of bus messages. Unfortunately, such classifiers were proven unreliable [1793]. A wider popularity of CAN-FD [1738], which offers a flexible data rate and larger messages (64B instead of 8B in CAN) will decrease overhead of security add-ons and may thus ease the development of more secure CAN communication in the future.

Many of the observed problems generalize beyond CAN to any bus system or even shared-medium network. Rogue components on a bus can suppress messages by invalidating them, anyone on a bus can see all messages, and there are no built-in protection against spoofing. Physical separation and segmentation of bus networks remains one of the key concepts to securing them. In addition, to add security guarantees to insecure bus protocols, we sometimes see complete protocol overhauls that typically break backward compatibility. For example, the insecure Modbus standard from 1979 [1736] has a secure alternative (Modbus/TCP Security Protocol [1794]) since 2018, which wraps bus messages in secure TLS-protected channels.

19.4 NETWORK SECURITY TOOLS

[1731, c8] [1733, c5,c8,c11,c12] [1734, c23] [1735, c6] [1732, c8]

Until now we have discussed attacks and defenses at the protocol level. We will now introduce additional and orthogonal tools to these protocol-level defenses. Many of these tools have become de facto standards on top of the aforementioned security schemes at the protocol level. We only provide a brief overview here. The effective deployment of these tools is covered in detail in the Security Operations & Incident Management Knowledge Area (Chapter 8).

19.4.1 Firewalling

Firewalls can be co-located with routers or implemented as specialised servers. In either case, they are gatekeepers, inspecting all incoming/outgoing traffic. Firewall systems are typically configured as bastion hosts, i.e., minimal systems hardened against attacks. They apply traffic filters based on a network's security policy and treat all network packets accordingly. The term filter is used for a set of rules configured by an administrator to inspect a packet and perform a matching action, e.g., let the packet through, drop the packet, drop and generate a notification to the sender via ICMP messages. Packets may be filtered according to their source and destination network addresses, protocol type (TCP, UDP, ICMP), TCP or UDP source/destination port numbers, TCP Flag bits (SYN/ACK), rules for traffic from a host or leaving the network via a particular interface and so on. Traditionally, firewalls were pure packet filters, which worked on inspecting header field only. By now, firewalls can also be stateful, i.e., they retain state information about flows and can map packets to streams. While stateful firewalls allow to monitor related traffic and can map communication to flows, this comes at the cost of maintaining (possibly lots of) state.

Rule	State	Src IP	Src Port	Dst IP	Dst Port	Proto	Action
#1	NEW	172.16.0.0/24	*	*	80, 443	TCP	ACCEPT
#2	NEW	*	*	172.16.20.5	22	TCP	ACCEPT
#3	ESTABLISHED	*	*	*	*	TCP	ACCEPT
#4	*	*	*	*	*	*	DROP

Figure 19.6: Firewalling example. Rule #1 allows outgoing HTTP(S), rule #2 allows incoming SSH.

Figure 19.6 shows a simple example firewall configuration. All internal hosts (here, in network `172.16.0.0/24`) are allowed to communicate to TCP ports 80/443 for HTTP/HTTPS to external hosts (rule #1). External hosts can connect to an internal SSH server via TCP on port 22 (rule #2). All follow-up communication of these connections is granted (rule #3). Any other communication is dropped (rule #4). In reality, firewall configurations can become incredibly more complex than this minimal example. Specifying complete and coherent policies is typically hard. It typically helps to first lay out a firewall decision diagram, which is then—ideally automatically—transferred into concrete, functionally equivalent firewall policies [1795]. Tools like Firewall Builder [1796] or Capirca [1797] can assist in this process.

Application Gateway (AG): Application gateways, aka application proxies, perform access control and thus facilitate any additional requirements of user authentication before a session is admitted. These AGs can also inspect content at the application layer, unless fully

encrypted. In a typical setting, the application gateway will use a firewall's services after performing authentication and policy enforcement. A client wanting to access an external service would connect to the AG first. The AG would prompt them for authentication before initiating a session to the external server. The AG would now establish the connection with the destination acting as a relay on behalf of the client, essentially creating two sessions like a PITM. Another interesting application of an AG is TLS termination. An incoming webserver TLS connection could be terminated at the AG, so that it could do the resource intensive encryption/decryption and pass the un-encrypted traffic to the back-end servers. In practice, the AGs are also configured to inspect encrypted outbound traffic where the clients are configured with corresponding certificates installed at the AG.

Circuit-level Gateway (CG): A CG is a proxy that functions as a relay for TCP connections, thus allowing hosts from a corporate Intranet to make TCP connections over the Internet. CGs are typically co-located with a firewall. The most widely used CG today is SOCKS. For end user applications, it runs transparently as long as the hosts are configured to use SOCKS in place of a standard socket interface. A CG is simple to implement compared to an AG, as it does not need to understand application layer protocols.

DMZ: Network design ensures careful firewall placements by segmenting networks. Typically, the Demilitarised Zone (DMZ) (aka a perimeter network) is created. All external untrusted users are restricted from using the services available in this zone. Typically, an organisation's public web server and authoritative DNS would reside in the DMZ. The rest of the network is partitioned into several security zones by a security architect. For example, a payment database would be deployed to an isolated network, so would an internal file server.

19.4.2 Intrusion Detection and Prevention Systems

Intrusion Detection Systems (IDSs) can provide valuable information about anomalous network behaviour. They inspect payload, higher-layer information and many more attributes of sessions beyond what a firewall can do. An IDS would monitor network traffic with the help of agents/sensors/monitors on the network and sets off alarms when it detects (or thinks it has) suspicious activity. Essentially, the IDS would compare the traffic against what it considers normal traffic and, using a range of techniques, would generate an alert. IDSs can operate purely on traffic statistics that can be derived from header data. Alternatively, Deep Packet Inspection (DPI) allows to inspect transport- or application-layer payloads to recognize known malicious communication patterns (e.g., of malware). There are several widely-used IDSs such as Snort [859], Zeek [1798] or Suricata [1799]. IDSs have been used in numerous contexts, such as for detecting malware [671, 1800, 1801, 1802], attacks in automotive networks [1803] or against unmanned vehicles [1804], software exploits [1805], DoS attacks [1806] or attacks in wireless ad-hoc networks [1807].

The accuracy of an IDS remains a fundamental operational challenge. False alarms are a huge problem for network/security administrators despite decades of research. An IDS may generate false positives for legitimate hosts carrying out suspicious yet benign behaviour. Likewise, a false negative occurs when malicious activity remains undetected.

Signature-based IDSs compare monitored traffic against a database of known malicious communication patterns. The database has to be continually updated, and despite all efforts, will never be complete. Signatures can be as simple as a source/destination IP address or other protocol headers, or match payload patterns. Rule specification goes beyond the scope of this chapter and is covered by more detailed textbooks [1808, 1809]. The following toy rule

checks generates an alert if the payload of TCP/80 connection to network 192.168.5.7/24 contains a 'GET' string.

```
alert tcp any any -> 192.168.5.7/24 80
(content:"GET"; msg:"GET_has_been_detected";)
```

A signature-based IDS generates a heavy workload, as it has to compare huge numbers of signatures. Speed of detection plays a key role in preventing these attacks. Several systems deploy parallel and distributed detection systems that can cope with high traffic rates on large networks and allow online detection; others exploit parallelism at the hardware level in order to overcome processing delays so that packets and flows can be processed at high speeds.

Anomaly-based IDSs compare monitored traffic to behavioral models built previously "normal" traffic during a learning phase. Instead of blocking certain patterns, anomaly detection allows all communication it deems as benign, and blocks the rest. The fundamentally hard problem here is to capture normal traffic that is both clean (i.e., does not contain malicious behavior) and sufficiently representative (i.e., it also captures benign behavior that will arise in the future). For example, an anomaly-based system based on statistical features could capture bandwidth usage, protocols, ports, arrival rate and burstiness [869]. In this example, a large percentage of port scans would be anomalous and generate an alert. Despite using machine learning techniques, anomaly-based IDSs accuracy remains unsatisfying in practical deployments [1810].

Another way of classifying IDSs is the point of monitoring for malicious behaviour. A Host Intrusion Detection System (HIDS) runs on individual hosts in the network. Most virus scan software would have this feature where they also monitor inbound and outbound traffic in addition to the usual virus scanning. This can be particularly helpful if the hosts have been compromised and form part of a bot to attack other servers/networks. In contrast, a network intrusion detection system is deployed at strategic locations within the network to monitor inbound and outbound traffic to and from the devices in various segments of the network.

Intrusion Prevention System (IPS): An IPS distinguishes itself from an IDS in that it can be configured to block potential threats by setting filtering criteria on routers/switches at various locations in the network. IPS systems monitor traffic in real time dropping any suspected malicious packets, blocking traffic from malicious source addresses or resetting suspect connections. In most cases, an IPS would also have IDS capabilities.

19.4.3 Network Security Monitoring

Several other network monitoring tools help to understand the security situation of a network. We will briefly discuss these monitoring methodologies and mention example uses cases.

Flow monitoring standards such as NetFlow [1811] or IPFIX [1812] aggregate statistical information of all communication streams within a network. They provide a sweet spot between recording all network communication and nothing at all. Flow aggregation typically requires little computational resources and has low storage demands, enabling for long-term storage. Flow data comes handy during network forensics, or even as input for anomaly detection [1813].

Network forensics enable administrators to extract application payload observed on their networks. When used as sniffer or when applied on recorded traffic, frameworks such as NetworkMiner [1814] or Xplico [1815] can, e.g., extract files, emails and HTTP sessions. They often come with further functionality to fingerprint the network hosts and to map network

hosts to locations. Having said this, unless augmented with the according key material, these forensic tools are limited to analyzing non-secure communication.

Network scans allow network administrators to enumerate hosts and services within their network (or, optionally, the entire Internet). There are numerous tools such as Nmap [1816] or Zmap [1767] that can send, e.g., ICMP and SYN probes at scale.

IP telescopes are publicly reachable network ranges that do not host any service or client. Given these networks are still routed, though, one can monitor any traffic sent to them and derive interesting observations. For example, IP telescopes help observing network scans by others [1817]. Similarly, they allow to spot backscatter [1818], i.e., responses of traffic that attackers have provoked when using the telescope's IP addresses in IP spoofing attacks (e.g., when assigning random IP addresses during SYN floods).

Honeypots are system used by defenders to trap attackers. They are intentionally vulnerable yet well-isolated client or server systems that are exposed to attackers. There is a wide diversity of client-side honeypots (e.g., to emulate browser vulnerabilities [1819]) and server-side honeypots (e.g., to emulate service vulnerabilities [1820, 1821], or to attract DDoS attacks [1822]). Observing the techniques attackers use to exploit these honeypots gives valuable insights into tactics and procedures.

Network reputation services can help to assess the trustworthiness of individual network entities such as IP addresses or domain names. Based on past behaviour observed from an entity, these mostly commercial providers publish a score that serves as reputation for others. Identifying badly reputed hosts in network traffic can help to detect known attackers or connections to botnets. Reputation services are, however, limited in coverage and accuracy due to volatile domaind and IP address usage of attacking hosts.

Finally, Security Information and Event Management (SIEM) systems collect events from security-critical sensors (e.g., IDS, firewalls, host-based sensors, system log files). A SIEM system then analyes these events to distill and raise security-critical incidents for further inspection. It is particularly the combination of multiple data sources (system log files, host-based anomaly sensors, firewall or IDS events) that makes SIEM so successful in detecting, e.g., brute force attacks, worm propagations, or scans.

19.4.4 SDN and NFV Security

The SDN architecture enables for novel threat detection and attack prevention capabilities [1823, 1824]. For example, the central SDN controller(s) can more accurately infer DDoS attacks, and automate mitigation strategies by dynamically reprogram switches to drop malicious traffic flows. Furthermore, infected hosts can automatically be routed to an isolated network region (sometimes called a *walled garden*) issuing quarantine notifications to users of infected systems. SDN allows for such an immediate network isolation via software-controlled changes in the network—which was tedious reconfiguration labor before. Similarly, network designs can be rapidly scaled to higher loads.

At the same time, the SDN management plane offers a unique attack vector. Intruders gaining power over the SDN controller undermine any security guarantees that SDN bring, and have additionally the power to reconfigure networks at will. It is therefore of utmost importance that the SDN controller software and the underlying platform follow strong security best practices, including hardened software and strict access control. Furthermore, the SDN platform has to be protected against new SDN-specific threats. For example, the SDN controllers use a

Spanning Tree Algorithm (SPTA) for topology updates. In a DoS attack, an adversary could advertise a fake link and force the SPTA to block legitimate ports. Similarly, being in a central position, SDN controllers can be target of DoS attacks [1825]. Furthermore, Hong et al. [1826] provide a number of attack vectors on practical SDN switch implementations. SDN switches are also prone to a *timing side channel attack* [1827]. For example, attackers can send a packet and measure the time it takes the switch to process this packet. For a new packet, the switch will need to fetch a new rule from the controller, thus resulting in additional delay over the flows that already have rules installed at the switch. Consequently, the attacker can determine whether an exchange between an IDS and a database server has taken place, or whether a host has visited a particular website. A possible countermeasure would introduce delay for the first few packets of every flow even if a rule exists [1828]. A more extensive analysis of SDN vulnerabilities in general can be found in a study by Zerkane et al. [1829].

Network Functions Virtualisation (NFV) aims to reduce capex and allow for the rapid introduction of new services to the market. Specialised network middleboxes such as firewalls, encoders/decoders, DMZs and deep packet inspection units are typically closed black box devices running proprietary software [1830]. NFV researchers have proposed the deployment of these middleboxes entirely as virtualised software modules and managed via standardised and open APIs. These modules are called Virtual Network Functions (VNFs). A large number of possible attacks concern the Virtual Machine (Hypervisor) as well as configuring virtual functions. Lal et al. [1831] provide a table of NFV security issues and best practice for addressing them. For example, an attacker can compromise a VNF and spawn other new VNFs to change the configuration of a network by blocking certain legitimate ports. The authors suggest hypervisor introspection and security zoning as mitigation techniques. Yang et al. [1832] provide a comprehensive survey on security issues in NFV.

19.4.5 Network Access Control

Networks are typically quite lax about which devices can become part of them. Section 19.3.4.1 described port-based device authentication devices, which demands secrets from trusted devices. Unfortunately, this still gives no security guarantees on the trustworthiness of network devices. For example, while a device may have been deemed trustworthy at times, system compromises may have caused the system to boot into an untrusted state. Network Access Control, usually implemented using the Trusted Network Connect (TNC) architecture [1833], enforces configurable security policies of devices when they join networks. These policies enforce that network clients have been booted into trustworthy configurations. This may even enable firewalls to precisely attribute which client software has caused certain traffic [1834]. Technically, to perform remote attestation, the verifier relies on unforgeable trusted hardware on the proving device. Technical details can be found in the Hardware Security Knowledge Area (Chapter 20).

The main practical drawback of such policies is that they attestate only the initial system state. Malicious runtime modifications to the system are not possible with TNC. Validating if a device was compromised *after* it entered a trusted boot state is still subject to research, e.g., in runtime-attestation schemes via remote control-flow enforcement [1835, 1836].

19.4.6 Zero Trust Networking

Zero trust networks radically give up the idea of blindly trusting devices within a assumed-to-be trusted part of a network. In zero trust networks, all devices are untrusted unless proven otherwise. This represent a paradigm shift which arose out of the challenges in defining centralized perimeters (e.g., a firewall) that split networks into trusted/untrusted domains. The main motivation here is that traditional networks keep losing control over which devices join the seemingly trusted side of a network (fueled by, e.g., bring-your-own-device). At the same time, devices temporarily jump from trusted to untrusted networks (e.g., work-from-home), before rejoining the trusted networks in a potentially modified state.

Migrating traditional network designs to zero trust networks is not trivial. NCSC provides a comprehensive tutorial which can serve as a great starting point [1837]. In essence, a transition to zero trust networks first requires a deep understanding of the assets in a network, such as users, devices, services and data. Similarly, administrators require capabilities to measure the security state of these assets. Any request to services must be authorized using strong multi-factor authentication (described in the Authentication, Authorisation & Accountability Knowledge Area (Chapter 14)), ideally paired with a single sign-on scheme not to frustrate users. Not all legacy services can readily be plugged into such a zero-trust setting, requiring adaptations to make them compatible to standard authentication schemes (e.g., OpenID Connect, OAuth, SAML).

One popular example of such a zero trust network design is *BeyondCorp* [1838]. It leverages network access control (see Section 19.4.5) to identify devices, and rigorously enforces user identification using a centralized single sign-on system. In BeyondCorp, previously internal application services become external ones, protected by an access proxy that rigorously enforces strong encryption and access control.

19.4.7 DoS Countermeasures

Denial of Service (DoS) attacks can roughly be categorized into two categories, depending on which resources they aim to exhaust. First, in volumetric DoS attacks, adversaries aim to exhaust the network bandwidth of a victim. Amplification attacks (see Section 19.3.2.4) are the most dominant instance of such attacks, but also large-scale Distributed Denial of Service (DDoS) attacks from remote-controlled botnets can leverage high attack bandwidths. Attack targets are typically individual services or networks, yet can also be entire links in the upper Internet hierarchy (and their depending ASs) that become congested [1839, 1840]. Volumetric attacks can be mitigated most effectively when traffic is stopped as early as possible before it reaches the target network. For example, commercial so-called scrubbing services help to filter malicious network traffic before it reaches the target. Technically, scrubbing services are high-bandwidth network providers that—with the help of their customers—place themselves between the Internet and an organization's perimeter. Alternatively, attack victims can null route traffic towards certain subnetworks via BGP advertisements to drop their traffic, or use BGP FlowSpec to filter traffic at powerful edge routers.

Second, in application-level DoS attacks, miscreants aim to cripple resources at the software layer. They typically aim to exhaust memory or computation resources (e.g., CPU). Here, defenses are quite application specific. For example, SYN cookies (see Section 19.3.2.3) and rate limiting protect TCP-based applications against connection floods. Also, CAPTCHAs may help to further distinguish between human- and or computer-generated communication, which is especially useful in the Web context.

19.5 CONCLUSION

[1733, c5] [1735, c8,c11] [1732, c6]

19.5.1 The Art of Secure Networking

We covered a broad arsenal of network security instruments and schemes that network architects, network operators and communication partners can employ. It is important to understand that there is no silver bullet to obtain "a secure" network or communication. Unfortunately, it is rarely even possible to *guarantee* that none of the security goals will be broken ever. Consequently, it requires a thorough combination of these principles to obtain a reasonable and satisfactory level of security.

Having said this, there are fundamentals that we should strive for, and for which we have proven and standardized means. Endpoints can securely communicate using TLS. Sometimes these endpoints do however not represent the final recipients of sensitive data, such as for email or messenger servers which may "buffer" messages until the recipient fetches them. In these cases, we can deploy asynchronous end-to-end security schemes at the application layer (e.g., PGP/SMIME) that tolerate untrusted middle hops. Network operators must be prepared for attackers from within their network, and from outsiders. To protect against external threats, they can deploy zero trust networking, or use firewalls for more centralized architectures. On top, an IDS helps to identify threats at the payload level that were unnoticed by the firewall, and network monitoring in general will allow for *a posterio* network forensics. Insider attacks are much harder to mitigate, especially if trusted devices have been compromised by attackers, yet port-based authentication or even network access control are good starting points. In any case, decent network defenses require a fundamental interplay of several security best practices.

19.5.2 Further Network Security Topics

Network security is too broad to be fully captured at reasonable detail within a single knowledge area. Therefore, we have excluded several topics that are closely related, yet (still) relatively special. We will very briefly outline them in the following.

Cloud and Data Center Security: As soon as organizations outsource computations (cloud) or information (data center), this immediately triggers security demands. They are only partially connected to network security, though. For example, to not expose data and computations to cloud operators, clients can store data in hardware-backed secure containers such as Intel SGX [1841]. Data centers and clouds run into risks that adversaries may abuse side channel to leak sensitive data from co-located services or systems—a topic that is discussed in detail in the Operating Systems & Virtualisation Knowledge Area (Chapter 11).

Delay-Tolerant Networks and Ad-hoc Sensors Networks: Not all networks guarantee that communication partners are online, reachable and responsive all the time. Sensor networks are one such example, where energy-constrained sensors just wake up periodically to exchange information, and usually hibernate. Similarly, the speed of light implies that devices in networks in space have significant message inter-arrival times (e.g., already over 2 seconds between Earth and Moon). In general, this requires delay-tolerant networks, which are incompatible with many of the aforementioned security principles, most of which assume reliable and quick responsiveness of communication endpoints. A detailed treatment of this subject goes

beyond this KA. A great starting point for further reading is Ivancic' security analysis on delay-tolerant networks [1842].

Network Covert Channels: Network covert channels aim to hide the pure existence of communication, e.g., using steganography. They allow two or more collaborating attacker processes to leak sensitive information despite network policies that should prevent such leakage. For example, attackers may encode sensitive information in TCP headers that will remain unnoticed by IDS [1843]. Similar covert channels are possible for other protocols, such as DNS [1844] or IP [1845]. Covert channels can be confined by carefully modeling and observing all protocols fields or patterns in general that could be abused for hiding information [1846].

Payment Networks: The banking sector foresees its own proprietary standards and network protocols. Exploring those in detail goes beyond the scope of this document, particularly also because protocols can be even specific to certain regions (e.g., *FinTS* in Germany) or special purposes (e.g., *3-D Secure* for securing credit card transactions). The rise of digital currencies such as Bitcoin which implement several protocols on their own add further complexity. Finally, stock exchanges nowadays heavily depend on reliable networks, and are extremely sensitive to timing attacks that require careful Quality-of-Service assurances [1847, 1848].

Physical-Layer Security: Our security analyses stopped at the logical part of the link layer. The physical part of this layer deserves further attention and indeed is a subject on its own. In fact, we witnessed several recent advancements in this field, such as Bluetooth Low Energy, distance bounding and positioning protocols, Near-Field Communication (NFC) or cellular networks. For a detailed treatment of this subject, we refer to the Physical Layer and Telecommunications Security Knowledge Area (Chapter 22).

Networking Infrastructure Security: We have so far assumed that networking components are fully trusted. However, with global supply chains that involve dozens of parties and countries during manufacturing a component, such assumption may be easily invalidated in practice. What happens if network infrastructure, which often is part of critical infrastructures, *cannot* be trusted, e.g., due to backdoors or software vulnerabilities? Answering this question is far from trivial, as it depends on which components and which security guarantees are at stake. One recent real-world example of such an analysis happens in 5G networks, where some countries ban hardware that is delivered by some other countries, simply because of lacking trust. This quickly turns into a non-networking issue that finds its solutions in other chapters, such as in the Software Security Knowledge Area (Chapter 15), the Secure Software Lifecycle Knowledge Area (Chapter 17) or Hardware Security Knowledge Area (Chapter 20). Discussing non-trustworthy networking components goes beyond the scope of this chapter.

Cross-Border Regulations: Networks that span several countries and thus legislations are quite interesting from a law perspective. There may be conflicts of law, e.g., regarding patents, export restrictions, or simply the question whether or not a digital signature is legally binding. These topics are addressed in depth in the Law & Regulation Knowledge Area (Chapter 3).

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[1731]	[1733]	[1734]	[1735]	[1732]
19.1 Security Goals and Attacker Models	c8	c1	c1	c6	c8
19.2 Networking Applications	c1				c1
19.3.1 Security at the Application Layer	c8		c6,c15,c19–c22		c8
19.3.2 Security at the Transport Layer	c8	c4,c6			c8
19.3.3 Security at the Internet Layer	c8	c5,c9	c17		c8
19.3.4 Security on the Link Layer	c8	c7			c8
19.4 Network Security Tools	c8	c5,c8,c11,c12	c23	c6	c8
19.5 Conclusion		c5		c8,c11	c6

Chapter 20
Hardware Security

Ingrid Verbauwhede | KU Leuven

INTRODUCTION

Hardware security covers a broad range of topics from trusted computing to Trojan circuits. To classify these topics we follow the different hardware abstraction layers as introduced by the Y-chart of Gajski & Kuhn. The different layers of the hardware design process will be introduced in section 20.1. It is linked with the important concept of a root of trust and associated threat models in the context of hardware security. Next follows section 20.2 on measuring and evaluating hardware security. The next sections gradually reduce the abstraction level. Section 20.3 describes secure platforms, i.e. a complete system or system-on-chip as trusted computing base. Next section 20.4 covers hardware support for software security: what features should a programmable processor include to support software security. This section is closely related to the Software Security Knowledge Area (Chapter 15). Register transfer level is the next abstraction level down, covered in section 20.5. Focus at this level is typically the efficient and secure implementation of cryptographic algorithms so that they can be mapped on ASIC or FPGA. This section is closely related to the Cryptography Knowledge Area (Chapter 10). All implementations also need protection against physical attacks, most importantly against side-channel and fault attacks. Physical attacks and countermeasures are described in section 20.6. Section 20.7 describes entropy sources at the lowest abstraction level, close to CMOS technology. It includes the design of random numbers generators and physically unclonable functions. The last technical section describes aspects related to the hardware design process itself. This chapter ends with the conclusion and an outlook on hardware security.

20.1 HARDWARE DESIGN CYCLE AND ITS LINK TO HARDWARE SECURITY

Hardware security is a very broad topic and many topics fall under its umbrella. In this section, these seemingly unrelated topics are grouped and ordered according to the design levels of abstraction as introduced by the Y-chart of Gajski & Kuhn [1849]. While Gajski & Kuhn propose a general approach to hardware design, in this chapter it is applied to the security aspects of hardware design and it is linked to threat models and the associated root of trust.

20.1.1 Short background on the hardware design process

Design abstraction layers are introduced in hardware design to reduce the complexity of the design. As indicated in 20.1, the lowest abstraction level a designer considers are individual transistors at the center of the figure. These transistors are composed together to form basic logic gates, such as NAND, NOR gates or flip-flops, called the logic level. Going one abstraction layer up, at register transfer level gates are grouped together to form modules, registers, ALU's, etc, and their operation is synchronized by a clock. These modules are then composed to form processors, specified by instruction sets, upon which applications and algorithms can be implemented.

By going up in the abstraction layers, details of underlying layers are hidden. This reduces design complexity at higher abstraction layers. The abstraction layers are represented by concentric circles in figure 20.1. Upon these circles, the Y-chart of Gajski & Kuhn introduces 3 design activities, represented by three axes: a behavioral axis, describing the behavior or *what*

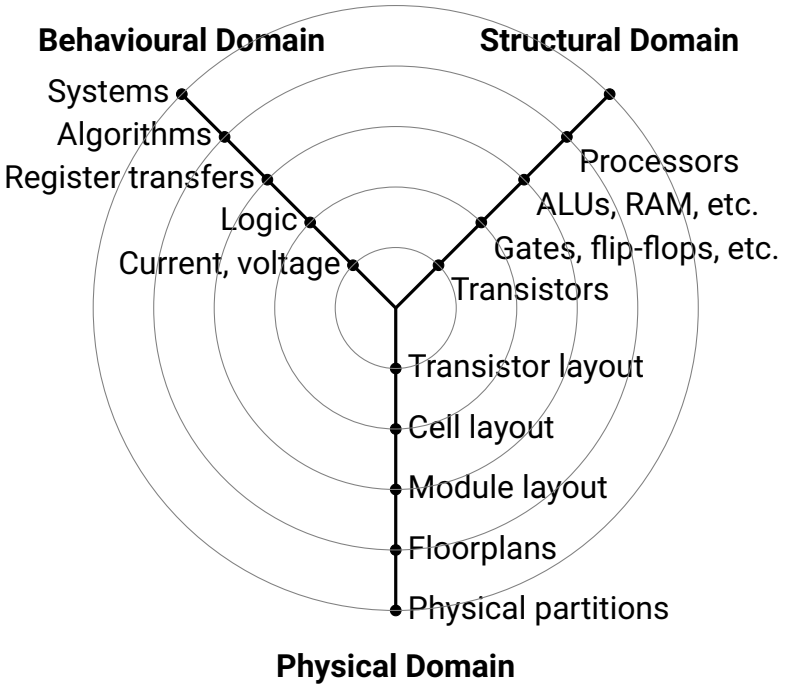


Figure 20.1: Gajski-Kuhn Y-chart

needs to be implemented (aka specifications), a structural axis describing *how* something is implemented and a physical axis, how the layouts are composed together at gate, module, chip, board level. An actual design activity is a 'walk' through this design space. Typically one starts with the specifications at the top of the behavioral domain. These specifications (=what) are decomposed in components at the same level of abstraction (=how) moving from the behavioral axis to the structural axis. A structural component at one abstraction level becomes a behavioral component at one level down.

As an example of a walk through the design space: Assume a hardware designer is requested to implement a light-weight, low power security protocol for an Internet of Things (IoT) device. This designer will only receive specifications on what needs to be designed: a security protocol aims at providing confidentiality and integrity (= what) and a set of cryptographic algorithms (= components) to support the protocol. The crypto-algorithms are provided as a behavioral specification to the hardware designer, who has the choice of implementing it as a dedicated co-processor, as an assembly program, or support it with a set of custom instructions. Depending on costs and volumes, a choice of a target CMOS technology or an FPGA platform is made. This behavioral level will be translated into a more detailed register-transfer level description (e.g. VHDL or Verilog). At the Register Transfer Level (RTL), decisions need to be made if this will be a parallel or sequential version, a dedicated or programmable design, with or without countermeasures against side-channel and fault attacks, etc.

Essential for the division in design abstraction layers, is the creation of models on how components behave. E.g. to simulate the throughput or energy consumption of a arithmetic unit, quality models of the underlying gates need to be available. Similarly, the Instruction Set Architecture is a model of a processor available to the programmer.

20.1.2 Root of trust

In the context of security, a root of trust is a model of an underlying component for the purpose of security evaluation. According to Anderson [1013]: "A root of trust is a component used to realize a security function, upon which a designer relies but of which the trustworthiness can not be explicitly verified." The designer uses one or multiple components to construct a security function, which then defines the trusted computing base. It is defined by the trusted computing group as follows: *"An entity can be trusted if it always behaves in the expected manner for the intended purpose."* [1850].

E.g. for an application developer, a Trusted Platform Module (TPM) or a Subscriber Identity Module (SIM) are a root of trust which the developer uses to construct a security application. For the TPM designer, the TPM is composition of smaller components which are composed together to provide security functionality. At the lowest hardware abstraction layers, basic roots of trust are the secure storage of the key in memory or the quality of the True Random Number Generator.

Hardware security is used as an enabler for software and system security. For this reason, hardware provides basic security services such as secure storage, isolation or attestation. The software or system considers the hardware as the trusted computing base. And thus from a systems or application view point, hardware has to behave as a trusted component. However, the hardware implementation can violate the trust assumption. E.g. Trojan circuits or side-channel attacks could leak the key or other sensitive data to an attacker. Hence, hardware itself also needs security. Moreover hardware needs security at all abstraction layers. Therefore, at every abstraction layer, a threat model and associated trust assumptions need to be made. An alternative definition for a root of trust in the context of design abstraction layers is therefore: "A root of trust is a component at a lower abstraction layer, upon which the system relies for its security. Its trustworthiness can either not be verified, or it is verified at a lower hardware design abstraction layer. "

20.1.3 Threat model

A threat model is associated with each root of trust. When using a root of trust, it is assumed that the threat model is not violated. This means that the threat model is also linked to the hardware abstraction layers. If we consider a root of trust at a particular abstraction layer, then all components that constitute this root of trust, are also considered trusted.

Example 1: security protocols assume that the secret key is securely stored and not accessible to the attacker. The root of trust, upon which the protocol relies, is the availability of secure memory to guard this key. For the protocol designer, this secure memory is a black box. The hardware designer has to decompose this requirement for a secure memory into a set of requirements at a lower abstraction layer. What type of memory will be used? On which busses will the key travel? Which other hardware components or software have access to the storage? Can there be side-channel leaks?

Example 2: It is during this translation of higher abstraction layer requirements from protocol or security application developers into lower abstraction layers for the hardware designers that many security vulnerabilities occur. Implementations of cryptographic algorithms used to be considered black boxes to the attacker: only inputs/outputs at the algorithm level are available to mount mostly mathematical cryptanalysis attacks. However, with the appearance of side-channel attacks (see section 20.6) this black box assumption no longer holds. Taking

side-channel leakage into account the attacker has the algorithm level information as well as the extra timing, power, electro-magnetic information as observable from the outside of the chip. Thus the attacker model moves from black box to gray box. It is still assumed that the attacker does not know the details of the internals, e.g. the contents of the key registers.

Example 3: for programmable processors, the model between hardware and software is traditionally considered the Instruction Set Architecture (ISA). The ISA is what is visible to the software programmer and the implementation of the ISA is left to the hardware designer. The ISA used to be considered the trust boundary for the software designer. Yet, with the discovery of micro-architectural side-channel attacks, such as Spectre, Meltdown, Foreshadow, this ISA model is no longer a black box, as also micro-architectural information and leakage are available to the attacker [1851].

20.1.4 Root of trust, threat model and hardware design abstraction layers

The decomposition in abstraction layers, in combination with Electronic Design Automation (EDA) tools, is one of the main reasons that the exponential growth of Moore's law was sustainable in the past decades and it still is. This approach works well when optimizing for performance, area, energy or power consumption. Yet for hardware security, no such general decomposition exists.

In this chapter, we propose to organise the different hardware security topics, their associated threat models and root of trust according to the hardware design abstraction layers, as there is no known other general body of knowledge available to organize the topics. This organization has the advantage that it can be used to identify the state of the art on different subtopics of hardware security. As an example, in the specific context of hardware implementations of cryptographic algorithms, the state of the art is well advanced and robust countermeasures exist to protect cryptographic implementations against a wide range of side-channel attacks, as shown in detail in section 20.5. Yet in the context of general processor security, e.g. to isolate process related data or to provide secure execution, new security hazards continue to be discovered on a regular basis.

In an attempt to order the topics, table 20.1 summarizes this organization. The different abstraction layers are identified (first column) from a hardware perspective. The highest level (system and software) sits on top of the hardware platform. E.g. a system designer assumes that a secure platform is available. Thus the secure platform is the root of trust, providing security functionality. The second column describes the functionality provided by the root of trust. The third column describes how this functionality might be implemented. E.g. at the highest abstraction layer this might be by providing a Trusted Execution Module or a secure element, etc. The fourth column describes the threat models and attack categories at that abstraction layer. E.g. at system level, the system designer assumes that they will receive a module that provides isolation, integrity, attestation, etc. The last column describes typical design activities at this particular design abstraction layer.

This exercise is repeated for each abstraction layer and described in detail in each of the following sections.

At the processor level, one can distinguish general purpose programmable processors and domain specific processors. General purpose processors should support a wide range of applications, which unfortunately typically include software vulnerabilities. Hardware features

Abstraction level	Root of trust - functionality	Structural (how) - examples	Example Threats	Typical HW design activities
System and application	Secure platforms	e.g. Trusted Execution (Trustzone, SGX, TEE), HSM, Secure Element	to support isolation, integrity, attestation, . . .	security application development
Processor	general purpose	e.g. shadow stack	SW vulnerabilities	ISA, HW/SW co-design
Processor	domain specific	Crypto specific RTL	Timing attacks	Constant number of clock cycles
Register Transfer	Crypto specific	Building blocks,	Side Channel Attack,	Logic synthesis
Logic	Resistance to SCA, Power, EM, fault	Masking, Circuit styles	Side Channel attack, fault	FPGA tools, standard cell design
Circuit and technology	Source of entropy	TRNG, PUF, Secure SRAM	Temperature, glitches	SPICE simulations
Physical	Tamper Resistance	Shields, sensors	Probing, heating	Layout activities

Table 20.1: Design abstraction layers linked to threat models, root of trust and design activities

are added to address these software vulnerabilities, such as a shadow stack or measures to support hardware control flow integrity. Domain specific processors typically focus on a limited functionality. They are typically developed as co-processors in larger systems-on-chip. Typical examples are co-processors to support public key or secret key cryptographic algorithms. Time at the processor level is typically measured in instruction cycles.

Both general purpose and domain specific processors are composed together from computational units, multipliers and ALU's, memory and interconnect. These modules are typically described at the register transfer level: constant-time and resistance against side-channel attacks become the focus. Time at this level is typically measured in clock cycles.

Multipliers, ALU's, memories, interconnect and bus infrastructure are created from gates and flip-flops at the logic level. At this design abstraction level, focus is on leakage through physical side-channels, power, electro-magnetic, and fault attacks. Time is typically measured in absolute time (nsec) based on the available standard cell libraries or FPGA platforms.

The design of entropy sources requires knowledge and insights into the behavior of transistors and the underlying Complementary Metal-Oxide-Semiconductor (CMOS) technology.The design of these hardware security primitives is therefore positioned at the circuit and transistor level. Similarly the design of sensors and shields against physical tampering require insight into the technology. At the circuit and technology level it is measured in absolute time, e.g. nsec delay or GHz clock frequency.

The table 20.1 does not aim to be complete. The idea is to illustrate each abstraction layer with an example. In the next sections, the hardware security goals and their associated threat models will be discussed in detail in relation to and relevance for each abstraction layer.

20.2 MEASURING HARDWARE SECURITY

Depending on the commercial application domain, several industrial and government organizations have issued standards or evaluation procedures. The most well known ones are the FIPS 140-2 (and the older FIPS 140-1), the Common Criteria (CC) evaluation and in the financial world the EMVCO. FIPS 140-2 mostly focuses on the implementation security of cryptographic algorithms. Common Criteria are applicable to IT security in general.

20.2.1 FIPS140-2

FIPS140-2 is a US NIST standard used for the evaluation of cryptographic modules. FIPS140-2 defines security levels from 1 to 4 (1 being the lowest). The following gives a description of the four levels from a physical hardware security point of view. Next to the physical requirements, there are also roles, services and authentication requirements (for more details see [1852] and other KAs).

Security level 1 only requires than an approved cryptographic algorithm be used, e.g. AES or SHA-3, but does not impose physical security requirements. Hence a software implementation could meet level 1. Level 2 requires a first level of tamper evidence. Level 3 also requires the tamper evidence, but on top requires tamper resistance.

NIST defines tampering as an intentional but unauthorized act resulting in the modification of a system, components of systems, its intended behavior, or data, [1853].

Tamper evidence means that there is a proof or testimony that tampering with a hardware module has happened. E.g. a broken seal indicates that a device was opened. A light sensor might observe that the lid of a chip package was lifted.

Tamper resistance means that on top of tamper evidence, protection mechanisms are added to the device. E.g. by extra coating or dense metal layers, it is difficult to probe the key registers.

Level 4 increases the requirements such that the cryptographic module can operate in physically unprotected environments. In this context, the physical side-channel attacks pose an important threat. If any of these physical components depend on sensitive data being processed, information is leaked. Since the device is under normal operation, a classic tamper evidence mechanism will not realize that the device is under attack. See later in section 20.6.

20.2.2 Common criteria and EMVCo

“Common Criteria for information technology security evaluation" is an international standard for IT product security (ISO/IEC 15408), in short known as Common Criteria (CC). CC is a very generic procedure applicable to the security evaluation of IT products. Several parties are involved in this procedure. The customer will define a set of security specifications for its product. The manufacturer will design a product according to these specifications. An independent evaluation lab will verify if the product fulfills the claims made in the security requirements. Certification bodies will issue a certification that the procedure was correctly followed and that the evaluation lab indeed confirmed the claims made. The set of security specifications are collected in a so-called protection profile.

Depending on the amount of effort put into the security evaluation, the CC defines different Evaluation Assurance Levels (EALs). It ranges from basic functional testing, corresponding

to EAL1, to formally verified design and tested, corresponding to the highest level EAL7. CC further subdivides the process of evaluation into several classes, where most of the classes verify the conformity of the device under test. The 5th class (AVA) deals with the actual vulnerability assessment. It is the most important class from a hardware security viewpoint as it searches for vulnerabilities and associated tests. It will assign a rating on the difficulty to execute the test, called the identification, and the possible benefit an attacker can gain from the penetration, called the exploitation. The difficulty is a function of the time required to perform the attack, the expertise of the attacker from layman to multiple experts, how much knowledge of the device is required from simple public information to detailed hardware source code, the number of samples required, and the cost and availability of equipment to perform the attack, etc. A high difficulty level will result in a high score and a high level of the AVA class. The highest score one can obtain is an AVA level of 5, which is required to obtain a top EAL score.

Its usage is well established in the field of smartcards and secure elements as they are used in telecom, financial, government ID's applications. It is also used in the field of Hardware Security Modules, Trusted Platform Modules and some more [1854]. For certain classes of applications minimum sets of requirements are defined into protection profiles. There exists protection profiles for Trusted Platform Module (TPM), Javacards, Biometric passports, SIM cards, secure elements, etc.

Since certification comes from one body, there exist agreements between countries so that the certifications in one country are recognized in other countries. As an exception EMVCo is a private organization to set the specifications for worldwide interoperability of payment transactions. It has its own certification procedure similar to CC.

Please note that the main purpose of a common criteria evaluation is to verify that an IT product delivers the claims promised in the profile. It does not mean that there are no vulnerabilities left. A good introduction to the topic can be found in [1855] and a list of certified products on [1854].

20.2.3 SESIP: Security Evaluation Standard for IoT Platforms

In the context of IoT security evaluation, a recent initiative is the SESIP Security Evaluation scheme [1856], currently at version 1.2. IoT devices are typically small, light-weight 'things', with limited accessibility via internet. Several levels of threat model for IoT are possible: from only remote internet access, over various remote software attack options, to also physical attack resistance. A comprehensive set of security functional requirements are defined: identification and attestation, product lifecycle, secure communication, software and physical attack resistance, cryptographic functionality including random number generation, and some compliance functionality to e.g. provide secure encrypted storage or provide reliable time. Similar to Common Criteria, SESIP provides several levels of assurance. Level 1 is the lowest level and consists of a self-assessment. The highest level of SESIP consists of a full CC evaluation similar to smart cards or secure elements. The levels in between cover from a black box penetration testing over white box penetration testing with or without time limitations.

20.3 SECURE PLATFORMS

This section describes the goals and the state-of-the-art in secure platforms. At this high level of abstraction the system designer receives a complete chip or board as trusted computing base. The system designers assume that the trusted root delivers a set of cryptographic functions, protected by the hardware and software inside the physical enclosure. Common to these platforms is that they are stand-alone pieces of silicon with a strict access policy. Depending on the provided functionality, the hardware tamper resistance and protection levels, and the communication interface, these secure platforms are used in different application fields (automotive, financial, telecom). Three important platforms are the Hardware Security Module (HSM), the Subscriber Identification Module or SIM and the Trusted Platform Module (TPM). These are briefly described next.

20.3.1 HSM Hardware Security Module

A HSM module will typically provide cryptographic operations, e.g. a set of public key and secret key algorithms, together with secure key management including secure generation, storage and deletion of keys. Essential to HSM's is that these operations occur in a hardened and tamper resistant environment. A TRNG and a notion of a real-time clock are usually also included. HSM's are mostly used in server back-end systems to manage keys or payment systems, e.g. in banking systems.

A HSM is used as a co-processor, attached to a host system. Its architecture typically includes a micro-processor/micro-controller, a set of crypto co-processors, secure volatile and non-volatile memory, TRNG, real-time clock, and I/O. The operations occur typically inside a tamper resistant casing. In previous generations, inside the casing multiple components reside on one board.

Recently, in some application domains, such as automotive, HSM functionality is no longer provided as a stand-alone module but is now integrated as a secure co-processor in a larger System on a Chip (SoC). Indeed Moore's law enables higher integration into one SoC. What exactly is covered under HSM functionality depends on the application domain. Therefore, compliance with security levels is also evaluated by specialized independent evaluation labs according to specific protection profiles.

20.3.2 Secure Element and Smartcard

Similar to an HSM, a Secure Element and a smart card provide a set of cryptographic algorithms, public key, secret key, HMAC, etc. together with secure key storage, generation and deletion. The main difference with an HSM are cost, size, and form factor. They are typically implemented as one single integrated circuit and have a much smaller form factor from around 50 cm^2 to less than 1 cm^2. The main difference between a smart card and a secure element sits in the form factor and the different markets they address. Secure elements are a more generic term, while smart cards have the very specific form factor of a banking card. They are produced in large volumes and need to be very cheap as they are used for SIM cards in cell phones and smart phones. They are also used in banking cards, pay-TV systems access cards, national identity cards and passports, and recently in IOT devices, vehicular systems and so on. Tamper resistance and physical protection are essential to secure elements. They are a clear instance of what in a computer architecture domain are called 'domain specific processors'. Specific

protection profiles exist depending the application domain: financial, automotive, pay-TV, etc.

A typical embedded secure element is one integrated circuit with no external components. It consists of a small micro-controller with cryptographic co-processors, secure volatile and non-volatile storage, TRNG, etc. I/O is usually limited, through a specific set of pins, or through a NFC wireless connection. Building a secure element is a challenge for a hardware designer, as one needs to combine security with non-security requirements of embedded circuits: small form factor (no external memory), low power and/or low energy consumption in combination with tamper resistance and resistance against physical attacks, such as side-channel and fault attacks (see section 20.6).

20.3.3 Trusted Platform Module (TPM)

The TPM module has been defined by the Trusted Computing Group (TCG), an industry association, to provide specific security functions to the Personal Computer (PC) platform. More specifically, the TPM is a root of trust embedded on the PC platform, so that PC+TPM platform can identify itself and its current configuration and running software [1850]. The TPM provides three specific roots of trust: the Root of Trust for Measurement (RTM), the Root of Trust for Storage (RTS), the Root of Trust for Reporting (RTR). Besides these three basic functions, other functionality of TPMs is being used: access to specific cryptographic functions, secure key storage, support for secure login, etc.

The TPM is implemented as a separate security module, much like a secure element but with a specific bus interface to a PC platform, e.g. through the LPC or I^2C bus interface. Its architecture at minimum consists of an embedded micro-controller, several crypto coprocessors, secure volatile and non-volatile storage for root keys and a high quality true random number generator. It includes hardware engines for hash functions (SHA1 and SHA256), public key (RSA and ECC), secret key (AES) and HMAC calculations. Since a TPM is a separate module, physical protection and tamper resistance is essential for security. Next to its main scope of integrity protection, TPM also has applications in disk encryption, digital rights management, etc.

The most recent TPM2.0 version broadens the application scope from PC oriented to also supporting networking, embedded, automotive, IoT, and so on. It also provides a more flexible approach in the functionality included. Four types of TPM are identified: the dedicated integrated circuit 'discrete element' TPM provides the highest security level. One step lower in protection level is the 'integrated TPM' as an IP module in a larger SoC. The lowest levels of protection are provided by the firmware and software TPM.

The adoption of TPMs has evolved differently from what was originally the focus of the TCG. Originally, the main focus was the support of a secure boot and the associated software stack, so that a complete measurement of the software installed could be made. The problem is that the complexity of this complete software base grows too quickly, making it too difficult to measure completely all variations in valid configurations. Thus TPMs are less used to protect a complete software stack up to the higher layers of software. Still most new PCs now have TPMs but they are used to protect the encryption keys, avoid firmware roll-back, and assist the boot process in general.

Starting from the original TPM, the Trusted Computing Group has broadened its scope and now has working groups on many different application, such as cloud, embedded systems, IoT, mobile, network equipment, and so on, [1857].

20.4 HARDWARE SUPPORT FOR SOFTWARE SECURITY AT ARCHITECTURE LEVEL

At the secure platform level, the complete module, i.e. hardware and its enclosed embedded software, are part of the trusted computing base. One level down on the abstraction layers, we make the assumption that all hardware is trusted, while software is no longer trusted. Indeed, software vulnerabilities are a major source of security weaknesses (see the Software Security Knowledge Area (Chapter 15)). To prevent the exploitation or to mitigate the effects of software vulnerabilities, a large variety of hardware modifications/additions to the processor architecture have been proposed in literature and have been included in commercial processors. We call this abstraction layer the hardware/software boundary: hardware forms the trust boundary, while software is no longer trusted. These security additions to the hardware typically have a cost in extra area and loss in performance.

The most important security objectives at this design abstraction level are to support protection, isolation and attestation for the software running on a processor platform [1858], [1859], [1860].

- Protection: "A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each others' data. These mechanisms also isolate the operating system from the user process" [1858]. In a traditional computer architecture, usually the OS kernel is part of the Trusted Computing Base (TCB), but the rest of the software is not.
- With isolation, a hardware mechanism is added that controls access to pieces of software and associated data. Isolation separates two parties: a software module might need protection from the surrounding software is one case. So, a Protected Model Architecture (PMA) provides a hardware guarantee that a piece of software runs unhindered from unwanted outside influences. The opposite case, if we want to limit the effects of possibly tainted software to its environment, it will be sandboxed or be put into a 'compartment.' Protected Module Architectures are a hardware only solution: the OS is not part of the TCB. More details are described in section 20.4.4
- With attestation, there is hardware support to demonstrate to a third party that the system, e.g. the code installed and/or running on a processor, is in a particular state. Attestation can be local or remote. Local attestation means that one software module can attest its state to another one on the same compute platform. Remote attestation means that a third party, outside the compute platform can get some guarantee about the state of a processor.

In the context of general purpose computing, Virtual Machines (VMs) and Hypervisors have been introduced to support multiple operating systems on one physical processor. This sharing of resources improves efficiency and reuse. It can however only be realized by a secure and efficient sharing of physical memory: virtual machines should only be allowed to use the portions of physical memory assigned to it. The organization and details of virtual memory are out of scope of hardware security and part of the Operating Systems & Virtualisation Knowledge Area (Chapter 11). The hardware supports protection by providing privileged instructions, control and status registers and sometimes support for multiple parallel threads.

In the context of embedded micro-controllers, with no operating system, and only one applica-

tion, the hardware support could be limited to only machine level support. Memory protection could be added as an optional hardware module to the processor.

Other more advanced security objectives to support software security might include:

- *Sealed storage* is the process of wrapping code and/or data with certain configuration, process or status values. Only under the correct configuration (e.g. program counter value, nonce, secret key, etc.) can the data be unsealed. *Dynamic root of trust* in combination with a *late launch* guarantees that even if the processor starts from an unknown state, it can enter a fixed known piece of code and known state. This typically requires special instructions to enter and exit the protected partition.
- *Memory protection* refers to the protection of data when it travels between the processor unit and the on-chip or off-chip memory. It protects against bus snooping or side-channel attacks or more active fault injection attacks.
- *Control flow integrity* is a security mechanism to prevent malware attacks from redirecting the flow of execution of a program. In hardware, the control flow of the program is compared on-the-fly at runtime with the expected control flow of the program.
- *Information flow analysis* is a security mechanism to follow the flow of sensitive data while it travels through the different components, from memory to cache over multiple busses into register files and processing units and back. This is important in the context of micro-architectural and physical side-channel attacks.

In the next subsections a representative set of hardware approaches to address the above software security challenges are presented. Some hardware techniques address multiple security objectives. Some are large complex approaches, others are simple dedicated hardware features.

As a side note: a large body of knowledge on software-only approaches is available in literature. Mostly, they offer a weaker level of security as they are not rooted in a hardware root of trust. E.g. for control flow integrity, software-only approaches might instruct the software code to check branches or jumps, while hardware support might calculate MACs on the fly and compare these to stored associated MACs.

20.4.1 Trusted Execution Environment (TEE)

TEE was originally an initiative of Global Platform, a consortium of companies, to standardize a part of the processor as a trusted secure part. TEE has since evolved and covers in general the hardware modifications made to processors to provide isolation and attestation to software applications. There is a large body of knowledge both from the industrial side as well as from the academic side.

TEE is a concept that provides a secure area of the main processor “to provide end-to-end security by protecting the execution of authenticated code, confidentiality, authenticity, privacy, system integrity and data access rights” [1861]. It is important that the TEE is isolated from the so-called Rich Execution Environment (REE), which includes the untrusted OS. The reasoning behind this split is that it is impossible to guarantee secure execution and to avoid malware in the normal world due to the complexity of the OS and all other applications running there. The rich resources are accessible from the TEE, while the opposite is not possible. Global Platform does not specify the specifics on how these security properties should be implemented. Three main hardware options are suggested. Option 1 assumes that every processor component on

the IC can be split into a trusted and a rich part, i.e. the processor core, the crypto accelerators, the volatile and non-volatile memory are all split. Option 2 assumes that there is a separate secure co-processor area on the SoC with a well-defined hardware interface to the rest of the SoC. Option 3 assumes a dedicated off-chip secure co-processor, much like a secure element.

Global Platform defines also a Common Criteria based protection profile (see section 20.2.2) for the TEE. It assumes that the package of the integrated circuit is a black box [1861] and thus secure storage is assumed by the fact that the secure asset remains inside the SoC. It follows the procedures of common criteria assurance package EAL2 with some extra features. It pays extra attention to the evaluation of the random number generator and the concept of monotonic increasing time.

20.4.2 IBM 4758 Secure coprocessor

An early example, even before the appearance of the TEE of Global Platform is the IBM 4758 secure processor. Physical hardware security was essential for this processor: it contained a board with a general purpose processor, DRAM, separate battery backed-DRAM, Flash ROM, crypto accelerator (for DES), a random number generator and more. All of these components were enclosed in a box with tamper resistant and tamper evidence measures. It was certified to FIPS 140-1, level 4 at that time [1862].

20.4.3 ARM Trustzone

ARM Trustzone is one well known instantiation of a TEE. It is part of a system of ARM processors integrated into System on a Chips (SoCs) mostly used for smartphones. The TEE is the secure part of the processor and it runs a smaller trusted OS. It is isolated from the non-secure world, called the Rich Execution Environment, which runs the untrusted rich OS. The main hardware feature to support this split is the Non-Secure (NS) bit. The AXI bus transactions are enhanced with a NS bit so that it can block the access of secure world resources by non-secure resources. Each AXI transaction comes with this bit set or reset. When the processor runs in the secure mode, then the transaction comes with the NS bit set to zero, which gives it access to both secure and non-secure resources. When the processor runs in normal mode, it can only access resources from the normal world. This concept is extended to the level 1 and level 2 cache. These caches store an extra information bit to indicate if the code can be accessed by a secure or non-secure master. Special procedures are foreseen to jump from secure to non-secure and vice-versa. This is supported by a special monitor mode which exists in the secure world.

The split applied by ARM Trustzone is however a binary split. Applications from different vendors could co-exist together in the secure world and so if one trusted component violates the system's security, the security can no longer be guaranteed. To address this issue, protected module architectures are introduced.

Trusted Execution Environments are also being created in open-source context, more specifically in the context of the RISC-V architecture.

20.4.4 Protected Module Architectures and HWSW co-design solutions

If multiple software applications want to run on the same platform isolated from each other, then hardware needs to isolate them from each other at a more fine granularity. This can be done by so-called protected module architectures. The basic idea is that small software modules can run protected from all other software running on the processor. And because they are small, their properties and behavior can be verified more thoroughly. The protection is provided by extra features added to the hardware in combination with an extremely small trusted software base if needed. In the Flicker project, the software TCB relies on only 250 lines of codes but requires a dedicated TPM chip [1863]. Table 12 of the review work of [1862], provides an in-depth comparison of several general purpose secure processor projects with their hardware and software TCB. The hardware TCB distinguishes between the complete mother board as TCB, e.g. for TPM usage, to CPU package only for SGX and other projects. The software TCB varies from a complete secure world as is the case for TrustZone to privileged containers in the case of SGX or a trusted hypervisor, OS or security monitor.

Even more advanced are solutions with a zero trusted software base: only the hardware is trusted. This is the case for the Sancus project [1864]. It implements a program counter based memory access control system. Extra hardware is provided to compare the current program counter with stored boundaries of the protected module. Access to data is only possible if the program counter is in the correct range of the code section. Progress of the program in the code section is also controlled by the hardware so that correct entry, progress and exit of the module can be guaranteed.

Intel's Software Guard Extension (SGX) are also a protection mechanism at small granularity. Software modules of an application are placed in memory enclaves. Enclaves are defined in the address space of a process, but access to enclaves is restricted. Enclaves are created, initialized, and cleared by possibly untrusted system software, but operating in the enclave can only be done by the application software. Minimizing the extra hardware to support SGX, and especially avoiding performance degradation is an important goal. The details of the hardware micro-architecture have not been disclosed: yet its most important parts are a memory encryption unit, a series of hardware enforced memory access checks and secure memory range registers [1862].

20.4.5 Light-weight and individual solutions

The above listed solutions are mostly suited for general purpose computing, i.e. for platforms on which a complex software stack will run. In literature, more solutions are proposed to provide extremely light weight solutions to support specific security requests. SMART is one early example: it includes a small immutable piece of bootROM, considered the root of trust, to support remote attestation [1865].

To protect against specific software attacks, more individual hardware countermeasures have been introduced. One example is a hardware shadow stack: to avoid buffer overflow attacks and to protect control flow integrity, return addresses are put on both the stack and the shadow stack. When a function loads a return address, the hardware will compare the return address of the stack to that of the shadow stack. They should agree for a correct return.

Another example is the protection of jump and return addresses to avoid buffer overflow attacks and other abuses of pointers. A simple but restrictive option is to use read-only memory, which fixes the pointer. A novel recent technique is the use of pointer authentication.

The authentication code relies on cryptographic primitives. A challenge for these algorithms is that they should create the authentication tag with very low latency to fit into the critical path of a microprocessor. The ARMV8-A architectures uses therefore a dedicated low-latency crypto algorithm Qarma [1866]. In this approach the unused bits in a 64-bit pointer are used to store a tag. This tag is calculated based on a key and on the program state, i.e. current address and function. These tags are calculated and verified on the fly.

Address Space Layout Randomization or Stack canaries area general software technique: its aim is to make it hard to predict the destination address of the jump. A detailed description can be found in the Software Security Knowledge Area (Chapter 15).

20.5 HARDWARE DESIGN FOR CRYPTOGRAPHIC ALGORITHMS AT RTL LEVEL

The hardware features discussed so far are added to general purpose compute platforms, i.e. to a programmable micro-processor or micro-controller. General purpose means that a platform is created of which the hardware designer does not know the future applications that will run on it. Flexibility, reflected in the instruction set, is then of importance. A second class of processors are domain-specific processors: they have limited or no programmability and designed for one or a small class of applications.

20.5.1 Design process from RTL to ASIC or FPGA

When a dedicated processor is built for one or a class of cryptographic algorithms, this gives a lot of freedom to the hardware designer. Typically, the hardware designer will, starting from the cryptographic algorithm description, come up with hardware architectures at the Register Transfer Level (RTL) taking into account a set of constraints. Area is measured by gate count at RTL level. Throughput is measured by bits/sec. Power consumption is important for cooling purposes and measured in Watt. Energy, measured in Joule, is important for battery operated devices. It is often expressed in the amount of operations or amount of bits that can be processed per unit energy. Hence the design goal is to maximize the operations/Joule or bits/Joule. The resistance to side channel attacks is measured by the number of measurements or samples required to disclose the key or other sensitive material. Flexibility and programmability are difficult to measure and are typically imposed by the application or class of applications that need to be supported: will the hardware support only one or a few algorithms, encryption and/or decryption, modes of operation, initialization, requirements for key storage, and so on.

A hardware architecture is typically described in a Hardware Description Language such as Verilog of VHDL. Starting from this description the two most important hardware platforms available to a hardware designer are ASIC and FPGA. An Application Specific Integrated Circuit (ASIC) is a dedicated circuit fabricated in silicon. Once fabricated (baked) it cannot be modified anymore. A Field Programmable Gate Array (FPGA) is a special type of programmable device: it consists of regular arrays of 1-bit cells, that can programmed by means of a bitstream. This special bitstream programs each cell to a specific function, e.g. a one bit addition, a register, a multiplexer, and so on. By changing the bit-stream the functionality of the FPGA changes. From the viewpoint of the Register Transfer Level (RTL) the actual design process for either FPGA or ASIC doesn't differ that much. Similar design options are available: the designer can

decide to go for serial or parallel architectures, making use of multiple design tricks to match the design with the requirements. The most well-known tricks are to use pipelining to increase throughput, or unrolling to reduce latency, time multiplexing to reduce area, etc.

From implementation viewpoint, at this register transfer abstraction level, a large body of knowledge and a large set of Electronic Design Automation (EDA) tools exist to map an application onto a FPGA or ASIC platform [1849]. Implementation results should be compared not only on the number of operations, but also on memory requirements (program memory and data memory), throughput and latency requirements, energy and power requirements, bandwidth requirements and the ease with which side-channel and fault attack countermeasures can be added. Please note that this large body of knowledge exists for implementations that focus on efficiency. However, when combining efficiency with security requirements, such as constant time execution or other countermeasures, there is a huge lack of supporting EDA tools (see section 20.8).

20.5.2 Cryptographic algorithms at RTL level

Cryptographic implementations are subdivided in several categories, enumerated below. The details of the cryptographic algorithms themselves are discussed in the Cryptography Knowledge Area (Chapter 10). Here only remarks related to the RTL implementation are made. In this section only notes specific to the hardware implementations are made.

- Secret key algorithms: both block ciphers and stream ciphers result usually in compact and fast implementations. Feistel ciphers are chosen for very area constrained designs as the encryption and decryption hardware is the same. This is e.g. not the case for the AES algorithm for which encryption and decryption require different units.
- Secret key: light-weight algorithms. For embedded devices, over the years, many light-weight algorithms have been developed and implemented, e.g. Present, Prince, Rectangle, Simon or Speck cipher. Focus in these cases is mostly on area cost. However, lately light-weight has been extended to include also low power, low energy and especially low-latency. Latency is defined as the time difference between input clear text and corresponding encrypted output or MAC. Having a short latency is important in real-time control systems, automotive, industrial IoT but also in memory encryption, control flow integrity applications etc. More knowledge will follow from the recent NIST call on light-weight crypto [1867].
- Secret key: block ciphers by themselves are not directly applicable in security application. They need to be combined with modes of operation to provide confidentiality or integrity, etc. (see the Cryptography Knowledge Area (Chapter 10)). In this context efficient implementations of authenticated encryption schemes are required: this is the topic of the CAESAR competition [1868]. From an implementation viewpoint, the sequential nature of the authenticated encryption schemes makes it very difficult to obtain high throughputs as pipelining cannot directly be applied.
- Hash algorithms require typically a much larger area compared to secret key algorithms. Especially the SHA3 algorithm and its different versions are large in area and slow in execution. Therefore, light-weight hash algorithms are a topic of active research.
- One important hardware application of hash functions is the mining of cryptocurrencies, such as Bitcoin, Etherium, Litecoin and others, based on SHA2, SHA256, SHA3, etc. To obtain the required high throughputs, massive parallelism and pipelining is applied. This

is however limited as hash algorithms are recursive algorithms and thus there is an upper bound on the amount of pipelining that can be applied [1869]. Cryptocurrencies form part of the more general technology of distributed ledgers, which is discussed in the Distributed Systems Security Knowledge Area (Chapter 12).

- The computational complexity of public key algorithms is typically 2 or 3 orders of magnitude higher than secret key and thus its implementation 2 to 3 orders slower or larger. Especially for RSA and Elliptic curve implementations, a large body of knowledge is available, ranging from compact [1870] to fast, for classic and newer curves [1871].
- Algorithms resistant to attacks of quantum computers, aka post-quantum secure algorithms, are the next generation algorithms requiring implementation in existing CMOS ASIC and FPGA technology. Computational bottle-necks are the large multiplier structures, with/without the Number Theoretic Transform, the large memory requirements and the requirements on random numbers that follow specific distributions. Currently, NIST is holding a competition on post-quantum cryptography [1872]. Thus it is expected that after the algorithms are decided, implementations in hardware will follow.
- Currently, the most demanding implementations for cryptographic algorithms are those used in homomorphic encryption schemes: the computational complexity, the size of the multipliers and especially the large memory requirements are the challenges to address [1873].

20.6 SIDE-CHANNEL ATTACKS, FAULT ATTACKS AND COUNTERMEASURES

This section first provides an overview of physical attacks on implementations of cryptographic algorithms. The second part discusses a wide range of countermeasures and some open research problems. Physical attacks, mostly side-channel and fault attacks, were originally of great concern to the developers of small devices that are in the hands of attackers, especially smart-cards and pay-TV systems. The importance of these attacks and countermeasures is growing as more electronic devices are easily accessible in the context of the IoT.

20.6.1 Attacks

At the current state of knowledge, cryptographic algorithms have become very secure against mathematical and cryptanalytical attacks: this is certainly the case for algorithms that are standardized or that have received an extensive review in the open research literature. Currently, the weak link is mostly the implementation of algorithms in hardware and software. Information leaks from the hardware implementation through side-channel and fault attacks. A distinction is made between passive or side-channel attacks versus active or fault attacks. A second distinction can be made based on the distance of the attacker to the device: attacks can occur remotely, close to the device still non-invasive to actual invasive attacks. More details on several classes of attacks are below.

Passive Side Channel Attacks General side-channel attacks are passive observations of a compute platform. Through data dependent variations of execution time, power consumption or electro-magnetic radiation of the device, the attacker can deduce information of secret internals. Variations of execution time, power consumption or electro-magnetic radiations

are typically picked up in close proximity of the device, while it is operated under normal conditions. It is important to note that the normal operation of the device is not disturbed. Thus the device is not aware that it is being attacked, which makes this attack quite powerful [980].

Side channel attacks based on variations on power consumption have been extensively studied. They are performed close to the device with access to the power supply or the power pins. One makes a distinction between Simple Power Analysis (SPA), Differential and Higher Order Power Analysis (DPA), and template attacks. In SPA, the idea is to first study the target for features that depend on the key. E.g. a typical target in timing and power attacks are if-then-else branches that are dependent on key bits. In public key algorithm implementations, such as RSA or ECC, the algorithm runs sequentially through all key bits. When the if-branch takes more or less computation time than the else-branch this can be observed from outside the chip. SPA attacks are not limited to public key algorithms, they have also been applied to secret key algorithms, or algorithms to generate prime numbers (in case they need to remain secret). So with knowledge of the internal operation of the device, SPA only requires to collect one or a few traces for analysis.

With DPA, the attacker collects multiple traces, ranging from a few tens for unprotected implementations to millions in case of protected hardware implementations. In this situation, the attacker exploits the fact that the instantaneous power consumption depends on the data that is processed. The same operation, depending on the same unknown sub-key, will result in different power consumption profiles if the data is different. The attacker will also built a statistical model of the device to estimate the power consumption as a function of the data and the different values of the subkey. Statistical analysis on these traces based on correlation analysis, mutual information and other statistical tests are applied to correlate the measured values to the statistical model.

Side channel attacks based on Electro-Magnetic radiations have been recognized early-on in the context of military communication and radio equipment. As a reaction, NATO and the governments of many countries have issued TEMPEST [1874]. It consists of specifications on the protection of equipment against unintentional electro-magnetic radiation but also against leakage of information through vibrations or sound. Electro-Magnetic radiation attacks can be mounted from a distance, as explained above, but also at close proximity to the integrated circuit. Electro-Magnetic probing on top of an integrated circuit can release very localized information of specific parts of an IC by using a 2D stepper and fine electro-magnetic probers. Thus electro-magnetic evaluation has the possibility to provide more fine grained leakage information compared to power measurements.

Timing attacks are another subclass of side-channel attacks [1453]. When the execution time of a cryptographic calculation or a program handling sensitive data, varies as a function of the sensitive data, then this time difference can be picked up by the attacker. A timing attack can be as simple as a key dependent different execution time of an if-branch versus an else-branch in a finite state machine. Cache attacks, which abuse the time difference between a cache hit and a cache miss are an important class of timing attacks [1875], [1876], .

With a template attack, the attacker will first create a copy or template of the target device [1877]. This template is used to study the behavior of the device for all or a large set of inputs and secret data values. One or a few samples of the target device are then compared to the templates in the database to deduce secret information from the device. Template attacks are typically used when the original device has countermeasures against multiple executions. E.g. it might have an internal counter to log the number of failed attempts. Templates can be

made based on timing, power or electro-magnetic information. As machine learning and AI techniques become more powerful, so will the attack possibility with template attacks.

Micro-architectural Side-channels Processor architectures are very vulnerable to timing attacks. The problem of information leaks and the difficulty of confinement between programs was already identified early on in [1878]. Later timing variations in cache hits and misses became an important class of timing attacks [1879]. Recently gaining a lot of attention are the micro-architectural side-channel attacks, such as Spectre, Meltdown, Foreshadow. They are also based on the observation of timing differences [1851][1879]. The strength of the attacks sits in the fact that they can be mounted remotely from software. Modern processors include multiple optimization techniques to boost performance not only with caches, but also speculative execution, out-of-order execution, branch predictors, etc. When multiple processes run on the same hardware platform, virtualization and other software techniques isolates the data of the different parties in separate memory locations. Yet, through the out-of-order execution or speculative execution (or many other variants) the hardware of the processor will access memory locations not intended for the process by means of so-called transient instructions. These instructions are executed but never committed. They have however touched memory locations, which might create side channel effects, such as variations in access time, and thus leak information.

Active fault attacks Fault attacks are active manipulations of hardware compute platforms [1880]. The result is that the computation itself or the program control flow is disturbed. Faulty or no outputs are released. Even if no output is released or the device resets itself, this decision might leak sensitive information. One famous example is published in [1881]: it describes an RSA signature implementation which makes use of the Chinese Remainder Theorem (CRT). With one faulty and one correct result signature, and some simple mathematical calculations, the secret signing key can be derived. Physical fault-attacks could be a simple clock glitching, power glitching, heating up or cooling down a device. These require close proximity to the device but are non-invasive.

With scaling of memories, more attack surfaces appear. A very specific attack on DRAM memories, is the RowHammer attack [989, 1882]. By repeating reading specific locations in DRAM memory, neighboring locations will loose their values. Thus by hammering certain locations, bit flips will occur in nearby locations.

With more expensive equipment, and with opening the lid of the integrated circuit or etching the silicon down, even more detailed information of the circuit can be obtained. Equipment that has been used include optical fault [1883], laser attacks [1884], Focused Ion Beam (FIB), a Scanning Electron Microscope (SEM) and other. The latter are typically equipment that has been designed for chip reliability and failure analysis. This equipment can also be used or misused for reverse engineering.

20.6.2 Countermeasures

There are no generic countermeasures that resist all classes of side-channel attacks. Depending on the threat model (remote/local access, passive/active, etc.) and the assumptions made on the trusted computing base (i.e. what is and what is not included in the root of trust), countermeasures have been proposed at several levels of abstraction. The most important categories are summarized below.

To resist timing attacks, the first objective is to provide hardware that executes the application or program in constant time independent of secret inputs, keys and internal state. Depending on the time granularity of the measurement equipment of the attacker, constant time countermeasures also need to be more fine grained. At the processor architecture level, constant time means a constant number of instructions. At the RTL level, constant time means a constant number of clock cycles. At logic and circuit level, constant time means a constant logic depth or critical path independent of the input data. At instruction level, constant time can be obtained by balancing execution paths and adding dummy instructions. Sharing of resources, e.g. through caches, make constant time implementations extremely difficult to obtain.

At RTL level, we need to make sure that all instructions run in the same number of clock cycles. dummy operations or dummy gates, depending on the granularity level. Providing constant time RTL level and gate level descriptions is however a challenge as design tools, both hardware and software compilers, will for performance reasons synthesize away the dummy operations or logic which were added to balance the computations.

As many side-channel attacks rely on a large number of observations or samples, randomisation is a popular countermeasure. It is used to protect against power, electro-magnetic and timing side-channel attacks. Randomisation is a technique that can be applied at algorithm level: it is especially popular for public key algorithms, which apply techniques such as scalar blinding, or message blinding [1885]. Randomisation applied at register transfer and gate level is called masking. Masking schemes randomise intermediate values in the calculations so that their power consumption can no longer be linked with the internal secrets. A large set of papers on gate level masking schemes is available, ranging from simple Boolean masking to threshold implementations that are provable secure under certain leakage models [1886]. Randomisation has been effective in practice especially as a public key implementation protection measure. The protection of secret key algorithms by masking is more challenging. Some masking schemes require a huge amount of random numbers, others assume leakage models that do not always correspond to reality. In this context, novel cryptographic techniques summarized under the label leakage resilient cryptography, are developed that are inherently resistant against side-channel attacks [1887, 1888]. At this stage, there is still a gap between theory and practice.

Hiding is another major class of countermeasures. The idea is to reduce the signal to noise ratio by reducing the signal strength. Shielding in the context of TEMPEST is one such example. Similarly, at gate level, reducing the power signature or electro-magnetic signature of standard cells or logic modules, will increase the resistance against power or electro-magnetic attacks. Simple techniques such as using a jittery or drifting clock, and large decoupling capacitances will also reduce the signal to noise ratio.

Sometimes solutions for leaking at one abstraction level, e.g. power side channels, can be addressed at a different abstraction level. Therefore, if there is a risk that an encryption key leaks from an embedded device, a cryptographic protocol that changes the key at a sufficiently

high frequency, will also avoid side-channel information leakage.

General purpose processors such as CPUs, GPUs, and micro-controllers can not be modified once fabricated. Thus protecting against micro-architectural attacks after fabrication by means of software patches and updates is extremely difficult and mostly at the cost of reduced performance [1851]. Micro-code updates are also a form of software, i.e. firmware update and not a hardware update. The main difference is that the translation from instructions to micro-code is a company secret, and thus for the user it looks like a hardware update. Providing generic solutions to programmable hardware is a challenge as it is unknown beforehand which application will run. Solutions to this problem will be a combined effort between hardware and software techniques.

Protection against fault attacks are made at the register transfer level, as well as at the circuit level. At RTL, protection against fault attacks is mostly based on redundancy either in space or in time and by adding checks based on coding, such as parity checks. The price is expensive as calculations are performed multiple times. One problem with adding redundancy is that it increases the attack surface of side-channels. Indeed, due to the redundant calculations, the attacker has more traces available to perform time, power or electro-magnetic side-channel attacks [1885]. At circuit level, monitors on the clock or power supply, might detect deviations from normal operations and raise an alarm.

Many type of circuit level sensors are added to integrated circuits. Examples are light sensors that detect that a lid of a package has been opened. Mesh metal sensors which are laid-out in top level metal layers can detect probing attacks. Temperature sensors detect heating or cooling of the integrated circuit. Antenna sensors to detect electro-magnetic probes close to the surface have been developed: these sensors measure a change in electro-magnetic fields. And sensors that detect manipulation of the power supply or clock can be added to the device. Note that adding sensors to detect active manipulation can again leak extra information to the side channel attacker.

Joint countermeasures against side-channel and fault attacks are challenging and an active area of research.

20.7 ENTROPY GENERATING BUILDING BLOCKS: RANDOM NUMBERS, PHYSICALLY UNCLONABLE FUNCTIONS

Sources of entropy are essential for security and privacy protocols. In this section two important sources of entropy related to silicon technology are discussed: random number generators and physically unclonable functions.

20.7.1 Random number generation

Security and privacy rely on strong cryptographic algorithms and protocols. A source of entropy is essential in these protocols: random numbers are used to generate session keys, nonces, initialization vectors, to introduce freshness, etc. Random numbers are also used to create masks in masking countermeasures, random shares in multi party computation, zero-knowledge proofs, etc. In this section the focus is on cryptographically secure random numbers as used in security applications. Random numbers are also used outside cryptography, e.g. in gaming, lottery applications, stochastic simulations, etc.

In general, random numbers are subdivided in two major classes: the Pseudo Random Number Generator (PRNG) also called Deterministic Random Bit Generator (DRBG) and the True Random Number Generator (TRNG) or Non-Deterministic Random Bit Generator (NRBG). The design, properties and testing of random numbers is described in detail by important standards, issued in the US by NIST. NIST has issued the NIST800-90A for deterministic random number generators, the NIST800-90B for entropy sources, and NIST800-90C for random bit generation constructions [1672], [1889] [1890] [1]. In Germany and by extension in most of Europe, the German BSI has issued two important standards: the AIS-20 for functionality classes and evaluation criteria for deterministic random number generators and the AIS-31 for physical random number generators [1891, 1892, 1893].

An ideal RNG should generate all numbers with equal probability. Secondly, these numbers should be independent from previous or next numbers generated by the RNG, called forward and backward secrecy. The probabilities are verified with statistical tests. Each standard includes a large set of statistical tests aimed at finding statistical weaknesses. Not being able to predict future values or derive previous values is important not only in many security applications, e.g. when this is used for key generation, but also in many gaming and lottery applications.

Pseudo-random number generators are deterministic algorithms that generate a sequence of bits or numbers that look random but are generated by a deterministic process. Since a PRNG is a deterministic process, when it starts with the same initial value, then the same sequence of numbers will be generated. Therefore it is essential that PRNG starts with a different start-up value each time the PRNG is initiated. This initial seed can either be generated by a slow true random number generated or at minimum by a non-repeating value, e.g. as provided by a monotonic increasing counter. A PRNG is called cryptographically secure if the attacker, who learns part of the sequence, is not able to compute any previous or future outputs. Cryptographically secure PRNGs rely on cryptographic algorithms to guarantee this forward and backward secrecy. Forward secrecy requires on top a regular reseeding to introduce new freshness into the generator. Hybrid RNG have an additional non-deterministic input to the PRNG.

PRNGs provide conditional security based on the computational complexity of the underlying cryptographic algorithms. See the Cryptography Knowledge Area (Chapter 10) for more details. In contrast, ideal true random number generators provide unconditional security as they are based on unpredictable physical phenomena. Thus their security is guaranteed independent of progress in mathematics and cryptanalysis.

The core of a true random number generator consists of an entropy source, which is a physical phenomena with a random behavior. In electronic circuits, noise or entropy sources are usually based on thermal noise, jitter and metastability. These noise sources are never perfect:

[1] NIST800-90C does not exist as a standard yet.

the bits they generate might show bias or correlation or other variations. Hence they don't have full entropy. Therefore, they are typically followed by entropy extractors or conditioners. These building blocks improve the entropy per bit of output. But as the entropy extractor are deterministic processes, they cannot increase the total entropy. So the output length will be shorter than the input length.

Due to environmental conditions, e.g. due to temperature or voltage variations, the quality of the generated numbers might vary over time. Therefore, the standards describe specific tests that should be applied at the start and continuously during the process of generating numbers. One can distinguish three main categories of tests. The first one is the total failure test, applied at the source of entropy. The second ones are online health tests to monitor the quality of the entropy extractors. The third ones are tests for the post-processed bits. The requirements for these tests are well described in the different standards and specialized text books [1894].

The challenge in designing TRNGs is first to provide a clear and convincing proof of the entropy source, second the design of online tests which at the same are compact and can detect a wide range of defects [1895]. The topic of attacks, countermeasures and sensors for TRNGs, especially in the context of IoT and embedded devices, is an active research topic.

20.7.2 Physically Unclonable Functions

From a hardware perspective, Physically Unclonable Functions (PUFs), are circuits and techniques to derive unique features from silicon circuits, similar to human biometrics [1896]. The manufacturing of silicon circuits results in unique process variations which cannot be physically cloned. The basic idea of PUFs is that these unique manufacturing features are magnified and digitized so that they can be used in security applications similar to the use of fingerprints or other biometrics. Process and physical variations such as doping fluctuations, line or edge widths of interconnect wires, result in variations of threshold voltages, transistor dimensions, capacitances, etc. Thus circuits are created that are sensitive to and amplify these variations.

The major security application for PUFs is to derive unique device specific keys, e.g. for usage in an IoT device or smart card. Traditionally, this storage of device unique keys is done in non-volatile memory, as the key has to remain in the chip even when the power is turned-off. Non-volatile memory requires however extra fabrication steps, which makes chips with non-volatile memory more expense than regular standard CMOS chips. Thus PUFs are promised as cheap alternative for secure non-volatile memory, because the unique silicon fingerprint is available without the extra processing steps. Indeed, each time the key is needed, it can be read from the post-processed PUF and directly used in security protocols. They can also replace fuses, which are large and their state is relatively easy to detect under a microscope.

The second security application is to use PUFs in identification applications, e.g. for access control or tracking of goods. The input to a PUF is called a challenge, the output the response. The ideal PUF has an exponential number of unique challenge response pairs, exponential in the number of circuit elements. The uniqueness of PUFs is measured by the inter-distance between different PUFs seeing the same challenge. The ideal PUF has stable responses: it replies with the same response, i.e. there is no noise in the responses. Moreover, PUF responses should be unpredictable and physically unclonable.

The ideal PUF unfortunately does not exist. In literature, two main classes of PUFs are defined,

characterized by the number of challenge-response pairs they can generate. So-called weak PUFs are circuits with a finite number of elements, with each element providing a high amount of entropy. The number of possible challenge-response pairs grows typically linear with the area of the integrated circuit. Hence they are called weak PUFs. The most well known example is the SRAM PUF [1897]. These PUFs are typically used for key generation. The raw PUF output material is not directly usable for key generation as the PUF responses are affected by noise. Indeed, subsequent readings of the same PUF might result in slightly varying noisy responses, typically up to 20%. Thus after the entropy extraction follows secure sketch (similar to error correction) circuits to eliminate the noise and compress the entropy to generate a full entropy key [1898]. The challenge for the PUF designer is to come up with process variations and circuits that can be used as key material, but which are not sensitive to transient noise. A second challenge is to keep all the post-processing modules compact so that the key-generation PUF can be included in embedded IoT devices.

The second class are the so-called strong PUFs. In this case, the number of challenge-response pairs grows large, ideally exponential, with the silicon area. The most well-known example is the arbiter PUF [1899]. A small number of silicon elements are combined together, e.g. to create a chain of multiplexers or comparators, so that simple combinations of the elements create the large challenge-response space. Also in this case, the effects of noise in the circuits needs to be taken into account. Strong PUFs are promised to be useful in authentication applications, e.g. for access control. Each time a challenge is applied to the PUF, a response unique to the chip will be sent. The verifier will accept the response if it can be uniquely tied to the prover. This requires that the PUF responses are registered in a form of a database beforehand during an enrollment phase.

The problem with strong PUFs is that there is a strong correlation between different challenge-response pairs of most circuits proposed in literature. Hence all of these circuits are broken with machine learning techniques [1900] and can not be used for authentication purposes. The fundamental problem is that very basic, mostly linear operations are used to combine PUF elements, which makes them easy targets for machine learning attacks. Ideally, these should be cryptographic or other computationally hard operations resistant to machine learning: unfortunately these cannot tolerate noise. Light-weight PUF based security protocols are an active area of research.

20.8 HARDWARE DESIGN PROCESS

In this section, several hardware security topics are described which are directly related to the lower design abstraction layers. One is the trust in the hardware design process itself. Directly related to this, is the problem of Trojan circuits. Also part of the hardware design process are circuit level techniques for camouflaging, logic locking, etc.

20.8.1 Design and fabrication of silicon integrated circuits

It is important to note that the hardware design process itself also needs to be trusted. Because of its design complexity, design at each abstraction layer relies on Electronic Design Automation (EDA) tools. The design, fabrication, packaging and test of silicon integrated circuits is an international engagement: silicon foundries are mostly located in Asia. Silicon design tools are most developed in the US, and silicon testing and packaging usually occur all over the world. For chips that end-up in critical infrastructure, such as telecommunication, military, aviation, trust and verification of the complete design cycle is essential.

Since silicon foundries and mask making are extremely expensive, very few countries and companies can still afford it and a huge consolidation has and is taking place in the industry. For critical infrastructure, governments demand more tools and techniques to increase the trustworthiness of this international design process. On this topic, large research projects are defined to come up with methods and tools to increase the trustworthiness of the design process and especially to assess the risk of Trojan insertions during the design process.

20.8.2 Trojan circuits

Trojan circuits are logic or gates added to large integrated circuits. As they are not part of the specified functionality, they are difficult to detect. They rely on the fact that they are extremely small in comparison with the large size of integrated circuits and SoCs. Trojan circuits are classified according to three main criteria [1901, 1902]. The first one is the physical characteristics of the Trojan, i.e. how is the Trojan inserted into the circuit. E.g. does it requires logic modifications or only layout modifications. The second one is the activation characteristic: will the Trojan be turned on by an internal or external event, etc. The third characteristic classifies the type of action taken by the Trojan, e.g. will it leak information or will it destroy functionality, etc. The knowledge area on this topic is summarized in [1901, 1902].

20.8.3 Circuit level techniques

To avoid visual inspection, circuit level *camouflaging* techniques are introduced [1903]. These are standard cells or other modules that visually look the same, or they look camouflaged by random extra material. This is done to avoid visual inspection and reverse engineering based on visual inspection.

Another techniques to avoid loss of intellectual property is *logic locking* [1904]. With this technique, extra gates are added to a circuit with a secret input. Only when the correct key is applied to the secret gates, will the circuit perform the correct functionality. This is an active research topic with logic locking schemes being proposed and attacked, with SAT solvers being a very useful tool in attacking the circuits.

20.8.4 Board Level Security

Integrated circuits are placed together on Printer Circuit Boards (PCBs). Many of the attacks and countermeasures mentioned before for integrated circuits, can be repeated for PCBs albeit at a different scale. While integrated circuits provide some level of protection because they are encapsulated in packages and use much smaller CMOS technologies, PCB's are less complex and somewhat easier to access. Therefore, for PCB's special coatings, and mechanical tamper evident and tamper resistant protection mechanisms could be provided. There have been some concerns that Trojan circuits could also be included at the board level.

20.8.5 Time

The concept of time and the concept of sequence of events are essential in security protocols. The TCG identifies three types of sequencing: a monotonic counter, a tick counter and actual trusted time [1850]. A monotonic counter always increases, but the wall clock time between two increments is unknown. The tick counter increases with a set frequency. It only increases when the power is on. At power-off the tick counter will reset. Therefore the tick counter is linked with a nonce and methods are foreseen to link this with a real wall clock time. Trusted time is the most secure. It makes sure that there is a link between the tick counter and the real wall clock time. From a hardware viewpoint it will require non-volatile memory, counters, crystals, continuous power, and an on chip clock generator. The connection to a real wall clock will require synchronization and an actual communication channel.

The importance of time is placed in a wider context in the Distributed Systems Security Knowledge Area (Chapter 12).

20.9 CONCLUSION

Hardware security is a very broad topic, covering many different topics. In this chapter, a classification is made based on the different design abstraction layers. At each abstraction layer, the threat model, root of trust and security goals are identified.

Because of the growth of IoT, edge and cloud computing, the importance of hardware security is growing. Yet, in many cases hardware security is in conflict with other performance optimisations, such as low power or limited battery operated conditions. In these circumstances, performance optimization is *the* most important design task. Yet it is also the most important cause of information leakage. This is the case at all abstraction layers: instruction level, architecture level and logic and circuit level.

Another trend is that hardware is becoming more 'soft'. This is an important trend in processor architecture, where FPGA functionality is added to processor architectures. The fundamental assumption that hardware is immutable is lost here. This will create a whole new class of attacks.

A last big challenge for hardware security is the lack of EDA tools to support hardware security. EDA tools are made for performance optimization and security is usually an afterthought. An added challenge is that it is difficult to measure security and thus difficult to balance security versus area, throughput or power optimisations.

Chapter 21
Cyber-Physical Systems Security

Alvaro Cardenas | University of California Santa Cruz

INTRODUCTION

Cyber-Physical Systems (CPSs) are engineered systems that are built from, and depend upon, the seamless integration of computation, and physical components. While automatic control systems like the steam governor have existed for several centuries, it is only in the past decades that the automation of physical infrastructures like the power grid, water systems, or chemical reactions have migrated from analogue controls to embedded computer-based control, often communicating through computer-based networks. In addition, new advances in medical implantable devices, or autonomous self-driving vehicles are increasing the role of computers in controlling even more physical systems.

While computers give us new opportunities and functionalities for interacting with the physical world, they can also enable new forms of attacks. The purpose of this Knowledge Area is to provide an overview of the emerging field of CPS security.

In contrast with other Knowledge Areas within CyBOK that can trace the roots of their field back to several decades, the work on CPS security is relatively new, and our community has not developed yet the same consensus on best security practices compared to cyber security fields described in other KAs. Therefore, in this document, we focus on providing an overview of research trends and unique characteristics in this field.

CPSs are diverse and can include a variety of technologies, for example, industrial control systems can be characterised by a hierarchy of technology layers (the Purdue model [1905]). However, the security problems in the higher layers of this taxonomy are more related to classical security problems covered in other KAs. Therefore, the scope of this document focuses on the aspects of CPSs more closely related to the sensing, control, and actuation of these systems (e.g., the lower layers of the Purdue model).

The rest of the Knowledge Area is organised as follows. In Section 21.1 we provide an introduction to CPSs and their unique characteristics. In Section 21.2, we discuss crosscutting security issues in CPSs generally applicable to several domains (e.g., the power grid or vehicle systems); in particular we discuss efforts for preventing, detecting, and responding to attacks. In Section 21.3, we summarise the specific security challenges in a variety of CPS domains, including the power grid, transportation systems, autonomous vehicles, robotics, and medical implantable devices. Finally, in Section 21.4, we examine the unique challenges CPS security poses to regulators and governments. In particular, we outline the role of governments in incentivising security protections for CPSs, and how CPS security relates to national security and the conduct of war.

CONTENT

21.1 CYBER-PHYSICAL SYSTEMS AND THEIR SECURITY RISKS

[1906, 1907, 1908]

The term Cyber-Physical Systems (CPSs) emerged just over a decade ago as an attempt to unify the common research problems related to the application of embedded computer and communication technologies for the automation of physical systems, including aerospace,

automotive, chemical production, civil infrastructure, energy, healthcare, manufacturing, new materials, and transportation. CPSs are usually composed of a set of networked agents interacting with the physical world; these agents include sensors, actuators, control processing units, and communication devices, as illustrated in Figure 21.1.

The term CPSs was coined in 2006 by Helen Gill from the National Science Foundation (NSF) in the United States [1906]. In their program announcement, NSF outlined their goal for considering various industries (such as water, transportation, and energy) under a unified lens: by abstracting from the particulars of specific applications in these domains, the goal of the CPS program is to reveal crosscutting fundamental scientific and engineering principles that underpin the integration of cyber and physical elements across all application sectors.

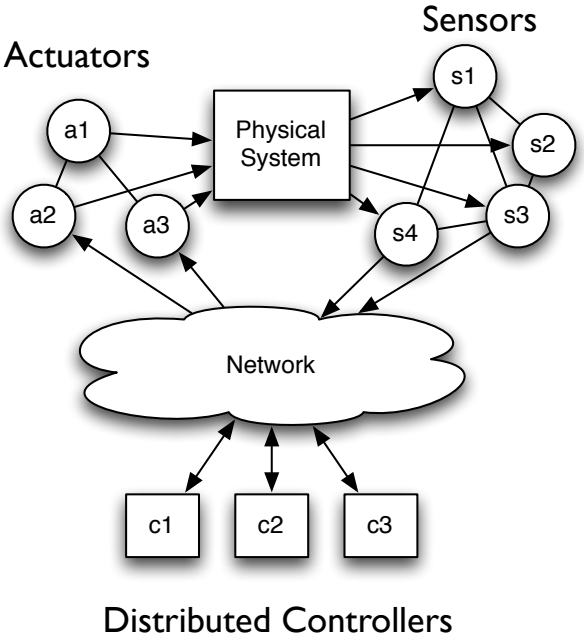


Figure 21.1: General architecture of cyber-physical systems [1909].

Soon after the CPS term was coined, several research communities rallied to outline and understand how CPSs *cyber security* research is fundamentally different when compared to conventional IT cyber security. Because of the crosscutting nature of CPSs, the background of early security position papers from 2006 to 2009 using the term CPSs, ranged from real-time systems [1910, 1911], to embedded systems [1912, 1913], control theory [1909], and cybersecurity [1908, 1913, 1914, 1915, 1916].

While cyber security research had been previously considered in other physical domains—most notably in the Supervisory Control and Data Acquisition (SCADA) systems of the power grid [1917]—these previous efforts focused on applying well-known IT cyber security best practices to control systems. What differentiates the early CPS security position papers was their crosscutting nature focusing on a *multi-disciplinary perspective* for CPS security (going beyond classical IT security). For example, while classical intrusion detection systems monitor purely cyber-events (network packets, operating system information, etc.), early CPSs papers bringing control theory elements [1908] suggested that intrusion detection systems for CPSs could also monitor the *physical* evolution of the system and then check it against a model of the expected dynamics as a way to improve attack detection.

CPS is related to other popular terms including the Internet of Things (IoT), Industry 4.0,

or the Industrial Internet of Things, but as pointed out by Edward Lee, *the term "CPS" is more foundational and durable than all of these, because it does not directly reference either implementation approaches (e.g., "Internet" in IoT) nor particular applications (e.g., "Industry" in Industry 4.0). It focuses instead on the fundamental intellectual problem of conjoining the engineering traditions of the cyber and physical worlds* [1906].

The rest of this section is organised as follows: in Section 21.1.1, we introduce general properties of CPS, then in Section 21.1.2, we discuss how physical systems have been traditionally protected from accidents and failures, and how these protections are not enough to protect the system against cyber-attacks. We finalise this section by discussing the security and privacy risks in CPSs along with summarising some of the most important real-world attacks on control systems in Section 21.1.3.

21.1.1 Characteristics of CPS

CPSs embody several aspects of embedded systems, real-time systems, (wired and wireless) networking, and control theory.

Embedded Systems: One of the most general characteristics of CPSs is that, because several of the computers interfacing directly with the physical world (sensors, controllers, or actuators) perform only a few specific actions, they do not need the general computing power of classical computers—or even mobile systems—and therefore they tend to have limited resources. Some of these embedded systems do not even run operating systems, but rather run only on *firmware*, which is a specific class of software that provides low-level control of the device hardware; devices without an operating systems are also known as *bare metal* systems. Even when embedded systems have an operating system, they often run a stripped-down version to concentrate on the minimal tools necessary for the platform.

Real-Time Systems: For safety-critical systems, the *time* in which computations are performed is important in order to ensure the correctness of the system [1918]. Real-time programming languages can help developers specify timing requirements for their systems, and Real-Time Operating System (RTOS) guarantee the time to accept and complete a task from an application [1919].

Network Protocols: Another characteristic of CPSs is that these embedded systems communicate with each other, increasingly over IP-compatible networks. While many critical infrastructures such as power systems have used serial communications to monitor remote operations in their SCADA systems, it is only in the past two decades that the information exchange between different parts of the system has migrated from serial communications to IP-compatible networks. For example, the serial communications protocol *Modbus* was released by Modicon in 1979, and subsequent serial protocols with more capabilities included IEC 60870-5-101 and DNP3 in the 1990s. All these *serial* protocols were later adapted to support IP networks in the late 1990s and early 2000s with standards such as Modbus/TCP, and IEC 60870-5-104 [1920, 1921].

Wireless: While most of the long-distance communications are done over wired networks, wireless networks are also a common characteristic of CPSs. Wireless communications for embedded systems attracted significant attention from the research community in the early 2000s in the form of *sensor networks*. The challenge here is to build networks on top of low-powered and lossy wireless links, where traditional concepts for routing like the "hop distance" to a destination are no longer applicable, and other *link quality* metrics are more reliable, e.g.,

the expected number of times a packet has to be sent before a one-hop transmission is successful. While most of the research on wireless sensor networks was done in abstract scenarios, one of the first real-world successful applications of these technologies was in large process control systems with the advent of WirelessHART, ISA100, and ZigBee [1922, 1923]. These three communications technologies were developed on top of the IEEE 802.15.4 standard, whose original version defined frames sizes so small, that they could not carry the header of IPv6 packets. Since Internet-connected embedded systems are expected to grow to billions of devices in the next years, vendors and standard organisations see the need to create embedded devices compatible with IPv6. To be able to send IPv6 packets in wireless standards, several efforts tried to tailor IPv6 to embedded networks. Most notably the Internet Engineering Task Force (IETF) launched the 6LoWPAN effort, originally to define a standard to send IPv6 packets on top of IEEE 802.15.4 networks, and later to serve as an adaptation layer for other embedded technologies. Other popular IETF efforts include the RPL routing protocol for IPv6 sensor networks, and CoAP for application-layer embedded communications [1924]. In the consumer IoT space some popular embedded wireless protocols include Bluetooth, Bluetooth Low Energy (BLE), ZigBee, and Z-Wave [1925, 1926].

Control: Finally, most CPSs observe and attempt to control variables in the physical world. Feedback control systems have existed for over two centuries, including technologies like the steam governor, which was introduced in 1788. Most of the literature in control theory attempts to model a physical process with differential equations and then design a controller that satisfies a set of desired properties such as stability and efficiency. Control systems were initially designed with analogue sensing and analogue control, meaning that the control logic was implemented in an electrical circuit, including a panel of relays, which usually encoded *ladder logic* controls. Analogue systems also allowed the seamless integration of control signals into a *continuous-time* physical process. The introduction of digital electronics and the microprocessor, led to work on **discrete-time control** [1927], as microprocessors and computers cannot control a system in *continuous time* because sensing and actuation signals have to be sampled at discrete-time intervals. More recently, the use of computer networks allowed digital controllers to be further away from the sensors and actuators (e.g., pumps, valves, etc.), and this originated the field of **networked-controlled systems** [1928]. Another recent attempt to combine the traditional models of physical systems (like differential equations) and computational models (like finite-state machines) is encapsulated in the field of **hybrid systems** [1929]. Hybrid systems played a fundamental role in the motivation towards creating a CPS research program, as they were an example of how combining models of computation and models of physical systems can generate new theories that enable us to reason about the properties of cyber- and physical-controlled systems.

Having discussed these general characteristics of CPSs, one caveat is that CPSs are diverse, and they include modern vehicles, medical devices, and industrial systems, all with different standards, requirements, communication technologies, and time constraints. Therefore, the general characteristics we associate with CPSs might not hold true in all systems or implementations.

Before we discuss cyber security problems, we describe how physical systems operating under automatic control systems have been protected from accidents and natural failures, and how these protections against non-malicious adversaries are not enough against strategic attackers (i.e., attackers that know that these protections are in place and try to either bypass them or abuse them).

21.1.2 Protections Against Natural Events and Accidents

Failures in the control equipment of physical infrastructures can cause irreparable harm to people, the environment, and other physical infrastructures. Therefore, engineers have developed a variety of protections against accidents and natural causes, including *safety systems, protection, fault-detection, and robustness*.

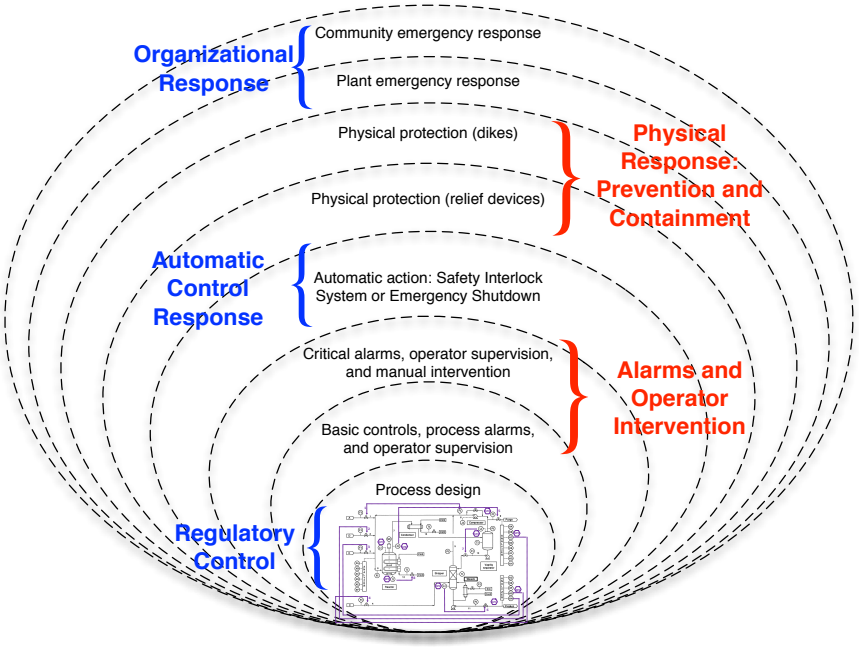


Figure 21.2: Layers of protection for safety-critical ICS.

Safety: The basic principle recommended by the general safety standard for control systems (IEC 61508) is to obtain requirements from a hazard and risk analysis including the likelihood of a given failure, and the consequence of the failure, and then design the system so that the safety requirements are met when all causes of failure are taken into account. This generic standard has served as the basis for many other standards in specific industries, for example, the process industry (refineries, chemical systems, etc.) use the IEC 61511 standard to design a Safety Instrumented System (SIS). The goal of a SIS is to prevent an accident by, e.g., closing a fuel valve whenever a high-pressure sensor raises an alarm. A more general defense-in-depth safety analysis uses *Layers of Protection* [1930], where hazards are mitigated by a set of layers starting from (1) basic low priority alarms sent to a monitoring station, to (2) the activation of SIS systems, to (3) mitigation safeguards such as physical protection systems (e.g., dikes) and (4) organisational response protocols for a plant emergency response/evacuation. Figure 21.2 illustrates these safety layers of protection.

Protection: A related concept to safety is that of protection in electric power grids. These protection systems include,

- Protection of Generators: when the frequency of the system is too low or too high, the generator will be automatically disconnected from the power grid to prevent permanent damage to the generator.
- Under Frequency Load Shedding (UFLS): if the frequency of the power grid is too low, controlled load shedding will be activated. This disconnection of portions of the electric distribution system is done in a controlled manner, while avoiding outages in safety-

critical loads like hospitals. UFLS is activated in an effort to increase the frequency of the power grid, and prevent generators from being disconnected.

- Overcurrent Protection: if the current in a line is too high, a protection relay will be triggered, opening the line, and preventing damage to equipment on each side of the lines.
- Over/Under Voltage Protection: if the voltage of a bus is too low or too high, a voltage relay will be triggered.

Reliability: While safety and protection systems try to prevent accidents, other approaches try to maintain operations even after failures in the system have occurred. For example, the electric system is designed and operated to satisfy the so-called N-1 security criterion, which means that the system could lose any one of its N components (such as one generator, substation, or transmission line) and continue operating with the resulting transients dying out to result in a satisfactory new steady-state operating condition, meaning that the reliable delivery of electric power will continue.

Fault Tolerance: A similar, but data-driven approach to detect and prevent failures falls under the umbrella of Fault Detection, Isolation, and Reconfiguration (FDIR) [1931]. Anomalies are detected using either a model-based detection system, or a purely data-driven system; this part of the process is also known as *Bad Data Detection*. Isolation is the process of identifying which device is the source of the anomaly, and reconfiguration is the process of recovering from the fault, usually by removing the faulty sensor (if there is enough sensor redundancy in the system).

Robust Control: Finally, another related concept is *robust control* [1932]. Robust control deals with the problem of uncertainty in the operation of a control system. These sources of unknown operating conditions can come from the environment (e.g., gusts of wind in the operation of planes), sensor noise, dynamics of the system not modelled by the engineers, or degradation of system components with time. Robust control systems usually take the envelope of least favourable operating conditions, and then design control algorithms so that the system operates safely, even in the worst-case uncertainty.

These mechanisms are not sufficient to provide security: Before *CPS security* was a mainstream field, there was a lot of confusion on whether safety, protection, fault-tolerance, and robust controls were enough to protect CPSs from cyber-attacks. However, as argued over a decade ago [1909], these protection systems generally assume independent, non-malicious failures, and in security, incorrect model assumptions are the easiest way for the adversary to bypass any protection. Since then, there have been several examples that show why these mechanisms do not provide security. For example Liu et al. [1933] showed how fault-detection (bad data detection) algorithms in the power grid can be bypassed by an adversary that sends incorrect data that is consistent with plausible power grid configurations, but at the same time is erroneous enough from the real values to cause problems to the system. A similar example for dynamic systems (systems with a “time” component) considers *stealthy attacks* [1934]. These are attacks that inject small false data in sensors so that the fault-detection system does not identify them as anomalies but, over a long-period of time, these attacks can drive the system to dangerous operating conditions. Similarly, the N-1 security criterion in the electric power grid assumes that if there is a failure, all protection equipment will react as configured, but an attacker can change the configuration of protection equipment in the power grid. In such a case, the outcome of an N-1 failure in the power grid will be completely unexpected, as equipment will react in ways that were unanticipated by the operators of the power grid,

leading to potential cascading failures in the bulk power system. Finally, in Section 21.1.3.1, we will describe how real-world attacks are starting to target some of these protections against accidents; for example, the Triton malware specifically targeted safety systems in a process control system.

Safety vs. Security: The addition of new security defences may pose safety concerns, for example, a power plant was shutdown because a computer rebooted after a patch [1935]. Software updates and patching might violate safety certifications, and preventing unauthorised users from accessing a CPS might also prevent first responders from access to the system in the case of an emergency (e.g., paramedics might need access to a medical device that prevents unauthorised connections). Security solutions should take these CPS safety concerns into account when designing and deploying new security mechanisms.

21.1.3 Security and Privacy Concerns

CPSs are at the core of health-care devices, energy systems, weapons systems, and transportation management. Industrial Control Systems systems, in particular, perform vital functions in critical national infrastructures, such as electric power distribution, oil and natural gas distribution, water and waste-water treatment, and intelligent transportation systems. The disruption of these CPSs could have a significant impact on public health, safety and lead to large economic losses.

For example, attacks on the power grid can cause blackouts, leading to interdependent cascading effects in other vital critical infrastructures such as computer networks, medical systems, or water systems creating potential catastrophic economic and safety effects in our society [1936]. Attacks on ground vehicles can create highway accidents [1937], attacks on GPS systems can mislead navigation systems and make drivers reach a destination desired by the attacker [1938], and attacks on consumer drones can let attackers steal, cause accidents or surreptitiously turn on cameras and microphones to monitor victims [1939].

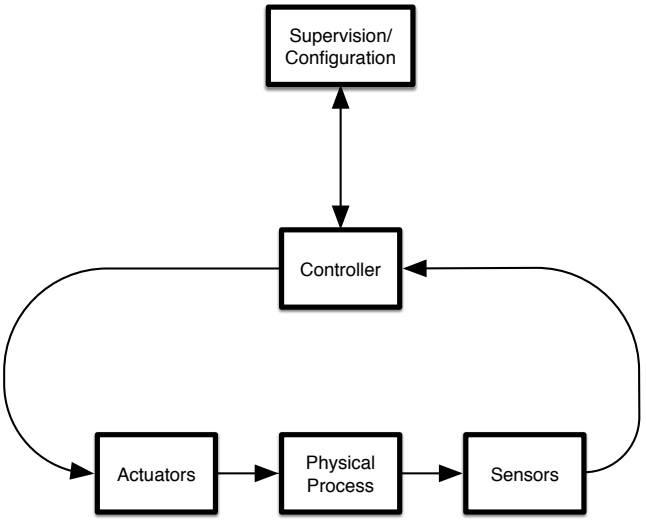


Figure 21.3: General Architecture of a CPS.

21.1.3.1 Attacks Against CPSs

In general, a CPS has a physical process under its control, a set of sensors that report the state of the process to a controller, which in turn sends control signals to actuators (e.g., a valve) to maintain the system in a desired state. The controller often communicates with a supervisory and/or configuration device (e.g., a SCADA system in the power grid, or a medical device programmer) which can monitor the system or change the settings of the controller. This general architecture is illustrated in Figure 21.3.

Attacks on CPSs can happen at any point in the general architecture, as illustrated in Figure 21.4, which considers eight attack points.

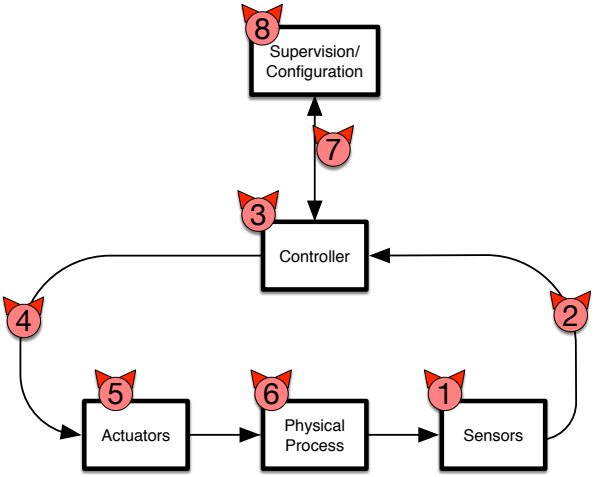


Figure 21.4: Attack Points in a CPS.

1. *Attack 1* represents an attacker who has compromised a sensor (e.g., if the sensor data is unauthenticated or if the attacker has the key material for the sensors) and injects false sensor signals, causing the control logic of the system to act on malicious data. An example of this type of attack is considered by Huang et al. [1940].
2. *Attack 2* represents an attacker in the communication path between the sensor and the controller, who can delay or even completely block the information from the sensors to the controller, so the controller loses observability of the system (loss of view), thus causing it to operate with *stale data*. Examples of these attacks include denial-of-service attacks on sensors [1941] and stale data attacks [1942].
3. *Attack 3* represents an attacker who has compromised the controller and sends incorrect control signals to the actuators. An example of this attack is the threat model considered by McLaughlin [1943].
4. *Attack 4* represents an attacker who can delay or block any control command, thus causing a denial of control to the system. This attack has been considered as a denial-of-service to the actuators [1941].
5. *Attack 5* represents an attacker who can compromise the actuators and execute a control action that is different to what the controller intended. Notice that this attack is different to an attack that directly attacks the controller, as this can lead to *zero dynamics attacks*. These types of attacks are considered by Teixeira et al. [1944].

6. *Attack 6* represents an attacker who can physically attack the system (e.g., physically destroying part of the infrastructure and combining this with a cyber-attack). This type of joint cyber and physical attack has been considered by Amin et al. [1945].
7. *Attack 7* represents an attacker who can delay or block communications to and from the supervisory control system or configuration devices. This attack has been considered in the context of SCADA systems [1946].
8. *Attack 8* represents an attacker who can compromise or impersonate the SCADA system or the configuration devices, and send malicious control or configuration changes to the controller. These types of attacks have been illustrated by the attacks on the power grid in Ukraine where the attackers compromised computers in the control room of the SCADA system [1947] and attacks where the configuration device of medical devices has been compromised [1948].

While traditionally most of the considered attacks on CPSs have been software-based, another property of CPSs is that the integrity of these systems can be compromised even without a computer-based exploit in what has been referred to as **transduction attacks** [1949] (these attacks represent a physical way to inject false signals, as covered by Attack 1 in Figure 21.4). By targeting the way sensors capture real-world data, the attacker can inject a false sensor reading or even a false actuation action, by manipulating the physical environment around the sensor [1949, 1950]. For example attackers can use speakers to affect the gyroscope of a drone [1951], exploit unintentional receiving antennas in the wires connecting sensors to controllers [1952], use intentional electromagnetic interference to cause a servo (an actuator) to follow the attacker's commands [1952], or inject inaudible voice commands to digital assistants [1953].

In addition to security and safety-related problems, CPSs can also have profound privacy implications unanticipated by designers of new systems. Warren and Brandeis stated in their seminal 1890 essay *The right to privacy* [132] that they saw a growing threat from recent inventions, like "instantaneous photographs" that allowed people to be unknowingly photographed, and new media industries, such as newspapers, that would publish photographs without their subjects' consent. The rise of CPS technologies in general, and consumer IoT in particular, are similarly challenging cultural assumptions about privacy.

CPS devices can collect physical data of diverse human activities such as electricity consumption, location information, driving habits, and biosensor data at unprecedented levels of granularity. In addition, the *passive* manner of collection leaves people generally unaware of how much information about them is being gathered. Furthermore, people are largely unaware that such collection exposes them to possible surveillance or criminal targeting, as the data collected by corporations can be obtained by other actors through a variety of legal or illegal means. For example, automobile manufacturers are remotely collecting a wide variety of driving history data from cars in an effort to increase the reliability of their products. Data known to be collected by some manufacturers include speed, odometer information, cabin temperature, outside temperature, battery status, and range. This paints a very detailed map of driving habits that can be exploited by manufacturers, retailers, advertisers, auto insurers, law enforcement, and stalkers, to name just a few.

Having presented the general risks and potential attacks to CPSs we finalise our first section by describing some of the most important real-world attacks against CPSs launched by malicious attackers.

21.1.3.2 High-Profile, Real-World Attacks Against CPSs

Control systems have been at the core of critical infrastructures, manufacturing and industrial plants for decades, and yet, there have been few confirmed cases of cyber-attacks (here we focus on attacks from malicious adversaries as opposed to attacks created by researchers for illustration purposes).

Non-targeted attacks are incidents caused by the same attacks that classical IT computers may suffer, such as the Slammer worm, which was indiscriminately targeting Windows servers but that inadvertently infected the Davis-Besse nuclear power plant [1954] affecting the ability of engineers to monitor the state of the system. Another non-targeted attack example was a controller being used to send spam in a water filtering plant [1955].

Targeted attacks are those where adversaries know that they are targeting a CPS, and therefore, *tailor their attack strategy with the aim of leveraging a specific CPS property*. We look in particular at attacks that had an effect in the physical world, and do not focus on attacks used to do reconnaissance of CPSs (such as Havex or BlackEnergy [1956]).

The first publicly reported attack on an SCADA system was the 2000 attack on Maroochy Shire Council's sewage control system[1] in Queensland, Australia [1958], where a contractor who wanted to be hired for a permanent position maintaining the system used commercially available radios and stolen SCADA software to make his laptop appear as a pumping station. During a 3-month period the attacker caused more than 750,000 gallons of untreated sewage water to be released into parks, rivers, and hotel grounds causing loss of marine life, and jeopardising public health. The incident cost the city council $176,000 in repairs, monitoring, clean-ups and extra security, and the contractor company spent $500,000 due to the incident [1959].

In the two decades since the Maroochy Shire attack there have been other confirmed attacks on CPSs [1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968]. However, no other attack has demonstrated the new sophisticated threats that CPSs face like the Stuxnet worm (discovered in 2010) targeting the Nuclear enrichment program in Natanz, Iran [701]. Stuxnet intercepted requests to read, write, and locate blocks on a Programmable Logic Controller (PLC). By intercepting these requests, Stuxnet was able to modify the data sent to, and returned from, the PLC, without the knowledge of the PLC operator. The more popular attack variant of Stuxnet consisted in sending incorrect rotation speeds to motors powering centrifuges enriching Uranium, causing the centrifuges to break down so that they needed to be replaced. As a result, centrifuge equipment had to be replaced regularly, slowing down the amount of enriched Uranium the Natanz plant was able to produce.

Two other high-profile confirmed attacks on CPSs were the December 2015 and 2016 attacks against the Ukrainian power grid [1969, 1970]. These attacks caused power outages and clearly illustrate the evolution of attack vectors. While the attacks in 2015 leveraged a remote access program that attackers had on computers in the SCADA systems of the distribution power companies, and as such a human was involved trying to send malicious commands, the attacks in 2016 were more automated thanks to the Industroyer malware [1971] which had knowledge of the industrial control protocols these machines use to communicate and could automatically craft malicious packets.

The most recent example in the arms race of malware creation targeting control systems

[1] There are prior reported attacks on control systems [1957] but there is no public information corroborating these incidents and the veracity of some earlier attacks has been questioned.

is the Triton malware [1972] (discovered in 2017 in the Middle-East) which targeted safety systems in industrial control systems. It was responsible for at least one process shutting down. Stuxnet, Industroyer, and Triton demonstrate a clear arms race in CPS attacks believed to be state sponsored. These attacks will have a profound impact on the way cyber-conflicts evolve in the future and will play an essential part in how wars may be waged, as we discuss in the last section of this chapter.

21.2 CROSSCUTTING SECURITY

[1973, 1974, 1975]

The first step for securing CPS is to identify the risks that these systems may have, and then prioritise how to address these risks with a defence-in-depth approach. Risk assessment consists of identifying assets in a CPS [1976], understanding their security exposure, and implementing countermeasures to reduce the risks to acceptable levels [1917, 1977, 1978, 1979, 1980]. Penetration testing is perhaps the most common way to understand the level of risk of the system and can be used to design a vulnerability management and patching strategy. The supply chain is also another risk factor, discussed further in the Risk Management & Governance Knowledge Area (Chapter 2).

One new area in CPSs is to identify the actuators or sensors that give the attacker maximum controlability of the CPS if they are compromised [1934, 1981, 1982, 1983, 1984] and then prioritise the protection of these devices.

Once the risks have been identified, a general defence-in-depth approach includes prevention, detection, and mitigation mechanisms. In this section we look at crosscutting security efforts to prevent, detect, and mitigate attacks, and the next section will look at specific CPS domains such as the power grid and intelligent transportation systems. This section is divided in three parts (1) preventing attacks (Section 21.2.1), (2) detecting attacks (Section 21.2.2), and (3) mitigating attacks (Section 21.2.3).

21.2.1 Preventing Attacks

The classical way to protect the first computer-based control systems was to have them isolated from the Internet, and from the corporate networks of the asset owners. As business practices changed, and efficiency reasons created more interconnections of control systems with other information technology networks, the concept of sub-network zone isolation was adopted by several CPS industries, most notably in the nuclear energy sector. This network isolation is usually implemented with the help of firewalls and *data diodes* [1985].

On the other hand, there are several ways to break the air gap, including *insider attacks*, or adding new connectivity to the network via mobile devices. Therefore, to prevent attacks in modern CPSs, designers and developers have to follow the same best security practices as classical IT systems; i.e., they need to follow a secure development life cycle to minimise software vulnerabilities, implement access control mechanisms, and provide strong cryptographic protections along with a secure key management system [1986].

While the best security practices of classical IT systems can give the *necessary* mechanisms for the security of control systems, these mechanisms alone are not *sufficient* for the defence-in-depth of CPSs. In this section we will discuss how, by understanding the interactions of the

CPS system with the physical world, we should be able to

1. better understand the consequences of an attack.
2. design novel attack-detection algorithms.
3. design new attack-resilient algorithms and architectures.

In the rest of this subsection we will focus on illustrating the challenges for implementing classical IT security best practices in CPSs, including the fact that several CPSs are composed of legacy systems, are operated by embedded devices with limited resources, and face new vulnerabilities such as analogue attacks.

Securing Legacy Systems: The life cycle of CPS devices can be an order of magnitude larger than regular computing servers, desktops, or mobile systems. Consumers expect that their cars last longer than their laptops, hospitals expect medical equipment to last over a decade, the assets of most industrial control systems last for at least 25 years [1987], and most of these devices will not be replaced until they are fully depreciated. Some of these devices were designed and deployed assuming a trusted environment that no longer exists. In addition, even if these devices were deployed with security mechanisms at the time, new vulnerabilities will eventually emerge and if the devices are no longer supported by the manufacturer, then they will not be patched. For example, after the Heartbleed vulnerability was discovered, major manufacturers pushed updates to mitigate this problem; however most embedded devices monitoring or controlling the physical world will not be patched (patching some safety-critical systems might even violate their safety certification). So even if a vendor used OpenSSL to create a secure communication channel between CPS devices originally, they also need to consider supporting the device over a long-time frame.

Therefore, to prevent attacks in CPSs we have to deal with (1) designing systems where security can be continuously updated, and (2) retrofitting security solutions for existing legacy systems [1988].

Some devices cannot be updated with these new secure standards, and therefore a popular way to add security to legacy networks is to add a **bump-in-the-wire** [1989]. Typically a bump-in-the-wire is a network appliance that is used to add integrity, authentication, and confidentiality to network packets exchanged between legacy devices. The legacy device thus sends unencrypted and unauthenticated packets and the network appliance will tunnel them over a secure channel to another bump-in-the-wire system at the other end of the communication channel that then removes the security protections and gives the insecure packet to the final destination. Note that a bump-in-the-wire can only protect the system from untrusted parties on a network, but if the end-point is compromised, a bump-in-the-wire won't be effective.

A similar concept has been proposed for wireless devices like implantable medical devices. Because some of these wireless devices communicate over insecure channels, attackers can listen or inject malicious packets. To prevent this, a **wireless shield** [1990] can be used near the vulnerable devices. The wireless shield will jam any communication attempt to the vulnerable devices except the ones from devices authorised by the owner of the shield. Wireless shields have also been proposed for other areas, such as protecting the privacy of consumers using BLE devices [1991]. Because of their disruptive nature, it is not clear if wireless shields will find practical applications in consumer applications.

Lightweight Security: While several embedded devices support classical cryptography, for some devices the performance of cryptographic algorithms in terms of energy consumption,

or latency, may not be acceptable [1992]. For symmetric cryptography, NIST has plans for the standardisation of a portfolio of lightweight cryptographic algorithms [1993] and the current CAESAR competition for an authenticated-encryption standard is evaluating the performance of their submissions in resource-constrained devices [1994]. For public-key algorithms, Elliptic Curve Cryptography generally offers the best balance of performance and security guarantees, but other lightweight public-key algorithms might be more appropriate depending on the requirements of the system [1995]. When it comes to exploit mitigation, the solutions are less clear. Most deeply embedded devices do not have support for data execution prevention, address space layout randomisation, stack canaries, virtual memory support, or cryptographically secure random number generators. In addition system-on-chip devices have no way to expand their memory, and real-time requirements might pose limitations on the use of virtual memory. However, there are some efforts to give embedded OS better exploit mitigation tools [1996].

Secure Microkernels: Another OS security approach is to try to formally prove the security of the kernel. The design of secure operating systems with formal proofs of security is an effort dating back to the *Orange Book* [1014]. Because the increasing complexity of code in monolithic kernels makes it hard to prove that operating systems are free of vulnerabilities, microkernel architectures that provide a minimal core of the functionality of an operating system have been on the rise. One example of such a system is the seL4 microkernel, which is notable because several security properties have been machine-checked with formal proofs of security [1034]. DARPA's HACMS program [1997] used this microkernel to build a quadcopter with strong safety and security guarantees [1997].

Preventing Transduction Attacks: As introduced in the previous section, *transduction attacks* represent one of the novel ways in which CPS security is different from classical IT security. Sensors are transducers that translate a physical signal into an electrical one, but these sensors sometimes have a coupling between the property they want to measure, and another analogue signal that can be manipulated by the attacker. For example, sound waves can affect accelerometers in wearable devices and make them report incorrect movement values [1998], and radio waves can trick pacemakers into disabling pacing shocks [1999]. Security countermeasures to prevent these attacks include the addition of better filters in sensors, improved shielding from external signals, anomaly detection, and sensor fusion [1950]. Some specific proposals include: drilling holes differently in a circuit board to shift the resonant frequency out of the range of the sensor, adding physical trenches around boards containing speakers to reduce mechanical coupling, using microfiber cloths for acoustic isolation, implementing low-pass filters that cut-off coupled signals, and secure amplifiers that prevent signal clipping [1949, 1998].

21.2.2 Detecting Attacks

Detecting attacks can be done by observing the internal state of a CPS device, by monitoring the interaction among devices to spot anomalous activities, or even using out-of-band channels.

In the first category, **Remote Attestation** is a field that has received significant attention for detecting malware in embedded systems because they usually do not have strong malware protections themselves [2000, 2001, 2002, 2003]. Remote attestation relies on the verification of the current internal state (e.g., RAM) of an untrusted device by a trusted verifier. There are three variants of remote attestation: software-based attestation, hardware-assisted attestation, and hybrid attestation. Software-based attestation does not rely on any special

security hardware in the device, but it has weak security guarantees and usually requires wireless range between the verifier and the device being checked. In contrast, hardware-based attestation (e.g., attestation with the support from a TPM, TrustZone or SGX) provides stronger security, but requires dedicated secure hardware in CPSs devices, which in turn increases their cost, which might not be affordable in some low-end embedded systems. Hybrid approaches attempt to find a middle ground by reducing the secure hardware requirements while overcoming the security limitations of pure software-based approaches [1865, 2004]. The minimal secure hardware requirements include a secure place to store the secret key, and safe code that has exclusive access to that key. A challenge for hybrid attestation is the fact that it needs to be non-interruptible and atomic (it has to run from the beginning to the end), and the (so far) relatively long (5-7 seconds [1865, 2004]) secure measurement of embedded memory might not be applicable for safety-critical real-time applications. In addition to academic work, industry is also developing standards to enhance the security of embedded systems with minimal silicon requirements. For example, the Trusted Computing Group (TCG) Device Identifier Composition Engine (DICE) is working on combining simple hardware capabilities to establish strong identity, attest software, and security policy, and assist in deploying software updates. We finalise our description of attestation by pointing out that most of the practical proposals for attestation work for initialisation, but building practical run-time attestation solutions remains a difficult challenge.

Network Intrusion Detection: The second category of solutions for detecting attacks relies on monitoring the interactions of CPS devices. In contrast with classical IT systems, where simple Finite-State models of network communications will fail, CPSs exhibit comparatively simpler network behaviour: servers change less frequently, there is a more stable network topology, a smaller user population, regular communication patterns, and networks host a smaller number of protocols. Therefore, intrusion detection systems, anomaly detection algorithms, and white listing access controls are easier to design and deploy than in classical IT systems [2005]. If the CPS designer can give a specification of the intended behaviour of the network, then any non-specified traffic can be flagged as an anomaly [2006]. Because most of the communications in CPS networks are between machines (with no human intervention), they happen automatically and periodically, and given their regularity, these communication patterns may be captured by finite state models like Deterministic Finite Automata [2007, 2008] or via Discrete-Time Markov Chains [2009, 2010]. While network specification is in general easier in CPS environments when compared to IT, it is still notoriously difficult to maintain.

Physics-Based Attack Detection: The major distinction of control systems with respect to other IT systems is the interaction of the control system with the physical world. In contrast to work in CPS intrusion detection that focuses on monitoring "cyber" patterns, another line of work studies how monitoring sensor (and actuation) values from physical observations, and control signals sent to actuators, can be used to detect attacks; this approach is usually called *physics-based* attack detection [1974]. The models of the physical variables in the system (their correlations in time and space) can be purely data-driven [2011], or based on physical models of the system [1934]. There are two main classes of physical anomalies: **historical anomalies** and **physical-law anomalies**.

Historical Anomalies: identify physical configuration we have not seen before. A typical example is to place limits on the observed behaviour of a variable [2012]. For example if during the learning phase, a water level in a tank is always between 1m and 2m, then if the water level ever goes above or below these values we can raise an alert. Machine learning models of the historical behaviour of the variables can also capture historical correlations of these variables. For example, they can capture the fact that when the tank of a water-level is high, the

water level of a second tank in the process is always low [2013]. One problem with historical anomalies is that they might generate a large number of false alarms.

Physical-Law Anomalies: A complementary approach to historical observations that may have fewer false alarms, is to create models of the physical evolution of the system. For example we have a sensor that monitors the height of a bouncing ball, then we know that this height follows the differential equations from Newton's laws of mechanics. Thus, if a sensor reports a trajectory that is not plausible given the laws of physics, we can immediately identify that something is not right with the sensor (a fault or an attack). Similarly, the physical properties of water systems (fluid dynamics) or the power grid (electromagnetic laws) can be used to create time series models that we can then use to confirm that the control commands sent to the field were executed correctly and that the information coming from sensors is consistent with the expected behaviour of the system. For example, if we open an intake valve we should expect that the water level in the tank should rise, otherwise we may have a problem with the control, actuator, or the sensor. Models of the physical evolution of the system have been shown to be better at limiting the short-term impact of stealthy attacks (i.e., attacks where the attacker creates a malicious signal that is within the margin of error of our physical models) [2014]. However, if the attack persists for a long time and drives the system to an unsafe region by carefully selecting a physically plausible trajectory, then historical models can help in detecting this previously unseen state [2015].

In addition to the physics of the system being controlled, devices (such as actuators) have dynamics as well, and these physical properties can also be used to monitor the proper behaviour of devices [2016].

Out-of-band Detection: Another way to passively monitor the physical system is through out-of-band channels [2017]. For example, Radio Frequency-based Distributed Intrusion Detection [2018] monitors radio frequency emissions from a power grid substation in order to check if there are malicious circuit breaker switching, transformer tap changes, or any activation of protecting relays without the direct request sent from the SCADA server. The basic idea is to correlate control commands sent by the SCADA server, with the radio frequency emissions observed in the substation. A potential drawback with this approach is that attackers can launch RF attacks mimicking the activation of a variety of electric systems, which can lead to security analysts losing confidence in the veracity of the alerts.

Active Detection: In addition to passively monitoring a CPS, an intrusion detection system can actively query devices to detect anomalies in how devices respond to these requests [2019]. In addition to a network query, the intrusion detection system can also send a *physical challenge* to change the system's physical behaviour. This approach is also known as **physical attestation** [2013, 2020, 2021], where a control signal is used to alter the physical world, and in response, it expects to see the changes done in the physical world reflected in the sensor values. For example, we can send signals to change the network topology of the power grid to see if the sensors report this expected change [2022], use a change in the field of vision of a camera to detect hacked surveillance cameras [2023], or use a watermarking signal in a control algorithm [2024]. The concept of active detection is related to research on *moving target defence* applied to cyber-physical systems [2025, 2026, 2027, 2028]. However, both active detection and moving target defence might impose unnecessary perturbations in a system by their change of the physical world for security purposes. Therefore, these techniques might be too invasive and costly. Consequently, the practicality of some of these approaches is uncertain.

21.2.3 Mitigating Attacks

Most of the efforts for mitigating faults in CPSs have focused on safety and reliability (the protection of the system against random and/or independent faults). Attack mitigation is an extension of safety and reliability protections for when the faults in the systems are not created at random by nature, but by an adversary.

Attack mitigation is related to the concept of **resilient control systems**, defined as those that maintain state awareness and an accepted level of operational normalcy in response to disturbances, including threats of an unexpected and malicious nature [2029].

There are two main types of mitigating technologies: i) proactive and ii) reactive. Proactive mitigation considers design choices deployed in the CPS prior to any attack. On the other hand, reactive responses only take effect once an attack has been detected, and they reconfigure the system online in order to minimise the impact of the attack. We first describe proactive approaches.

Conservative Control: One of the first ideas for mitigating the impact of attacks was to operate the system with enough safety margins so that if an attack ever occurred, it would be harder for the attacker to reach an unsafe region. One intuitive idea for this type of control algorithm is to use Model Predictive Control (MPC) to design a control strategy that predicts that an attack will happen starting at the next time step [1941], and therefore plans an optimal control action that will attempt to keep the system safe if the attack happens. Operating a CPS conservatively usually comes at the cost of suboptimal operation and extra costs when the system is not under attack.

Resilient Estimation: Resilient estimation algorithms attempt to obtain this state of a system, even if a subset of sensors is compromised [2030, 2031]. The basic idea is to use the knowledge of a CPS and the correlations of all sensor values. With enough redundancy in sensor measurements, a resilient estimation algorithm can reject attempted attacks and still obtain an accurate state estimate. This idea is similar to error correcting codes in information theory, where a subset of the bits transmitted can be corrupted, but the error correcting code reconstructs the original message. The drawback, however, is that not all CPSs will have a variety of correlated sensors to check the consistency of others, so this approach depends on the properties of the system.

Sensor Fusion: Resilient estimation algorithms usually assume a variety of multi-modal sensors to achieve their security guarantees. This is also the idea behind sensor fusion, where sensors of different types can help "confirm" the measurement of other sensors [2032, 2033, 2034]. A basic example of sensor fusion in automotive systems is to verify that both the LiDAR readings and the camera measurements report consistent observations.

Virtual Sensors: When we use *physical-laws* anomaly detection systems, we have, in effect, a model of the physical evolution of the system. Therefore, one way to mitigate attacks on the sensors of a CPS is to use a physical model of the system to come up with the expected sensor values that can then be provided to the control algorithm [1934, 2015, 2035]. By removing a sensor value with its expected value obtained from the system model, we are effectively controlling a system using open-loop control, which might work in the short-term, but may be risky as a long-term solution, as all physical models are not perfect, and the error between the real-world and the model simulation can increase over time. Another important consideration when designing virtual sensors as an attack-response mechanism, is to evaluate the safety of the system whenever the system is activated due to a false alarm [1934].

Constraining Actuation: A similar principle of operating conservatively is to physically constrain the actuators of a CPS so that if the attacker ever succeeds in gaining access to the system, it is restricted in how fast it can change the operation of the system. This approach can guarantee, for example, the safety of vehicle platooning systems, even when the attacker has complete control of one of the vehicles [2036].

Inertial Resets: Another idea to mitigate attacks is to reset and diversify the system as frequently as possible so that attackers are unable to gain persistent control of the system [2037, 2038]. The basic idea is that a full software reset of the system will make the system boot again in a trusted state, eliminating the presence of an attacker. This requires the system to have a trusted computing base that can boot the system in a secure state where the malware is not loaded yet. However, turning off a system that is in operation is a potentially dangerous action, and it is not clear if this proposal will be practical.

Reactive Control Compensation: When sensors or controllers are under attack, new actions are generated in order to maintain the safety of the system. Inspired by the literature on *fault-tolerant control*, one idea is to attempt to estimate the attack signal, and then generate a compensating action to eliminate it [2039]. The problem with this approach is that it does not consider strategic adversaries; however game-theoretic approaches can address that limitation. In game-theoretic models, an attacker compromises a set of control signals $u_k^a \in R^{ma}$ and the defender uses the remaining controllers $u_k^d \in R^{md}$ to deploy a defence action. The game between the attacker and the defender can be simultaneous (zero-sum or minimax game) [2040, 2041, 2042] or sequential (e.g., Stackelberg game) [2043, 2044, 2045]. One of the challenges with game theory is that, in order to model and prove results, the formulation needs to be simplified, and in addition, models need to add a number of extra assumptions that might not hold in practice.

Safe Control Actions: Another reactive approach is to change or even prevent a potentially malicious control action from acting on the system. The idea of having a High Assurance Controller (HAC) as a backup to a High Performance Controller (HPC) predates work on CPS security, and was proposed as a safety mechanism to prevent complex and hard-to verify HPCs from driving the system to unsafe states [2046]. A more recent and security-oriented approach is to use the concept of a *reference monitor* to check if the control action will result in any unsafe behaviour before it is allowed to go into the field [1943]. The proposed approach depends on a controller of controllers (C^2), which mediates all control signals sent by the controller to the physical system. In particular, there are three main properties that C^2 attempts to hold: 1) *safety* (the approach must not introduce new unsafe behaviours, i.e., when operations are denied the 'automated' control over the plant, it should not lead the plant to an unsafe behaviour); 2) *security* (mediation guarantees should hold under all attacks allowed by the threat model); and 3) *performance* (control systems must meet real-time deadlines while imposing minimal overhead).

All the security proposals for preventing, detecting, and responding to attacks presented in this section are generally applicable to CPSs. However, there are unique properties of each CPS application that can make a difference in how these solutions are implemented. Furthermore, some unique properties of a particular CPS domain can lead to new solutions (such as the *touch-to-access* principle proposed for implantable medical devices [2047]). In the next section we change focus from general and abstract CPS descriptions, to domain-specific problems and solutions.

21.3 CPS DOMAINS

[1973, 2048, 2049, 2050, 2051, 2052, 2053, 2054]

Having presented general principles for securing CPSs, in this section we discuss domain-specific security problems for CPSs. In particular we focus on industrial control systems, electrical power grids, transportation systems, vehicles, robots, medical devices, and consumer IoT.

21.3.1 Industrial Control Systems

Industrial control systems represent a wide variety of networked information technology systems connected to the physical world [2055]. Depending on the application, these control systems are also called Process Control Systems (PCSs) in the chemical industry, or Distributed Control Systems (DCSs) if the devices used for supervision and control are procured using a monolithic architecture.

Control systems are usually composed of a set of networked agents, consisting of sensors, actuators, control processing units such as Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), and communication devices. For example, the oil and gas industry uses integrated control systems to manage refining operations at plant sites, remotely monitor the pressure and flow of gas pipelines, and control the flow and pathways of gas transmission. Water utilities can remotely monitor well levels and control the wells' pumps; monitor flows, tank levels, or pressure in storage tanks; monitor pH, turbidity, and chlorine residual; and control the addition of chemicals to the water.

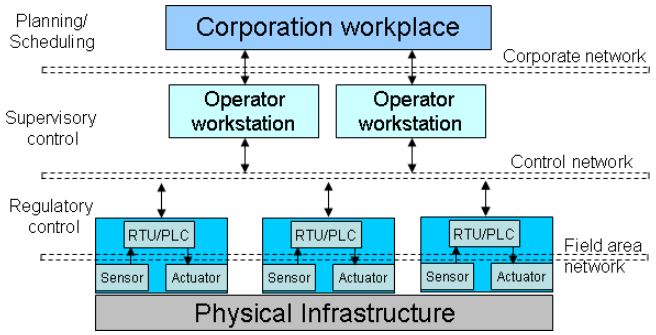


Figure 21.5: Bottom Layers of Industrial Control Systems [1908].

Control systems have a layered hierarchy [1905], which can be used for network segmentation and to ensure access control. Figure 21.5 shows an illustration of the lower layers of this hierarchy.

The top layers operate using mostly traditional Information Technology: computers, operating systems, and related software. They control the business logistic system, which manages the basic plant production schedule, material use, shipping and inventory levels, and also plant performance, and keep data historians for data-driven analytics (e.g., predictive maintenance).

The supervisory control layer is where the Supervisory Control and Data Acquisition (SCADA) systems and other servers communicate with remote control equipment like Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs). The communication between servers in a control room and these control equipment is done via a Supervisory Control Network (SCN).

Regulatory control is done at the lower layer, which involves instrumentation in the field, such as sensors (thermometers, tachometers, etc.) and actuators (pumps, valves, etc.). While traditionally this interface has been analogue (e.g., 4-20 milliamperes), the growing numbers of sensors and actuators as well as their increased intelligence and capabilities, has given rise to new Field Communication Networks (FCNs) where the PLCs and other types of controllers interface with remote Input/Output boxes or directly with sensors and actuators using new Ethernet-based industrial protocols like ENIP and PROFINET, and wireless networks like WirelessHART. Several ring topologies have also been proposed to avoid a single point of failure for these networks, such as the use of Device Level Ring (DLR) over ENIP.

SCN and FCN networks represent Oblivious Transfer (OT) networks, and they have different communication requirements and different industrial network protocols. While SCN can tolerate delays of up to the order of seconds, FCN typically require an order of magnitude of lower communication delays, typically enabling communications between devices with a period of 400 us.

Intrusion detection is a popular research topic for protecting control systems, and this includes using network security monitors adapted to industrial protocols [2005, 2007, 2008, 2009, 2010, 2056, 2057], and physics-based anomaly detection [1934, 2011, 2012, 2014, 2058, 2059]. The layer where we monitor the physics of the system can have a significant impact on the types of attacks that can be detected [2060].

In particular the adversary can compromise and launch attacks from (1) SCADA servers [2061], (2) controllers/PLCs [2062], (3) sensors [1933], and (4) actuators [2063], and each of these attacks can be observable at different layers of the system.

Most of the work on network security monitoring for industrial control systems has deployed network intrusion detection systems at the SCN. However, if an anomaly detection system is only deployed in the supervisory control network then a compromised PLC can send manipulated data to the field network, while pretending to report that everything is normal back to the supervisory control network. In the Stuxnet attack, the attacker compromised a PLC (Siemens 315) and sent a manipulated control signal u^a (which was different from the original u, i.e., $u^a \neq u$). Upon reception of u^a, the frequency converters periodically increased and decreased the rotor speeds well above and below their intended operation levels. While the status of the frequency converters y was then relayed back to the PLC, the compromised PLC reported a manipulated value $y_a \neq y$ to the control centre (claiming that devices were operating normally). A similar attack was performed against the Siemens 417 controller [2062], where attackers captured 21 seconds of valid sensor variables at the PLC, and then replayed them continuously for the duration of the attack, ensuring that the data sent through the SCN to the SCADA monitors would appear normal [2062]. A systematic study of the detectability of various ICS attacks (controller, sensor, or actuator attacks) was given by Giraldo et al. [2060], and the final recommendation is to deploy system monitors at the field network, as well as at the supervisory network, and across different loops of the control system.

In addition to attack detection, preventing the system from reaching unsafe states is also an active area of research [1943, 2064, 2065, 2066, 2067]. The basic idea is to identify that a control action can cause a problem in the system, and therefore a reference monitor will prevent this control signal from reaching the physical system. Other research areas include the retrofitting of security in legacy systems [1988, 2068], and malware in industrial control devices [2069, 2070]. A concise survey of *research* in ICS security was given by Krotofil and Gollmann [2071], and reviews of state-of-the-art practices in the field of ICS security include the work of Knowles et al. and Cherdantseva et al. [1980, 2072].

A problem for studying industrial control systems is the diversity of platforms, including the diversity of devices (different manufacturers with different technologies) and applications (water, chemical systems, oil and gas, etc.). Therefore one of the big challenges in this space is the reproducibility of results and the generality of industrial control testbeds [2073].

21.3.2 Electric Power Grids

At the turn of the century, the US National Academy of Engineering selected the top 20 engineering achievements of the twentieth century (the achievements that most improved people's quality of life) and at the top of this list, was the power grid [2074]. In the approximately 140 years since their inception, electric grids have extended transmission lines to 5 billion people around the world, bringing light, refrigeration, and many other basic services to people across the globe.

The power grid has three major parts: (1) generation, (2) transmission, and (3) distribution. Electric power is generated wherever it is convenient and economical, and then it is transmitted at high voltages (100kV-500kV) in order to minimise energy losses—electrical power is equal to voltage times electrical current ($P = VI$), (and given a constant power, high voltage lines have less electrical current), and therefore there is less energy lost as heat as the current moves through the transmission lines. Geographically, a distribution system is located in a smaller region thereby energy losses are less of a concern while safety (preventing accidents, fires, electrocutions, etc.) is more important, therefore they are operated at lower voltages.

The transmission system is an interconnected, redundant network that spans large regions (usually one country). Large generation plants and the transmission network (the first two parts of the power grid) are usually referred to as the **Bulk Power System**, and this bulk power system is responsible for the reliable delivery of electricity to large areas. A disruption in the bulk power grid can cause a country-level blackout that would require several days of a blackstart period to restart the system. In contrast, distribution systems (the third part of the grid) are much smaller, their networks are radial (non-redundant), and a failure in their system usually only causes a localised outage (e.g., a blackout in a neighborhood). This is the reason most government and industry efforts have prioritised the creation of standards for security in the bulk power system [2050].

One of the most popular lines of work related to the security of power systems is the study of false data injection attacks in order to cause the algorithms in the power grid to misbehave. The most popular of this type of attacks are the false data injection attacks against state estimation. In the power grid, operators need to estimate the phase angles x_k from the measured power flow y_k in the transmission grid. As mentioned in the section about CPS safety, bad data detection algorithms were meant to detect random sensor faults, not strategic attacks, and as Liu et al. [1933, 2075] showed, it is possible for an attacker to create false sensor signals that will not raise an alarm (experimental validation in software used by the energy sector was later confirmed [2076]). There has been a significant amount of follow up research focusing on false data injection for state estimation in the power grid, including the work of Dán and Sandberg[2077], who study the problem of identifying the best k sensors to protect in order to minimise the impact of attacks, and Kosut et al. [2078], who consider attackers trying to minimise the error introduced in the estimate, and defenders with a new detection algorithm that attempts to detect false data injection attacks. Further work includes [1982, 2022, 2079, 2080, 2081].

21.3.2.1 Smart Grids

While the current power grid architecture has served well for many years, there is a growing need to modernise the world's electric grids to address new requirements and to take advantage of the new technologies. This modernisation includes the integration of renewable sources of energy, the deployment of smart meters, the exchange of electricity between consumers and the grid, etc. Figure 21.6 illustrates some of these concepts. The rationale for modernising the power grid includes the following reasons:

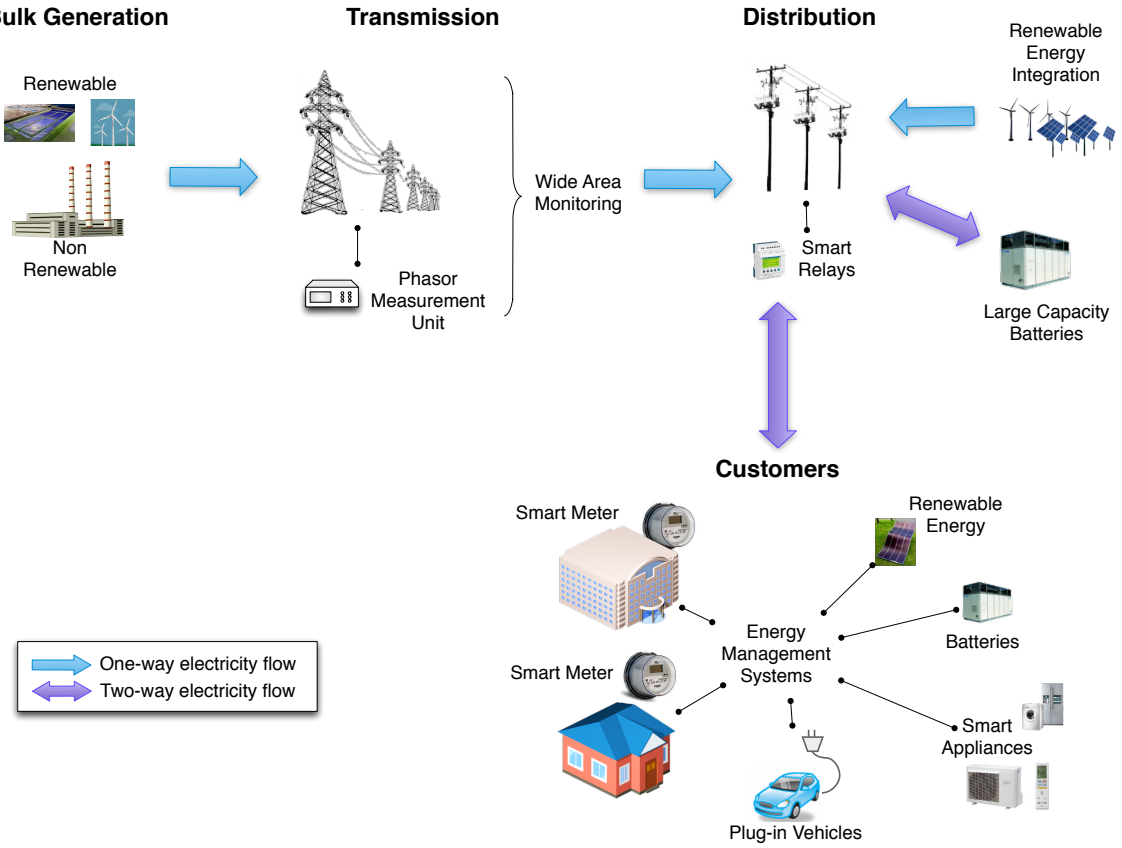


Figure 21.6: Modernization of the power grid [2082].

Efficiency: One of the main drivers of the smart grid programs is the need to make more efficient use of the current assets. The peak demand for electricity is growing every year and so utility companies need to spend more money each year in new power plants and their associated infrastructures. However, the peak demand is only needed 16% of the time and so the equipment required to satisfy this peak demand will remain idle for the rest of the time.

One of the goals for the smart grid is to change the grid from *load following* to *load shaping* by giving incentives to consumers for reducing electricity consumption at the times of peak demand. Reducing peak demand – in addition to increasing the grid stability – can enable utilities to postpone or avoid the construction of new power stations. The control or incentive actions used to shape the load is usually called *Demand Response*.

Efficiency also deals with the integration of the new and renewable generation sources, such as wind and solar power with the aim of reducing the carbon footprint.

Reliability: The second main objective of modernising the power grid is reliability, especially at the distribution layer (the transmission layer is more reliable). By deploying new sensors and

actuators throughout the power grid, operators can receive real-time, fine-grained data about the status of the power grid, that enables better situational awareness, faster detection of faults (or attacks), and better control of the system, resulting in fewer outages. For example, the deployment of smart meters is allowing distribution utilities to automatically identify the location and source of an outage.

Consumer choice: The third objective is to address the lack of transparency the current power grid provides to consumers. Currently, most consumers receive only monthly updates about their energy usage. In general, consumers do not know their electricity consumption and prices that they are paying at different times of the day. They are also not informed about other important aspect of their consumption such as the proportion of electricity that was generated through renewable resources. Such information can be used to shape the usage pattern (i.e., the load). One of the goals of the smart grid is to offer consumers real-time data and analytics about their energy use. Smart appliances and energy management systems will automate homes and businesses according to consumer preferences, such as cost savings or by making sure more renewable energy is consumed.

To achieve these objectives, the major initiatives associated with the smart grid are the advanced metering infrastructure, demand response, transmission and distribution automation, distributed energy resources, and the integration of electric vehicles.

While modernising the power grid will bring many advantages, it can also create new threat vectors. For example, by increasing the amount of collected consumer information, new forms of attack will become possible [2083]. Smart grid technologies can be used to infer the location and behaviour of users including if they are at home, the amount of energy that they consume, and the type of devices they own [2084, 2085]).

In addition to new privacy threats, another potential new attack has been referred to as *load-altering attack*. Load-altering attacks have been previously studied in demand-response systems [2086, 2087, 2088, 2089, 2090, 2091]. Demand-response programs provide a new mechanism for controlling the demand of electricity to improve power grid stability and energy efficiency. In their basic form, demand-response programs provide incentives (e.g., via dynamic pricing) for consumers to reduce electricity consumption during peak hours. Currently, these programs are mostly used by large commercial consumers and government agencies managing large campuses and buildings, and their operation is based on informal incentive signals via phone calls by the utility or by the demand-response provider (e.g., a company such as Enel X) asking the consumer to lower their energy consumption during the peak times. As these programs become more widespread (targeting residential consumers) and automated (giving utilities or demand-response companies the ability to directly control the load of their customers remotely) the attack surface for load-altering attacks will increase. The attacks proposed consider that the adversary has gained access to the company controlling remote loads and can change a large amount of the load to affect the power system and cause either inefficiencies to the system, economic profits for the attacker, or potentially cause enough load changes to change the frequency of the power grid and cause large-scale blackouts. Demand-response systems can be generalised by transactive energy markets, where *prosumers* (consumers with energy generation and storage capabilities) can trade energy with each other, bringing their own privacy and security challenges [2092].

More recently Soltan et al. [2093] studied the same type of load-altering attacks but when the attacker creates a large-scale botnet with hundreds of thousands of high-energy IoT devices (such as water heaters and air conditioners). With such a big botnet the attacker can cause (i) frequency instabilities, (ii) line failures, and (iii) increased operating costs. A followup work

by Huang et al. [2094] showed that creating a system blackout—which would require a black start period of several days to restart the grid—or even a blackout of a large percentage of the bulk power grid can be very difficult in part because the power grid has several protections to load changes, including under-frequency load shedding.

21.3.3 Transportation Systems and Autonomous Vehicles

Modern vehicular applications leverage ubiquitous sensing and actuation capabilities to improve transportation operations [2095] thanks to technologies such as smart phones [2096], participatory sensing [2097], and wireless communication networks [2098]. Modern functionalities include *Traffic flow control* with ramp metering at freeway on-ramps and signal timing plans at signalised intersections to reduce congestion; *Demand management* which focuses on reducing the excess traffic during peak hours; *Incident management* which targets resources to alleviate incident hot spots; and *Traveler information* which is used to reduce traveler buffer time, i.e., the extra time the travelers must account for, when planning trips.

While this large-scale collection of sensor data can enable various societal advantages, it also raises significant privacy concerns. To address these emerging privacy concerns from sensor data, many techniques have been proposed, including differential privacy [509].

Although privacy is an important concern for these systems, it is unfortunately not the only one. Widespread vulnerabilities such as those from traffic sensors [1968, 2099, 2100] can be readily exploited [2101, 2102, 2103, 2104]. For example, Wang et al. [2103] showed that attackers can inject false data in crowdsourced services to cause false traffic congestion alarms and fake accidents, triggering the services to automatically reroute traffic.

Similar problems can be found on commercial flights. Not only are airplanes being modernised while introducing potentially new attack vectors by attempting to attack avionic systems through the entertainment network [2105] but air traffic systems might also be vulnerable to attacks. A new technology complementing (or potentially replacing) radar systems is the Automatic Dependent Surveillance-Broadcast (ADS-B) system. ADS-B consists of airplanes sharing their GPS coordinates with each other and with air traffic control systems, but these systems are currently unauthenticated and unencrypted, posing security and privacy problems [2106].

21.3.3.1 Ground, Air, and Sea Vehicles

Software problems in the sensors of vehicles can cause notorious failures, as the Ariane 5 rocket accident [2107], which was caused by software in the inertial navigation system shut down causing incorrect signals to be sent to the engines. With advances in manufacturing and modern sensors, we are starting to see the proliferation of Unmanned Vehicles (UVs) in the consumer market as well as across other industries. Devices that were only available to government agencies have diversified their applications ranging from agricultural management to aerial mapping and freight transportation [2108]. Out of all the UVs available in the commercial market (aerial, ground and sea vehicles) unmanned aerial vehicles seem to be the most popular kind with a projected 11.2 billion dollar global market by 2020 [2109].

The expansion of unmanned aerial vehicles has increased security and privacy concerns. In general, there is a lack of security standards for drones and it has been shown that they are vulnerable to attacks that target either the cyber and/or physical elements [2052, 2110]. From

the point of view of privacy, drones can let users spy on neighbours [2111, 2112], and enable literal *helicopter parenting* [2113].

Attacks remotely accessing someone else's drone (e.g., a neighbour) to take photos or videos, stealing drones wirelessly (e.g., an attacker in a vehicle can take over a drone and ask it to follow the vehicle), and taking down a drone operated by someone else (which can lead to charges like mishandling a drone in public, which in turn has resulted in reckless endangerment convictions) [1939].

UVs have multiple sensors that aid them to assess their physical environments such as accelerometers, gyroscopes, barometers, GPS and cameras. While reliance on sensor data without any form of validation has proven to be an effective trade-off in order to maintain the efficiency demands of real-time systems, it is not a sustainable practice as UVs become more pervasive. *Transduction attacks* on sensors have shown that accelerometers, gyroscopes, and even cameras used by drones for stabilisation can be easily attacked, causing the drone to malfunction, crash, or even be taken over by the attacker [1951, 1998, 2114].

Even on many operational warships, remote monitoring of equipment is now done with a hardwired LAN by systems such as the Integrated Condition Assessment System (ICAS) [2115]. ICAS are generally installed with connections to external Programmable Logic Controllers (PLCs), which are used in Supervisory Control and Data Acquisition (SCADA) systems to direct the movement of control equipment that performs actual manipulation of physical devices in the ship such as propulsion and steering (rudder) devices [2115, 2116]. Therefore, the secure operation of ships is highly related to the security of industrial control systems.

For ground vehicles, one of the areas of interest is the security of the Controller Area Network (CAN). The CAN system is a serial broadcast bus designed by Bosch in 1983 to enable the communication of Electrical Control Units (ECUs) in cars. Examples of ECUs include brake systems, the central timing module, telematic control units, gear control, and engine control. The CAN protocol, however, does not have any security mechanism, and therefore an attacker who can enter the CAN bus in a vehicle (e.g., through a local or remote exploit) can spoof any ECU to ignore the input from drivers, and disable the brakes or stop the engine [2117]. Therefore, research has considered ways to retrofit lightweight security mechanisms for CAN systems [1791], or how to detect spoofed CAN messages based on the physical-layer characteristics of the signal [2118] (voltage level profiles, timing, frequency of messages, etc.). However, the security of some of these systems remains in question [2119].

Autonomous vehicles will also face new threats, for example, a malicious vehicle in an automated platoon can cause the platoon to behave erratically, potentially causing accidents [2120]. Finally, new functionalities like a remote kill-switch can be abused by attackers, for example, an attacker remotely deactivated hundreds of vehicles in Austin, Texas, leaving their owners without transportation [2121].

21.3.4 Robotics and Advanced Manufacturing

Security in manufacturing has been for many years a part of critical infrastructure security but, as the manufacturing process became more sophisticated, the threats have increased. Wells et al. [2053] give a high-level view about the concerns of this industry. They also mention that quality control techniques traditionally used in the manufacturing industry can be leveraged to detect attacks.

Attacks can target the structural integrity (scale, indent, or vertex) or material integrity (strength, roughness, or color) of the manufactured products [2122]. Physical tests, for example, non-destructive tests such as visual inspection, weight measure, dimension measure, 3D laser scanning, interferometry, X-ray, CT, and destructive mechanical tests like employing the tensile and yield properties of the material can help us in detecting attacks.

Robotic systems in automated assembly lines can also be used to create damaged parts or cause safety problems [2123]. Safety accidents with robots date back to 1979, when a worker at Ford motor company was killed by a robot. As pointed out by P.W. Singer, the Ford worker might have been the first, but he would be far from the last, as robots have killed various other people [2124]. Beyond manufacturing, robotic weapons also pose significant challenges. For example, in 2007 a software glitch in an antiaircraft system sporting two cannons began firing hundreds of high-explosive rounds, and by the time they were emptied, nine soldiers were dead, and fourteen seriously injured [2124]. We will discuss later in this document how new advances in CPSs may change the way nations wage future wars.

21.3.5 Medical Devices

Due to their safety and privacy risks, embedded medical devices are another CPS domain that has received significant attention in the literature.

While not an attack, the software error of the Therac-25 is one of the most well-known classical examples of how software problems can harm and even kill people. The Therac-25 was a computer-controlled radiation therapy machine that gave massive radiation overdoses to patients resulting in deaths and injuries [2125]. Our concern here is if these problems are not accidental but malicious?

Modern Implantable Medical Devices (IMDs) include pacemakers, defibrillators, neurostimulators, and drug delivery systems. These devices can usually be queried and reprogrammed by a doctor, but this also opens these devices up to security and privacy threats, in particular when an attacker can impersonate the device used by the doctor to modify the settings of IMDs.

Rushanan et al. [2054] and Camara et al. [2126] describe the types of adversaries that medical devices will be subject to, including the ability to eavesdrop all communication channels (passive) or read, modify and inject data (active). In order to mitigate possible attacks in the telemetry interface, they propose authentication (e.g., biometric, distance bounding, out of band channels, etc.), and the use of an external wearable device that allows or denies access to the medical device depending on whether this extra wearable device is present. In addition to prevention, they also discuss attack detection by observing patterns to distinguish between safe and unsafe behaviour.

In particular, a novel proposal to study proper authentication of the programmer with the IMD is the *touch-to-access* principle [2047, 2127]. The basic idea is that the patient has a biometric signal (such as the time between heart beats) that should only be available to other devices

in direct contact with the patient. This “secret” information is then used by the programmer and the IMD as a fuzzy password to bootstrap their security association.

A key challenge is to make sure that the biometric signal being used to give access via *touch-to-access*, is not remotely observable. However, heart beats can be inferred with side information including a webcam [2128], and an infrared laser [2129].

Security goes beyond implantable devices. As healthcare computer and software infrastructure introduces new technology, the industry will need to increase its security efforts. Medical data is a prime target for theft and privacy violations, and denial of service attacks in the form of ransomware [2130].

21.3.6 The Internet of Things

Consumer Internet of Things (IoT) devices are found everywhere: in our houses as voice-assistant devices, home automation smart devices, smart appliances, and surveillance systems; in healthcare as wearable technology including fitness devices and health-monitoring devices; in education including Internet-connected educational children toys; and for entertainment including remote controlled Wi-Fi devices.

As our lives become more dependent on these systems, their security has become an important, growing concern. The security of these devices depends on the integrity of the software and firmware they execute and the security mechanisms they implement.

New attack vectors make IoT devices attractive to criminals, like bad actors using vulnerable IoT devices to orchestrate massive Distributed Denial of Service (DDoS) attacks (the Mirai botnet) [665, 2131], attackers who compromised a fish tank to penetrate the internal network of a casino [2132], or attackers demanding ransomware from a hotel so they could let their guests enter their rooms [1961].

A large number of the IoT devices included in large IoT botnets [665, 2131] include Internet-connected cameras. Internet-connected cameras have given rise to multiple reports of unauthorised access by attackers [2133], and video feeds of multiple cameras are openly available online and discoverable through IoT web indexing platforms like Shodan [2134], potentially compromising the privacy of consumers who do not check the default configuration mechanisms. The threats to IoT go beyond privacy fears and DDoS attacks. Vulnerabilities in consumer IoT products including drones, IoT cameras, smart toys for children, and intimate devices can lead not only to privacy invasions but also to physical damages (drones being used to harm people), abuse, and harassment [2135]. Understanding the consequences of these new type of physical and mental abuses will require the involvement of more social scientists and legal scholars to help us define a framework on how to reason about them.

An area that has attracted significant attention from the research community is the security of voice-activated digital assistants. For example, researchers leveraged microphone non-linearities to inject inaudible voice commands to digital assistants [1953]. Other recent work includes the use of new attacks like “voice squatting” or “voice masquerading” to take over voice-controlled applications [2136]. For example the consumer might want to open the application “Capital One”, but an attacker can make an application available called “Capital Won” and the voice-controlled personal assistant might open the second functionality. In the “voice masquerading” attack, an attacker application might remain in control of the system and pretend to be following the consumer’s commands to open other functionalities, while in reality it is impersonating the desired functionalities.

Several of the security solutions for consumer IoT have proposed the idea of having a centralised IoT secure hub that mediates the communications between IoT devices in a home, and the Internet [2137]. One of the problems of relying on an external device to mediate IoT communications is that the connections between IoT device and the cloud servers may be encrypted, and therefore this hub will need to make security decisions with encrypted traffic [2138]. On the other hand, end-to-end encrypted communications can also prevent consumers from auditing their IoT devices to make sure they are not violating their privacy expectations. One option to address this problem is to ask the vendor of the IoT device to disclose their key (and rotate their key) to a trusted third party (called "auditor") that can decrypt and show the results to the owners of the data [2139].

In short, the proliferation of vulnerable IoT devices is raising new security and privacy concerns, while making IoT devices attractive to attackers. Insecurities in these devices range from insecure-by-design implementations (e.g., devices that have backdoors for troubleshooting) to their inability to apply software updates to patch vulnerable firmware. One of the biggest problems for improving the security of IoT and CPSs is that market forces do not incentivise vendors to compete for better security. In the next section we will discuss the causes of this lack of security and some potential solutions.

21.4 POLICY AND POLITICAL ASPECTS OF CPS SECURITY

[2124, 2140, 2141]

In this final section of the paper we summarise some of the industry- and government-led efforts to try to improve the security of CPSs, and how to leverage the new field of CPS security for attacks and wars.

21.4.1 Incentives and Regulation

Most industries in the CPS domain have rarely seen attacks sabotaging their physical process, in part because CPS attacks are hard to monetise by criminals. In addition to being rare, attacks on CPSs are not openly reported, and this lack of actuarial data leads to low quality risk estimates; as the US Department of Energy (DoE) stated in their Energy Delivery Systems Cyber Security Roadmap [2142]: "Making a strong business case for cyber security investments is complicated by the difficulty of quantifying risk in an environment of (1) rapidly changing, (2) unpredictable threats, (3) with consequences that are hard to demonstrate."

In summary, market incentives alone are insufficient to improve the security posture of CPSs, and as a result, our CPS infrastructures remain fairly vulnerable to computer attacks and with security practices that are decades behind the current security best practices used in enterprise IT domains. This market failure for improving the security of CPSs has resulted in several calls for government intervention [2143, 2144, 2145].

Regulation: Mandating cyber security standards that the CPS industries have to follow is a possible government intervention, and there is some precedent for this idea. Before 2003, the North American Electric Reliability Corporation (NERC) merely suggested standards to the power systems operators in the US but after the August 2003 blackout, regulations that were once optional are now mandatory [2049]. However, CPS industries have pushed back against regulation, arguing that regulations (e.g., mandating compliance to specific security standards)

will stifle innovation, and that more regulation tends to create a culture of *compliance* instead of a culture of *security*.

Some states in the US are starting to take regulation into their hands; for example, the recently proposed California Senate Bill SB-327 will make California the first state in the US with an IoT cyber security law—starting in 2020, any manufacturer of a device that connects "directly or indirectly" to the Internet must equip it with "reasonable" security features, designed to prevent unauthorised access, modification, or information disclosure.

The European Union Agency for cyber security proposed the EU Network and Information Security directive [2146] as the first piece of EU-wide cyber security legislation, where operators of essential services such as those outlined in this KA have to comply with these new sets of standards.

Another alternative to imposing regulation broadly, is to use the governments' "power of the purse" by mandating cyber security standards only to companies that want to do business with the government. The goal would be that once the best security practices are developed to meet the standards for working with the government, then they will spread to other markets and products. This approach is a reasonable balance between incentives and regulation. Only CPS and IoT vendors working with the Federal government will have to follow specific security standards, but once they are implemented, the same security standards will benefit other markets where they reuse the technologies.

One of the notable exceptions to the lack of regulation is the nuclear energy industry. Because of the highly safety-critical nature of this industry, nuclear energy is highly regulated in general, and in cyber security standards in particular, with processes such as the Office for Nuclear Regulation (ONR) Security Assessment Principles in the UK [2147].

Incentives: A complementary way to nudge companies to improve their cyber security posture is for governments to nurture a cyber-insurance market for CPS protection. So, instead of asking companies to follow specific standards, governments would demand firms to have cyber-insurance for their operations [2148, 2149, 2150, 2151]. There is a popular view that under certain conditions, the insurance industry can incentivise investments in protection [2152]. The idea is that premiums charged by the insurance companies would reflect the cyber security posture of CPS companies; if a company follows good cyber security practices, the insurance premiums would be low, otherwise, the premiums would be very expensive (and this would in principle incentivise the company to invest more in cyber security protections). It is not clear if this cyber-insurance market will grow organically, or if it would need to be mandated by the government.

It is unclear if government incentives to improve security in CPSs will require first a catastrophic cyber-attack, but it appears that, in the future, the choice will no longer be between government regulation and no government regulation, but between *smart government regulation and stupid regulation* [2140].

21.4.2 Cyber-Conflict

Computer networks enable an extension to the way we interact with others, and any conflict in the *real-world*, will have its representation in cyberspace; including (cyber-)crime, activism, bullying, espionage, and war [1916].

Cybercriminals compromise computers anywhere they can find them (even in control systems). These attacks may not be targeted (i.e., they do not have the intention of harming control systems), but may cause negative side effects: control systems infected with malware may operate inappropriately. The most famous non-targeted attack on control systems occurred in 2003, when the Slammer worm affected the computerised safety monitoring system at the Davis-Besse nuclear power plant in the US. While the plant was not connected to the Internet, the worm entered the plant network via a contractor's infected computer connected by telephone directly to the plant's network, thereby bypassing the firewall [1954]. A more recent example of a non-targeted attack occurred in 2006, when a computer system that managed the water treatment operations of a water filtering plant near Harrisburgh Pensylvania, was compromised and used to send spam and redistribute illegal software [1955]. More recently, ransomware has also been used to attack CPSs, like the attack on the Austrian hotel [1961], where guests were unable to get their room keys activated until the hotel paid the ransom.

Disgruntled employees are a major source of targeted computer attacks against control systems [780, 1960, 1963]. These attacks are important from a security point of view because they are caused by insiders: individuals with authorised access to computers and networks used by control systems. So, even if the systems had proper authentication and authorisation, as well as little information publicly available about them, attacks by insiders would still be possible. Because disgruntled employees generally act alone, the potential consequences of their attacks may not be as damaging as the potential harm caused by larger organised groups such as terrorists and nation states.

Terrorists, and activists are another potential threat to control systems. While there is no concrete evidence that terrorists or activists have targeted control systems via cyber-attacks, there is a growing threat of such an attack in the future.

Nation states are establishing military units with computer security expertise for any future conflicts. For example, the US established Cyber Command [2153] to conduct full spectrum *operations* (offensive capabilities) in 2009, and several other countries also announced similar efforts around the same time. The role of computer networks in warfare has been a topic of academic discussion since 1998 [2154], and CPSs are playing a foundational difference on how wars are waged, from robotic units and unmanned vehicles supporting soldiers in the field, to discussions of cyberwar [2155].

In addition to land, air, sea and space, cyberspace is now considered by many nations as an additional theatre of conflict. International treaties have developed public international law concerning two main principles in the law of war (1) *jus ad bellum* the right to wage a war, and (2) *jus in bellum* acceptable wartime conduct. Two sources have considered how the law of war applies to cyberspace [2141]: (1) The Tallinn Manual, and (2) the Koh Speech.

The Tallinn manual is a non-binding study by NATO's cooperative cyber-defence center of excellence, on how the law of war applies to cyber conflicts, and the Koh Speech was a speech given by Harold Koh, a US State Department legal advisor, which explained how the US interprets international law applied to cyberspace. Both of these sources agree that a key reason to authorise the use of force (*jus ad bellum*) as a response to a cyber operation, is

when the physical effects of a cyber-attack are comparable to kinetic effects of other armed conflicts, for example, when a computer attack triggers a nuclear plant meltdown, opens a dam upriver, or disables air-traffic control. The argument is that the effects of any of these attacks are similar to what a missile strike from an enemy would look like. In contrast, when there is no physical harm, the problem of determining when a cyber-attack can be considered a *use of force* by the enemy is unresolved, so cyber-attacks to the financial, or election infrastructure of a nation may not clear the bar to be considered an act of war.

Once nations are engaged in war, the question is how to leverage computer attacks in a way that is consistent with acceptable wartime conduct (*jus in bellum*). The conventional norm is that attacks must distinguish between military and non-military objectives. Military objectives can include war-fighting, war-supporting, and war-sustaining efforts. The problem in attacking critical infrastructures is that some of the infrastructures supporting these efforts are in dual-use by the military as well as by the civilian population. For example, a large percentage of military communications in the US use civilian networks at some stage, and the power grid supports military as well as civilian infrastructures.

Another factor to consider in designing CPS attacks is that the "law of war" in general prohibits uncontrollable or unpredictable attacks, in particular those that deny the civilian population of indispensable objects, such as food or water. While physical weapons have a limited geographical area of impact, cyberweapons can have more uncontrollable side-effects; for example, worms can replicate and escape their intended target network and infect civilian infrastructures. Therefore, nations will have to extensively test any cyberweapon to minimise unpredictable consequences.

In short, any future conflict in the physical world will have enabling technologies in the cyber-world, and computer attacks may be expected to play an integral part in future conflicts. There is a large grey area regarding what types of computer attacks can be considered an act of force, and a future challenge will be to design cyber-attacks that only target military objectives and minimise civilian side effects. At the same time, attack attribution in cyber-space will be harder, and nation-states might be able to get away with sabotage operations without facing consequences. It is a responsibility of the international community to design new legal frameworks to cover cyber-conflicts, and for nation states to outline new doctrines covering how to conduct cyber-operations with physical side effects.

Finally, cyberwar is also related to the discussion in the last section about cyber-insurance. For example, after the NotPetya cyberattack in 2017 [2156], several companies who had purchased cyber-insurance protections sought to get help from their insurance companies to cover part of their loses. However, some insurance companies denied the claims citing a *war exclusion* which protects insurers from being saddled with costs related to damage from war. Since then insurers have been applying the war exemption to avoid claims related to digital attacks [2]. This type of collateral damage from cyber-attacks might be more common in the future, and presents a challenge for insurance industries in their quest to quantify the risk of correlated large-scale events.

[2] **https://www.nytimes.com/2019/04/15/technology/cyberinsurance-notpetya-attack.html**

21.4.3 Industry Practices and Standards

We finalise the CPS Security KA by referencing various industry and government efforts for improving the security of CPSs. There are several industrial and government-led efforts to improve the security of control systems. One of the most important security standards in this space started with the Instruction Set Architecture (ISA) standard ISA 99, which later became a US standard with ANSI 62443 and finally an international cyber security standard for control systems known as IEC 62443 [2157].

The US National Institute of Standards and Technology (NIST) has guidelines for security best practices for general IT in Special Publication 800-53. US Federal agencies must meet NIST SP 800-53, but industry in general (and industry dealing with the US government in particular) uses these recommendations as a basis for their security posture. To address the security of control systems in particular, NIST has also published a Guide to Industrial Control System (ICS) Security [2048], a guideline to smart grid security in NIST-IR 762 [2158], and a guideline for IoT security and privacy [1973]. Although these recommendations are not enforceable, they can provide guidance for analysing the security of most utility companies. A more recent effort is the NIST cyber security framework for protecting critical infrastructure, which was initiated by an Executive Order from then US President Obama [2159], as an effort to improve the security posture of critical infrastructures.

Another notable industry-led effort for protecting critical infrastructures is the North American Electric Reliability Corporation (NERC) cyber security standards for control systems [2050]. NERC is authorised to enforce compliance to these standards, and it is expected that all electric utilities operating the bulk power system in North America are fully compliant with these standards.

All of these standards are general and flexible. Instead of prescribing specific technology solutions, they give a high-level overview of the variety of security technologies available (e.g., authentication, access control, network segmentation, etc.), and then give a set of general procedures for protecting systems, starting with (1) gathering data to identify the attack surface of a given system (this includes a basic network enumeration procedure that seeks to enumerate all devices and services available in the network of the asset owner), (2) building a security policy based on the attack surface of the system, and (3) deploy the security countermeasures, including network segmentation, or network security monitoring.

In addition to these general security standards for control systems, the industries that develop and maintain specific industrial control protocols, such as those used for SCADA, e.g., IEC 104, or those in the process industry, e.g., PROFINET, have also released standards and documentation for securing industrial networks. Recall that most of these industrial protocols were developed before security was a pressing concern for industrial control systems, therefore the communication links were not authenticated or encrypted. The new standard IEC 62351 is meant to guide asset owners on how to deploy a secure network to authenticate and encrypt network links, and other organisations have released similar support, such as, providing security extensions for PROFINET[3]. Instead (or in addition) to using these end-to-end application layer security recommendations, some operators might prefer to use lower-layer security protections of IP networks, including TLS and IPSec.

In the IoT domain, ETSI, the European Standards Organisation developed the first globally-applicable security standard for consumer IoT. ETSI TS 103 645 establishes a security baseline for Internet-connected consumer products and provide a basis for future IoT certification. This

[3] https://www.profibus.com/download/pi-white-paper-security-extensions-for-profinet/

standard builds closely on the UK's Code of Practice for Consumer IoT Security [2160]. Another more specific IoT standard by the Internet Engineering Task Force (IETF) for IoT devices is the Manufacturer Usage Description (MUD) standard [2161]. The goal of this standard is to automate the creation of network *white lists*, which are used by network administrators to block any unauthorised connection by the device. Other IoT security standards being developed by the IETF include protocols for communications security, access control, restricting communications, and firmware and software updates [2162].

All these industry efforts and standards have essentially three goals: (1) create awareness of security issues in control systems, (2) help operators of control systems and security officers design a security policy, and (3) recommend basic security mechanisms for prevention (authentication, access controls, etc), detection, and response to security breaches. For the most part industry efforts for protecting CPSs are based on the same technical principles from general Information Technology systems. Therefore, industry best practices are behind general IT security best practices and the most recent CPS security research discussed in this KA. We hope that in the next decade CPS security research becomes mature enough to start having an impact on industry practices.

CONCLUSIONS

As technology continues to integrate computing, networking, and control elements in new cyber-physical systems, we also need to train a new generation of engineers, computer scientists, and social scientists to be able to capture the multidisciplinary nature of CPS security, like transduction attacks. In addition, as the technologies behind CPS security mature, some of them will become industry-accepted best practices while others might be forgotten. In 2018, one of the areas with greatest momentum is the industry for network security monitoring (intrusion detection) in cyber-physical networks. Several start-up companies in the US, Europe, and Israel offer services for profiling and characterising industrial networks, to help operators better understand what is allowed and what should be blocked. On the other hand, there are other CPS security research areas that are just starting to be analysed, like the work on attack mitigation, and in particular, the response to alerts from intrusion detection systems.

We are only at the starting point for CPS security research, and the decades to come will bring new challenges as we continue to integrate physical things with computing capabilities.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	[2163]	*Other*
21.1 Cyber-Physical Systems and their Security Risks		
21.1.1 Characteristics of CPS	c1	[1906]
21.1.2 Protections Against Natural Events and Accidents		[1907]
21.1.3 Security and Privacy Concerns		[1908]
21.2 Crosscutting Security		
21.2.1 Preventing Attacks	c6,c9	[1973]
21.2.2 Detecting Attacks	c18	[1974]
21.2.3 Mitigating Attacks		[1975]
21.3 CPS Domains		
21.3.1 Industrial Control Systems		[2048]
21.3.2 Electric Power Grids	c25	[2049, 2050]
21.3.3 Transportation Systems and Autonomous Vehicles	c26, c29	[2051, 2052]
21.3.4 Robotics and Advanced Manufacturing		[2053]
21.3.5 Medical Devices	c27	[2054]
21.3.6 The Internet of Things		[1973]
21.4 Policy and Political Aspects of CPS Security		
21.4.1 Incentives and Regulation		[2140]
21.4.2 Cyber-Conflict		[2124, 2141]
21.4.3 Industry Practices and Standards		[2048]

Chapter 22
Physical Layer and Telecommunications Security

Srdjan Čapkun | ETH Zurich

INTRODUCTION

This Knowledge Area is a review of the most relevant topics in wireless physical layer security. The physical phenomenon utilized by the techniques presented in this Knowledge Area is the radiation of electromagnetic waves. The frequencies considered hereinafter consist of the entire spectrum that ranges from a few Hertz to frequencies beyond those of visible light (optical spectrum). This Knowledge Area covers concepts and techniques that exploit the way these signals propagate through the air and other transmission media. It is organised into sections that describe security mechanisms for wireless communication methods as well as some implications of unintended radio frequency emanations.

Since most frequencies used for wireless communication reside in the radio frequency spectrum and follow the well-understood laws of radio propagation theory, the majority of this Knowledge Area is dedicated to security concepts based on physical aspects of radio frequency transmission. The chapter therefore starts with an explanation of the fundamental concepts and main techniques that were developed to make use of the wireless communication layer for confidentiality, integrity, access control and covert communication. These techniques mainly use properties of physical layer modulations and signal propagation to enhance the security of systems.

After having presented schemes to secure the wireless channel, the Knowledge Area continues with a review of security issues related to the wireless physical layer, focusing on those aspects that make wireless communication systems different from wired systems. Most notably, signal jamming, signal annihilation and jamming resilience. The section on jamming is followed by a review of techniques capable of performing physical device identification (i.e., device fingerprinting) by extracting unique characteristics from the device's (analogue) circuitry.

Following this, the chapter continues to present approaches for performing secure distance measurements and secure positioning based on electromagnetic waves. Protocols for distance measurements and positioning are designed in order to thwart threats on the physical layer as well as the logical layer. Those attack vectors are covered in detail, together with defense strategies and the requirements for secure position verification.

Then, the Knowledge Area covers unintentional wireless emanations from devices such as from computer displays and summarises wireless side-channel attacks studied in literature. This is followed by a review on spoofing of analogue sensors. Unintentional emissions are in their nature different from wireless communication systems, especially because these interactions are not structured. They are not designed to carry information, however, they also make use of—or can be affected by—electromagnetic waves.

Finally, after having treated the fundamental concepts of wireless physical security, this Knowledge Area presents a selection of existing communication technologies and discusses their security mechanisms. It explains design choices and highlights potential shortcomings while referring to the principles described in the earlier sections. Included are examples from near-field communication and wireless communication in the aviation industry, followed by the security considerations of cellular networks. Security of global navigation systems and of terrestrial positioning systems is covered last since the security goals of such systems are different from communication systems and are mainly related to position spoofing resilience.

CONTENT

22.1 PHYSICAL LAYER SCHEMES FOR CONFIDENTIALITY, INTEGRITY AND ACCESS CONTROL

[1990, 2164, 2165, 2166, 2167, 2168]

Securing wireless networks is challenging due to the shared broadcast medium which makes it easy for remote adversaries to eavesdrop, modify and block the communication between devices. However, wireless communication also offers some unique opportunities. Radio signals are affected by reflection, diffraction, and scattering, all of which contribute to a complex multi-path behaviour of communicated signals. The channel response, as measured at the receiver, can therefore be modelled as having frequency and position dependent random components. In addition, within the short time span and in the absence of interference, communicating parties will measure highly correlated channel responses. These responses can therefore be used as shared randomness, unavailable to the adversary, and form a basis of secure communication.

It should be noted that modern-day cryptography provides many different protocols to assure the confidentiality, integrity and authenticity of data transmitted using radio signals. If the communicating parties are associated with each other or share a mutual secret, cryptographic protocols can effectively establish secure communication by making use of cryptographic keying material. However, if mere information exchange is not the only goal of a wireless system (e.g., in a positioning system), or if no pre-shared secrets are available, cryptographic protocols operating at higher layers of the protocol stack are not sufficient and physical-layer constructs can be viable solutions. The main physical layer schemes are presented in the following sections.

22.1.1 Key Establishment based on Channel Reciprocity

The physical-layer randomness of a wireless channel can be used to derive a shared secret. One of the main security assumptions of physical-layer key establishment schemes is that the attacker is located at least half a wavelength away from the communicating parties. According to wireless communication theory, it can be assumed that the attacker's channel measurements will be de-correlated from those computed by the communicating parties if they are at least half a wavelength apart. The attacker will therefore likely not have access to the measured secret randomness. If the attacker injects signals during the key generation, the signal that it transmits will, due to channel distortions, be measured differently at communicating parties, resulting in key disagreement.

Physical layer key establishment schemes operate as follows. The communicating parties (Alice and Bob) first exchange pre-agreed, non-secret, data packets. Each party then measures the channel response over the received packets. The key agreement is then typically executed in three phases.

Quantisation Phase: Alice and Bob create a time series of channel properties that are measured over the received packets. Example properties include RSSI and the CIR. Any property that is believed to be non-observable by the attacker can be used. The measured time series are then quantised by both parties independently. This quantisation is typically based on fixed or

dynamic thresholds.

Information reconciliation phase: Since the quantisation phase is likely to result in disagreeing sequences at Alice and Bob, they need to reconcile their sequences to correct for any errors. This is typically done leveraging error correcting codes and privacy amplification techniques. Most schemes use simple level-crossing algorithms for quantisation and do not use coding techniques. However, if the key derivation uses methods based on channel states whose distributions are not necessarily symmetric, more sophisticated quantisation methods, such as approximating the channel fading phenomena as a Gaussian source, or (multi-level) coding is needed [2165].

Key Verification Phase: In this last phase, communicating parties confirm that they established a shared secret key. If this step fails, the parties need to restart key establishment.

Most of the research in physical-layer techniques has been concerned with the choice of channel properties and of the quantisation technique. Even if physical-layer key establishment techniques seem attractive, many of them have been shown to be vulnerable to active, physically distributed and multi-antenna adversaries. However, in a number of scenarios where the devices are mobile, and where the attacker is restricted, they can be a valuable replacement or enhancement to traditional public-key key establishment techniques.

22.1.2 MIMO-supported Approaches: Orthogonal Blinding, Zero-Forcing

Initially, physical-layer key establishment techniques were proposed in the context of single-antenna devices. However, with the emergence of MIMO devices and beam-forming, researchers have proposed to leverage these new capabilities to further secure communication. Two basic techniques that were proposed in this context are orthogonal blinding and zero forcing. Both of these techniques aim to enable the transmitter to wirelessly send confidential data to the intended receiver, while preventing the co-located attacker from receiving this data. Although this might seem infeasible, since as well as the intended receiver, the attacker can receive all transmitted packets. However, MIMO systems allow transmitters to 'steer' the signal towards the intended receiver. For beam-forming to be effective, the transmitter needs to know some channel information for the channels from its antennas to the antennas of the receiver. As described in [2167], these channels are considered to be secret from the attacker. In Zero-Forcing, the transmitter knows the channels to the intended receiver as well as to the attacker. This allows the transmitter to encode the data such that it can be measured at the receiver, whereas the attacker measures nothing related to the data. In many scenarios, assuming the knowledge of the channel to the attackers is unrealistic. In Orthogonal Blinding, the transmitter doesn't know the channel to the attacker, but knows the channels to the receiver. The transmitter then encodes the data in the way that the receiver can decode the data, whereas the attacker will receive data mixed with random noise. The attacker therefore cannot decode the data. In order to communicate securely, the transmitter and the receiver do not need to share any secrets. Instead, the transmitter only needs to know (or measure) the channels to the intended receivers. Like physical-layer key establishment techniques, these techniques have been show to be vulnerable to multi-antenna and physically distributed attackers. They were further shown to be vulnerable to known-plaintext attacks.

22.1.3 Secrecy Capacity

Secrecy capacity is an information-theoretical concept that attempts to determine the maximal rate at which a wireless channel can be used to transmit confidential information without relying on higher-layer encryption, even if there is an eavesdropper present. A famous result by Shannon [2169] says that, for an adversary with unbounded computing power, unconditionally secure transmission can only be achieved if a one-time-pad cipher is used to encrypt the transmitted information. However, Wyner later showed that if the attacker's channel slightly degrades the information, that is, the channel is noisy, the secrecy capacity can indeed be positive under certain conditions [2170]. This means it is possible to convey a secret message without leaking any information to an eavesdropper. Csiszár and Korner extended Wyner's result by showing that the secrecy capacity is non-zero, unless the adversary's channel (wiretap channel) is less noisy than the channel that carries the message from the legitimate transmitter to the receiver [2171]. These theoretical results have been refined for concrete channel models by assuming a certain type of noise (e.g., Gaussian) and channel layout (e.g., SIMO and MIMO). Researchers have managed to derive explicit mathematical expressions and bounds even when taking into account complex phenomena such as fading which is present in wireless channels [2172].

A practical implementation of the concept of secrecy capacity can mainly be achieved using the two methods described above. Either the communicating parties establish a secret key by extracting features from the wireless channel (see 22.1.1) or they communicate with each other using intelligent coding and transmission strategies possibly relying on multiple antennas (see 22.1.2). Therefore, the study of secrecy capacity can be understood as the information-theoretical framework for key establishment and MIMO-supported security mechanisms in the context of wireless communication.

22.1.4 Friendly Jamming

Similar to Orthogonal Blinding, Friendly Jamming schemes use signal interference generated by collaborating devices to either prevent an attacker from communicating with the protected device, or to prevent the attacker from eavesdropping on messages sent by protected devices. Friendly Jamming can therefore be used for both confidentiality and access control. Unlike Orthogonal Blinding, Friendly Jamming doesn't leverage the knowledge of the channel to the receiver. If a collaborating device (i.e., the friendly jammer) wants to prevent unauthorised communication with the protected device it will jam the receiver of the protected device. If it wants to prevent eavesdropping, it will transmit jamming signals in the vicinity of the protected device. Preventing communication with a protected device requires no special assumptions on the location of the collaborating devices. However, protecting against eavesdropping requires that the eavesdropper is unable to separate the signals from the protected device from those originating at the collaborating device. For this to hold, the channel from the protected device to the attacker should not be correlated to the channel from the collaborating device to the attacker. To ensure this, the protected device and the collaborating device need to be typically placed less than half a carrier wavelength apart. This assumption is based on the fact that, in theory, an attacker with multiple antennas who tries to tell apart the jamming signal from the target signal requires the two transmitters to be separated by more than half a wavelength. However, signal deterioration is gradual and it has been shown that under some conditions, a multi-antenna attacker will be able to separate these signals and recover the transmitted messages.

Friendly jamming was originally proposed for the protection of those medical implants (e.g., already implanted pacemakers) that have no abilities to perform cryptographic operations. The main idea was that the collaborating device (i.e. 'the shield') would be placed around the user's neck, close to the pacemaker. This device would then simultaneously receive and jam all communication from the implant. The shield would then be able to forward the received messages to any other authorised device using standard cryptographic techniques.

22.1.5 Using Physical Layer to Protect Data Integrity

Research into the use of physical layer for security is not only limited to the protection of data confidentiality. Physical layer can also be leveraged to protect data integrity. This is illustrated by the following scenario. Assuming that two entities (Alice and Bob) share a common radio communication channel, but do not share any secrets or authentication material (e.g., shared keys or authenticated public keys), how can the messages exchanged between these entities be authenticated and how can their integrity be preserved in the presence of an attacker? Here, by message integrity, we mean that the message must be protected against any malicious modification, and by message authentication we mean that it should be clear who is the sender of the message.

One basic technique that was proposed in this context is *integrity codes*, a modulation scheme that provides a method of ensuring the integrity (and a basis for authentication) of a message transmitted over a public channel. Integrity codes rely on the observation that, in a mobile setting and in a multi-path rich environment, it is hard for the attacker to annihilate randomly chosen signals.

Integrity codes assume a synchronised transmission between the transmitter and a receiver, as well as the receiver being aware that it is in the range of the transmitter. To transmit a message, the sender encodes the binary message using a unidirectional code (e.g., a Manchester code), resulting in a known ration of 1s and 0s within an encoded message (for Manchester code, the number of 1s and 0s will be equal). This encoded message is then transmitted using on-off keying, such that each 0 is transmitted as an absence of signal and each 1 as a random signal. To decode the message and check its integrity, the receiver simply measures the energy of the signal. If the energy in a time slot is above a fixed threshold, the bit is interpreted as a 1 and if it is below a threshold, it is interpreted as a 0. If the ratio of bits 1 and 0 corresponds to the encoding scheme, the integrity of the message is validated. Integrity codes assume that the receiver knows when the transmitter is transmitting. This means that their communication needs to be scheduled or the transmitter needs to always be transmitting.

22.1.6 Low probability of intercept and Covert Communication

LPI signals are such signals that are difficult to detect for the unintended recipient. The simplest form of LPI is communication at a reduced power and with high directionality. Since such communication limits the range and the direction of communication, more sophisticated techniques were developed: Frequency Hopping, Direct Sequence Spread Spectrum and Chirping. In Frequency Hopping the sender and the receiver hop between different frequency channels thus trying to avoid detection. In Direct Sequence Spread Spectrum the information signal is modulated with a high rate (and thus high bandwidth) digital signal, thus spreading across a wide frequency band. Finally, Chirps are high speed frequency sweeps that carry information. The hopping sequence or chirp sequence constitute a secret shared between

receiver and transmitter. This allows the legitimate receiver to recombine the signal while an eavesdropper is unable to do so.

Covert communication is parasitic and leverages legitimate and expected transmissions to enable unobservable communication. Typically, such communication hides within the expected and tolerated deviations of the signal from its nominal form. One prominent example is embedding of communicated bits within the modulation errors.

22.2 JAMMING AND JAMMING-RESILIENT COMMUNICATION

[2173, 2174]

Communication jamming is an interference that prevents the intended receiver(s) from successfully recognising and decoding the transmitted message. It happens when the jammer injects a signal which, when combined with the legitimate transmission, prevents the receiver from extracting the information contained in the legitimate transmission. Jamming can be surgical and affect only the message preamble thus preventing decoding, or can be comprehensive and aim to affect every symbol in the transmission.

Depending on their behaviour, jammers can be classified as *constant* or *reactive*. Constant jammers transmit permanently, irrespective of the legitimate transmission. Reactive jammers are most agile as they sense for transmission and then jam. This allows them to save energy as well as to stay undetected. Jammer strength is typically expressed in terms of their output power and their effectiveness as the jamming-to-signal ratio at the receiver. Beyond a certain jamming-to-signal ratio, the receiver will not be able to decode the information contained in the signal. This ratio is specific to particular receivers and communication schemes. The main parameters that influence the success of jamming are transmission power of the jammer and benign transmitter, their antenna gains, communication frequency, and their respective distances to the benign receiver. These parameters will determine the jamming-to-signal ratio.

Countermeasures against jamming involve concealing from the adversary which frequencies are used for communication at which time. This uncertainty forces the adversary to jam a wider portion of the spectrum and therefore weakens their impact on the legitimate transmission, effectively reducing the jamming-to-signal ratio. Most common techniques include Chirp, FHSS and DSSS. Typically, these techniques rely on pre-shared secret keys, in which case we call the communication 'coordinated'. Recently, to enable jamming resilience in scenarios in which keys cannot be pre-shared (e.g., broadcast), uncoordinated FHSS and DSSS schemes were also proposed.

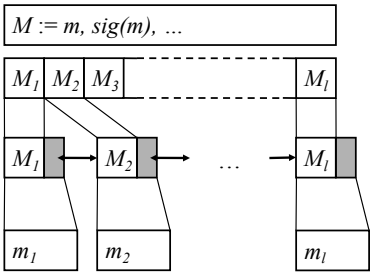


Figure 22.1: In UFH, the fragment linking protect against message insertion attack.

22.2.1 Coordinated Spread Spectrum Techniques

Coordinated Spread Spectrum techniques are prevalent jamming countermeasures in a number of civilian and military applications. They are used not only to increase resilience to jamming, but also to cope with interference from neighboring devices. Spreading is used in practically all wireless communication technologies, in e.g.,802.11, cellular, Bluetooth, global satellite positioning systems.

Spread spectrum techniques are typically effective against jammers that cannot cover the entire communication spectrum at all times. These techniques make a sender spread a signal over the entire available band of radio frequencies, which might require a considerable amount of energy. The attacker's ability to impact the transmission is limited by the achieved processing gain of the spread-spectrum communication. This gain is the ratio by which interference can be suppressed relative to the original signal, and is computed as a ratio of the spread signal radio frequency bandwidth to the un-spread information (baseband) bandwidth.

Spread-spectrum techniques use randomly generated sequences to spread information signals over a wider band of frequencies. The resulting signal is transmitted and then de-spread at the receivers by correlating it with the spreading sequence. For this to work, it is essential that the transmitter and receiver share the same secret spreading sequence. In FHSS, this sequence is the set of central frequencies and the order in which the transmitter and receiver switch between them in synchrony. In DSSS, the data signal is modulated with the spreading sequence; this process effectively mixes the carrier signal with the spreading sequence, thus increasing the frequency bandwidth of the transmitted signal. This process allows for both narrow band and wide band jamming to be suppressed at the receiver. Unless the jammer can guess the spreading code, its jamming signal will be spread at the receiver, whereas the legitimate transmission will be de-spread, allowing for its detection. The secrecy of the spreading codes is therefore crucial for the jamming resilience of spread spectrum systems. This is why a number of civilian systems that use spreading with public spreading codes, such as the GPS and 802.11b, remain vulnerable to jamming.

22.2.2 Uncoordinated Spread Spectrum Techniques

In broadcast applications and in applications in which communication cannot be anticipated as scheduled, there is still a need to protect such communication from jamming.

To address such scenarios, uncoordinated spread spectrum techniques were proposed: UFH and UDSSS. These techniques enable anti-jamming broadcast communication without pre-shared secrets. uncoordinated frequency hopping relies on the fact that even if the sender hops in a manner that is not coordinated with the receiver, the throughput of this channel will

be non-zero. In fact, if the receiver is broadband, it can recover all the messages transmitted by the sender. UFH however, introduces new challenges. Given that the sender and the receiver are not synchronised, and short message fragments transmitted within each hop are not authenticated, the attacker can inject fragments that make the reassembly of the packets infeasible. To prevent this, UFH includes fragment linking schemes that make this reassembly possible even under poisoning.

UDSSS follows the principle of DSSS in terms of spreading the data using spreading sequences. However, in contrast to anti-jamming DSSS where the spreading sequence is secret and shared exclusively by the communication partners, in UDSSS, a public set of spreading sequences is used by the sender and the receivers. To transmit a message, the sender repeatedly selects a fresh, randomly selected spreading sequence from the public set and spreads the message with this sequence. Hence, UDSSS neither requires message fragmentation at the sender nor message reassembly at the receivers. The receivers record the signal on the channel and despread the message by applying sequences from the public set, using a trial-and-error approach. The receivers are not synchronised to the beginning of the sender's message and thus record for (at least) twice the message transmission time. After the sampling, the receiver tries to decode the data in the buffer by using code sequences from the set and by applying a sliding-window protocol.

22.2.3 Signal Annihilation and Overshadowing

Unlike jamming where the primary goal of the attacker is to prevent information from being decoded at the receiver, signal annihilation suppresses the signal at the receiver by introducing destructive interference. The attacker's goal is to insert a signal which cancels out the legitimate transmitter's signal at the antenna of the receiver. This typically means that the attacker will generate a signal identical to the legitimate transmission only with a different polarity. Jamming attacks typically increase the energy on the channel and thus are more easily detected than signal annihilation which reduces the energy typically below the threshold of signal detection.

The goal of overshadowing is similar to jamming and signal annihilation in the sense that the attacker aims to prevent the receiver from decoding a legitimate signal. However, instead of interfering with the signal by adding excessive noise to the channel or cancelling out the signal (i.e., signal annihilation), the attacker emits their own signal at the same time and overshadows the legitimate signal. As a result, the receiver only registers the adversarial signal which is often orders of magnitude higher in amplitude than the legitimate signal. Practical overshadowing attacks were shown to be effective against QPSK modulation [2175] and more recently against cellular LTE systems [2176].

Malicious signal overshadowing can not only deceive the receiver into decoding different data than intended, it can also be used to alter any physical properties the receiver may extract during signal reception, such as angle of arrival or time of arrival. Overshadowing attacks have been shown to be particularly effective against systems that rely on physical layer properties including positioning and ranging systems.

22.3 PHYSICAL-LAYER IDENTIFICATION

[2177]

Physical-Layer Identification techniques enable the identification of wireless devices by unique characteristics of their analogue (radio) circuitry; this type of identification is also referred to as Radio Fingerprinting. More precisely, physical-layer device identification is the process of fingerprinting the analogue circuitry of a device by analysing the device's communication at the physical layer for the purpose of identifying a device or a class of devices. This type of identification is possible due to hardware imperfections in the analogue circuitry introduced at the manufacturing process. These imperfections are remotely measurable as they appear in the transmitted signals. While more precise manufacturing and quality control could minimise such artefacts, it is often impractical due to significantly higher production costs.

Physical-layer device identification systems aim at identifying (or verifying the identity of) devices or their affiliation classes, such as their manufacturer. Such systems can be viewed as pattern recognition systems typically composed of: *an acquisition setup* to acquire signals from devices under identification, also referred to as identification signals, a *feature extraction module* to obtain identification-relevant information from the acquired signals, also referred to as fingerprints, and a *fingerprint matcher* for comparing fingerprints and notifying the application system requesting the identification of the comparison results. Typically, there are two modules in an identification system: one for enrollment and one for identification. During enrollment, signals are captured either from each device or each (set of) class-representative device(s) considered by the application system. Fingerprints obtained from the feature extraction module are then stored in a database (each fingerprint may be linked with some form of unique ID representing the associated device or class). During identification, fingerprints obtained from the devices under identification are compared with reference fingerprints stored during enrollment. The task of the identification module can be twofold: either recognise (identify) a device or its affiliation class from among many enrolled devices or classes (1:N comparisons), or verify that a device identity or class matches a claimed identity or class (1:1 comparison).

The identification module uses statistical methods to perform the matching of the fingerprints. These methods are classifiers trained with Machine Learning techniques during the enrollment phase. If the module has to verify a 1:1 comparison, the classifier is referred to as binary. It tries to verify a newly acquired signal against a stored reference pattern established during enrollment. If the classifier performs a 1:N comparison, on the other hand, it attempts to find the reference pattern in a data base which best matches with the acquired signal. Often, these classifiers are designed to return a list of candidates ranked according to a similarity metric or likelihood that denotes the confidence for a match.

22.3.1 Device under Identification

Physical-layer device identification is based on fingerprinting the analogue circuitry of devices by observing their radio communication. Consequently, any device that uses radio communication may be subject to physical-layer identification. So far, it has been shown that a number of devices (or classes of devices) can be identified using physical-layer identification. These include analogue VHF, Bluetooth, WiFi, RFID and other radio transmitters.

Although what enables a device or a class of devices to be uniquely identified among other devices or classes of devices is known to be due to imperfections introduced at the manufacturing phase of the analogue circuitry, the actual device's components causing these have not always been clearly identified in all systems. For example, VHF identification systems are based on the uniqueness of transmitters' frequency synthesisers (local oscillators), while in RFID systems some studies only suggested that the proposed identification system may rely on imperfections caused by the RFID device's antennas and charge pumps. Identifying the exact components may become more difficult when considering relatively-complex devices. In these cases, it is common to identify in the whole analogue circuitry, or in a specific sub-circuit, the cause of imperfections. For example, IEEE 802.11 transceivers were identified considering modulation-related features; the cause of hardware artefacts can be then located in the modulator subcircuit of the transceivers. Knowing the components that make devices uniquely identifiable may have relevant implications for both attacks and applications, which makes the investigation of such components an important open problem and research direction.

22.3.2 Identification Signals

Considering devices communicating through radio signals, that is, sending data according to some defined specification and protocol, identification at the physical layer aims at extracting unique characteristics from the transmitted radio signals and to use those characteristics to distinguish among different devices or classes of devices. We define identification signals as the signals that are collected for the purpose of identification. Signal characteristics are mainly based on observing and extracting information from the properties of the transmitted signals, like amplitude, frequency, or phase over a certain period of time. These time-windows can cover different parts of the transmitted signals. Mainly, we distinguish between data and non-data related parts. The data parts of signals directly relate to data (e.g., preamble, midamble, payload) transmission, which leads to considered data-related properties such as modulation errors, preamble (midamble) amplitude, frequency and phase, spectral transformations. Non-data-related parts of signals are not associated with data transmission. Examples include the turn-on transients, near-transient regions, RF burst signals. These have been used to identify active wireless transceivers (IEEE 802.11, 802.15.4) and passive transponders (ISO 14443 HF RFID).

The characteristics extracted from identification signals are called features. Those can be predefined or inferred. Predefined features relate to well-understood signal characteristics. Those can be classified as in-specification and out-specification. Specifications are used for quality control and describe error tolerances. Examples of in-specification characteristics include modulation errors such as frequency offset, I/Q origin offset, magnitude and phase errors, as well as time-related parameters such as the duration of the response. Examples of out-specification characteristics include clock skew and the duration of the turn-on transient.

Differently from predefined features, where the considered characteristics are known in advance prior to recording of the signals, we say that features are inferred when they are extracted

from signals, for example, by means of some spectral transformations such as Fast Fourier Transform (FFT) or Discrete Wavelet Transform (DWT), without a-priori knowledge of a specific signal characteristic. For instance, wavelet transformations have been applied on signal turn-on transients and different data-related signal regions. The Fourier transformation has also been used to extract features from the turn-on transient and other technology-specific device responses. Both predefined and inferred features can be subject to further statistical analysis in order to improve their quality and distinguishing power.

22.3.3 Device Fingerprints

Fingerprints are sets of features (or combinations of features, that are used to identify devices. The properties that fingerprints need to present in order to achieve practical implementations are (similar to those of biometrics):

1. Universality. Every device (in the considered device-space) should have the considered features.
2. Uniqueness. No two devices should have the same fingerprints.
3. Permanence. The obtained fingerprints should be invariant over time.
4. Collectability. It should be possible to capture the identification signals with existing (available) equipments.

When considering physical-layer identification of wireless devices, we further consider:

5. Robustness. Fingerprints should not be subject, or at least, they should be evaluated with respect to external environmental aspects that directly influence the collected signal like radio interference due to other radio signals, surrounding materials, signal reflections, absorption, etc., as well as positioning aspects like the distance and orientation between the devices under identification and the identification system. Furthermore, fingerprints should be robust to device-related aspects like temperature, voltage level, and power level. Many types of robustness can be acceptable for a practical identification system. Generally, obtaining robust features helps in building more reliable identification systems.
6. Data-Dependency. Fingerprints can be obtained from features extracted from a specific bit pattern (data-related part of the identification signal) transmitted by a device under identification (e.g., the claimed ID sent in a packet frame). This dependency has particularly interesting implications if the fingerprints can be associated with both devices and data transmitted by those devices. This might strengthen authentication and help prevent replay attacks.

22.3.4 Attacks on Physical Layer Identification

The large majority of research works have focused on exploring feature extraction and matching techniques for physical-layer device identification. Only recently the security of these techniques started being addressed. Different studies showed that their identification system may be vulnerable to hill-climbing attacks if the set of signals used for building the device fingerprint is not carefully chosen. This attack consists of repeatedly sending signals to the device identification system with modifications that gradually improve the similarity score between these signals and a target genuine signal. They also demonstrated that transient-based approaches could easily be disabled by jamming the transient part of the signal while still enabling reliable communication. Furthermore, impersonation attacks on modulation-based identification techniques were developed and showed that low-cost software-defined radios as well as high end signal generators could be used to reproduce modulation features and impersonate a target device with a success rate of 50-75%. Modulation-based techniques are vulnerable to impersonation with high accuracy, while transient-based techniques are likely to be compromised only from the location of the target device. The authors pointed out that this is mostly due to presence of wireless channel effects in the considered device fingerprints; therefore, the channel needed to be taken into consideration for successful impersonation.

Generally, these attacks can be divided into two groups: *signal re(P)lay* and *feature replay attacks*. In a signal replay attack, the attacker's goal is to observe analogue identification signals of a target device, capture them in a digital form (digital sampling), and then transmit (replay) these signals towards the identification system by some appropriate means. The attacker does not modify the captured identification signals, that is, the analogue signal and the data payload are preserved. This attack is similar to message replay in the Dolev-Yao model in which an attacker can observe and manipulate information currently in the air at will. Unlike in signal replay attacks, where the goal of the attack is to reproduce the captured identification signals in their entirety, feature replay attack creates, modifies or composes identification signals that reproduce only the features considered by the identification system. The analogue representation of the forged signals may be different, but the features should be the same (or at the least very similar).

22.4 DISTANCE BOUNDING AND SECURE POSITIONING

[2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185]

Secure distance measurement (i.e., distance bounding) protocols were proposed to address the issue of the verification of proximity between (wireless) devices. Their use is broad and ranges from the prevention of relay attacks to enabling secure positioning.

Securing distance measurement requires secure protocols on the logical layer and a distance measurement technique resilient to physical layer attacks. To attack distance measurement, an attacker can exploit both data-layer as well as physical-layer weaknesses of distance measurement techniques and protocols. Data-layer attacks can be, to a large extent, prevented by implementing distance bounding protocols. However, physical-layer attacks are of significant concern as they can be executed independently of any higher-layer cryptographic primitive that is implemented.

22.4.1 Distance Bounding Protocols

Secure distance measurement protocols aim at preventing distance shortening and enlargement attacks. When they only prevent distance shortening, they are also called distance bounding protocols, where at the end of the protocol a secure upper bound on the distance is calculated. These protocols are typically executed with different trust assumptions. Devices measuring the distance (typically named verifier and prover) can be mutually trusted, in which case the protocol aims at preventing distance manipulation by an external attacker. If one of the devices, the prover, is untrusted, it will try to manipulate the measured distance. Other scenarios include the untrusted prover being helped by third parties to cheat on its distance. Distance bounding literature describes four main types of attacks 'frauds' corresponding to the above scenarios: distance fraud, mafia fraud, terrorist fraud and distance hijacking.

First investigations of distance bounding protocols started with the work of Beth and Desmedt [2179], and by Brands and Chaum [2180]. These protocols, as well as many that followed, are designed as cryptographic challenge-response protocols with RTT of flight measurements. One of the key insights of Brands and Chaum was to minimise the processing at the prover so that the prover cannot cheat on its distance to the verifier. Namely, this protocol requires that the prover only computes single bit XOR during the time-critical phase of the protocol. This translates into strong security guarantees as long as the prover cannot implement a faster XOR than assumed by the verifier. Hancke and Kuhn [2186] proposed an alternative protocol that uses register selection as a prover processing function. This design reduces the number of protocols steps by allowing the verifier and the prover to pre-agree on the nonces that will be used in the protocol exchange. Many protocols followed these two designs, notably addressing other types of frauds (especially terrorist fraud), as well as the robustness to message loss, performance in terms of protocol execution time, and privacy of distance measurement.

22.4.2 Distance Measurement Techniques

Establishing proximity requires estimating the physical distance between two or more wireless entities. Typically, the distance is estimated either by observing the changes in the signal's physical properties (e.g., amplitude, phase) that occur as the signal propagates or by estimating the time taken for the signal to travel between the entities.

A radio signal experiences a loss in its signal strength as it travels through the medium. The amount of loss or attenuation in the signal's strength is proportional to the square of the distance travelled. The distance between the transmitter and the receiver can therefore be calculated based on the free space path loss equation. In reality, the signal experiences additional losses due to its interaction with the objects in the environment which are difficult to account for accurately. This directly affects the accuracy of the computed distance and therefore advanced models such as the Rayleigh fading and log-distance path loss models are typically used to improve the distance estimation accuracy. Bluetooth-based proximity sensing tags (e.g., Apple iBeacon and passive keyless entry and Start Systems) use the strength of the received Bluetooth signal also referred to as the Received Signal Strength Indicator (RSSI) value as a measure of proximity.

Alternatively, the devices can measure the distance between them by estimating the phase difference between a received continuous wave signal and a local reference signal. The need for keeping track of the number of whole cycles elapsed is eliminated by using signals of different frequencies typically referred to as multi-carrier phase-based ranging. Due to their low

complexity and low power consumption, phase based ranging is used in several commercial products.

Finally, the time taken for the radio waves to travel from one point to another can be used to measure the distance between the devices. In RF-based RTT based distance estimation the distance d between two entities is given by $d = (t_{rx} - t_{tx}) \times c$, where c is the speed of light, t_{tx} and t_{rx} represent the time of transmission and reception respectively. The measured time-of-flight can either be one way time-of-flight or a round-trip time-of-flight. One way time-of-flight measurement requires the clocks of the measuring entities to be tightly synchronised. The errors due to mismatched clocks are compensated in the round-trip time-of-flight measurement.

The precise distance measurement largely depends on the system's ability to estimate the time of arrival and the physical characteristics of the radio frequency signal itself. The ranging precision is roughly proportional to the bandwidth of the ranging signal. Depending on the required level of accuracy, time-of-flight based distance measurement systems use either Impulse-Radio Ultra Wideband (IR-UWB) or Chirp-Spread Spectrum (CSS) signals. IR-UWB systems provide centimeter-level precision while the precision of CSS systems is of the order of 1–2m. There are a number of commercially available wireless systems that use chirp and UWB round-trip time-of-flight for distance measurement today.

22.4.3 Physical Layer Attacks on Secure Distance Measurement

With the increasing availability of low-cost software-defined radio systems, an attacker can eavesdrop, modify, compose, and (re)play radio signals with ease. This means that the attacker has full control of the wireless communication channel and therefore is capable of manipulating all messages transmitted between the two entities. In RSSI-based distance estimation, an attacker can manipulate the measured distance by manipulating the received signal strength at the verifier. The attacker can simply amplify the signal transmitted by the prover before relaying it to the verifier. This will result in an incorrect distance estimation at the verifier. Commercially available solutions claim to secure against relay attacks by simply reducing or attenuating the power of the transmitted signal. However, an attacker can trivially circumvent such countermeasures by using higher gain amplifiers and receiving antennas.

Similarly, an attacker can also manipulate the estimated distance between the verifier and the prover in systems that use the phase or frequency property of the radio signal. For instance, the attacker can exploit the maximum measurable property of phase or frequency-based distance measurement systems and execute distance reduction attacks. The maximum measurable distance, i.e., the largest value of distance d_{max} that can be estimated using a phase-based proximity system, directly depends on the maximum measurable phase. Given that the phase value ranges from 0 to 2π and then rolls over, the maximum measurable distance also rolls over after a certain value. An attacker can leverage this maximum measurable distance property of the system in order to execute the distance decreasing relay attack. During the attack, the attacker simply relays (amplifies and forwards) the verifier's interrogating signal to the prover. The prover determines the phase of the interrogating signal and re-transmits a response signal that is phase-locked with the verifier's interrogating signal. The attacker then receives the prover's response signal and forwards it to the verifier, however with a time delay. The attacker chooses the time delay such that the measured phase differences reaches its maximum value of 2 and rolls over. In other words, the attacker was able to prove to the verifier that the prover is in close proximity (e.g., 1m away) even though the prover was far from the verifier.

In Time of Flight (ToF) based ranging systems, the distance is estimated based on the time elapsed between the verifier transmitting a ranging packet and receiving an acknowledgement back from the prover. In order to reduce the distance measured, an attacker must decrease the signal's round trip time of flight. Based on the implementation, an attacker can reduce the estimated distance in a time-of-flight based ranging system in more than one way. Given that the radio signals travel at a speed of light, a 1 ns decrease in the time estimate can result in a distance reduction of 30cm.

The first type of attack on time-of-flight ranging leverages the predictable nature of the data contained in the ranging and the acknowledgement packets. A number of time-of-flight ranging systems use pre-defined data packets for ranging, making it trivial for an attacker to predict and generate their own ranging or acknowledgment signal. An attacker can transmit the acknowledgment packet even before receiving the challenge ranging packet. Several works have shown that the de-facto standard for IR-UWB, IEEE 802.15.4a does not automatically provide security against distance decreasing attacks. In [2187] it was shown that an attacker can potentially decrease the measured distance by as much as 140 meters by predicting the preamble and payload data with more than 99% accuracy even before receiving the entire symbol. In a 'Cicada' attack, the attacker continuously transmits a pulse with a power greater than that of the prover. This degrades the performance of energy detection based receivers, resulting in reduction of the distance measurements. In order to prevent such attacks it is important to avoid predefined or fixed data during the time critical phase of the distance estimation scheme.

In addition to having the response packet dependent on the challenge signal, the way in which these challenge and response data are encoded in the radio signals affects the security guarantees provided by the ranging or localisation system. An attacker can predict the bit (early detect) even before receiving the symbol completely. Furthermore, the attacker can leverage the robustness property of modern receivers and transmit arbitrary signal until the correct symbol is predicted. Once the bit is predicted (e.g., early-detection), the attacker stops transmitting the arbitrary signal and switches to transmitting the bit corresponding to the predicted symbol, i.e., the attacker 'commits' to the predicted symbol, commonly known as late commit. In such a scenario, the attacker needn't wait for the entire series of pulses to be received before detecting the data being transmitted. After just a time period, the attacker would be able to correctly predict the symbol.

As described previously, round-trip time-of-flight systems are implemented either using chirp or impulse radio ultrawideband signals. Due to their long symbol lengths, both implementations have been shown to be vulnerable to early-detect and late-commit attacks. In the case of chirp-based systems, an attacker can decrease the distance by more than 160 m and in some scenarios even up to 700 m. Although IR-UWB pulses are of short duration (typically 2–3 ns long), data symbols are typically composed of a series of UWB pulses. Furthermore, IEEE 802.15.4a IR-UWB standard allows long symbol lengths ranging from 32 ns to as large as 8μs. Therefore, even the smallest symbol length of 32 ns allows an attacker to reduce the distance by as much as 10 m by performing early-detect and late-commit attacks. Thus, it is clear that in order to guarantee proximity and secure a wireless proximity system against early detect and late-commit attacks, it is necessary to keep the symbol length as short as possible.

Design of a physical layer for secure distance measurement remains an open topic. However, research so far has yielded some guiding principles for its design. Only radio RTT with single-pulse or multi-pulse UWB modulation has been shown to be secure against physical layer attacks. As a result, the IEEE 802.15.4z working group started the standardization of a new

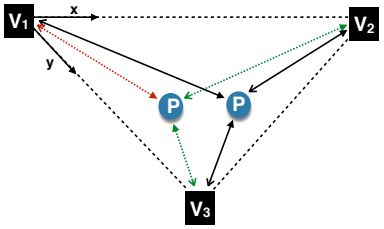


Figure 22.2: If the computed location of the prover is in the verification triangle, the verifiers conclude that this is a correct location. To spoof the position of prover inside the triangle, the attacker would need to reduce at least one of the distance bounds.

physical layer for UWB secure distance measurement.

The first attempt at formalizing the requirements for secure distance measurement based on the Time of Arrival (ToA) of transmitted messages can be found in [2185]. Said work presents a formal definition of Message Time of Arrival Codes (MTACs), the core primitive in the construction of systems for secure ToA measurement. If implemented correctly, MTACs provide the ability to withstand reduction and enlargement attacks on distance measurements. It is shown that systems based on UWB modulation can be implemented such that the stated security requirements are met and therefore constitute examples of MTAC schemes.

22.4.4 Secure Positioning

Secure positioning systems allow positioning anchors (also called verifiers) to compute the correct position of a node (also called the prover) or allow the prover to determine its own position correctly despite manipulations by the attacker. This means that the attacker cannot convince the verifiers or the prover that the prover is at a position that is different from its true position. This is also called spoofing-resilience. A related property is the one of secure position verification which means that the verifiers can verify the position of an untrusted prover. It is generally assumed that the verifiers are trusted. No restrictions are posed on the attacker as it fully controls the communication channel between the provers and the verifiers.

The analysis of broadcast positioning techniques, such as GNSS has shown that such techniques are vulnerable to spoofing if the attacker controls the signals at the antenna of the GNSS receiver.

These type of approaches have been proposed to address this issue: *Verifiable Multilateration* and *Secure Positioning based on Hidden Stations*.

Verifiable Multilateration relies on secure distance measurement / distance bounding. It consists of distance bound measurements to the prover from at least three verifiers (in 2D) and four verifiers (in 3D) and of subsequent computations performed by the verifiers or by a central system. Verifiable Multilateration has been proposed to address both secure positioning and position verification. In the case of secure positioning, the prover is trusted and mafia-fraud-resilient distance bounding is run between the prover and each of the verifiers. The verifiers form verification triangles / triangular pyramids (in 3D) and verify the position of the prover within the triangle / pyramid. For the attacker to spoof a prover from position P to P' within a triangle/pyramid, the attacker would need to reduce at least one of the distance bounds that are measured to P. This follows from the geometry of the triangle/pyramid. Since Distance bounding prevents distance reduction attacks, Verifiable Multilateration prevents spoofing attacks within the triangle/pyramid. The attacker can only spoof P to P' that is

outside of the triangle/pyramid, causing the prover and the verifiers to reject the computed position. Namely, the verifiers and the prover only accept the positions that are within the area of coverage, defined as the area covered by the verification triangles/pyramids. Given this, when the prover is trusted, Verifiable Multilateration is resilient to all forms of spoofing by the attacker. Additional care needs to be given to the management of errors and the computation of the position when distance measurement errors are taken into account.

When used for position verification, Verifiable Multilateration is run with an untrusted prover. Each verifier runs a distance-fraud resilient distance bounding protocol with the prover. Based on the obtained distance bounds, the verifiers compute the provers' position. If this position (within some distance and position error bounds) falls within the verification triangle/pyramid, the verifiers accept it as valid. Given that the prover is untrusted, it can enlarge any of the measured distances, but cannot reduce them since this is prevented by the use of distance bounding protocols. Like in the case of secure positioning, the geometry of the triangle/pyramid then prevents the prover from claiming a false position. Unlike in the case of secure positioning, position verification is vulnerable to cloning attacks, in which the prover shares its key to its clones. These clones can then be strategically placed to the verifiers and fake any position by enlarging distances to each individual verifier. This attack can be possibly addressed by tamper resistant hardware or device fingerprinting.

Another approach to secure positioning and position verification is to prevent the attacker from deterministically spoofing the computed position by making the positions of the verifiers unpredictable for the attacker (either a malicious prover or an external attacker). Verifier positions can therefore be hidden or the verifiers can be mobile. When the verifiers are hidden they should only listen to the beacons sent by the nodes to not disclose their positions. Upon receiving the beacons, the base stations compute the nodes location with TDOA and check if this location is consistent with the time differences.

22.5 COMPROMISING EMANATIONS AND SENSOR SPOOFING

[1953, 1998, 1999, 2188, 2189, 2190, 2191, 2192, 2193]

Electronic devices emit electromagnetic waves in the form of radio and audio signals, produce heat and create vibration, all of which could correlate with confidential information that the devices process or store. Such emanations, or more generally referred to as side channels, are prevalent and have been extensively studied.

Remote sensor spoofing is the (physical) opposite of compromising emanations. Instead of eavesdropping on electromagnetic leakage, an attacker injects signals that spoof the value measured by a sensor or receiver and thereby (adversely) affects the system relying on the sensor readings and measurements. This is particularly critical in autonomous and other cyber-physical systems that have direct consequences on the safety of the surrounding people and infrastructure.

22.5.1 Compromising Emanations

In the military context, techniques for exploiting and protecting against unwanted emission in communication systems date back to World War II and have over the time have been collected in an umbrella-term called TEMPEST. The first public demonstration of low-cost attacks on commercial systems using compromising emanations was done in 1985 by Wim van Eck [2194]. This attack demonstrated that information displayed on CRT monitors can be successfully eavesdropped from a distance of hundreds of meters. This demonstration prompted research into the sources of such emanations as well as into protective measures. It also highlighted that not only radio emissions leak information. In general, there are four categories of such emanations: acoustic, optical, thermal, and electromagnetic.

Detailed studies of the sources and features that lead to such compromises have been carried out over the years, and on multiple occasions, it was demonstrated that compromising emanations from analogue and digital displays resulted from information being transmitted through analogue video cables and through high-speed Digital Serial Interface (DVI) cables. However, more recent works show that such emanations are not restricted to cables and, to aggravate the situation, compromising emissions are not necessarily caused by analogue or digital displays only.

Some attacks described in research showed that high-frequency sounds caused by vibration of electronic components (capacitors and coils) in the computer's voltage regulation circuit can be used to infer prime factors and therefore derive RSA encryption keys. Sounds emanating from key presses on a keyboard were used to infer what a user is typing. The resulting vibrations can, for instance, be sensed by the accelerometer of a phone located nearby. Finally, reflections from different objects in the vicinity of computer screens, such as spoons, bottles and user's retina were used to infer information show on a display.

The increasing availability of phones that integrate high quality sensors, such as cameras, microphones and accelerometers makes it easier to mount successful attacks since no dedicated sensor equipment needs to be covertly put in place.

To avoid unwanted signal emissions, devices can be held at a distance, can be shielded and signals that are transmitted should be filtered in order to remove high-frequency components that might reflect switching activity in the circuitry. Moreover, it is generally advised to place a return wire close to the transmission wire in order to avoid exploitation of the return current. In general, wires and communication systems bearing confidential information should be separated (air-gapped) from non-confidential systems.

22.5.2 Sensor Compromise

Analogue sensors have been shown to be particularly vulnerable to spoofing attacks. Similar to compromising emanations, sensor spoofing depends on the type of the physical phenomena the sensor captures. It can be acoustic, optical, thermal, mechanic or electromagnetic.

Nowadays, many electronic devices, including self-driving cars, medical devices and closed-loop control systems, feature analogue sensors that help observe the environment and make decisions in a fully autonomous way. These systems are equipped with sophisticated protection mechanisms to prevent unauthorised access or compromise via the device's communication interfaces, such as encryption, authentication and access control. Unfortunately, when it comes to data gathered by sensors, the same level of protection is often not available or difficult to achieve since adversarial interactions with a sensor can be hard to model and

predict. As a result, unintentional and especially intentional EMI targeted at analogue sensors can pose a realistic threat to any system that relies on readings obtained from an affected sensor.

EMI has been used to manipulate the output of medical devices as well as to compromise ultrasonic ranging systems. Research has shown that consumer electronic devices equipped with microphones are especially vulnerable to the injection of fabricated audio signals [1999]. Ultrasonic signals were used to inject silent voice commands, and acoustic waves were used to affect the output of MEMS accelerometers. Accelerometers and intertial systems based on MEMS are, for instance, used extensively in (consumer-grade) drones and multi-copters.

Undoubtedly, sensor spoofing attacks have gained a lot of attention and will likely impact many future cyber-physical devices. System designers therefore have to take great care and protect analogue sensors from adversarial input as an attacker might trigger a critical decision on the application layer of such a device by exposing it to intentional EMI. Potential defence strategies include, for example, (analogue) shielding of the devices, measuring signal contamination using various metrics, or accommodating dedicated EMI monitors to detect and flag suspicious sensor readings.

A promising strategy that follows the approach of quantifying signal contamination to detect EMI sensor spoofing is presented in [2193]. The sensor output can be turned on and off according to a pattern unknown to the attacker. Adversarial EMI in the wires between sensor and the circuitry converting the reading to a digital value, i.e., the ADC, can be detected during the times the sensor is off since the sensor output should be at a known level. In case there are fluctuations in the readings, an attack is detected. Such an approach is thought to be especially effective when used to protect powered or non-powered passive sensors. It has been demonstrated to successfully thwart EMI attacks against a microphone and a temperature sensor system. The only modification required is the addition of an electronic switch that can be operated by the control unit or microcontroller to turn the sensor on and off. A similar sensor spoofing detection scheme can be implemented for active sensors, such as ultrasonic and infrared sensors, by incorporating a challenge-response like mechanism into the measurement acquisition process [2195]. An active sensor often has an emitting element and a receiving element. The emitter releases a signal that is reflected and captured by the receiver. Based on the properties of the received signal, the sensor can infer information about the entity or the object that reflected the signal. The emitter can be turned off randomly and during that time the receiver should not be able to register any incoming signal. Otherwise, an attack is detected and the sensor reading is discarded.

22.6 PHYSICAL LAYER SECURITY OF SELECTED COMMUNICATION TECHNOLOGIES

[2196, 2197, 2198, 2199]

This section presents security mechanisms of a selection of existing wireless communication techniques that are in use today. The main focus is on physical-layer security constructs as well as any lack thereof. The communication techniques that are discussed in detail are near-field communication, air traffic communication networks, cellular networks and global navigation satellite systems.

22.6.1 Near-field communication (NFC)

Near-field communication commonly refers to wireless communication protocols between two small (portable) electronic devices. The standard is used for contact-less payment and mobile payment systems in general. NFC-enabled devices can also exchange identity information, such as keycards, for access control, and negotiate parameters to establish a subsequent high-bandwidth wireless connection using more capable protocols.

NFC is designed to only transmit and receive data to a distance of up to a few centimeters. Even if higher-layer cryptographic protocols are used, vanilla NFC protocols do not offer secure communication and can not guarantee that two communicating devices are indeed only a short distance apart. NFC is vulnerable to eavesdropping, man-in-the-middle attacks and message relay attacks.

Even nowadays, standard NFC is deployed in security-critical contexts due to the assumption that communicating devices are in close proximity. Research has shown, however, that this assumption can not be verified reliably using NFC protocols. The distance can be made almost arbitrarily large by relaying messages between NFC-enabled devices. The attack works as follows: The benign NFC devices are made to believe that they are communicating with each other, but they are actually exchanging data with a modified smartphone. An adversary can strategically place a smartphone next to each benign NFC device while the smartphones themselves use a communication method that can cover long distances, such as WiFi. They simply forward the messages the benign devices are sending to each other. Such an attack is also referred to as a wormhole attack where communicating parties are tricked into assuming that they are closer than they actually are. This is a problem that cannot be solved using techniques on the logical layer or on the data layer.

Obviously, most of the described attacks can be mitigated by shielding the NFC devices or enhance the protocol with two-factor authentication, for example. Such mechanisms unfortunately transfer security-relevant decisions to the user of an NFC system. Countermeasures that do not impose user burden can roughly be categorised into physical layer methods and the augmentation with context- or device-specific identifiers [2196].

Protocol augmentation entails context-aware NFC devices that incorporate location information into the NFC system to verify proximity. The location sensing can be implemented with the help of a variety of different services, each with its own accuracy and granularity. Conceivable are, for instance, GNSS/GPS based proximity verification or leveraging the cell-ID of the base station to which the NFC device is currently closest in order to infer a notion of proximity.

Physical layer methods that have been suggested in research literature are timing restrictions and distance bounding. Enforcing strict timing restraints on the protocol messages can be understood as a crude form of distance bounding. As discussed in Section 4.1, distance bounding determines an upper bound on the physical distance between two communicating devices. While distance bounding is considered the most effective approach, it still remains to be shown if secure distance bounding can be implemented in practice for small NFC-enabled devices.

22.6.2 Air Traffic Communication Networks

Throughout different flight phases commercial and non-commercial aviation uses several wireless communication technologies to exchange information with aviation authorities on the ground as well as between airborne vehicles. Often legacy systems are still in use and security has never been part of the design of such systems.

While new proposals suggest to overhaul these systems and to tightly integrate security measures into the data layer, such as encryption and message authentication, air traffic communication networks are not only used for information transmission, but also to extract physical layer features from the signal in order to perform aircraft location positioning.

A prominent example is ADS-B. An ADS-B transponder periodically (or when requested) broadcasts the aircraft's position information, such as coordinates, that have been obtained through an on-board GNSS receiver. Most versions of ADS-B only support unauthenticated messages and therefore, this technology suffers from active and passive attacks, i.e., eavesdropping, modifying, injecting and jamming messages. It is, for instance, possible to prevent an aircraft's location from being tracked by Air Traffic Control (ATC) by simply jamming the respective messages. Similarly, an adversary could create ghost planes by emitting fabricated transponder messages. A sophisticated attacker could even fully distort the view ATC has on its airspace.

Multilateration (MLAT) can be seen as a technology that mitigates some of the shortcomings of unauthenticated ADS-B and is therefore usually deployed in conjunction with ADS-B. MLAT does not rely on the transmitted information encapsulated in the message, but makes use of the physical and geometrical constellation between the transmitter (i.e., transponder of the aircraft) and several receivers. MLAT systems extract physical layer properties from the received messages. The time of arrival of a message is recorded at different co-located receivers and, using the propagation speed of the signal, the location of the aircraft's transponder can be estimated. Multilateration techniques infer the aircraft's location even if the contents of the ADS-B messages are incorrect and thus MLAT provides a means to crosscheck the location information disseminated by the aircraft's transponder.

Although MLAT offers additional security based on physical layer properties, a distributed adversary can still manipulate ADS-B messages. In addition to altering the location information, an attacker can modify or inject signals that affect the time-of-arrival measurement at the receivers. If the attacker has access to multiple distributed antennas and is able to coordinate adversarial signal emission precisely, attacks similar to those on standard ADS-B are feasible. However, the more receivers used to record the signals, the more difficult such attacks become. Unfortunately, MLAT is not always an effective solution in aviation as strategic receiver placement is crucial and time of arrival calculations can be susceptible to multi-path interference [2197].

22.6.3 Cellular Networks

Cellular networks provide voice, data and messaging communication through a network of base stations, each covering one or more cells. The security provisions of these networks are mainly governed by the standards that were adopted in the GSM Association and later in the Third Generation Partnership Plan (3GPP).

Second Generation (2G) 'GSM' networks were introduced during the 1990s, and restricted their services to voice and text messaging. 2G networks were capable of carrying data via a Circuit-Switched Data Service (CSD) which operated in a manner similar to the dial-up modems, just over cellular networks. Further development of email and web services resulted in a need for enhanced speeds and services

3GPP improved 2G GSM standard with packet switched data service, resulting in the general packet radio service (GPRS). Like GSM, GPRS made use of the Home Location Register (HLR), a component that was responsible for subscriber key management and authentication. However, GPRS enhanced GSM by adding the Serving GPRS Support Node (SGSN) for data traffic routing and mobility management for better data traffic delivery. Third Generation (3G) of cellular networks, also known as Universal Mobile Telecommunications Systems (UMTS), introduced a number of improvements over 2G networks, including security enhancements, as well as increased uplink and downlink speeds and capacities. Fourth Generation (4G) cellular networks, also known as Long Term Evolution (LTE) introduced further increase in transmission speeds and capacities.

One of the main security properties that cellular networks aim to protect is the confidentiality of the communication of the link between the mobile station, and the base station and correct billing. The security of cellular networks has evolved with network generations, but all generations have the same overarching concept. Subscribers are identified via their (Universal) subscriber identity modules their International Mobile Subscriber Identity (IMSI) number and its related secret key. IMSI and the keys are used to authenticate subscribers as well as to generate the necessary shared secrets to protect the communication to the cellular network.

2G security focused on the confidentiality of the wireless link between the mobile station and the base station. This was achieved through the authentication via a challenge-response protocol, 2G Authentication and Key Agreement (AKA). This protocol is executed each time when a mobile station initiates a billable operation. 2G AKA achieved authentication based on a long term key K_i shared between the subscriber SIM card and the network. This key is used by the network to authenticate the subscriber and to derive a session key K_c. This is done within in a challenge response protocol, executed between the SGSN and the mobile station. Before the execution of the protocol, SGSN receives from the HLR the K_c, a random value $RAND$ and an expected response $XRES$. Both K_c and $XRES$ are generated within the HLR based on $RAND$ and K_i. When the mobile station attempts to authenticate to the network it is sent $RAND$. To authenticate, the mobile station combines its long term key K_i (stored on its SIM card) with the received $RAND$ to generate RES and K_c. The mobile station sends RES to the SGSN which compares it to $XRES$. If the two values match, the mobile station is authenticated to the network. The SGSN then sends the K_c to the base station to which the mobile station is connected in order to protect the mobile to base station wireless link.

2G AKA offered very limited protection. It used inadequate key size (56-64 bits), and weak authentication and key generation algorithms (A3,A5 and A8) which were, once released, broken, allowing for eavesdropping and message forgery. Furthermore, AKA was designed to provide only one-way authentication of the mobile station to the network. Since the network

did not authenticate to the mobile stations this enabled attacks by fake base stations violating users location privacy and confidentiality of their communication.

In order to address the 2G security shortcomings, 3G networks introduced new 3G Authentication and Key Agreement (3G AKA) procedures. 3G AKA replaced the weak cryptographic algorithms that were used in 2G and provided mutual authentication between the network and the mobile stations. Like in 2G, the goal of the protocol is the authentication (now mutual) of the network and the mobile station. The input into the protocol is a secret key K shared between the HLR and the subscriber. The outcome of the protocol are two keys, the encryption/confidentiality key CK and the integrity key IK. The generation of two keys allows the network and the mobile station to protect the integrity and confidentiality of their communication using two different keys, in line with common security practices. CK and IK are each 128 bits long which is considered adequate.

The authentication and key derivation is performed as follows. The HLR first generates the random challenge $RAND$, from it the expected response $XRES$, the keys CK and IK and the authentication token $AUTN$. It then sends these values to the SGSN. The SGSN sends the $RAND$ as well as the $AUTN$ to the mobile station (also denoted as User Equipment (UE)), which will then use its long term key K to generate the response RES and to verify if $AUTN$ was generated by the HLR. The $AUTN$ is from the shared key and the counter maintained by both the HLR and the mobile station. Upon receiving the RES from the mobile station, SGSN will compare it with the $XRES$ and if they match, will forward the CK and IK to the base station. The base and mobile station can now use these keys to protect their communication.

3G, however, still didn't resolve the vulnerabilities within the operator's networks. CK and IK are transmitted between different entities in the network. They are transmitted between SGSN and the associated base station as well as between different base stations during mobility. This allows network attackers to record these keys and therefore eavesdrop on wireless connections.

4G (LTE) security architecture preserved many of the core elements of 2G and 3G networks, but aimed to address the shortcomings of 3G in terms of the protection of the in-network traffic through the protection of network links and redistribution of different roles. For example, the long term key storage was moved from the HLR to the Home Subscriber Server (HSS). Mobility management was moved from the SGSN to the Mobility Management Engine (MME).

5G security architecture evolves 4G but follows a similar set of principles and entities. 5G introduces a new versions of Authentication and Key Agreement (AKA) protocols that was designed to fix the issues found in 4G, however with mixed success [1227].

22.6.4 GNSS Security and Spoofing Attacks

GNSS such as GPS and Galileo provide global navigation service through satellites that are orbiting the earth approximately 20,000km above the ground. Satellites are equipped with high-precision atomic clocks which allows the satellites to remain synchronised. Satellites transmit navigation messages at central frequencies of 1575.42MHz (L1) and 1227.60MHz (L2). direct sequence spreading is used to enable acquisition and to protect the signals carrying those messages from spoofing and jamming attacks. Civilian codes are public and therefore do not offer such protection, whereas military and special interest codes are kept confidential. Navigation messages carry data including satellite clock information, the ephemeris (information related to the satellite orbit) and the almanac (the satellite orbital and

clock information). Satellite messages are broadcasted and the reception of messages from four of more satellites will allow a receiver to calculate its position. This position calculation is based on trilateration. The receiver measures the times of arrival of the satellite signals, converts them into distances (pseudoranges), and then calculates its position as well as its clock offset with respect to the satellite clocks.

A GPS signal spoofing attack is a physical-layer attack in which an attacker transmits specially crafted radio signals that are identical to authentic satellite signals. Civilian GPS is easily vulnerable to signal spoofing attacks. This is due to the lack of any signal authentication and the publicly known spreading codes for each satellite, modulation schemes, and data structure. In a signal spoofing attack, the objective of an attacker may be to force a target receiver to (i) compute an incorrect position, (ii) compute an incorrect time or (iii) disrupt the receiver. Due to the low power of the legitimate satellite signal at the receiver, the attacker's spoofing signals can trivially overshadow the authentic signals. In a spoofing attack, the GPS receiver typically locks (acquires and tracks) onto the stronger, attacker's signal, thus ignoring the satellite signals.

An attacker can influence the receiver's position and time estimate in two ways: (i) by manipulating the contents of the navigation messages (e.g., the location of satellites, navigation message transmission time) and/or (ii) by modifying the arrival time of the navigation messages. The attacker can manipulate the receiver time of arrival by temporally shifting the navigation message signals while transmitting the spoofing signals. We can classify spoofing attacks based on how synchronous (in time) and consistent (with respect to the contents of the navigation messages) the spoofing signals are in comparison to the legitimate GPS signals currently being received at the receiver's true location.

Non-Coherent and Modified Message Contents: In this type of attack, the attacker's signals are both unsynchronised and contain different navigation message data in comparison to the authentic signals. Attackers who use GPS signal generators to execute the spoofing attack typically fall under this category. An attacker with a little know-how can execute a spoofing attack using these simulators due to their low complexity, portability and ease of use. Some advanced GPS signal generators are even capable of recording and replaying signals, however not in real-time. In other words, the attacker uses the simulator to record at one particular time in a given location and later replays it. Since they are replayed at a later time, the attacker's signals are not coherent and contain different navigation message data than the legitimate signals currently being received.

Non-Coherent but Unmodified Message Contents: In this type of attack, the navigation message contents of the transmitted spoofing signals are identical to the legitimate GPS signals currently being received. However, the attacker temporally shifts the spoofing signal thereby manipulating the spoofing signal time of arrival at the target receiver. For example, attackers capable of real-time recording and replaying of GPS signals fall under this category as they will have the same navigation contents as that of the legitimate GPS signals, however shifted in time. The location or time offset caused by such an attack on the target receiver depends on the time delay introduced both by the attacker and due to the propagation time of the relayed signal. The attacker can precompute these delays and successfully spoof a receiver to a desired location.

Coherent but Modified Message Contents: The attacker generates spoofing signals that are synchronised to the authentic GPS signals. However, the contents of the navigation messages are not the same as that of the currently seen authentic signals. For instance, phase-coherent signal synthesisers are capable of generating spoofing signals with the same code phase

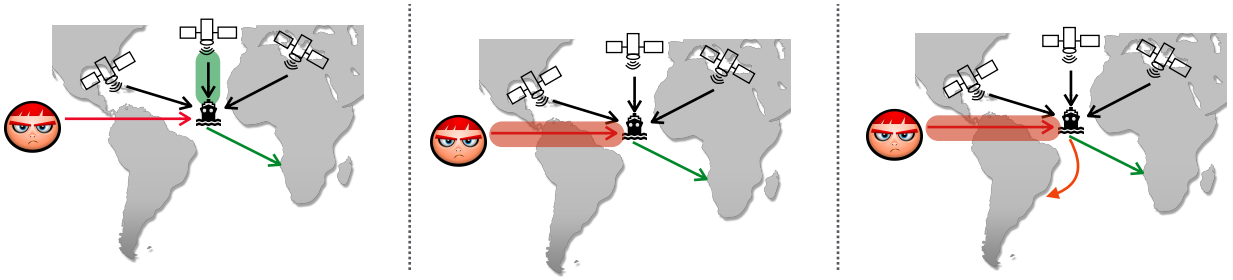


Figure 22.3: Seamless takeover attack on GPS. The spoofing aligns its signal with the legitimate signal and slowly increase the transmit power. Once receiver locks on to attacker's signal, he starts to manipulate it.

as the legitimate GPS signal that the target receiver is currently locked on to. Additionally, the attacker modifies the contents of the navigation message in real-time (and with minimal delay) and replays it to the target receiver. A variety of commercial GPS receivers were shown to be vulnerable to this attack and in some cases, it even caused permanent damage to the receivers.

Coherent and Unmodified Message Contents: Here, the attacker does not modify the contents of the navigation message and is completely synchronised to the authentic GPS signals. Even though the receiver locks on to the attacker's spoofing signals (due to the higher power), there is no change in the location or time computed by the target receiver. Therefore, this is not an attack in itself but is an important first step in executing the seamless takeover attack.

The seamless takeover attack is considered one of the strongest attacks in literature. In a majority of applications, the target receiver is already locked on to the legitimate GPS satellite signals. The main steps are highlighted in Figure 22.3. The goal of an attacker is to force the receiver to stop tracking the authentic GPS signals and lock onto the spoofing signals without causing any signal disruption or data loss. This is because the target receiver can potentially detect the attack based on the abrupt loss of GPS signal. In a seamless takeover attack, first, the attacker transmits spoofing signals that are synchronised with the legitimate satellite signals and are at a power level lower than the received satellite signals. The receiver is still locked on to the legitimate satellite signals due to the higher power and hence there is no change in the ships route. The attacker then gradually increases the power of the spoofing signals until the target receiver stops tracking the authentic signal and locks on to the spoofing signals. Note that during this takeover, the receiver does not see any loss of lock, in other words, the takeover was seamless. Even though the target receiver is now locked on to the attacker, there is still no change in the route as the spoofing signals are both coherent with the legitimate satellite signals as well as there is no modification to the contents of the navigation message itself. Now, the attacker begins to manipulate the spoofing signal such that the receiver computes a false location and begins to alter its course. The attacker can either slowly introduce a temporal shift from the legitimate signals or directly manipulate the navigation message contents to slowly deviate the course of the ship to a hostile destination.

If an attacker controls all the signals that arrive at the receiver's antenna(s) the receiver cannot detect spoofing. However, if the attack is remote, and the attacker cannot fully control the signals at the receiver, anomaly detection techniques can be used to detect spoofing. In particular, Automatic Gain Control (AGC) values, Received Signal Strength (RSS) from individual satellites, carrier phase values, estimated noise floor levels, number of visible satellites all can be used to detect spoofing. Particularly interesting are techniques based

on tracking and analysis of autocorrelation peaks that are used for the detection of GNSS signals. Distortion, the number and the behaviour over time of these peaks can be used to detect even the most sophisticated seamless takeover attacks.

The detection of GNSS spoofing can be improved if spoofing signals are simultaneously received by several receivers. This can be used for the detection of spoofing as well as for spoofer localisation. If the receivers know their mutual distances (e.g., are placed at fixed distances), the spoofer needs to preserve those distances when performing the spoofing attack. When a single spoofer broadcasts its signals, it will result in all receivers being spoofed to the same position, therefore enabling detection. This basic detection technique can be generalised to several receivers, allowing even the detection of distributed spoofers.

Finally, GNSS spoofing can be made harder through the authentication and hiding of GNSS signals. Although currently civilian GNSS systems do not support authentication, digital signatures as well as hash-based signatures such as TESLA can be added to prevent the attacker from generating GNSS signals. This would, however, not prevent all spoofing attacks since the attacker can still selectively delay navigation messages and therefore modify the computed position. This attack can be prevented by the use of spreading with delayed key disclosure. Even this approach still does not fully prevent against spoofing by broadband receivers that are able to relay full GNSS frequency band between locations.

Military GPS signals are authenticated, and try to achieve low-probability of intercept as well as jamming resilience via the use of secret spreading codes. This approach prevents some of the spoofing attacks, but still fails to fully prevent record-and-relay attacks. In addition, this approach does not scale well since secret spreading codes need to be distributed to all intended receivers, increasing the likelihood of their leakage and reducing usability.

In conclusion, although newly proposed and deployed countermeasures make it more difficult for the attacker to spoof GNS systems like GPS, currently no measure fully prevents spoofing under strong attacker models. This is an area of active research.

CONCLUSION

As we have shown in this knowledge area, the wireless physical layer presents both challenges and opportunities. Challenges typically come from the broadcast nature of wireless communication and from it not being protected against confidentiality and integrity violations. Physical layer is typically application agnostic. Opportunities stem from the stochastic nature of the channel as well as from its robustness to fine-grained manipulations. Under different attacker models, physical layer can support both highly usable and secure solutions.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

The table below lists the reference material that serves as the basis for for this chapter and explains how it relates to the different topics. Whenever possible, references are further divided into sub-topics.

Topic	Key references	Other references
22.1 Physical Layer Schemes for Confidentiality, Integrity and Access Control		
22.1.1 Key Establishment based on Channel Reciprocity	[2164, 2165, 2166]	[2200, 2201, 2202, 2203, 2204, 2205]
22.1.2 MIMO-supported Approaches: Orthogonal Blinding, Zero-Forcing	[2164, 2167]	[2206, 2207, 2208, 2209]
22.1.3 Secrecy Capacity	[2169, 2170, 2171, 2172]	[2210, 2211, 2212, 2213]
22.1.4 Friendly Jamming	[1990, 2164]	[2214, 2215, 2216, 2217]
22.1.5 Using Physical Layer to Protect Data Integrity	[2164, 2168]	[2218]
22.1.6 Low probability of intercept and Covert Communication	[2164]	[2219, 2220, 2221]
22.2 Jamming and Jamming-Resilient Communication	[2173, 2174]	[2222, 2223, 2224, 2225]
22.3 Physical-Layer Identification	[2177]	[2226, 2227, 2228]
22.4 Distance Bounding and Secure Positioning		
22.4.1 Distance Bounding Protocols	[2178, 2179, 2180]	[2186, 2229, 2230, 2231, 2232, 2233]
22.4.2 Distance Measurement Techniques	[2178, 2182]	[2234, 2235, 2236, 2237]
22.4.3 Physical Layer Attacks on Secure Distance Measurement	[2178][2181, 2182, 2183]	[2187, 2238, 2239, 2240, 2241]
22.4.4 Secure Positioning	[2184]	[2242, 2243, 2244, 2245, 2246]
22.5 Compromising Emanations and Sensor Spoofing		
22.5.1 Compromising Emanations	[2188, 2189, 2190, 2191, 2192]	[2247, 2248, 2249, 2250, 2251]
22.5.2 Sensor Compromise	[1953, 1998, 1999, 2193, 2195]	[1951, 1952, 2252, 2253, 2254]
22.6 Physical Layer Security of Selected Communication Technologies		
22.6.1 Near-field communication (NFC)	[2196]	[2255, 2256, 2257]
22.6.2 Air Traffic Communication Networks	[2197]	[2106, 2258, 2259, 2260]
22.6.3 Cellular Networks	[2198]	[2261, 2262, 2263]
22.6.4 GNSS Security and Spoofing Attacks	[2199]	[2264, 2265, 2266, 2267, 2268]

ACKNOWLEDGEMENTS

The author would like to specially thank Marc Roeschlin for his valuable input. Thanks to Aanjhan Ranganathan, Davide Zanetti, Boris Danev, Christina Popper, Kasper Rasmussen and Nils Tippenhauer for allowing the reproduction of selected text and figures from their publications within this document.

VI Appendix

Bibliography

[1] A. Rashid, G. Danezis, H. Chivers, E. Lupu, A. Martin, M. Lewis, and C. Peersman, "Scoping the cyber security body of knowledge," *IEEE Security & Privacy*, vol. 16, no. 3, pp. 96–102, 2018. [Online]. Available: https://doi.org/10.1109/MSP.2018.2701150

[2] J. Hallett, R. Larson, and A. Rashid, "Mirror, mirror, on the wall: What are we teaching them all? Characterising the focus of cybersecurity curricular frameworks," in *2018 USENIX Workshop on Advances in Security Education, ASE 2018, Baltimore, MD, USA, August 13, 2018.*, 2018. [Online]. Available: https://www.usenix.org/conference/ase18/presentation/hallett

[3] P. Bourque, R. E. Fairley *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[4] HM Government, UK, "National cyber security strategy 2016–2021," 2016. [Online]. Available: https://www.gov.uk/government/publications/national-cyber-security-strategy-2016-to-2021

[5] ETSI/CEN/CENELEC Cybersecurity Coordination Group (CSCG) and ENISA, "Definition of Cybersecurity gaps and overlaps in standardisation," ENISA, Report, Dec. 2015. [Online]. Available: https://www.enisa.europa.eu/publications/definition-of-cybersecurity

[6] ISO/IEC, "Information technology – security techniques – information security management systems – overview and vocabulary," ISO/IEC, – 27000:2018, 2018.

[7] D. Gollmann, *Computer Security*, 3rd ed. Wiley, 2011.

[8] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975. [Online]. Available: https://doi.org/10.1109/PROC.1975.9939

[9] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. 9, 1883.

[10] R. Ross, M. McEvilley, and J. C. Oren, "Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems," NIST, Tech. Rep. NIST.SP.800-160 Volume 1, Nov. 2016. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-160v1

[11] J. Reason, *The human contribution: unsafe acts, accidents and heroic recoveries*. CRC Press, 2008.

[12] W. Pieters and A. van Cleeff, "The precautionary principle in a world of digital dependencies," *IEEE Computer*, vol. 42, no. 6, pp. 50–56, 2009.

[13] R. Anderson and T. Moore, "The economics of information security," *Science*, vol. 314, no. 5799, pp. 610—-613, 2006.

[14] R. Anderson, "Why information security is hard-an economic perspective," in *17th Annual Computer Security Applications Conference (ACSAC 2001), 11-14 December 2001, New Orleans, Louisiana, USA*, 2001, pp. 358–365.

[15] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko, "Botcoin: Monetizing stolen cycles," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[16] P. Pearce, V. Dave, C. Grier, K. Levchenko, S. Guha, D. McCoy, V. Paxson, S. Savage, and G. M. Voelker, "Characterizing large-scale click fraud in ZeroAccess," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 141–152.

[17] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics: an empirical analysis of spam marketing conversion," in *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, 2008, pp. 3–14.

[18] C. Herley and D. A. F. Florêncio, "Nobody sells gold for the price of silver: Dishonesty, uncertainty and the underground economy," in *8th Annual Workshop on the Economics of Information Security, WEIS 2009, University College London, England, UK, June 24-25, 2009*, 2009.

[19] E. van de Sandt, "Deviant security: The technical computer security practices of cyber criminals," Ph.D. dissertation, Department of Computer Science, University of Bristol, UK, 2019.

[20] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "SoK: P2PWNED-modeling and evaluating the resilience of peer-to-peer botnets," in *IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 97–111.

[21] H. Berghel, "Hiding data, forensics, and anti-forensics," *Commun. ACM*, vol. 50, no. 4, pp. 15–20, 2007.

[22] O. Renn, "The role of risk perception for risk management," *Reliability Engineering & System Safety*, vol. 59, no. 1, pp. 49 – 62, 1998, risk Perception Versus Risk Analysis. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832097001191

[23] P. Slovic, "Perception of risk," *Science*, vol. 236, no. 4799, pp. 280–285, 1987. [Online]. Available: https://science.sciencemag.org/content/236/4799/280

[24] O. Renn, *Risk Governance: Coping With Uncertainty in a Complex World*. Springer, 2008.

[25] D. Hillson, "Extending the risk process to manage opportunities," *International Journal of Project Management*, vol. 20, no. 3, pp. 235 – 240, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0263786301000746

[26] B. Schneier, *Beyond Fear: Thinking Sensibly About Security in an Uncertain World*. Berlin, Heidelberg: Springer-Verlag, 2003.

[27] "ALARP "at a glance"," accessed: 2019-06-13. [Online]. Available: http://www.hse.gov.uk/risk/theory/alarpglance.htm

[28] "NIST special publication 800-39 - managing information security risk," accessed: 2019-04-15. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf

[29] "ISO.IEC 31000:2018 - risk management - guidelines," accessed: 2019-09-06. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso:31000:ed-2:v1:en

[30] "NIS directive," accessed: 2019-04-15. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2016/1148/oj

[31] NCSC, "NIS directive principles," accessed: 2019-04-15. [Online]. Available: https://www.ncsc.gov.uk/guidance/table-view-principles-and-related-guidance

[32] W. Brun and K. H. Teigen, "Verbal probabilities: Ambiguous, context-dependent, or both?" *Organizational Behavior and Human Decision Processes*, vol. 41, no. 3, pp. 390 – 404, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0749597888900362

[33] P. Slovic, B. Fischhoff, and S. Lichtenstein, *Rating the Risks*. Boston, MA: Springer US, 1981, pp. 193–217. [Online]. Available: https://doi.org/10.1007/978-1-4899-2168-0_17

[34] A. O'Hagan, C. E. Buck, A. Daneshkhah, J. R. Eiser, P. H. Garthwaite, D. J. Jenkinson, J. E. Oakley, and T. Rakow, *Uncertain Judgements: Eliciting Expert Probabilities*. Chichester: John Wiley, 2006. [Online]. Available: http://oro.open.ac.uk/17948/

[35] NCSC, "Risk management guidance," accessed: 2019-04-15. [Online]. Available: https://www.ncsc.gov.uk/collection/risk-management-collection?curPage=/collection/risk-management-collection/essential-topics/introduction-risk-management-cyber-security-guidance

[36] S. Liu, "The internet of things (IoT)* units installed base by category from 2014 to 2020 (in billions)," Statista, Tech. Rep., 01 2002. [Online]. Available: https://www.statista.com/statistics/370350/internet-of-things-installed-base-by-category

[37] "UK government report - securing cyber resilience in health and care: October 2018 update," accessed: 2019-04-15. [Online]. Available: https://www.gov.uk/government/publications/securing-cyber-resilience-in-health-and-care-october-2018-update

[38] "Global cybersecurity index," accessed: 2019-04-15. [Online]. Available: https://www.itu.int/en/ITU-D/Cybersecurity/Pages/global-cybersecurity-index.aspx

[39] "Cyber readiness index," accessed: 2019-04-15. [Online]. Available: http://www.potomacinstitute.org/academic-centers/cyber-readiness-index

[40] E. Millstone, P. Van Zwanenberg, C. Marris, L. Levidow, and H. Torgersen, "Science in trade disputes related to potential risks: comparative case studies." *IPTS technical report series EUR 21301 EN, European Commission Joint Research Centre / IPTS Institute for Prospective Technological Studies, Brussels/Luxembourg*, 01 2004.

[41] B. Rohrmann and O. Renn, "Risk perception research - an introduction," *Cross-Cultural Risk Perception: A Survey of Empirical Studies*, pp. 11–54, 01 2000.

[42] B. Fischhoff, S. Lichtenstein, P. Slovic, S. Derby, and R. Keeney, *Acceptable risk*. Cambridge University Press, New York, NY, 1981.

[43] A. Sasse and I. Flechais, *Usable Security: Why Do We Need It? How Do We Get It? In: Cranor, LF and Garfinkel, S, (eds.) Security and Usability: Designing secure systems that people can use.* Sebastopol, US: O'Reilly, 2005.

[44] D. Weirich and M. A. Sasse, "Pretty good persuasion: A first step towards effective password security in the real world," in *Proceedings of the 2001 Workshop on New Security Paradigms*, ser. NSPW '01. New York, NY, USA: ACM, 2001, pp. 137–143. [Online]. Available: http://doi.acm.org/10.1145/508171.508195

[45] W. Leiss, "OECD guidance document for risk communication for chemical risk management," 01 2002.

[46] S. Dekker, *Just culture: Balancing safety and accountability, 2nd edition*. CRC Press, 01 2012.

[47] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Pearson Education, 01 2007.

[48] M. A. Sasse and A. Rashid, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Human Factors, version 1.0.1. [Online]. Available: https://www.cybok.org/

[49] IRGC, "Introduction to the IRGC risk governance framework, revised version. Lausanne: EPFL international risk governance center," 01 2017.

[50] R. E. Lundgren and A. H. McMakin, *Principles of Risk Communication*. IEEE, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6582032

[51] L. Cox, "What's wrong with risk matrices?" *Risk analysis : an official publication of the Society for Risk Analysis*, vol. 28, pp. 497–512, 05 2008.

[52] J. Rasmussen, "Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models," *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 257–266, 1983.

[53] Joint Task Force Transformation Initiative, "Security and privacy controls for federal information systems and organizations," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-53, Revision 4, 2014.

[54] "ISO.IEC 27005:2018 information technology – security techniques – information security risk management," accessed: 2019-04-15. [Online]. Available: https://www.iso.org/standard/75281.html

[55] NCSC, "Component-driven risk assessment," accessed: 2019-04-15. [Online]. Available: https://www.ncsc.gov.uk/collection/risk-management-collection?curPage=/collection/risk-management-collection/component-system-driven-approaches/understanding-component-driven-risk-management

[56] ——, "System-driven risk assessment," accessed: 2019-04-15. [Online]. Available: https://www.ncsc.gov.uk/collection/risk-management-collection?curPage=/collection/risk-management-collection/component-system-driven-approaches/understanding-system-driven-risk-management

[57] D. S. Herrmann, *Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI*, 1st ed. Boston, MA, USA: Auerbach Publications, 2007.

[58] A. Shostack, *Threat Modeling: Designing for Security*, 1st ed. Wiley Publishing, 2014.

[59] S. Rass, "On game-theoretic risk management (part three) - modeling and applications," 2017.

[60] J. Jones, *An Introduction to Factor Analysis of Information Risk (FAIR)*. Risk Management Insight, 2005.

[61] "What is open fair?" accessed: 2019-06-13. [Online]. Available: https://blog.opengroup.org/2017/01/24/what-is-open-fair/

[62] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[63] "Risk assesment method comparison," accessed: 2019-06-13. [Online]. Available: https://www.sisainfosec.com/americas/blogs/comparison-between-iso-27005-octave-nist-sp-800-30/

[64] N. Leveson, *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.

[65] "TOGAF," accessed: 2019-04-15. [Online]. Available: https://www.opengroup.org/togaf

[66] Open Group, "TOGAF - risk assessment," accessed: 2019-06-13. [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap27.html

[67] "Dependency modeling," accessed: 2019-04-15. [Online]. Available: https://publications.opengroup.org/c133

[68] P. Burnap, Y. Cherdantseva, A. Blyth, P. Eden, K. Jones, H. Soulsby, and K. Stoddart, "Determining and sharing risk data in distributed interdependent systems," *Computer*, vol. 50, no. 04, pp. 72–79, apr 2017.

[69] "SABSA," accessed: 2019-04-15. [Online]. Available: https://sabsa.org/

[70] A. Jones and D. Ashenden, *Risk Management for Computer Security: Protecting Your Network & Information Assets*. Newton, MA, USA: Butterworth-Heinemann, 2005.

[71] T. Atlantic, "The Obama doctrine," accessed: 2019-06-13. [Online]. Available: https://www.theatlantic.com/magazine/archive/2016/04/the-obama-doctrine/471525/

[72] A. Shostack, ""Think like an attacker" is an opt-in mistake," accessed: 2019-06-13. [Online]. Available: https://adam.shostack.org/blog/2016/04/think-like-an-attacker-is-an-opt-in-mistake/

[73] H. Debar, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Security Operations & Inicident Management, version 1.0.2. [Online]. Available: https://www.cybok.org/

[74] "ISO.IEC 27035 - information security incident management," accessed: 2019-04-15. [Online]. Available: https://www.iso.org/standard/62071.html?browse=tc

[75] NCSC, "10 steps to incident management," accessed: 2019-04-15. [Online]. Available: https://www.ncsc.gov.uk/collection/10-steps-to-cyber-security?curPage=/collection/10-steps-to-cyber-security/the-10-steps/incident-management

[76] Y. Hou, J. Such, and A. Rashid, "Understanding security requirements for industrial control system supply chains," in *Proceedings of the 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems*, ser. SEsCPS '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 50–53. [Online]. Available: https://doi.org/10.1109/SEsCPS.2019.00016

[77] B. A. Sabbagh and S. Kowalski, "A socio-technical framework for threat modeling a software supply chain," *IEEE Security Privacy*, vol. 13, no. 4, pp. 30–39, July 2015.

[78] J. Goldsmith and T. Wu, *Who Controls the Internet? Illusions of a Borderless World*. New York, NY: Oxford University Press, 2006.

[79] O. W. Holmes, *The Common Law*. Boston: Little, Brown and Company, 1881.

[80] J. P. Barlow. (1996) A Declaration of the Independence of Cyberspace. [Online]. Available: https://www.eff.org/cyberspace-independence

[81] L. Lessig, *Code: Version 2.0*. Basic Books, 2006.

[82] C. Millard and R. Carolina, "Commercial transactions on the global information infrastructure: A European perspective," *The John Marshall Journal of Information Technology & Privacy Law*, vol. 14, no. 2, pp. 269–301, 1996.

[83] D. R. Johnson and D. Post, "Law and Borders–The Rise of Law in Cyberspace," *Stanford Law Review*, vol. 48, pp. 1367–1402, 1996.

[84] C. Reed, *Internet law: text and materials*, 2nd ed. Cambridge University Press, 2004.

[85] "Report of the Group of Governmental Experts on Developments in the Field of Information and Telecommunications in the Context of International Security," Report A/68/98, United Nations General Assembly, 2013.

[86] "Group of Governmental Experts on Developments in the Field of Information and Telecommunications in the Context of International Security," Report A/70/174, United Nations General Assembly, 2015.

[87] M. N. Schmitt, Ed., *Tallinn Manual 2.0 on the International Law Applicable to Cyber Warfare*. Cambridge University Press, 2017.

[88] D. van der Linden and A. Rashid, "The Effect of Software Warranties on Cybersecurity," *ACM SIGSOFT Software Engineering Notes*, vol. 43, no. 4, pp. 31–35, 2018.

[89] "Report of the High Commissioner for Human Rights on the Right to Privacy in the Digital Age," A/HRC/39/29, United Nations, 2018.

[90] D. Rowland, U. Kohl, and A. Charlesworth, *Information Technology Law*, 5th ed. London: Routledge, 2017.

[91] A. Murray, *Information Technology Law: the Law and Society*, 3rd ed. Oxford University Press, 2016.

[92] I. Walden, *Computer Crimes and Digital Investigations*, 2nd ed. Oxford: Oxford University Press, 2016.

[93] T. J. Holt, A. M. Bossler, and K. C. Seigfried-Spellar, *Cybercrime and Digital Forensics: An Introduction*, 2nd ed. Routledge: Taylor and Francis, 2018.

[94] J. Clough, *Principles of Cybercrime*, 2nd ed. Cambridge University Press, 2015.

[95] "American Banana Co. v. United Fruit Co." 213 U.S. 347 (US S.Ct), 1909.

[96] "United States v. Alcoa," 148 F.2d 416 (2d Cir. *sitting by designation of the US S.Ct*), 1945.

[97] "Wood Pulp," 1988 ECR 05193, 1988.

[98] J. J. Friedberg, "The convergence of law in an Era of Political Integration: The wood pulp case and the Alcoa effects doctrine," *University of Pittsburgh Law Review*, vol. 52, pp. 289–326, 1991.

[99] "Foreign Corrupt Practices Act," US Statutes, vol. 91, pp. 1494-1500, 1977.

[100] "Directive 2011/92/EU of the European Parliament and of the Council of 13 December 2011 on combating the sexual abuse and sexual exploitation of children and child pornography, and replacing Council Framework Decision 2004/68/JHA," *Official Journal of the European Union*, vol. L 335, pp. 1–14, 2011.

[101] "PROTECT Act," US Statutes, vol. 117, pp. 649-695, 2003.

[102] "LICRA v. Yahoo! Inc & Yahoo France," (Tribunal de Grande Instance de Paris, 22 May 2000), *affirmed in* LICRA & UEJF v. Yahoo! Inc & Yahoo France (Tribunal de Grande Instance de Paris, 20 November 2000), 2000.

[103] (2001) Convention on Cybercrime. European Treaty Series 185. Council of Europe. [Online]. Available: https://www.coe.int/en/web/conventions/full-list/-/conventions/rms/0900001680081561

[104] "Directive 2013/40/EU of the European Parliament and of the Council of 12 August 2013 on attacks against information systems and replacing Council Framework Decision 2005/222/JHA," *Official Journal of the European Union*, vol. L 218, pp. 8–14, 2013.

[105] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC," *Official Journal of the European Union*, vol. L 119, pp. 1–88, 2016.

[106] "Google Spain SL, Google Inc. v. AEPD, Mario Costeja González," C-131/12, ECLI:EU:C:2014:317, 2014.

[107] "Guidelines 3/2018 on the territorial scope of the GDPR (Article 3) – Version for public consultation," European Data Protection Board, 2018.

[108] K. Kopel, "Operation Seizing Our Sites: How the Federal Government is Taking Domain Names Without Prior Notice," *Berkeley Technology Law Journal*, vol. 28, pp. 859–900, 2013.

[109] J. Mellyn, "Reach out and Touch Someone: The Growing Use of Domain Name Seizure as a Vehicle for the Extraterritorial Enforcement of U.S. Law," *Georgetown Journal of International Law*, vol. 42, pp. 1241–1264, 2011.

[110] A. M. Froomkin, "When We Say US(tm), We Mean It!" *Houston Law Review*, vol. 41, no. 3, pp. 839–884, 2004.

[111] "Libyan Arab Foreign Bank v. Bankers Trust Co," [1989] QB 728, 1989.

[112] R. Cranston, E. Avgouleas, K. Van Zwieten, and T. Van Sante, *Principles of Banking Law*, 3rd ed. Oxford: Oxford University Press, 2018.

[113] R. McLaughlin, "Authorizations for maritime law enforcement operations," *International Review of the Red Cross*, vol. 98, no. 2, pp. 465–490, 2016.

[114] "Twentieth Century Fox and others v. British Telecommunications plc," [2011] EWHC 1981, 2011.

[115] P. M. Connorton, "Tracking terrorist financing through SWIFT: when US subpoenas and foreign privacy law collide," *Fordham Law Review*, vol. 76, pp. 283–322, 2007.

[116] "In the Matter of a Warrant to Search a Certain E-Mail Account Controlled and Maintained by Microsoft Corporation," 829 F.3d 197, 2d Cir, 2016.

[117] "Stored Communications Act," *codified at* 18 U.S.C 2701 et seq, 1986.

[118] "CASE NOTE: Privacy — Stored Communications Act — Second Circuit Holds That The Government Cannot Compel An Internet Service Provider To Produce Information Stored Overseas. — Microsoft Corp. v. United States, 829 F.3d 197 (2d Cir. 2016)," *Harvard Law Review*, vol. 130, pp. 769–776, 2016.

[119] "United States v. Microsoft Corporation," No. 17-2; 584 U.S. ___, 138 S.Ct 1186 (US S.Ct), 2018.

[120] "Comprehensive Study on Cybercrime (Draft)," United Nations Office on Drugs and Crime, February 2013.

[121] "T-CY Guidance Note # 3: Transborder access to data (Article 32)," T-CY (2013)7, Council of Europe, Cybercrime Convention Committee (T-CY), 3 December 2014.

[122] "Transborder access and jurisdiction: What are the options?" T-CY (2012)3 (provisional), Council of Europe, Cybercrime Convention Committee (T-CY), Ad-hoc Sub-group on Jurisdiction and Transborder Access to Data, 6 December 2012.

[123] "ECRI report on the Russian Federation (fifth monitoring cycle)," European Commission against Racism and Intolerance (ECRI), 5 March 2019. [Online]. Available: https://rm.coe.int/fifth-report-on-the-russian-federation/1680934a91

[124] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Special Publication 800-145, NIST, 2011.

[125] (2018) Cloud security guidance: 2.1 Physical location and legal jurisdiction. National Cyber Security Centre (UK). [Online]. Available: https://www.ncsc.gov.uk/collection/cloudsecurity/implementing-the-cloud-security-principles/asset-protection-and-resilience#physical

[126] C. Millard, "Forced Localization of Cloud Services: Is Privacy the Real Driver?" *IEEE Cloud Computing*, vol. 2, no. 2, pp. 10–14, 2015.

[127] A. Chander and U. P. Lê, "Data Nationalism," *Emory LJ*, vol. 64, pp. 677–739, 2014.

[128] A. Savelyev, "Russia's new personal data localization regulations: A step forward or a self-imposed sanction?" *Computer Law & Security Review*, vol. 32, no. 1, pp. 128–145, 2016.

[129] S. Livingston and G. Greenleaf, "Data localisation in China and other APEC jurisdictions," *Privacy Laws & Business International Report*, vol. 143, pp. 22–26, 2016.

[130] A. D. Mitchell and J. Hepburn, "Don't Fence Me In: Reforming Trade and Investment Law to Better Facilitate Cross-Border Data Transfer," *Yale JL & Tech.*, vol. 19, pp. 182–237, 2017.

[131] "Regulation (EU) 2018/1807 of the European Parliament and of the Council of 14 November 2018 on a framework for the free flow of non-personal data in the European Union," *Official Journal of the European Union*, vol. L 303, pp. 59–68, 2018.

[132] S. D. Warren and L. D. Brandeis, "The Right to Privacy," *Harvard Law Review*, vol. 4, no. 5, pp. 193–220, 1890.

[133] "Universal Declaration of Human Rights," United Nations, 1948.

[134] "Charter of Fundamental Rights of the European Union," *Official Journal of the European Union*, vol. C 303, pp. 1–16, 2007.

[135] "US Constitution, Amendment IV," 1791.

[136] "Olmstead v. United States," 277 U.S. 438 (US S.Ct), 1928.

[137] "Katz v. United States," 389 U.S. 347 (US S.Ct), 1967.

[138] "Halford v. The United Kingdom," (20605/92) ([1997] 24 EHRR 523, 1997.

[139] (2019) Guide to International Law and Surveillance (2.0). Privacy International. [Online]. Available: https://privacyinternational.org/sites/default/files/2019-04/Guide%20to%20International%20Law%20and%20Surveillance%202.0.pdf

[140] Special Representative of the Secretary-General on the issue of human rights and transnational corporations and other business enterprises, "Guiding Principles on Business and Human Rights: Implementing the United Nations "Protect, Respect and Remedy" Framework," United Nations, 2011.

[141] "Smith v. Maryland," 442 U.S. 735 (US S.Ct), 1979.

[142] A. M. Rutkowski, "International Signals Intelligence Law: Provisions and History," *Lawfare Research Paper Series*, vol. 42, pp. 933–988, 2017.

[143] N. Jupillat, "From the Cuckoo's Egg to Global Surveillance: Cyber Espionage that Becomes Prohibited Intervention," *North Carolina Journal of International Law*, vol. 42, pp. 933–988, 2016.

[144] J. Hurwitz, "Encryption$^{\text{Congress}}$ mod (Apple+ CALEA)," *Harvard Journal of Law & Technology*, vol. 30, pp. 355–424, 2017.

[145] HIPCAR, "Interception of Communications: Model Policy Guidelines & Legislative Texts," International Telecommunications Union, 2012.

[146] "Council Resolution of 17 January 1995 on the lawful interception of telecommunications," *Official Journal of the European Communities*, vol. C 329, pp. 1–6, 1995.

[147] (2019) Lawful Interception. ETSI. [Online]. Available: https://www.etsi.org/technologies/lawfulinterception

[148] L. Sacharoff, "Unlocking the Fifth Amendment: Passwords and Encrypted Devices," *Fordham Law Revue*, vol. 87, pp. 203–251, 2018.

[149] M. J. Weber, "Warning-Weak Password: The Courts' Indecipherable Approach to Encryption and the Fifth Amendment," *University of Illinois Journal of Law*, pp. 455–486, 2016.

[150] "Investigatory Powers Act 2016," United Kingdom, 2016.

[151] "Mapp v. Ohio," 367 U.S. 643 (US S.Ct), 1961.

[152] (2014) R -v- Coulson and others: Sentencing Remarks of Mr Justice Saunders. [Online]. Available: https://www.judiciary.uk/wp-content/uploads/2014/07/sentencing-remarks-mr-jsaunders-r-v-coulson-others.pdf

[153] "Handbook on European data protection law," European Union Agency for Fundamental Rights, 2018.

[154] (2019) GDPR: guidelines, recommendations, best practices. European Data Protection Board. [Online]. Available: https://edpb.europa.eu/our-work-tools/general-guidance/gdpr-guidelinesrecommendations-best-practices_en

[155] R. Carolina. (2017) Why the EU Has Issued Relatively Few Data Protection Adequacy Determinations? A Reply. Lawfare. [Online]. Available: https://www.lawfareblog.com/why-eu-has-issued-relatively-few-data-protection-adequacydeterminations-reply

[156] "Directive (EU) 2016/680 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data by competent authorities for the purposes of the prevention, investigation, detection or prosecution of criminal offences or the execution of criminal penalties, and on the free movement of such data, and repealing Council Framework Decision 2008/977/JHA," *Official Journal of the European Union*, vol. L 119, pp. 89–131, 2016.

[157] (2018) D2.2: Review report on Directive 2016/680 aimed at the judiciary. INtroduction of the data protection reFORM to the judicial system (INFORM). [Online]. Available: http://informproject.eu/wp-content/uploads/2018/05/D2.2.pdf

[158] (2018) D2.5 Review report on Directive (EU) 2016/680 aimed at the legal practitioners. INtroduction of the data protection reFORM to the judicial system (INFORM). [Online]. Available: http://informproject.eu/wp-content/uploads/2018/05/D2.5.pdf.

[159] "Breyer v. Germany," Case C-582/14, 2016.

[160] M. Elliot, K. O'Hara, C. Raab, C. M. O'Keefe, E. Mackey, C. Dibben, H. Gowans, K. Purdam, and K. McCullagh, "Functional anonymisation: Personal data and the data environment," *Computer Law & Security Review*, vol. 34, pp. 204–221, 2018.

[161] S. Stalla-Bourdillon and A. Knight, "Anonymous Data v. Personal Data - False Debate: An EU Perspective on Anonymization, Pseudonymization and Personal Data," *Wisconsin International Law Journal*, vol. 34, no. 2, pp. 284–322, 2016.

[162] "ISO/IEC 29100:2011 Information technology - Security techniques - Privacy framework," 2011.

[163] "Guide to Protecting the Confidentiality of Personally Identifiable Information," Special Publication 800-122, NIST, 2010.

[164] P. M. Schwartz and D. J. Solove, "Reconciling Personal Information in the United States and European Union," *California Law Review*, vol. 102, no. 4, pp. 877–916, 2014.

[165] M. J. Wiebe, "Applying the Video Privacy Protection Act to Modern Technology [Ellis v. Cartoon Network, Inc., 803 F. 3d 1251 (11th Cir. 2015)]," *Washburn Law Journal*, vol. 57, pp. 169–202, 2018.

[166] "In re Nickelodeon Consumer Privacy Litig." 827 F.3d 262, 278 (3d Cir. 2016), *cert denied*, 137 S.Ct. 624 (2017), 2016.

[167] "Eichenberger v. ESPN, Inc." 876 F.3d 979 (9th Cir.), 2017.

[168] (2019) Guide to Data Protection. Information Commissioner's Office (UK). [Online]. Available: https://ico.org.uk/for-organisations/guide-to-data-protection

[169] The INFORM Project. [Online]. Available: http://informproject.eu

[170] "UK Data Protection Act 2018," United Kingdom, 2018.

[171] "Schrems v. Data Protection Commissioner," Case C 362/14, ECLI:EU:C:2015:650, 2015.

[172] "Guidelines 2/2018 on derogations of Article 49 under Regulation 2016/679," European Data Protection Board, 2018.

[173] E. Preston and P. Turner, "The Global Rise of a Duty to Disclose Information Security Breaches," *The John Marshall Journal of Information Technology & Privacy Law*, vol. 22, pp. 457–492, 2004.

[174] N. Robinson, V. Horvath, J. Cave, A. P. Roosendaal, and M. Klaver, "Data and security breaches and cyber-security strategies in the EU and its international counterparts," European Union, 2013.

[175] A. Daly, "The introduction of data breach notification legislation in Australia: A comparative view," *Computer Law & Security Review*, vol. 34, pp. 477–495, 2018.

[176] "Wm Morrison Supermarkets PLC v. Various Claimants," [2018] EWCA Civ 2339, 2018.

[177] R. Broadhurst, "Cybercrime: the human factor," R. Leukfeldt and T. Holt, Eds. Routledge, forthcoming, 2019, ch. Child Sex Abuse Images and Exploitation Materials. [Online]. Available: https://ssrn.com/abstract=3384499

[178] (1990) Computer Misuse Act 1990. [Online]. Available: https://www.legislation.gov.uk/ukpga/1990/18/contents

[179] "Computer Fraud and Abuse Act," *codified at* 18 U.S.C §1030 et seq.

[180] O. S. Kerr, *Computer Crime Law*, 4th ed. West, 2018.

[181] (2019) Chart of signatures and ratifications of Treaty 185. Council of Europe. [Online]. Available: https://www.coe.int/en/web/conventions/full-list/-/conventions/treaty/185/signatures?p_auth=x7nTJy1r

[182] O. S. Kerr, "Cybercrime's Scope: Interpreting Access and Authorization in Computer Misuse Statutes," *New York University Law Review*, vol. 78, pp. 1596–1668, 2003.

[183] ——, "Norms of Computer Trespass," *Columbia Law Review*, vol. 116, no. 4, pp. 1143–1184, 2016.

[184] "United States v. Nosal (*Nosal II*)," 844 F.3d 1024 (9th Cir.), 2016.

[185] "United States v. Drew," 259 FRD 449 (C.D. Cal.), 2009.

[186] "The Code for Crown Prosecutors," Director of Public Prosecutions (UK), 2018.

[187] R. Carolina, "Legal Aspects of Software Protection Devices," *Computer Law and Security Report*, vol. 11, pp. 188–193, Jul-Aug 1995.

[188] G. J. Edwards, "Self-Help Repossession of Software: Should Repossession Be Available in Article 2B of the UCC," *University of Pittsburgh Law Review*, vol. 58, pp. 763–788, 1997.

[189] N. Schmidle, "Digital Vigilantes," The New Yorker, 7 May 2018.

[190] S. Curry, "Hack-Back: Vigilantism In The Connected World," *Forbes*, 7 January 2019. [Online]. Available: https://www.forbes.com/sites/samcurry/2019/01/07/hack-back-vigilantism-in-the-connected-world/#379ca3a55437

[191] (2019) Should Companies Risk Going on the Cyber Offensive? Brink. [Online]. Available: http://www.brinknews.com/should-companies-risk-going-on-the-cyber-offensive/

[192] H. Beale, Ed., *Chitty on Contracts*, 33rd ed. Sweet & Maxwell, 2018.

[193] "Directive 2000/31/EC of the European Parliament and of the Council of 8 June 2000 on certain legal aspects of information society services, in particular electronic commerce, in the Internal Market (Directive on electronic commerce)," *Official Journal of the European Communities*, vol. L 178, pp. 1–16, 2000.

[194] "Fair and Accurate Credit Transactions Act of 2003," United States Statutes at Large, vol. 117, p. 1952, 2003.

[195] "Directive (EU) 2015/2366 Of The European Parliament And Of The Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 200," *Official Journal of the European Communities*, vol. L 337, pp. 35–127, 2015.

[196] "Uniform Commercial Code, Article 4A," The American Law Institute and the National Conference of Commissioners on Uniform State Laws.

[197] "Regulation (EC) No 593/2008 of the European Parliament and of the Council of 17 June 2008 on the law applicable to contractual obligations (Rome I)," *Official Journal of the European Communities*, vol. L 177, pp. 6–16, 2008.

[198] "Umpqua Bank, et al v. Target Corp, (complaint)," MDL No. 14-2522 (PAM/JJK), Fed Dist Minnesota, (complaint as filed Aug 1, 2014).

[199] "Dittman, et al v. UPMC," No. 43 WAP 2017, 196 A.3d 1036 (Pa S.Ct), 2018.

[200] "The T.J. Hooper," 60 F.2d 737, 2d Cir, 1932.

[201] R. A. Epstein, “The Path to "The TJ Hooper": The Theory and History of Custom in the Law of Tort,” *The Journal of Legal Studies*, vol. 21, no. 1, pp. 1–38, 1992.

[202] “United States v. Caroll Towing,” 159 F.2d 169, 2d Cir., 1947.

[203] P. J. Kelley, “The Carroll Towing Company Case and the Teaching of Tort Law,” *Saint Louis University Law Journal*, vol. 45, pp. 731–758, 2001.

[204] “Restatement (Third) of Torts: Products Liability,” American Law Institute, 1997.

[205] “Council Directive of 25 July 1985 on the approximation of the laws, regulations and administrative provisions of the Member States concerning liability for defective products (85/374/EEC),” *Official Journal of the European Communities*, vol. L 210, pp. 29–33, 1985.

[206] “Evaluation of Council Directive 85/374/EEC of 25 July 1985 on the approximation of the liability for defective products,” European Commission, 2018.

[207] “Liability for emerging digital technologies,” SWD(2018) 137, European Commission, 2018.

[208] “Barnett v. Chelsea & Kensington Hospital [1969],” 1 QB 428, 1969.

[209] “Dillon v. Twin State Gas & Elec. Co.” 85 N.H. 449, New Hampshire, 1932.

[210] “Wagon Mound (No. 1) [1961] UKPC 2 ,” Privy Council, 1961.

[211] “Hedley Byrne & Co Ltd v. Heller & Partners Ltd,” [1964] AC 465, 1963.

[212] “Regulation (EC) No 864/2007 of the European Parliament and of the Council of 11 July 2007 on the law applicable to non-contractual obligations (Rome II),” *Official Journal of the European Union*, vol. L 199, pp. 40–49, 2007.

[213] “WIPO Copyright Treaty,” World Intellectual Property Organization, Geneva, 1996.

[214] S. P. Calandrillo and E. M. Davison, “The Dangers of the Digital Millennium Copyright Act: Much Ado about Nothing,” *William and Mary Law Review*, vol. 50, pp. 349–415, 2008.

[215] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1997.

[216] G. Vetter, “Patenting Cryptographic Technology,” *Chicago-Kent Law Review*, vol. 84, pp. 757–776, 2009.

[217] L. Khansa and C. W. Zobel, “Assessing innovations in cloud security,” *Journal of Computer Information Systems*, vol. 54, no. 3, pp. 45–56, 2014.

[218] T. L. James, L. Khansa, D. F. Cook, O. Bruyaka, and K. B. Keeling, “Using network-based text analysis to analyze trends in microsoft’s security innovations,” *Computers & Security*, vol. 36, pp. 49–67, 2013.

[219] “Uniform Trade Secrets Act,” Uniform Law Commission, National Conference of Commissioners on Uniform State Laws, 1985.

[220] J. H. Pooley, M. A. Lemley, and P. J. Toren, “Understanding the Economic Espionage Act of 1996,” *Tex. Intell. Prop. LJ*, vol. 5, pp. 177–229, 1996.

[221] "Directive (EU) 2016/943 of the European Parliament and of the Council of 8 June 2016 on the protection of undisclosed know-how and business information (trade secrets) against their unlawful acquisition, use and disclosure," *Official Journal of the European Union*, vol. L 157, pp. 1–18, 2016.

[222] PricewaterhouseCoopers, "The scale and impact of industrial espionage and theft of trade secrets through cyber," European Commission, 2018.

[223] D. S. Levine and C. B. Seaman, "The DTSA at One: An Empirical Study of the First Year of Litigation Under the Defend Trade Secrets Act," *Wake Forest Law Review*, vol. 53, pp. 105–156, 2018.

[224] J. Lane, "NTP, Inc. v. Research in Motion, Ltd.: Inventions Are Global, But Politics Are Still Local-An Examination of the BlackBerry Case," *Berkeley Tech. LJ*, vol. 21, pp. 59–77, 2006.

[225] P. Samuelson and S. Scotchmer, "The Law and Economics of Reverse Engineering," *Yale Law Journal*, vol. 111, pp. 1575–1663, 2002.

[226] P. J. Weiser, "The Internet, Innovation, and Intellectual Property Policy," *Columbia Law Review*, vol. 103, pp. 534–613, 2003.

[227] P. Samuelson, "Anticircumvention Rules: Threat to Science," *Science*, vol. 293, no. 5537, pp. 2028–2031, 2001.

[228] C. Zieminski, "Game Over for Reverse Engineering: How the DMCA and Contracts Have Affected Innovation," *Journal of Technology Law & Policy*, vol. 13, pp. 289–339, 2008.

[229] "Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs," *Official Journal of the European Union*, vol. L 111, pp. 16–22, 2009.

[230] P. Samuelson, "Freedom to Tinker," *Theoretical Inquiries in Law*, vol. 17, no. 2, pp. 562–600, 2016.

[231] R. Verdult and F. D. Garcia, "Cryptanalysis of the Megamos Crypto automotive immobilizer," *USENIX; login*, vol. 40, no. 6, pp. 17–22, 2015.

[232] "Volkswagen Aktiengesellschaft vs Garcia, et al," [2013] EWHC 1832 (Ch), 2013.

[233] R. Carolina and K. G. Paterson. (2013) Megamos Crypto, Responsible Disclosure, and the Chilling Effect of Volkswagen Aktiengesellschaft vs Garcia, et al. [Online]. Available: http://www.origin.co.uk/download/43/

[234] R. Verdult, F. D. Garcia, and B. Ege, "Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 687–702.

[235] "Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979)," 1979.

[236] "Prince plc v. Prince Sports Groups," [1998] F.S.R. 21, 1998.

[237] G. Sartor, "Providers Liability: From the eCommerce Directive to the Future," European Parliament, 2017.

[238] K. Perset, "The Economic and Social Role of Internet Intermediaries," *OECD, Directorate for Science, Technology and Industry, OECD Digital Economy Papers*, 2010.

[239] T. Verbiest, G. Spindler, and G. M. Riccio. (2007) Study on the liability of internet intermediaries. [Online]. Available: http://dx.doi.org/10.2139/ssrn.2575069

[240] M. B. Kaya, "The regulation of Internet intermediaries under Turkish law: Is there a delicate balance between rights and obligations?" *Computer Law & Security Review*, vol. 33, p. 759–774, 2017.

[241] "UNCITRAL Model Law on Electronic Commerce with Guide to Enactment 1996 with additional article 5 bis as adopted in 1998," United Nations, 1999.

[242] A. M. Froomkin, "The Essential Role of Trusted Third Parties in Electronic Commerce," *Oregon Law Review*, vol. 75, pp. 49–115, 1996.

[243] S. Mason, *Electronic Signatures in Law*, 4th ed. University of London, School of Advanced Study, Institute of Advanced Legal Studies, 2017.

[244] "Digital Signature Guidelines: Legal Infrastructure for Certification Authorities and Secure Electronic Commerce," American Bar Association, 1996.

[245] "Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures," *Official Journal of the European Union*, vol. L 13, pp. 12–20, 1999.

[246] "Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC," *Official Journal of the European Union*, vol. L 257, pp. 73–114, 2014.

[247] S. Room, *Butterworths Data Security Law & Practice*. LexisNexis, 2009.

[248] D. W. Arner, J. Barberis, and R. P. Buckey, "FinTech, RegTech, and the Reconceptualization of Financial Regulation," *Northwestern Journal of International Law & Business*, vol. 37, pp. 371–414, 2017.

[249] M. G. Porcedda, "Patching the patchwork: appraising the EU regulatory framework on cyber security breaches," *Computer Law & Security Review*, vol. 34, pp. 1077–1098, 2018.

[250] "Directive (EU) 2016/1148 of the European Parliament and of the Council of 6 July 2016 concerning measures for a high common level of security of network and information systems across the Union," *Official Journal of the European Union*, vol. L 194, pp. 1–30, 2016.

[251] "The Network and Information Systems Regulations 2018," S.I. 2018 No 506, 2018.

[252] D. Thaw, "The Efficacy of Cybersecurity Regulation," *Georgia State University Law Review*, vol. 30, pp. 287–374, 2014.

[253] "Regulation (EU) 2019/881 of the European Parliament and of the Council of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No 526/2013 (Cybersecurity Act)," *Official Journal of the European Union*, vol. L 151, pp. 15–69, 2017.

[254] "Code of Practice for Consumer IoT Security," UK Government: Department for Digital, Culture, Media & Sport, 2018.

[255] "TS 103 645 V1.1.1 Cyber Security for Consumer Internet of Things," ETSI, 2019.

[256] "Junger v. Daley," 209 F.3d 481 (6th Cir.), 2000.

[257] J. R. Roig, "Decoding First Amendment Coverage of Computer Source Code in the Age of YouTube, Facebook, and the Arab Spring," *N.Y.U. Annual Survey of American Law*, vol. 68, pp. 319–395, 2012.

[258] "Cyber and International Law in the 21st Century," Speech delivered at Chatham House, Royal Institute for International Affairs, Attorney General of the UK, Jeremy Wright, QC, MP, 23 May 2018. [Online]. Available: https://www.gov.uk/government/speeches/cyber-and-international-law-in-the-21st-century

[259] I. Y. Liu, "The due diligence doctrine under Tallinn Manual 2.0," *Computer Law & Security Review*, vol. 33, p. 390–395, 2017.

[260] C. Lotriionte, "Countering State-Sponsored Cyber Economic Espionage under International Law," *North Carolina Journal of International Law and Commercial Regulation*, vol. 40, no. 2, pp. 443–542, 2015.

[261] ——, "Reconsidering the Consequences for State-Sponsored Hostile Cyber Operations Under International Law," *The Cyber Defense Review*, vol. 3, no. 2, pp. 73–144, 2018.

[262] M. Libicki, "The Coming of Cyber Espionage Norms," in *2017 9th International Conference on Cyber Conflict (CyCon)*. IEEE, 2017, pp. 1–17.

[263] "The Joint Service Manual of the Law of Armed Conflict," Joint Service Publication 383, UK Ministry of Defence, 2004.

[264] "Department of Defense Law of War Manual," US Department of Defense, December 2016.

[265] "Law of Armed Conflict Deskbook," International and Operational Law Department, the United States Army Judge Advocate General's Legal Center and School (TJAGLCS), 2015.

[266] P. Pascucci, "Distinction and Proportionality in Cyberwar: Virtual Problems with a Real Solution," *Minnesota Journal of International Law*, vol. 26, no. 2, pp. 419–460, 2017.

[267] M. Dark, R. Epstein, L. Morales, T. Countermine, Q. Yuan, M. Ali, M. Rose, and N. Harter, "A Framework for Information Security Ethics Education," in *10th Colloquium for Information Systems Security Education-University of Maryland*, vol. 4, 2006, pp. 109–115.

[268] ACM History. Association for Computing Machinery. [Online]. Available: https://www.acm.org/about-acm/acm-history

[269] "ACM Code of Ethics and Professional Conduct," Association for Computing Machinery, 2018.

[270] Using the Code. Association for Computing Machinery. [Online]. Available: https://ethics.acm.org/code-of-ethics/using-the-code/

[271] (2019) Accredited Companies - Regions and Services. Crest (International). [Online]. Available: https://www.crest-approved.org/accredited-companies/index.html

[272] "CREST Code of Conduct for CREST Qualified Individuals (version 8.0)," Crest (GB) Ltd., 2016.

[273] A. M. Matwyshyn, A. Cui, A. D. Keromytis, and S. J. Stolfo, "Ethics in Security Vulnerability Research," in *IEEE Security & Privacy*. IEEE Computer Society, March/April 2010, pp. 67–72.

[274] A. Maurushat, *Disclosure of Security Vulnerabilities: Legal and Ethical Issues*. Springer, 2013.

[275] "The Equities Process," National Cyber Security Centre (UK), 2018. [Online]. Available: https://www.gchq.gov.uk/information/equities-process

[276] "Responsible Release Principles for Cyber Security Vulnerabilities," Australian Signals Directorate, 2019. [Online]. Available: https://www.asd.gov.au/publications/Responsible-Release-Principles-for-Cyber-Security-Vulnerabilities.pdf

[277] "Vulnerabilities Equities Process (*redacted version disclosed 2016 under FOIA)*," National Security Agency (US). [Online]. Available: https://www.eff.org/files/2016/01/18/37-3_vep_2016.pdf

[278] (2019) EFF v. NSA, ODNI - Vulnerabilities FOIA. Electronic Frontier Foundation. [Online]. Available: https://www.eff.org/cases/eff-v-nsa-odni-vulnerabilities-foia

[279] N. Asokan, "Ethics in Information Security," *IEEE Security & Privacy*, May/June 2017.

[280] "Bug Bounties: Working Towards a Fairer and Safer Marketplace," CREST, 2018. [Online]. Available: https://www.crest-approved.org/wp-content/uploads/CREST-Bug-Bounties-2018.pdf

[281] M. Goldstein, A. Stevenson, and L. Picker, "Hedge Fund and Cybersecurity Firm Team Up to Short-Sell Device Maker," *The New York Times*, 8 September 2016.

[282] "ISO/IEC 29147:2014 Information Technology - Security techniques - Vulnerability disclosures," 2014.

[283] "ISO/IEC 30111:2013 Information technology - Security techniques - Vulnerability handling processes," 2013.

[284] "Coordinated Vulnerability Disclosure: the Guideline," National Cyber Security Centre (NL), 2018.

[285] "NCSC vulnerability disclosure co-ordination," National Cyber Security Centre (UK), 2018. [Online]. Available: https://www.ncsc.gov.uk/blog-post/ncsc-vulnerability-disclosure-co-ordination

[286] D. Feldman, "The Nature of Legal Scholarship," *Modern Law Review*, vol. 52, pp. 498–517, 1989.

[287] G. A. Spann, "Baby M and the Cassandra Problem," *Georgetown Law Journal*, vol. 76, pp. 1719–1739, 1987.

[288] K. Takayanagi, "Contact of the Common Law with the Civil Law in Japan," *The American Journal of Comparative Law*, vol. 4, pp. 60–69, 1955.

[289] C. Reed and A. Murray, *Rethinking the Jurisprudence of Cyberspace*. Edward Elgar Publishing, 2018.

[290] S. M. Solaiman, "Legal personality of robots, corporations, idols and chimpanzees: a quest for legitimacy," *Artificial Intelligence and Law*, vol. 25, no. 2, pp. 155–179, 2017.

[291] P. Cerka, J. Grigiene, and G. Sirbikyte, "Is it possible to grant legal personality to artificial intelligence software systems?" *Computer Law & Security Review*, vol. 33, no. 5, pp. 685–699, 2017.

[292] ——, "Liability for damages caused by artificial intelligence," *Computer Law & Security Review*, vol. 31, no. 3, pp. 376–389, 2015.

[293] D. Gerard, *Attack of the 50 Foot Blockchain: Bitcoin, Blockchain, Ethereum & Smart Contracts*, 2017.

[294] "Criminal Liability: Insanity and Automatism. A Discussion Paper – Summary for non-specialists," 2013. [Online]. Available: http://www.lawcom.gov.uk/app/uploads/2015/06/insanity_discussion_summary.pdf

[295] "Regulation (EU) No 1215/2012 of the European Parliament and of the Council of 12 December 2012 on jurisdiction and the recognition and enforcement of judgments in civil and commercial matters," *Official Journal of the European Union*, vol. L 351, pp. 1–32, 2012.

[296] L. Kasdan, Director, "Silverado," FILM, Columbia Pictures, US, 1985.

[297] A. M. Froomkin, "Wrong Turn in Cyberspace: Using ICANN to Route around the APA and the Constitution," *Duke Law Journal*, vol. 50, pp. 17–184, 2000.

[298] R. H. Weber, "'Rose is a rose is a rose is a rose'–what about code and law?" *Computer Law & Security Review*, vol. 34, pp. 701–706, 2018.

[299] (2019) URL List. Internet Watch Foundation. [Online]. Available: https://www.iwf.org.uk/become-a-member/services-for-members/url-list

[300] "Joint Civil Society Response to Discussion Guide on a 2nd Additional Protocol to the Budapest Convention on Cybercrime," The Electronic Frontier Foundation (EFF), European Digital Rights (EDRi), Association for Civil Rights (ADC), Derechos Digitales, Elektronisk Forpost Norge (EFN), IPANDETEC, Karisma Foundation, OpenMedia, Panoptykon Foundation, R3D: Red en Defensa de los Derechos Digitales, Samuelson-Glushko Canadian Internet Policy and Public Interest Clinic (CIPPIC), SonTusDatos (Artículo 12, A.C.) and TEDIC, 28 June 2018. [Online]. Available: https://www.eff.org/files/2018/08/02/globalcoalition-civilsociety-t-cy_201816-final1.pdf

[301] D. Beyleveld and R. Brownsword, *Consent in the Law*. Hart, 2007.

[302] "Convention for the Protection of Human Rights and Fundamental Freedoms (European Convention on Human Rights)," Council of Europe, 1950.

[303] "Explanations Relating to the Charter of Fundamental Rights," *Official Journal of the European Union*, vol. C 303, pp. 17–35, 2007.

[304] "Carpenter v. US," 585 U.S. ___, 138 S.Ct. 2206 (US S.Ct), 2018.

[305] "Law enforcement disclosure and demands for customer data, Legal Annexe: Overview of legal powers," Vodafone Group Plc, 2017. [Online]. Available: https://www.vodafone.com/content/dam/vodafone-images/sustainability/drf/pdf/vodafone_drf_law_enforcement_disclosure_legal_annexe_2016.pdf

[306] (2016) Canary Watch – One Year Later. Electronic Frontier Foundation. [Online]. Available: https://www.eff.org/deeplinks/2016/05/canary-watch-one-year-later

[307] "Brady v. Maryland," 373 U.S. 83 (US S.Ct), 1963.

[308] M. Mourby, E. Mackey, M. Elliot, H. Gowans, S. E. Wallace, J. Bell, H. Smith, S. Aidinlis, and J. Kaye, "Are 'pseudonymised' data always personal data? Implications of the GDPR for administrative data research in the UK," *Computer Law & Security Review*, vol. 34, pp. 222–233, 2018.

[309] L. Rocher, J. M. Hendrickx, and Y.-A. de Montjoye, "Estimating the success of re-identifications in incomplete datasets using generative models," *Nature Communications*, vol. 10, no. 3069, 2019. [Online]. Available: https://doi.org/10.1038/s41467-019-10933-3

[310] J. Joerling, "Data Breach Notification Laws: An Argument for a Comprehensive Federal Law to Protect Consumer Data," *Washington University Journal of Law & Policy*, vol. 32, pp. 467–488, 2010.

[311] "R v. Gold and Schifreen," [1988] AC 1063, [1988] Crim LR 437 (HL), 1988.

[312] B. Sterling, *The Hacker Crackdown: Law and Disorder on the Electronic Frontier*. Bantam Books, 1992. [Online]. Available: https://github.com/bdesham/the-hacker-crackdown

[313] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, 1989.

[314] "Computer Crime Statutes," National Conference of State Legislatures. [Online]. Available: http://www.ncsl.org/research/telecommunications-and-information-technology/computer-hacking-and-unauthorized-access-laws.aspx

[315] S. W. Brenner, "State Cybercrime Legislation in the United States of America: A Survey," *Richmond Journal of Law and Technology*, vol. 7, no. 3, pp. 28–?, 2001. [Online]. Available: http://scholarship.richmond.edu/jolt/vol7/iss3/4

[316] (2009) BBC team exposes cyber crime risk. BBC. [Online]. Available: http://news.bbc.co.uk/1/hi/programmes/click_online/7932816.stm

[317] M. Perrow. (2009) Click's botnet experiment. [Online]. Available: https://www.bbc.co.uk/blogs/theeditors/2009/03/click_botnet_experiment.html

[318] R. Carolina. (2009) Opinion: The unanticipated consequences of BBC Click's botnet crime. [Online]. Available: https://www.computerweekly.com/opinion/Opinion-The-unanticipated-consequences-of-BBC-Clicks-botnet-crime

[319] ——. (2009) Opinion: BBC Click exploited world's poor and vulnerable. [Online]. Available: https://www.computerweekly.com/opinion/Opinion-BBC-Click-exploited-worlds-poor-and-vulnerable

[320] "Guidelines Manual," United States Sentencing Commission, November 2018. [Online]. Available: https://www.ussc.gov/guidelines/2018-guidelines-manual-annotated

[321] D. Etcovitch and T. van der Merwe, "Coming in from the Cold: A Safe Harbor from the CFAA and the DMCA §1201 for Security Researchers," Research Publication No. 2018-4, Berkman Klein Center, 2018.

[322] U. Kinis, "From Responsible Disclosure Policy (RDP) towards State Regulated Responsible Vulnerability Disclosure Procedure (hereinafter – RVDP): The Latvian approach," *Computer Law & Security Review*, vol. 34, pp. 508–522, 2018.

[323] "Palsgraf v. Long Island Railroad Co." 248 N.Y. 339, 162 N.E. 99, (N.Y Ct of Appeals), 1928.

[324] "Cooney v. Chicago Public Schools," 943 N.E.2d 23 (Illinois Appellate Ct, 1st District, 2010), *appeal denied* 949 N.E.2d 657 (Illinois S.Ct, 2011), 2010.

[325] J. De Bruyne and J. Werbrouck, "Merging self-driving cars with the law," *Computer Law & Security Review*, vol. 34, pp. 1150–1153, 2018.

[326] N. E. Vellinga, "From the testing to the deployment of self-driving cars: Legal challenges to policymakers on the road ahead," *Computer Law & Security Review*, vol. 33, p. 847–863, 2017.

[327] T. Mackie, "Proving liability for highly and fully automated vehicle accidents in Australia," *Computer Law & Security Review*, vol. 34, p. 1314–1332, 2018.

[328] S. Pettypiece and E. Dexheimer, "Target Reaches $67 Million Agreement With Visa Over Breach," *Bloomberg*, 18 August 2015.

[329] J. Stempel and N. Bose, "Target in $39.4 million settlement with banks over data breach," *Reuters*, 3 December 2015.

[330] "Analysis of the Economic Impact of the Development Risk Clause as provided by Directive 85/374/EEC on Liability for Defective Products," European Commission, Ares(2014)3310430 - 07/10/2014, Fondazione Rosselli, 2014.

[331] F. D. Prager, "The Influence of Mr. Justice Story on American Patent Law," *The American Journal of Legal History*, vol. 5, pp. 254–264, 1961.

[332] "Ungar v. Sugg," (1892) 9 RPC 113, 1892.

[333] C. Yang, "The BlackBerry Widow's Tale," *Bloomberg*, 18 December 2005. [Online]. Available: https://www.bloomberg.com/news/articles/2005-12-18/the-blackberry-widows-tale

[334] P. Samuelson, "Intellectual Property and the Digital Economy: Why the Anti-circumvention Regulations Need to be Revised," *Berkley Technology Law Journal*, vol. 14, pp. 519–566, 1999.

[335] "Haberman v. Jackel International," [1999] FSR 683, 1999.

[336] (2019) Woodhull Freedom Foundation et al. v. United States. Electronic Frontier Foundation. [Online]. Available: https://www.eff.org/cases/woodhull-freedom-foundation-et-al-v-united-states

[337] R. Romano. (2019) A new law intended to curb sex trafficking threatens the future of the internet as we know it. [Online]. Available: https://www.vox.com/culture/2018/4/13/17172762/fosta-sesta-backpage-230-internet-freedom

[338] S. Mason, "Documents signed or executed with electronic signatures in English law," *Computer Law & Security Review*, vol. 34, pp. 933–954, 2018.

[339] J. E. Cohen, "The Zombie First Amendment," *William and Mary Law Review*, vol. 56, pp. 1119–1158, 2015.

[340] B. Lee, "Where Gutenberg Meets Guns: The Liberator, 3D-printed Weapons, and the First Amendment," *North Carolina Law Review*, vol. 92, pp. 1393–1425, 2014.

[341] E. T. Jensen, "The Tallinn Manual 2.0: Highlights and Insights," *Georgetown Journal of International Law*, vol. 48, pp. 735–778, 2016.

[342] P. Wilson, "The myth of international humanitarian law," *International Affairs*, vol. 93, no. 3, pp. 563–579, 2017.

[343] "Glossary of Key Information Security Terms," NISTIR 7298 (revision 2), National Institute of Standards and Technology, 2013.

[344] J. Markoff, "Apple's Engineers, if Defiant, Would be in Sync With Ethics Code," *The New York Times*, 18 March 2016.

[345] "United States v. Arthur Andersen LLP," 374 F.3d 281 (5th Cir), *reversed by* [346], 2004.

[346] "Arthur Andersen LLP v. United States," 544 U.S. 696 (US S.Ct), 2005.

[347] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: A usability evaluation of PGP 5.0." in *USENIX Security Symposium*, vol. 348, 1999.

[348] S. Ruoti, J. Andersen, S. Heidbrink, M. O'Neill, E. Vaziripour, J. Wu, D. Zappala, and K. E. Seamons, ""We're on the same Page": A usability study of secure email using pairs of novice users," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, 2016, pp. 4298–4308.

[349] S. Ruoti, J. Andersen, T. Hendershot, D. Zappala, and K. E. Seamons, "Private webmail 2.0: Simple and easy-to-use secure email," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*, 2016, pp. 461–472.

[350] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. E. Seamons, "A comparative usability study of key management in secure email," in *Fourteenth Symposium on Usable Privacy and Security, SOUPS 2018, Baltimore, MD, USA, August 12-14, 2018.*, 2018, pp. 375–394.

[351] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[352] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith, ""If you want, i can store the encrypted password." a password-storage field study with freelance developers," in *Proceedings of the 2019 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI, 2019, Glasgow, UK, May 4 – 9, 2019*, 2019.

[353] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong?: A qualitative usability study," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 311–328.

[354] D. M. Ashenden, L. Coles-Kemp, and K. O'Hara, "Why should I?: Cybersecurity, the security of the state and the insecurity of the citizen," *Politics & Governance*, vol. 6, no. 2, pp. 41–48, 2018.

[355] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The emperor's new security indicators," in *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, 2007, pp. 51–65.

[356] M. E. Zurko, C. Kaufman, K. Spanbauer, and C. Bassett, "Did you ever have to make up your mind? What notes users do when faced with a security decision," in *18th Annual*

Computer Security Applications Conference (ACSAC 2002), 9-13 December 2002, Las Vegas, NV, USA, 2002, pp. 371–381.

[357] S. Egelman, L. F. Cranor, and J. I. Hong, "You've been warned: an empirical study of the effectiveness of web browser phishing warnings," in *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, 2008, pp. 1065–1074.

[358] P. Kumaraguru, S. Sheng, A. Acquisti, L. F. Cranor, and J. I. Hong, "Teaching Johnny not to fall for phish," *ACM Trans. Internet Techn.*, vol. 10, no. 2, pp. 7:1–7:31, 2010.

[359] C. Herley, "More is not the answer," *IEEE Security & Privacy*, vol. 12, no. 1, pp. 14–19, 2014.

[360] M. A. Sasse, S. Brostoff, and D. Weirich, "Transforming the 'weakest link'—a human/computer interaction approach to usable and effective security," *BT Technology Journal*, vol. 19, no. 3, pp. 122–131, 2001.

[361] S. L. Pfleeger, M. A. Sasse, and A. Furnham, "From weakest link to security hero: Transforming staff security behavior," *Journal of Homeland Security and Emergency Management*, vol. 11, no. 4, pp. 489–510, 2014.

[362] R. Reeder, E. C. Kowalczyk, and A. Shostack, "Helping engineers design NEAT security warnings," in *Proceedings of the Symposium On Usable Privacy and Security (SOUPS), Pittsburgh, PA*, 2011.

[363] R. Biddle, S. Chiasson, and P. C. Van Oorschot, "Graphical passwords: Learning from the first twelve years," *ACM Computing Surveys (CSUR)*, vol. 44, no. 4, p. 19, 2012.

[364] F. Monrose and M. K. Reiter, "Graphical passwords," *Security and Usability*, pp. 147–164, 2005.

[365] S. Chiasson, A. Forget, R. Biddle, and P. C. van Oorschot, "Influencing users towards better passwords: persuasive cued click-points," in *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction-Volume 1*. British Computer Society, 2008, pp. 121–130.

[366] R. Jhawar, P. Inglesant, N. Courtois, and M. A. Sasse, "Make mine a quadruple: Strengthening the security of graphical one-time pin authentication," in *Network and System Security (NSS), 2011 5th International Conference on*. IEEE, 2011, pp. 81–88.

[367] S. Uellenbeck, M. Dürmuth, C. Wolf, and T. Holz, "Quantifying the security of graphical passwords: The case of Android unlock patterns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 161–172. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516700

[368] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor, "How does your password measure up? the effect of strength meters on password creation," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, 2012, pp. 65–80.

[369] M. Golla and M. Dürmuth, "On the accuracy of password strength meters," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, 2018, pp. 1567–1582.

[370] J. Foer, *Moonwalking with Einstein: The Art and Science of Remembering Everything*. Penguin Books, 2012.

[371] M. Steves, D. Chisnell, A. Sasse, K. Krol, M. Theofanos, and H. Wald, "Report: Authentication diary study," National Institute of Standards and Technology, Tech. Rep., 2014.

[372] G. Sauer, J. Lazar, H. Hochheiser, and J. Feng, "Towards A universally usable human interaction proof: Evaluation of task completion strategies," *TACCESS*, vol. 2, no. 4, pp. 15:1–15:32, 2010.

[373] E. Bursztein, A. Moscicki, C. Fabry, S. Bethard, J. C. Mitchell, and D. Jurafsky, "Easy does it: more usable CAPTCHAs," in *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, 2014, pp. 2637–2646.

[374] C. Fidas, A. G. Voyiatzis, and N. M. Avouris, "On the necessity of user-friendly CAPTCHA," in *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, 2011, pp. 2623–2626.

[375] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, "Progressive authentication: Deciding when to authenticate on mobile phones," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, 2012, pp. 301–316.

[376] A. Beautement, M. A. Sasse, and M. Wonham, "The compliance budget: managing security behaviour in organisations," in *Proceedings of the 2008 New Security Paradigms Workshop*. ACM, 2009, pp. 47–58.

[377] S. Furnell and K.-L. Thomson, "Recognising and addressing 'security fatigue'," *Computer Fraud & Security*, vol. 2009, no. 11, pp. 7–11, 2009.

[378] K. Holtzblatt and H. Beyer, *Contextual design: Design for life*. Morgan Kaufmann, 2016.

[379] I. Kirlappos, S. Parkin, and M. A. Sasse, "Shadow security as a tool for the learning organization," *ACM SIGCAS Computers and Society*, vol. 45, no. 1, pp. 29–37, 2015.

[380] B. E. Litzky, K. A. Eddleston, and D. L. Kidder, "The good, the bad, and the misguided: How managers inadvertently encourage deviant behaviors," *Academy of Management Perspectives*, vol. 20, no. 1, pp. 91–103, 2006.

[381] K. M. Ramokapane, A. Rashid, and J. M. Such, ""I feel stupid I can't delete...": A study of users' cloud deletion practices and coping strategies," in *Thirteenth Symposium on Usable Privacy and Security, SOUPS 2017, Santa Clara, CA, USA, July 12-14, 2017.*, 2017, pp. 241–256.

[382] K. M. Ramokapane, A. Mazeli, and A. Rashid, "Skip, skip, skip, accept!!!: A study on the usability of smartphone manufacturer provided default features and user privacy," in *Proceedings of Privacy Enhancing Technologies (PoPETS)*, 2019.

[383] K. Greene, J. M. Franklin, and J. M. Kelsey, "Tap on, tap off: onscreen keyboards and mobile password entry," NIST, Tech. Rep., 2015.

[384] W. Melicher, D. Kurilova, S. M. Segreti, P. Kalvani, R. Shay, B. Ur, L. Bauer, N. Christin, L. F. Cranor, and M. L. Mazurek, "Usability and security of text passwords on mobile

devices," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, 2016, pp. 527–539.

[385] K. Krol, E. Philippou, E. De Cristofaro, and M. A. Sasse, ""They brought in the horrible key ring thing!" analysing the usability of two-factor authentication in UK online banking," *ArXiv Preprint ArXiv:1501.04434*, 2015.

[386] B. Craggs and A. Rashid, "Smart cyber-physical systems: Beyond usable security to security ergonomics by design," in *3rd IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS@ICSE 2017, Buenos Aires, Argentina, May 21, 2017*, 2017, pp. 22–25.

[387] I. Kirlappos and M. A. Sasse, "Security education against phishing: A modest proposal for a major rethink," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 24–32, 2012.

[388] S. A. Naqvi, R. Chitchyan, S. Zschaler, A. Rashid, and M. Südholt, "Cross-document dependency analysis for system-of-system integration," in *Foundations of Computer Software. Future Trends and Techniques for Development, 15th Monterey Workshop 2008, Budapest, Hungary, September 24-26, 2008, Revised Selected Papers*, 2008, pp. 201–226.

[389] E. Hollnagel, "Is safety a subject for science?" *Safety Science*, vol. 67, pp. 21–24, 2014.

[390] C. Perrow, "Organizing to reduce the vulnerabilities of complexity," *Journal of Contingencies and Crisis Management*, vol. 7, no. 3, pp. 150–155, 1999.

[391] D. Kahneman, *Thinking, Fast and Slow*. Penguin Books, 2012.

[392] M. Beyer, S. Ahmed, K. Doelemann, S. Arnell, S. Parkin, A. Sasse, and N. Passingham, "Awareness is only the first step," Hewlett Packard Enterprise, Tech. Rep., 2015.

[393] R. Wash, "Folk models of home computer security," in *Proceedings of the Sixth Symposium on Usable Privacy and Security*. ACM, 2010, p. 11.

[394] J. Nicholson, L. M. Coventry, and P. Briggs, "Introducing the cybersurvival task: Assessing and addressing staff beliefs about effective cyber protection," in *Fourteenth Symposium on Usable Privacy and Security, SOUPS 2018, Baltimore, MD, USA, August 12-14, 2018.*, 2018, pp. 443–457.

[395] S. Frey, A. Rashid, P. Anthonysamy, M. Pinto-Albuquerque, and S. A. Naqvi, "The good, the bad and the ugly: A study of security decisions in a cyber-physical systems game," *IEEE Transactions on Software Engineering*, vol. 45, no. 5, pp. 521–536, 2017.

[396] E. Wenger, *Communities of practice: Learning, meaning, and identity*. Cambridge university press, 1999.

[397] A. Joinson and L. Piwek, "Technology and the formation of socially positive behaviours," *Beyond Behaviour Change: Key Issues, Interdisciplinary Approaches and Future Directions*, vol. 157, 2016.

[398] T. Denning, A. Lerner, A. Shostack, and T. Kohno, "Control-alt-hack: the design and evaluation of a card game for computer security awareness and education," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, 2013, pp. 915–928.

[399] C. Bravo-Lillo, L. F. Cranor, J. S. Downs, and S. Komanduri, "Bridging the gap in

computer security warnings: A mental model approach," *IEEE Security & Privacy*, vol. 9, no. 2, pp. 18–26, 2011.

[400] L. J. Camp, "Mental models of privacy and security," *IEEE Technol. Soc. Mag.*, vol. 28, no. 3, pp. 37–46, 2009.

[401] D. Florêncio, C. Herley, and A. Shostack, "FUD: a plea for intolerance," *Commun. ACM*, vol. 57, no. 6, pp. 31–33, 2014.

[402] B. McSweeney and M. Bill, *Security, identity and interests: a sociology of international relations*. Cambridge University Press, 1999, vol. 69.

[403] P. Roe, "The 'value' of positive security," *Review of International Studies*, vol. 34, no. 4, pp. 777–794, 2008.

[404] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[405] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security APIs," *IEEE Security & Privacy*, vol. 14, no. 5, 2016.

[406] P. Dunphy, J. Vines, L. Coles-Kemp, R. Clarke, V. Vlachokyriakos, P. Wright, J. McCarthy, and P. Olivier, "Understanding the experience-centeredness of privacy and security technologies," in *Proceedings of the 2014 New Security Paradigms Workshop*. ACM, 2014, pp. 83–94.

[407] C. P. Heath, P. A. Hall, and L. Coles-Kemp, "Holding on to dissensus: Participatory interactions in security design," *Strategic Design Research Journal*, vol. 11, no. 2, pp. 65–78, 2018.

[408] A. Beautement, I. Becker, S. Parkin, K. Krol, and M. A. Sasse, "Productive security: A scalable methodology for analysing employee security behaviours," in *Twelfth Symposium on Usable Privacy and Security, SOUPS 2016, Denver, CO, USA, June 22-24, 2016*, 2016, pp. 253–270.

[409] M. E. Zurko and R. T. Simon, "User-centered security," in *Proceedings of the 1996 workshop on New security paradigms*. ACM, 1996, pp. 27–33.

[410] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028088

[411] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 289–305.

[412] T. Lopez, T. T. Tun, A. K. Bandara, M. Levine, B. Nuseibeh, and H. Sharp, "An anatomy of security conversations in stack overflow," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, 2019, pp. 31–40.

[413] S. Arzt, S. Nadi, K. Ali, E. Bodden, S. Erdweg, and M. Mezini, "Towards secure integration of cryptographic software," in *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2015, Pittsburgh, PA, USA, October 25-30, 2015*, 2015, pp. 1–13.

[414] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: why do java developers struggle with cryptography APIs?" in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, 2016, pp. 935–946.

[415] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, and R. Kamath, "CogniCrypt: supporting developers in using cryptography," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, 2017, pp. 931–936.

[416] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. R. Murphy-Hill, "Just-in-time static analysis," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*, 2017, pp. 307–317.

[417] N. Patnaik, J. Hallett, and A. Rashid, "Usability smells: An analysis of developers' struggle with crypto libraries," in *Fifteenth Symposium on Usable Privacy and Security (SOUPS), Santa Clara, USA*. USENIX Association, 2019.

[418] D. D. Caputo, S. L. Pfleeger, M. A. Sasse, P. Ammann, J. Offutt, and L. Deng, "Barriers to usable security? three organizational case studies," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 22–32, 2016.

[419] J. M. Haney, M. Theofanos, Y. Acar, and S. S. Prettyman, ""We make it a big deal in the company": security mindsets in organizations that develop cryptographic products," in *Fourteenth Symposium on Usable Privacy and Security, SOUPS 2018, Baltimore, MD, USA, August 12-14, 2018.*, 2018, pp. 357–373.

[420] R. Stevens, D. Votipka, E. M. Redmiles, C. Ahern, P. Sweeney, and M. L. Mazurek, "The battle for New York: A case study of applied digital threat modeling at the enterprise level," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 621–637.

[421] UN General Assembly, "Universal declaration of human rights," December 1948, article 12. [Online]. Available: https://www.un.org/en/udhrbook/pdf/udhr_booklet_en_web.pdf

[422] S. D. Warren and L. D. Brandeis, "The right to privacy," in *Ethical Issues in the Use of Computers*, D. G. Johnson and J. W. Snapper, Eds. Wadsworth Publ. Co., 1985.

[423] A. Rouvroy and Y. Poullet, "The right to informational self-determination and the value of self-development: Reassessing the importance of privacy for democracy," in *Reinventing Data Protection?*, S. Gutwirth, Y. Poullet, P. De Hert, C. de Terwangne, and S. Nouwt, Eds. Springer Netherlands, 2009.

[424] A. F. Westin, *Privacy and Freedom*. Atheneum, 1967.

[425] P. E. Agre and M. Rotenberg, Eds., *Technology and Privacy: The New Landscape*. MIT Press, 1998.

[426] European Commission, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," *Official Journal of the European Union*, 2016.

[427] G. Danezis and S. Gürses, "A critical review of 10 years of privacy technology," *Surveillance Cultures: A Global Surveillance Society?*, 2010.

[428] S. Gürses and C. Diaz, "Two tales of privacy in online social networks," *IEEE Security and Privacy*, vol. 11, no. 3, pp. 29–37, 2013.

[429] H. Nissenbaum, "Privacy as contextual integrity," *Washington Law Review*, 2 2004.

[430] G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[431] N. Borisov, I. Goldberg, and E. A. Brewer, "Off-the-record communication, or, why not to use PGP," in *WPES*. ACM, 2004.

[432] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology - EUROCRYPT*, 2001.

[433] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1982.

[434] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, 2014.

[435] M. Marlinspike, "Advanced cryptographic ratcheting)." [Online]. Available: https://whispersystems.org/blog/advanced-ratcheting/

[436] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *IEEE Symposium on Security and Privacy (S&P 2009)*, 2009.

[437] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Díaz, "A survey on routing in anonymous communication protocols," *ACM Comput. Surv.*, 2018.

[438] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: how Diffie-Hellman fails in practice," *Commun. ACM*, 2019.

[439] "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, 2018.

[440] M. W. Lucas, *PGP & GPG: Email for the Practical Paranoid*, 1st ed. No Starch Press, 2006.

[441] C. Alexander and I. Goldberg, "Improved user authentication in off-the-record messaging," in *WPES*, 2007.

[442] N. Unger and I. Goldberg, "Improved strongly deniable authenticated key exchanges for secure messaging," *PoPETs*, 2018.

[443] Open Whisper Systems, "Signal Protocol library for Java/Android." [Online]. Available: https://github.com/signalapp/libsignal-protocol-java

[444] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *IEEE European Symposium on Security and Privacy, EuroS&P*, 2017.

[445] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *PoPETs*, 2018.

[446] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000.

[447] N. Borisov, G. Danezis, and I. Goldberg, "DP5: A private presence service," *PoPETs*, 2015.

[448] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "PIR-Tor: Scalable anonymous communication using private information retrieval," in *20th USENIX Security Symposium*, 2011.

[449] R. R. Toledo, G. Danezis, and I. Goldberg, "Lower-cost $\in$-private information retrieval," *PoPETs*, 2016.

[450] W. Aiello, Y. Ishai, and O. Reingold, "Priced oblivious transfer: How to sell digital goods," in *Advances in Cryptology - EUROCRYPT 2001*, 2001.

[451] J. Camenisch, M. Dubovitskaya, and G. Neven, "Unlinkable priced oblivious transfer with rechargeable wallets," in *Financial Cryptography and Data Security*, 2010.

[452] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *2018 IEEE Symposium on Security and Privacy, SP 2018*, 2018.

[453] S. Sasy, S. Gorbunov, and C. W. Fletcher, "Zerotrace : Oblivious memory primitives from intel SGX," in *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[454] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 85–100. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043566

[455] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *ACM Conference on Computer and Communications Security*, 2015.

[456] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *ACM Conference on Computer and Communications Security*, 2016.

[457] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov, "The tao of inference in privacy-protected databases," *PVLDB*, 2018.

[458] P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," in *ACM Conference on Computer and Communications Security*, 2018.

[459] P. Grubbs, T. Ristenpart, and V. Shmatikov, "Why your encrypted database is not secure," in *HotOS*, 2017.

[460] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving

encrypted databases," in *ACM Conference on Computer and Communications Security*, 2015.

[461] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI*, 2017.

[462] L. Melis, G. Danezis, and E. D. Cristofaro, "Efficient private statistics with succinct sketches," in *NDSS*, 2016.

[463] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases - real-world applications of secure multi-party computation," *Comput. J.*, 2018.

[464] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *PoPETs*, 2017.

[465] M. Nagy, E. D. Cristofaro, A. Dmitrienko, N. Asokan, and A. Sadeghi, "Do I know you?: Efficient and privacy-preserving common friend-finder protocols and applications," in *Annual Computer Security Applications Conference, ACSAC*, 2013.

[466] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding P2P bots with structured graph analysis," in *19th USENIX Security Symposium*, 2010.

[467] P. Baldi, R. Baronio, E. D. Cristofaro, P. Gasti, and G. Tsudik, "Countering GATTACA: efficient and secure testing of fully-sequenced human genomes," in *ACM Conference on Computer and Communications Security, CCS*, 2011.

[468] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography*, 2010.

[469] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004.

[470] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient circuit-based PSI via cuckoo hashing," in *EUROCRYPT (3)*, 2018.

[471] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "PrETP: Privacy-preserving electronic toll pricing," in *USENIX Security Symposium*, 2010.

[472] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *WPES*, 2011.

[473] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy, SP*, 2015.

[474] C. Hazay and K. Nissim, "Efficient set operations in the presence of malicious adversaries," in *Public Key Cryptography*, 2010.

[475] J. Camenisch, "Concepts around privacy-preserving attribute-based credentials - making authentication with anonymous credentials practical," in *Privacy and Identity Management*, ser. IFIP Advances in Information and Communication Technology, 2013.

[476] M. Koning, P. Korenhof, G. Alpár, and J.-H. Hoepman, "The ABC of ABC: An analysis of attribute-based credentials in the light of data protection, privacy and identity," *HotPETS*, 2014.

[477] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich, "How to win the clonewars: efficient periodic n-times anonymous authentication," in *13th ACM Conference on Computer and Communications Security, CCS*, 2006.

[478] R. Henry and I. Goldberg, "Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting," in *12th annual ACM Workshop on Privacy in the Electronic Society, WPES*, 2013.

[479] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "Blacklistable anonymous credentials: blocking misbehaving users without ttps," in *ACM Conference on Computer and Communications Security, CCS*, 2007.

[480] J. Camenisch, M. Drijvers, and J. Hajny, "Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs," in *ACM on Workshop on Privacy in the Electronic Society, WPES*, 2016.

[481] IBM, "Identity mixer." [Online]. Available: https://www.zurich.ibm.com/identity_mixer/

[482] Privacy by Design Foundation, "IRMA: I Reveal My Attributes." [Online]. Available: https://privacybydesign.foundation/irma-explanation/

[483] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *IEEE Symposium on Security and Privacy, SP*, 2014.

[484] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from Bitcoin," in *IEEE Symposium on Security and Privacy, SP*, 2013, pp. 397–411.

[485] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium*, 2014.

[486] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific Reports*, 2013.

[487] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," *PLOS Genetics*, 2008.

[488] P. Golle and K. Partridge, "On the anonymity of home/work location pairs," in *Pervasive Computing*, 2009.

[489] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002.

[490] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *International Conference on Data Engineering, ICDE*, 2006.

[491] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *ICDE*. IEEE Computer Society, 2007.

[492] K. El Emam and F. K. Dankar, "Protecting Privacy Using k-Anonymity," *Journal of the American Medical Informatics Association*, 2008.

[493] B. Gedik and L. Liu, "Protecting location privacy with personalized k-anonymity: Architecture and algorithms," *IEEE Transactions on Mobile Computing*, 2008.

[494] M. Stegelmann and D. Kesdogan, "Gridpriv: A smart metering architecture offering k-anonymity," in *Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2012.

[495] E. Balsa, C. Troncoso, and C. Díaz, "OB-PWS: obfuscation-based private web search," in *IEEE Symposium on Security and Privacy*, 2012.

[496] A. Gadotti, F. Houssiau, L. Rocher, and Y. de Montjoye, "When the signal is in the noise: The limits of diffix's sticky noise," *CoRR*, vol. abs/1804.06752, 2018.

[497] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Preserving privacy in GPS traces via uncertainty-aware path cloaking," in *ACM Conference on Computer and Communications Security, CCS*, 2007.

[498] K. Chatzikokolakis, C. Palamidessi, and M. Stronati, "A predictive differentially-private mechanism for mobility traces," in *Privacy Enhancing Technologies - 14th International Symposium, PETS*, 2014.

[499] R. Chow and P. Golle, "Faking contextual data for fun, profit, and privacy," in *ACM Workshop on Privacy in the Electronic Society, WPES*, 2009.

[500] S. T. Peddinti and N. Saxena, "On the privacy of web search based on query obfuscation: A case study of trackmenot," in *10th International Symposium Privacy Enhancing Technologies PETS*, 2010.

[501] J. L. Raisaro, J. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J.-P. Hubaux, "Medco: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data," *IEEE/ACM transactions on computational biology and bioinformatics*, 2018.

[502] J. J. Kim, "A method for limiting disclosure in microdata based on random noise and transformation," in *Proceedings of the section on survey research methods*. American Statistical Association, 1986.

[503] W. E. Yancey, W. E. Winkler, and R. H. Creecy, "Disclosure risk assessment in perturbative microdata protection," in *Inference Control in Statistical Databases, From Theory to Practice*, 2002.

[504] C. Dwork, "Differential privacy," in *ICALP (2)*, ser. Lecture Notes in Computer Science, 2006.

[505] S. Oya, C. Troncoso, and F. Pérez-González, "Is geo-indistinguishability what you are looking for?" in *WPES*, 2017.

[506] S. Meiser and E. Mohammadi, "Tight on budget?: Tight bounds for r-fold approximate differential privacy," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[507] K. Chatzikokolakis, M. E. Andrés, N. E. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *Privacy Enhancing Technologies*, 2013.

[508] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *ACM Conference on Computer and Communications Security*, 2017.

[509] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 901–914.

[510] J. M. Abowd, "The U.S. census bureau adopts differential privacy," in *KDD*, 2018.

[511] S. E. Oh, S. Li, and N. Hopper, "Fingerprinting keywords in search queries over Tor," *PoPETs*, vol. 2017, 2017.

[512] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks," in *IEEE Symposium on Security and Privacy, S&P 2011*, 2011.

[513] G. Danezis and C. Diaz, "A survey of anonymous communication channels," Microsoft Research, Tech. Rep. MSR-TR-2008-35, 2008.

[514] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *13th USENIX Security Symposium*, 2004.

[515] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. F. Syverson, "Users get routed: traffic correlation on Tor by realistic adversaries," in *ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[516] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, 1981.

[517] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix anonymity system," in *USENIX Security Symposium, USENIX Security*, 2017.

[518] G. Danezis and I. Goldberg, "Sphinx: A compact and provably secure mix format," in *IEEE Symposium on Security and Privacy (S&P)*, 2009.

[519] G. Acar, M. Juárez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "FPDetective: dusting the web for fingerprinters," in *2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[520] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "FP-STALKER: tracking browser fingerprint evolutions," in *IEEE Symposium on Security and Privacy, SP*, 2018.

[521] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *IEEE Symposium on Security and Privacy*, 2013.

[522] P. Laperdrix, B. Baudry, and V. Mishra, "FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques," in *ESSoS*, 2017.

[523] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2012.

[524] L. Olejnik, M. Tran, and C. Castelluccia, "Selling off user privacy at auction," in *Network and Distributed System Security Symposium, NDSS*, 2014.

[525] H. Zang and J. Bolot, "Anonymization of location data does not work: a large-scale measurement study," in *Conference on Mobile Computing and Networking, MOBICOM*, 2011, pp. 145–156.

[526] J. Pang and Y. Zhang, "Deepcity: A feature learning framework for mining location check-ins," in *Conference on Web and Social Media, ICWSM*, 2017.

[527] S. Gambs, M. Killijian, and M. N. del Prado Cortez, "Show me how you move and I will tell you who you are," *Trans. Data Privacy*, 2011.

[528] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, Lester and Pierre: Three protocols for location privacy," in *Privacy Enhancing Technologies*, 2007.

[529] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *NDSS*, 2011.

[530] Z. Lin, D. F. Kune, and N. Hopper, "Efficient private proximity testing with GSM location sketches," in *Financial Cryptography*, 2012.

[531] M. Li, K. Sampigethaya, L. Huang, and R. Poovendran, "Swing & swap: user-centric approaches towards maximizing location privacy," in *WPES*, 2006.

[532] J. Krumm, "Inference attacks on location tracks," in *Pervasive*, 2007.

[533] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *MobiSys*, 2003.

[534] J. Krumm, "Realistic driving trips for location privacy," in *Pervasive*, 2009.

[535] A. Acquisti, I. Adjerid, and L. Brandimarte, "Gone in 15 seconds: The limits of privacy transparency and control," *IEEE Security Privacy*, 2013.

[536] M. Madejski, M. Johnson, and S. M Bellovin, "The failure of online social network privacy settings," Columbia University, Tech. Rep. CUCS-010-11, 2011.

[537] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," in *USENIX Security Symposium*, 2018.

[538] J. Bonneau, J. Anderson, and L. Church, "Privacy suites: shared privacy for social networks," in *Symposium on Usable Privacy and Security, SOUPS*, 2009.

[539] J. Lin, B. Liu, N. M. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Symposium on Usable Privacy and Security, SOUPS*, 2014.

[540] Q. Ismail, T. Ahmed, K. Caine, A. Kapadia, and M. K. Reiter, "To permit or not to permit, that is the usability question: Crowdsourcing mobile apps' privacy permission settings," *PoPETs*, 2017.

[541] World Wide Web Consortium (W3C), "Platform for Privacy Preferences (P3P)," 2007. [Online]. Available: https://www.w3.org/P3P

[542] L. Cranor, M. Langheinrich, and M. Marchiori, "A P3P preference exchange language 1.0 (APPEL 1.0)," World Wide Web Consortium, Working Draft WD-P3P-preferences-20020415, 2002.

[543] J. Byun and N. Li, "Purpose based access control for privacy protection in relational database systems," *VLDB J.*, 2008.

[544] G. Karjoth, M. Schunter, and M. Waidner, "Platform for enterprise privacy practices:

Privacy-enabled management of customer data," in *Privacy Enhancing Technologies, PET*, 2002.

[545] S. Wilson, F. Schaub, R. Ramanath, N. M. Sadeh, F. Liu, N. A. Smith, and F. Liu, "Crowdsourcing annotations for websites' privacy policies: Can it really work?" in *WWW*, 2016.

[546] H. Liu, P. Maes, and G. Davenport, "Unraveling the taste fabric of social networks," *Int. J. Semantic Web Inf. Syst.*, 2006.

[547] Y. Wang, P. G. Leon, K. Scott, X. Chen, A. Acquisti, and L. F. Cranor, "Privacy nudges for social media: an exploratory facebook study," in *International World Wide Web Conference, WWW*, 2013, pp. 763–770.

[548] A. Acquisti, I. Adjerid, R. Balebako, L. Brandimarte, L. F. Cranor, S. Komanduri, P. G. Leon, N. Sadeh, F. Schaub, M. Sleeper, Y. Wang, and S. Wilson, "Nudges for privacy and security: Understanding and assisting users' choices online," *ACM Comput. Surv.*, 2017.

[549] T. Paul, D. Puscher, and T. Strufe, "Improving the usability of privacy settings in facebook," *CoRR*, vol. abs/1109.6046, 2011.

[550] D. Biswas and V. Niemi, "Transforming privacy policies to auditing specifications," in *HASE*, 2011.

[551] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux, "UnLynx: A decentralized system for privacy-conscious data sharing," *PoPETs*, 2017.

[552] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *ACM Conference on Computer and Communications Security*. ACM, 2001.

[553] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "SoK: Making sense of censorship resistance systems," *PoPETs*, 2016.

[554] D. J. Solove, ""I've got nothing to hide" and other misunderstandings of privacy," *San Diego Law Review*, 2007.

[555] J. Benaloh, R. L. Rivest, P. Y. A. Ryan, P. B. Stark, V. Teague, and P. L. Vora, "End-to-end verifiability," *CoRR*, vol. abs/1504.03778, 2015.

[556] D. Boneh and P. Golle, "Almost entirely correct mixing with applications to voting," in *ACM Conference on Computer and Communications Security, CCS*, 2002.

[557] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *USENIX Security Symposium*, 2002.

[558] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *AUSCRYPT*, 1992.

[559] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security Symposium*, 2008.

[560] A. Kiayias, M. Korman, and D. Walluck, "An internet voting system supporting user privacy," in *Annual Computer Security Applications Conference (ACSAC 2006)*, 2006.

[561] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Towards Trustworthy Elections*, 2010.

[562] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *2008 IEEE Symposium on Security and Privacy (S&P)*, 2008.

[563] J. Berg, "The dark side of e-petitions? Exploring anonymous signatures," 2017.

[564] C. Diaz, E. Kosta, H. Dekeyser, M. Kohlweiss, and G. Nigusse, "Privacy preserving electronic petitions," *Identity in the Information Society*, 2008.

[565] A. Sonnino, M. Al-Bassam, S. Bano, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," *CoRR*, vol. abs/1802.07344, 2018. [Online]. Available: http://arxiv.org/abs/1802.07344

[566] M. C. Tschantz, S. Afroz, anonymous, and V. Paxson, "SoK: Towards grounding censorship circumvention in empiricism," in *IEEE Symposium on Security and Privacy, SP*, 2016.

[567] R. Newman, "The Church of Scientology vs. anon.penet.fi." [Online]. Available: http://www.spaink.net/cos/rnewman/anon/penet.html

[568] R. Anderson, "The eternity service," in *Proceedings of Pragocrypt '96*, 1996.

[569] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[570] G. Tian, Z. Duan, T. Baumeister, and Y. Dong, "A traceback attack on freenet," in *INFOCOM*, 2013.

[571] S. Roos, F. Platzer, J. Heller, and T. Strufe, "Inferring obfuscated values in Freenet," in *NetSys*, 2015.

[572] B. Levine, M. Liberatore, B. Lynn, and M. Wright, "Statistical detection of downloaders in freenet," in *IWPE@SP*, 2017.

[573] M. Waldman and D. Mazières, "Tangler: a censorship-resistant publishing system based on document entanglements," in *ACM Conference on Computer and Communications Security*, 2001.

[574] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph: protocol obfuscation for Tor bridges," in *ACM Conference on Computer and Communications Security*, 2012.

[575] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "StegoTorus: a camouflage proxy for the Tor anonymity system," in *ACM Conference on Computer and Communications Security*, 2012.

[576] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *IEEE Symposium on Security and Privacy*, 2013.

[577] C. Brubaker, A. Houmansadr, and V. Shmatikov, "Cloudtransport: Using cloud storage for censorship-resistant networking," in *Privacy Enhancing Technologies*, 2014.

[578] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," *PoPETs*, 2015.

[579] R. McPherson, A. Houmansadr, and V. Shmatikov, "CovertCast: Using live streaming to evade internet censorship," *PoPETs*, 2016.

[580] The Tor Project, "Tor Pluggable Transports," 2007. [Online]. Available: https://obfuscation.github.io/

[581] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. Mankins, and W. T. Strayer, "Decoy routing: Toward unblockable internet communication," in *FOCI*, 2011.

[582] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the network infrastructure," in *USENIX Security Symposium*, 2011.

[583] C. Bocovich and I. Goldberg, "Secure asymmetry and deployability for decoy routing systems," *PoPETs*, 2018.

[584] M. Nasr, H. Zolfaghari, and A. Houmansadr, "The waterfall of liberty: Decoy routing circumvention that resists routing attacks," in *ACM Conference on Computer and Communications Security*, 2017.

[585] S. Gürses, C. Troncoso, and C. Diaz, "Engineering privacy by design reloaded," in *Amsterdam Privacy Conference*, 2015.

[586] I. Wagner and D. Eckhoff, "Technical privacy metrics: A systematic survey," *ACM Comput. Surv.*, 2018.

[587] J. Hoepman, "Privacy design strategies - (extended abstract)," in *29th IFIP TC 11 International Conference, SEC*, 2014.

[588] M. Sikorski and A. Honig, *Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[589] W. Stallings and L. Brown, *Computer Security: Principles and Practice, 4th Edition*. Pearson, 2018.

[590] McAfee, "Fileless malware execution with powershell is easier than you may realize," 2017. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-fileless-malware-execution.pdf

[591] ars TECHNICA, "A rash of invisible, fileless malware is infecting banks around the globe," 2017. [Online]. Available: https://arstechnica.com/information-technology/2017/02/a-rash-of-invisible-fileless-malware-is-infecting-banks-around-the-globe/?comments=1&post=32786675

[592] Lockheed Martin, "The cyber kill chain." [Online]. Available: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[593] MITRE, "ATT&CK knowledge base." [Online]. Available: https://attack.mitre.org

[594] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution." in *The Network and Distributed System Security Symposium (NDSS)*, 2016.

[595] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel *et al.*, "Sok: state of the art of war: Offensive techniques in binary analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 138–157.

[596] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in *The Network and Distributed System Security Symposium (NDSS)*, 2008.

[597] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A platform for in-vivo multi-path analysis of software systems," *ACM Sigplan Notices*, vol. 46, no. 3, pp. 265–278, 2011.

[598] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing mayhem on binary code," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2012, pp. 380–394.

[599] D. A. Ramos and D. R. Engler, "Under-constrained symbolic execution: Correctness checking for real code." in *USENIX Security Symposium*, 2015, pp. 49–64.

[600] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson, "GQ: Practical containment for measuring modern malware systems," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 397–412.

[601] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 51–62.

[602] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *IEEE Symposium on Security and Privacy*. IEEE, 2007.

[603] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection." in *USENIX Security Symposium*, 2014, pp. 287–301.

[604] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Automatic reverse engineering of malware emulators," in *30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 94–109.

[605] S. Mariani, L. Fontana, F. Gritti, and S. D'Alessio, "PinDemonium: a DBI-based generic unpacker for Windows executables," in *Black Hat USA 2016*, 2016.

[606] E. Kenneally, M. Bailey, and D. Maughan, "A framework for understanding and applying ethical principles in network and security research," in *Workshop on Ethics in Computer Security Research (WECSR '10)*, 2010.

[607] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *Journal in computer virology*, vol. 7, no. 2, pp. 107–119, 2011.

[608] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197.

[609] M. Zalewski, "American fuzzy lop." [Online]. Available: http://lcamtuf.coredump.cx/afl/

[610] I. Yun, S. Lee, M. Xu, Y. Jang, and T. Kim, "QSYM: A practical concolic execution engine tailored for hybrid fuzzing," in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[611] C. Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," in *Communications of the ACM*, 2013.

[612] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, "BAP: A binary analysis platform," in *International Conference on Computer Aided Verification*. Springer, 2011.

[613] C. Cadar, D. Dunbar, and D. R. Engler, "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *8th USENIX Symposium on Operating Systems Design and Implementation*, vol. 8, 2008, pp. 209–224.

[614] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "BitBlaze: A new approach to computer security via binary analysis," in *International Conference on Information Systems Security*. Springer, 2008, pp. 1–25.

[615] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, "Directed greybox fuzzing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2329–2344.

[616] V. Kuznetsov, J. Kinder, S. Bucur, and G. Candea, "Efficient state merging in symbolic execution," *ACM Sigplan Notices*, vol. 47, no. 6, pp. 193–204, 2012.

[617] T. Avgerinos, A. Rebert, S. K. Cha, and D. Brumley, "Enhancing symbolic execution with veritesting," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1083–1094.

[618] "The unicorn emulator." [Online]. Available: https://www.unicorn-engine.org/

[619] "The QEMU emulator." [Online]. Available: https://www.qemu.org/

[620] "The bochs emulator." [Online]. Available: http://bochs.sourceforge.net/

[621] "The VirtualBox." [Online]. Available: https://www.virtualbox.org/

[622] "The KVM." [Online]. Available: https://www.linux-kvm.org/

[623] "The VMware." [Online]. Available: https://www.vmware.com/

[624] "The VMware ESXi." [Online]. Available: https://www.vmware.com/products/esxi-and-esx.html/

[625] "The Hyper-V." [Online]. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/

[626] "The Xen." [Online]. Available: https://www.xenproject.org/

[627] P. Royal, "Entrapment: Tricking malware with transparent, scalable malware analysis," 2012, talk at Black Hat.

[628] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting system emulators," in *International Conference on Information Security*. Springer, 2007, pp. 1–18.

[629] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux Malware," in *IEEE Symposium on Security & Privacy*, 2018.

[630] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 1009–1024.

[631] J. T. Bennett, N. Moran, and N. Villeneuve, "Poison ivy: Assessing damage and extracting intelligence," *FireEye Threat Research Blog*, 2013.

[632] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, "Protocol-independent adaptive replay of application dialog." in *NDSS*, 2006.

[633] J. Caballero and D. Song, "Automatic protocol reverse-engineering: Message format extraction and field semantics inference," *Computer Networks*, vol. 57, no. 2, pp. 451–474, 2013.

[634] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 481–500.

[635] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 2003, pp. 290–299.

[636] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee, "Impeding malware analysis using conditional code obfuscation," in *NDSS*, 2008.

[637] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 421–430.

[638] G. Bonfante, J. Fernandez, J.-Y. Marion, B. Rouxel, F. Sabatier, and A. Thierry, "Codisasm: medium scale concatic disassembly of self-modifying binaries with overlapping instructions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 745–756.

[639] "Vmprotect." [Online]. Available: https://vmpsoft.com

[640] R. R. Branco, G. N. Barbosa, and P. D. Neto, "Scientific but not academical overview of malware anti-debugging, anti-disassembly and AntiVM technologies," in *Anti-Disassembly and Anti-VM Technologies, Black Hat USA Conference*, 2012.

[641] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained malware analysis using stealth localized-executions," in *IEEE Symposium on Security and Privacy*. IEEE, 2006.

[642] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[643] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su, "X-Force: Force-executing binary programs for security applications," in *The 23rd USENIX Security Symposium (USENIX Security 14*, 2014, pp. 829–844.

[644] L.-K. Yan, M. Jayachandra, M. Zhang, and H. Yin, "V2E: Combining hardware virtualization and softwareemulation for transparent and extensible malware analysis," *ACM Sigplan Notices*, vol. 47, no. 7, pp. 227–238, 2012.

[645] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 289–300.

[646] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *USENIX Security*, 2006.

[647] D. Denning and P. G. Neumann, *Requirements and model for IDES-a real-time intrusion-detection expert system*. SRI International, 1985.

[648] H. S. Javitz and A. Valdes, "The NIDES statistical component: Description and justification," *Contract*, vol. 39, no. 92-C, p. 0015, 1993. [Online]. Available: http://www.csl.sri.com/papers/statreport/

[649] K. Ilgun, R. Kemmerer, and P. Porras, "State transition analysis: A rule-based intrusion detection approach," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, pp. 181–199, 1995.

[650] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[651] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[652] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE, 1999, pp. 120–132.

[653] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 255–264.

[654] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 15–pp.

[655] A. Kantchelian, J. D. Tygar, and A. D. Joseph, "Evasion and hardening of tree ensemble classifiers," *arXiv preprint arXiv:1509.07892*, 2015.

[656] G. Cleary, M. Corpin, O. Cox, H. Lau, B. Nahorney, D. O'Brien, B. O'Gorman, J.-P. Power, S. Wallace, P. Wood, and C. Wueest, "Internet security threat report," Symantec, Tech. Rep., 2018.

[657] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *USENIX security symposium*, 2010, pp. 273–290.

[658] C. Kolbitsch, E. Kirda, and C. Kruegel, "The power of procrastination: detection and mitigation of execution-stalling malicious code," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 285–296.

[659] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 203–222.

[660] P. Szor, *The Art of Computer Virus Research and Defense*. Symantec Press, 2005, ch. Advanced code evolution techniques and computer virus generator kits.

[661] K. Stevens and D. Jackson, "Zeus banking trojan report," *Atlanta: SecureWorks*, 2010.

[662] N. Falliere and E. Chien, "Zeus: King of the bots," Symantec, Tech. Rep. Security Response, 2009. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/security-response-zeus-king-of-bots-09-en.pdf

[663] B. Krebs, "Feds to charge alleged SpyEye trojan author." [Online]. Available: https://krebsonsecurity.com/2014/01/feds-to-charge-alleged-spyeye-trojan-author/#more-24554

[664] D. Gilbert, "Inside SpyEye: How the russian hacker behind the billion-dollar malware was taken down," Oct 2017, international Business Times. [Online]. Available: https://www.ibtimes.com/inside-spyeye-how-russian-hacker-behind-billion-dollar-malware-was-taken-down-2357477

[665] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai botnet," in *USENIX Security Symposium*, 2017, pp. 1093–1110.

[666] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for UNIX processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.

[667] S. Hao, A. Kantchelian, B. Miller, V. Paxson, and N. Feamster, "Predator: proactive recognition and elimination of domain abuse at time-of-registration," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1568–1579.

[668] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse." in *NDSS*, 2014.

[669] D. Y. Huang, D. McCoy, M. M. Aliapoulios, V. G. Li, L. Invernizzi, E. Bursztein, K. McRoberts, J. Levin, K. Levchenko, and A. C. Snoeren, "Tracking ransomware end-to-end," in *Tracking Ransomware End-to-end*. IEEE Symposium on Security & Privacy, 2018.

[670] Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, "Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking," in *27th USENIX Security Symposium*. USENIX Association, 2018, pp. 1705–1722.

[671] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting malware infection through IDS-driven dialog correlation," in *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[672] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and Efficient Malware Detection at the End Host," in *USENIX security symposium*, 2009, pp. 351–366.

[673] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *USENIX security symposium*, vol. 12, 2012.

[674] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[675] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[676] M. McCoyd and D. Wagner, "Background class defense against adversarial examples," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 96–102.

[677] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 463–480.

[678] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 39–57.

[679] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[680] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "LEMNA: Explaining deep learning based security applications," in *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS '18)*, 2018.

[681] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, and others, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 99–108.

[682] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proceedings of the 26th USENIX Security Symposium*, 2017.

[683] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, "Beheading hydras: Performing effective botnet takedowns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 121–132.

[684] S. Alrwais, X. Liao, X. Mi, P. Wang, X. Wang, F. Qian, R. Beyah, and D. McCoy, "Under the shadow of sunshine: Understanding and detecting bulletproof hosting on legitimate service provider networks," in *IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 805–823.

[685] S. Alrabaee, P. Shirani, M. Debbabi, and L. Wang, "On the feasibility of malware authorship attribution," in *International Symposium on Foundations and Practice of Security*. Springer, 2016, pp. 256–272.

[686] Y. Chen, P. Kintis, M. Antonakakis, Y. Nadji, D. Dagon, W. Lee, and M. Farrell, "Financial lower bounds of online advertising abuse," in *International conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 231–254.

[687] Y. Nadji, M. Antonakakis, R. Perdisci, and W. Lee, "Understanding the prevalence and use of alternative plans in malware with network games," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 1–10.

[688] M. Konte, N. Feamster, and J. Jung, "Fast flux service networks: Dynamics and roles in hosting online scams," Georgia Institute of Technology, Tech. Rep., 2008.

[689] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks." in *NDSS*, 2008.

[690] FBI New York Field Office, "Operation ghost click: International cyber ring that infected millions of computers dismantled," April 2012. [Online]. Available: https://www.fbi.gov/news/stories/international-cyber-ring-that-infected-millions-of-computers-dismantled

[691] W. Meng, R. Duan, and W. Lee, "DNS changer remediation study," in *M3AAWG 27th General Meeting*, 2013.

[692] *Civil Action No: 1:11cv1017 (JCC/IDD), Microsoft Corporation v. Dominique Alexander Piatti, Dotfree Group SRO John Does 1–22, Controlling a computer botnet thereby injuring Microsoft and its customers*. UNITED STATES DISTRICT COURT FOR THE EASTERN DISTRICT OF VIRGINIA, Feb 2013.

[693] B. Bartholomew and J. A. Guerrero-Saade, "Wave your false flags!" [Online]. Available: https://securelist.com/wave-your-false-flags/76273/

[694] M. McGuire and S. Dowling, "Cyber crime: A review of the evidence," *Summary of Key Findings and Implications. Home Office Research Report*, vol. 75, 2013.

[695] N. E. Willard, *Cyberbullying and cyberthreats: Responding to the challenge of online social aggression, threats, and distress*. Research Press, 2007.

[696] H. Glickman, "The Nigerian "419" advance fee scams: prank or peril?" *Canadian Journal of African Studies/La Revue Canadienne Des ÉTudes Africaines*, vol. 39, no. 3, pp. 460–489, 2005.

[697] N. Christin, "Traveling the silk road: A measurement analysis of a large anonymous online marketplace," in *international Conference on World Wide Web (WWW)*. ACM, 2013, pp. 213–224.

[698] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 581–590.

[699] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: analysis of a botnet takeover," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2009, pp. 635–647.

[700] T. Jordan and P. Taylor, *Hacktivism and cyberwars: Rebels with a cause?* Routledge, 2004.

[701] K. Zetter, *Countdown to Zero Day: Stuxnet and the launch of the world's first digital weapon*. Broadway books, 2014.

[702] S. Zannettou, T. Caulfield, W. Setzer, M. Sirivianos, G. Stringhini, and J. Blackburn, "Who let the trolls out? towards understanding state-sponsored trolls," *CoRR*, vol. abs/1811.03130, 2018. [Online]. Available: http://arxiv.org/abs/1811.03130

[703] M. Kjaerland, "A classification of computer security incidents based on reported attack data," *Journal of Investigative Psychology and Offender Profiling*, vol. 2, no. 2, pp. 105–120, 2005.

[704] R. Leukfeldt, E. Kleemans, and W. Stol, "A typology of cybercriminal networks: from low-tech all-rounders to high-tech specialists," *Crime, Law, and Social Change*, pp. 21–37, 2017.

[705] K. Thomas, D. Huang, D. Wang, E. Bursztein, C. Grier, T. J. Holt, C. Kruegel, D. McCoy, S. Savage, and G. Vigna, "Framing dependencies introduced by underground commoditization," in *Workshop on the Economics of Information Security*, 2015.

[706] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna, "The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns," *LEET*, vol. 11, pp. 4–4, 2011.

[707] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@ spam: the underground on 140 characters or less," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 27–37.

[708] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th annual computer security applications conference*. ACM, 2010, pp. 1–9.

[709] E. Bursztein, B. Benko, D. Margolis, T. Pietraszek, A. Archer, A. Aquino, A. Pitsillidis, and S. Savage, "Handcrafted fraud and extortion: Manual account hijacking in the wild," in *ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*. ACM, 2014, pp. 347–358.

[710] J. Clough, "A world of difference: The budapest convention of cybercrime and the challenges of harmonisation," *Monash UL Rev.*, vol. 40, p. 698, 2014.

[711] C. Kershaw, S. Nicholas, and A. Walker, "Crime in England and Wales 2007/08: Findings from the british crime survey and police recorded crime," *Home Office Statistical Bulletin*, vol. 7, no. 8, 2008.

[712] Y. Jewkes, *Handbook of Internet Crime*. Routledge, 2010, ch. Public Policing and Internet Crime.

[713] D. Chatzakou, N. Kourtellis, J. Blackburn, E. De Cristofaro, G. Stringhini, and A. Vakali, "Mean birds: Detecting aggression and bullying on Twitter," in *Proceedings of the 2017 ACM on web science conference*. ACM, 2017, pp. 13–22.

[714] R. Slonje and P. K. Smith, "Cyberbullying: Another main type of bullying?" *Scandinavian Journal of Psychology*, vol. 49, no. 2, pp. 147–154, 2008.

[715] R. M. Kowalski and S. P. Limber, "Electronic bullying among middle school students," *Journal of Adolescent Health*, vol. 41, no. 6, pp. S22–S30, 2007.

[716] J. Suler, "The online disinhibition effect," *Cyberpsychology & Behavior*, vol. 7, no. 3, pp. 321–326, 2004.

[717] A. N. Joinson, "Disinhibition and the internet," in *Psychology and the Internet*. Elsevier, 2007, pp. 75–92.

[718] G. E. Hine, J. Onaolapo, E. De Cristofaro, N. Kourtellis, I. Leontiadis, R. Samaras, G. Stringhini, and J. Blackburn, "Kek, cucks, and god emperor Trump: A measurement study of 4chan's politically incorrect forum and its effects on the web," in *International Conference on Web and Social Media (ICWSM)*. AAAI, 2017.

[719] P. Snyder, P. Doerfler, C. Kanich, and D. McCoy, "Fifteen minutes of unwanted fame: Detecting and characterizing doxing," in *Proceedings of the 2017 Internet Measurement Conference*. ACM, 2017, pp. 432–444.

[720] D. Chatzakou, N. Kourtellis, J. Blackburn, E. De Cristofaro, G. Stringhini, and A. Vakali, "Hate is not binary: Studying abusive behavior of #gamergate on Twitter," in *Proceedings of the 28th ACM conference on hypertext and social media*. ACM, 2017, pp. 65–74.

[721] D. Freed, J. Palmer, D. E. Minchala, K. Levy, T. Ristenpart, and N. Dell, "Digital technologies and intimate partner violence: A qualitative analysis with multiple stakeholders," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. CSCW, p. 46, 2017.

[722] D. Freed, J. Palmer, D. Minchala, K. Levy, T. Ristenpart, and N. Dell, ""A stalker's paradise": How intimate partner abusers exploit technology," in *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2018.

[723] A. Mishra and D. Mishra, "Cyber stalking: a challenge for web security," in *Cyber Warfare and Cyber Terrorism*. IGI Global, 2007, pp. 216–226.

[724] B. Wittes, C. Poplin, Q. Jurecic, and C. Spera, "Sextortion: Cybersecurity, teenagers, and remote sexual assault," *Center for Technology at Brookings. Https://www. Brookings. Edu/wp-content/uploads/2016/05/sextortion1-1. Pdf. Accessed*, vol. 16, 2016.

[725] H. Whittle, C. Hamilton-Giachritsis, A. Beech, and G. Collings, "A review of online grooming: Characteristics and concerns," *Aggression and Violent Behavior*, vol. 18, no. 1, pp. 62–70, 2013.

[726] J. Wolak, D. Finkelhor, and K. Mitchell, "Is talking online to unknown people always risky? Distinguishing online interaction styles in a national sample of youth internet users," *Cyberpsychology & Behavior*, vol. 11, no. 3, pp. 340–343, 2008.

[727] A. Rashid, P. Greehwood, and J. Walkerdine, "Technological solutions to offending," in *Understanding and preventing online sexual exploitation of children*. Routledge, 2013, pp. 244–259.

[728] C. May-Chahal, C. Mason, A. Rashid, J. Walkerdine, P. Rayson, and P. Greenwood, "Safeguarding cyborg childhoods: Incorporating the on/offline behaviour of children into everyday social work practices," *British Journal of Social Work*, vol. 44, no. 3, pp. 596–614, 2012.

[729] M. Latapy, C. Magnien, and R. Fournier, "Quantifying paedophile activity in a large P2P system," *Information Processing & Management*, vol. 49, no. 1, pp. 248–263, 2013.

[730] C. Peersman, C. Schulze, A. Rashid, M. Brennan, and C. Fischer, "iCOP: live forensics to reveal previously unknown criminal media on P2P networks," *Digital Investigation*, vol. 18, pp. 50–64, 2016.

[731] C. Guitton, "A review of the available content on Tor hidden services: The case against further development," *Computers in Human Behavior*, vol. 29, no. 6, pp. 2805–2815, 2013.

[732] J. Huang, G. Stringhini, and P. Yong, "Quit playing games with my heart: Understanding online dating scams," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 216–236.

[733] M. Dittus, J. Wright, and M. Graham, "Platform criminalism: The'last-mile'geography of the darknet market supply chain," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 277–286.

[734] M. T. Whitty and T. Buchanan, "The online romance scam: A serious cybercrime," *Cyberpsychology, Behavior, and Social Networking*, vol. 15, no. 3, pp. 181–183, 2012.

[735] Y. Park, D. McCoy, and E. Shi, "Understanding craigslist rental scams," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 3–21.

[736] M. Edwards, G. Suarez-Tangil, C. Peersman, G. Stringhini, A. Rashid, and M. Whitty, "The geography of online dating fraud," in *Workshop on Technology and Consumer Protection*. IEEE, 2018.

[737] C. Herley, "Why do Nigerian scammers say they are from Nigeria?" in *Workshop on the Economics of Information Security (WEIS)*, 2012.

[738] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The second generation onion router," in *Usenix Security*, 2004.

[739] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," online, 2008.

[740] K. Soska and N. Christin, "Measuring the longitudinal evolution of the online anonymous marketplace ecosystem," in *USENIX Security Symposium*, 2015, pp. 33–48.

[741] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 41–52.

[742] B. Krebs, *Spam nation: The inside story of organized cybercrime-from global epidemic to your front door*. Sourcebooks, Inc., 2014.

[743] S. Hinde, "Spam: the evolution of a nuisance," *Computers & Security*, vol. 22, no. 6, pp. 474–478, 2003.

[744] B. S. McWilliams, *Spam kings: the real story behind the high-rolling hucksters pushing porn, pills, and%*@)# enlargements*. " O'Reilly Media, Inc.", 2014.

[745] G. Stringhini, O. Hohlfeld, C. Kruegel, and G. Vigna, "The harvester, the botmaster, and the spammer: on the relations between the different actors in the spam landscape," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, pp. 353–364.

[746] D. McCoy, A. Pitsillidis, G. Jordan, N. Weaver, C. Kreibich, B. Krebs, G. M. Voelker, S. Savage, and K. Levchenko, "Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs," in *USENIX Security Symposium*. USENIX Association, 2012, pp. 1–1.

[747] D. Samosseiko, "The Partnerka—What is it, and why should you care," in *Proc. of Virus Bulletin Conference*, 2009.

[748] N. Spirin and J. Han, "Survey on web spam detection: principles and algorithms," *Acm Sigkdd Explorations Newsletter*, vol. 13, no. 2, pp. 50–64, 2012.

[749] X. Han, N. Kheir, and D. Balzarotti, "Phisheye: Live monitoring of sandboxed phishing kits," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 1402–1413.

[750] T. Moore and R. Clayton, "Examining the impact of website take-down on phishing," in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*. ACM, 2007, pp. 1–13.

[751] ——, "Evil searching: Compromise and recompromise of internet hosts for phishing," in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 256–272.

[752] J. Onaolapo, E. Mariconti, and G. Stringhini, "What happens after you are pwnd: Understanding the use of leaked webmail credentials in the wild," in *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016, pp. 65–79.

[753] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the zeus botnet crimeware toolkit," in *Annual International Conference on Privacy Security and Trust (PST)*. IEEE, 2010, pp. 31–38.

[754] A. Haslebacher, J. Onaolapo, and G. Stringhini, "All your cards are belong to us:

Understanding online carding forums," in *Electronic Crime Research (eCrime), 2017 APWG Symposium on*. IEEE, 2017, pp. 41–51.

[755] N. Scaife, C. Peeters, and P. Traynor, "Fear the reaper: characterization and fast detection of card skimmers," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1–14.

[756] M. McCormick, "Data theft: a prototypical insider threat," in *Insider Attack and Cyber Security*. Springer, 2008, pp. 53–68.

[757] J. R. Nurse, O. Buckley, P. A. Legg, M. Goldsmith, S. Creese, G. R. Wright, and M. Whitty, "Understanding insider threat: A framework for characterising attacks," in *2014 IEEE Security and Privacy Workshops*. IEEE, 2014, pp. 214–228.

[758] S. Hao, K. Borgolte, N. Nikiforakis, G. Stringhini, M. Egele, M. Eubanks, B. Krebs, and G. Vigna, "Drops for stuff: An analysis of reshipping mule scams," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 1081–1092.

[759] A. Bouveret, *Cyber Risk for the Financial Sector: A Framework for Quantitative Assessment*. International Monetary Fund, 2018.

[760] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," 2019.

[761] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "MineSweeper: an in-depth look into drive-by cryptocurrency mining and its defense," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1714–1730.

[762] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2018, pp. 70–76.

[763] G. O'Gorman and G. McDonald, *Ransomware: A growing menace*. Symantec Corporation, 2012.

[764] A. L. Young and M. Yung, "Cryptovirology: The birth, neglect, and explosion of ransomware," *Communications of the ACM*, vol. 60, no. 7, pp. 24–26, 2017.

[765] D. Y. Huang, M. M. Aliapoulios, V. G. Li, L. Invernizzi, E. Bursztein, K. McRoberts, J. Levin, K. Levchenko, A. C. Snoeren, and D. McCoy, "Tracking ransomware end-to-end," in *IEEE Symposium on Security and Privacy*. IEEE, 2018, pp. 618–631.

[766] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2015, pp. 3–24.

[767] M. Karami, Y. Park, and D. McCoy, "Stress testing the booters: Understanding and undermining the business of DDoS services," in *Proceedings of the 25th International Conference on World Wide Web*. ACM, 2016, pp. 1033–1043.

[768] B. Brevini, A. Hintz, and P. McCurdy, *Beyond WikiLeaks: implications for the future of communications, journalism and society*. Springer, 2013.

[769] M. Conway, "Cyberterrorism: Hype and reality," *Information Warfare: Separating Hype From Reality*, pp. 73–93, 2007.

[770] T. J. Holt, M. Stonhouse, J. Freilich, and S. M. Chermak, "Examining ideologically motivated cyberattacks performed by far-left groups," *Terrorism and Political Violence*, vol. 0, pp. 1–22, 2019.

[771] S. Milan, *Routledge Companion to Alternative and Community Media*. London: Routledge, 2015, ch. Hactivism as a radical media practice.

[772] L. Goode, "Anonymous and the political ethos of hacktivism," *Popular Communication*, vol. 13, no. 1, pp. 74–86, 2015.

[773] G. Greenwald, *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Macmillan, 2014.

[774] H.-j. Woo, Y. Kim, and J. Dominick, "Hackers: Militants or merry pranksters? A content analysis of defaced web pages," *Media Psychology*, vol. 6, no. 1, pp. 63–82, 2004.

[775] A. K. Al-Rawi, "Cyber warriors in the middle east: The case of the syrian electronic army," *Public Relations Review*, vol. 40, no. 3, pp. 420–428, 2014.

[776] D. Maimon, A. Fukuda, S. Hinton, O. Babko-Malaya, and R. Cathey, "On the relevance of social media platforms in predicting the volume and patterns of web defacement attacks," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 4668–4673.

[777] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 833–844.

[778] M. B. Line, A. Zand, G. Stringhini, and R. Kemmerer, "Targeted attacks against industrial control systems: Is the power industry prepared?" in *Proceedings of the 2nd Workshop on Smart Energy Grid Security*. ACM, 2014, pp. 13–22.

[779] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[780] J. Slay and M. Miller, "Lessons learned from the Maroochy water breach," in *Critical Infrastructure Protection*, vol. 253/2007. Springer Boston, November 2007, pp. 73–82.

[781] S. Le Blond, A. Uritesc, C. Gilbert, Z. L. Chua, P. Saxena, and E. Kirda, "A look at targeted attacks through the lense of an NGO." in *USENIX Security Symposium*. USENIX Association, 2014, pp. 543–558.

[782] D. Alperovitch *et al.*, *Revealed: operation shady RAT*. McAfee, 2011, vol. 3.

[783] A. Badawy, E. Ferrara, and K. Lerman, "Analyzing the digital traces of political manipulation: The 2016 Russian interference Twitter campaign," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 258–265.

[784] K. Starbird, A. Arif, T. Wilson, K. Van Koevering, K. Yefimova, and D. Scarnecchia, "Ecosystem or echo-system? exploring content sharing across alternative media domains," in *Twelfth International AAAI Conference on Web and Social Media*, 2018.

[785] S. Zannettou, T. Caulfield, E. De Cristofaro, M. Sirivianos, G. Stringhini, and J. Blackburn, "Disinformation warfare: Understanding state-sponsored trolls on Twitter and their influence on the web," in *Companion Proceedings of The 2019 World Wide Web Conference*. ACM, 2019, pp. 218–226.

[786] S. Zannettou, M. Sirivianos, J. Blackburn, and N. Kourtellis, "The web of false information: Rumors, fake news, hoaxes, clickbait, and various other shenanigans," *Journal of Data and Information Quality (JDIQ)*, vol. 11, no. 3, p. 10, 2019.

[787] K. Levchenko, A. Pitsillidis, N. Chachra, B. Enright, M. Félegyházi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu *et al.*, "Click trajectories: End-to-end analysis of the spam value chain," in *IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 431–446.

[788] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu *et al.*, "The ghost in the browser: Analysis of web-based malware," *HotBots*, vol. 7, pp. 4–4, 2007.

[789] B. Stone-Gross, C. Kruegel, K. Almeroth, A. Moser, and E. Kirda, "FIRE: finding rogue networks," in *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2009, pp. 231–240.

[790] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis *et al.*, "Manufacturing compromise: the emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 821–832.

[791] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: the commoditization of malware distribution," in *USENIX Security Symposium*. USENIX Association, 2011, pp. 13–13.

[792] B. Stone-Gross, R. Abman, R. A. Kemmerer, C. Kruegel, D. G. Steigerwald, and G. Vigna, "The underground economy of fake antivirus software," in *Workshop on the Economics of Information Security and Privacy (WEIS)*. Springer, 2013, pp. 55–78.

[793] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2013, pp. 127–140.

[794] P. Knight, "ILOVEYOU: viruses, paranoia, and the environment of risk," *The Sociological Review*, vol. 48, no. S2, pp. 17–30, 2000.

[795] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks," *Journal of Information Security and Applications*, vol. 22, pp. 113–122, 2015.

[796] D. Y. Wang, M. Der, M. Karami, L. Saul, D. McCoy, S. Savage, and G. M. Voelker, "Search+ seizure: The effectiveness of interventions on SEO campaigns," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2014, pp. 359–372.

[797] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 281–290. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772720

[798] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna, "The dark alleys of madison avenue: Understanding malicious advertisements," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 373–380.

[799] M. Konte, R. Perdisci, and N. Feamster, "Aswatch: An as reputation system to expose bulletproof hosting ases," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 625–638.

[800] C. Y. Cho, J. Caballero, C. Grier, V. Paxson, and D. Song, "Insights from the inside: A view of botnet management from infiltration," *LEET*, vol. 10, pp. 1–1, 2010.

[801] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamcraft: An inside look at spam campaign orchestration," in *LEET*. USENIX Association, 2009.

[802] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with DNS traffic analysis," *IEEE/Acm Transactions on Networking*, vol. 20, no. 5, pp. 1663–1677, 2012.

[803] J. Lusthaus, *Industry of Anonymity: Inside the Business of Cybercrime*. Harvard University Press, 2018.

[804] J. Iedemska, G. Stringhini, R. Kemmerer, C. Kruegel, and G. Vigna, "The tricks of the trade: What makes spam campaigns successful?" in *Security and Privacy Workshops (SPW), 2014 IEEE*. IEEE, 2014, pp. 77–83.

[805] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re: CAPTCHAs-Understanding CAPTCHA-Solving services in an economic context," in *USENIX Security Symposium*, vol. 10. USENIX Association, 2010, p. 3.

[806] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: The role of the underground market in Twitter spam and abuse," in *USENIX Security Symposium*. USENIX Association, 2013, pp. 195–210.

[807] E. De Cristofaro, A. Friedman, G. Jourjon, M. A. Kaafar, and M. Z. Shafiq, "Paying for likes?: Understanding facebook like fraud using honeypots," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2014, pp. 129–136.

[808] G. Stringhini, G. Wang, M. Egele, C. Kruegel, G. Vigna, H. Zheng, and B. Y. Zhao, "Follow the green: growth and dynamics in Twitter follower markets," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2013, pp. 163–176.

[809] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker, "Dirty jobs: The role of freelance labor in web service abuse," in *USENIX Security Symposium*. USENIX Association, 2011, pp. 14–14.

[810] D. Florêncio and C. Herley, "Phishing and money mules," in *International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2010, pp. 1–5.

[811] R. Anderson, C. Barton, R. Böhme, R. Clayton, M. J. Van Eeten, M. Levi, T. Moore, and S. Savage, "Measuring the cost of cybercrime," in *Workshop on the economics of information security and privacy (WEIS)*. Springer, 2013, pp. 265–300.

[812] T. Moore and N. Christin, "Beware the middleman: Empirical analysis of bitcoin-exchange risk," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 25–33.

[813] K. Moeller, R. Munksgaard, and J. Demant, "Flow my FE the vendor said: Exploring violent and fraudulent resource exchanges on cryptomarkets for illicit drugs," *American Behavioral Scientist*, vol. 61, no. 11, pp. 1427–1450, 2017.

[814] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[815] P. J. Brantingham, P. L. Brantingham *et al.*, *Environmental criminology*. Sage Publications Beverly Hills, CA, 1981.

[816] D. N. Khey and V. A. Sainato, "Examining the correlates and spatial distribution of organizational data breaches in the United States," *Security Journal*, vol. 26, no. 4, pp. 367–382, 2013.

[817] S. Hinduja and B. Kooi, "Curtailing cyber and information security vulnerabilities through situational crime prevention," *Security Journal*, vol. 26, no. 4, pp. 383–402, 2013.

[818] T. Rid and B. Buchanan, "Attributing cyber attacks," *Journal of Strategic Studies*, vol. 38, no. 1-2, pp. 4–37, 2015.

[819] A. PAdmos, https://github.com/arnepadmos/resources/tree/master/methodologies/model, 2017.

[820] L. P. Swiler and C. Phillips, "A graph-based system for network-vulnerability analysis," Sandia National Labs., Albuquerque, NM (United States), Tech. Rep., 1998.

[821] J. P. McDermott, "Attack net penetration testing," in *NSPW*, 2000, pp. 15–21.

[822] B. Farinholt, M. Rezaeirad, P. Pearce, H. Dharmdasani, H. Yin, S. Le Blond, D. McCoy, and K. Levchenko, "To catch a ratter: Monitoring the behavior of amateur DarkComet RAT operators in the wild," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 770–787.

[823] B. Cheswick, "An evening with Berferd in which a cracker is lured, endured, and studied," in *USENIX Security Symposium*, 1992, pp. 20–24.

[824] D. Moore, C. Shannon *et al.*, "Code-red: a case study on the spread and victims of an internet worm," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 273–284.

[825] A. Dwyer, "The NHS cyber-attack: A look at the complex environmental conditions of WannaCry," *RAD Magazine*, vol. 44, 2018.

[826] L. E. Cohen and M. Felson, "Social change and crime rate trends: A routine activity approach (1979)," in *Classics in Environmental Criminology*. CRC Press, 2016, pp. 203–232.

[827] R. V. Clarke and D. B. Cornish, "Modeling offenders' decisions: A framework for research and policy," *Crime and Justice*, vol. 6, pp. 147–185, 1985.

[828] R. V. G. Clarke, *Situational crime prevention*. Criminal Justice Press Monsey, NY, 1997.

[829] M. R. Albert, "E-buyer beware: why online auction fraud should be regulated," *American Business Law Journal*, vol. 39, no. 4, pp. 575–644, 2002.

[830] H. Liu, K. Levchenko, M. Félegyházi, C. Kreibich, G. Maier, G. M. Voelker, and S. Savage, "On the effects of registrar-level intervention," in *LEET*, 2011.

[831] D. B. Cornish, "The procedural analysis of offending and its relevance for situational prevention," *Crime Prevention Studies*, vol. 3, pp. 151–196, 1994.

[832] R. Wortley, A. Sidebottom, N. Tilley, and G. Laycock, *Routledge Handbook of Crime Science*. Routledge, 2018.

[833] D. Huang, K. Thomas, C. Grier, D. Wang, E. Burztein, T. Holt, C. Kruegel, D. McCoy, S. Savage, and G. Vigna, "Framing dependencies introduced by underground commoditization," in *Workshop on the Economics of Information Security*, 2015.

[834] J. P. Anderson *et al.*, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[835] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, no. 2, pp. 222–232, 1987.

[836] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, p. 7, 2008.

[837] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 3, pp. 186–205, Aug. 2000.

[838] A. Patel, M. Taghavi, K. Bakhtiyari, and J. C. Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review," *Journal of network and computer applications*, vol. 36, no. 1, pp. 25–41, 2013.

[839] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

[840] R. Sommer and A. Feldmann, "Netflow: Information loss or win?" in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. ACM, 2002, pp. 173–174.

[841] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[842] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju, "Visflowconnect: netflow visualizations of link relationships for security situational awareness," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*. ACM, 2004, pp. 26–34.

[843] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: techniques and challenges," *Computers & Security*, vol. 70, pp. 238–254, 2017.

[844] T. Deshpande, P. Katsaros, S. Basagiannis, and S. A. Smolka, "Formal analysis of the DNS bandwidth amplification attack and its countermeasures using probabilistic model checking," in *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*. IEEE, 2011, pp. 360–367.

[845] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS amplification attack revisited," *Computers & Security*, vol. 39, pp. 475–485, 2013.

[846] A. Herzberg and H. Shulman, "DNS authentication as a service: preventing amplification attacks," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 356–365.

[847] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its potential for DDoS attacks: a comprehensive measurement study," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 449–460.

[848] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive DNS analysis service to detect and report malicious domains," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 14, 2014.

[849] A. Ramachandran, D. Dagon, and N. Feamster, "Can DNS-based blacklists keep up with bots?" in *CEAS*. Citeseer, 2006.

[850] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 435–448.

[851] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from hell? reducing the impact of amplification DDoS attacks." in *USENIX Security Symposium*, 2014, pp. 111–125.

[852] N. Feamster, J. Jung, and H. Balakrishnan, "An empirical study of bogon route advertisements," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, pp. 63–70, 2005.

[853] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 291–302.

[854] E. Biersack, Q. Jacquemart, F. Fischer, J. Fuchs, O. Thonnard, G. Theodoridis, D. Tzovaras, and P.-A. Vervier, "Visual analytics for BGP monitoring and prefix hijacking identification," *IEEE Network*, vol. 26, no. 6, 2012.

[855] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack, "Heap: reliable assessment of bgp hijacking attacks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1849–1861, 2016.

[856] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[857] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4. ACM, 2006, pp. 15–27.

[858] X. Chen, H. Bos, and C. Giuffrida, "Codearmor: Virtualizing the code space to counter disclosure attacks," in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 2017, pp. 514–529.

[859] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[860] S. Patton, W. Yurcik, and D. Doss, "An achilles' heel in signature-based ids: Squealing false positives in snort," in *Proceedings of RAID*, vol. 2001. Citeseer, 2001.

[861] E. Albin and N. C. Rowe, "A realistic experimental comparison of the Suricata and Snort intrusion-detection systems," in *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*. IEEE, 2012, pp. 122–127.

[862] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, 2011.

[863] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection:

methods, systems and tools," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[864] T. Cruz, L. Rosa, J. Proença, L. Maglaras, M. Aubigny, L. Lev, J. Jiang, and P. Simoes, "A cybersecurity detection framework for supervisory control and data acquisition systems," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2236–2246, 2016.

[865] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "MADAM: a multi-level anomaly detector for android malware," in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer, 2012, pp. 240–253.

[866] M. Almgren, H. Debar, and M. Dacier, "A lightweight tool for detecting web server attacks." in *Proceedings of NDSS*, 2000.

[867] E. Tombini, H. Debar, L. Mé, and M. Ducassé, "A serial combination of anomaly and misuse idses applied to http traffic," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 428–437.

[868] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000.

[869] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, Fourth 2008.

[870] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[871] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*. ACM, 2009, pp. 23–31.

[872] P. Laskov and N. Šrndić, "Static detection of malicious javascript-bearing pdf documents," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 373–382.

[873] D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications security*. ACM, 2013, pp. 119–130.

[874] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 02, p. 56, 2014.

[875] A.-D. Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Çamtepe, and S. Albayrak, "Monitoring smartphones for anomaly detection," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 92–106, 2009.

[876] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.

[877] U. Franke and J. Brynielsson, "Cyber situational awareness–a systematic review of the literature," *Computers & Security*, vol. 46, pp. 18–31, 2014.

[878] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[879] M. A. Erlinger and M. Wood, "Intrusion Detection Message Exchange Requirements," RFC 4766, Mar. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4766.txt

[880] B. Feinstein, D. Curry, and H. Debar, "The Intrusion Detection Message Exchange Format (IDMEF)," RFC 4765, Mar. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4765.txt

[881] G. Matthews and B. Feinstein, "The Intrusion Detection Exchange Protocol (IDXP)," RFC 4767, Mar. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4767.txt

[882] J. Steinberger, A. Sperotto, M. Golling, and H. Baier, "How to exchange security events? overview and evaluation of formats and protocols," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 261–269.

[883] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 85–103.

[884] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, p. 202.

[885] O. S. Saydjari, "Cyber defense: art to science," *Communications of the ACM*, vol. 47, no. 3, pp. 52–57, 2004.

[886] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 245–254.

[887] J. Zhou, M. Heckman, B. Reynolds, A. Carlson, and M. Bishop, "Modeling network intrusion detection alerts for correlation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 1, p. 4, 2007.

[888] B. Morin, L. Mé, H. Debar, and M. Ducassé, "A logic-based model to support alert correlation in intrusion detection," *Information Fusion*, vol. 10, no. 4, pp. 285–299, 2009.

[889] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 54–68.

[890] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 366–375.

[891] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the IOC game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 755–766.

[892] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[893] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Computing Surveys (CSUR)*, vol. 39, no. 1, p. 3, 2007.

[894] N. Hachem, H. Debar, and J. Garcia-Alfaro, "HADEGA: A novel MPLS-based mitigation solution to handle network attacks," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*. IEEE, 2012, pp. 171–180.

[895] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, "ArOMA: An SDN based autonomic DDoS mitigation framework," *Computers & Security*, vol. 70, pp. 482–499, 2017.

[896] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *International Conference on Information Security and Cryptology*. Springer, 2005, pp. 186–198.

[897] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer." in *USENIX Security Symposium*, vol. 8. Baltimore, MD, 2005.

[898] R. Böhme, G. Schwartz *et al.*, "Modeling cyber-insurance: Towards a unifying framework." in *WEIS*, 2010.

[899] A. Motzek, G. Gonzalez-Granadillo, H. Debar, J. Garcia-Alfaro, and R. Möller, "Selection of pareto-efficient response plans based on financial and operational assessments," *EURASIP Journal on Information Security*, vol. 2017, no. 1, p. 12, 2017.

[900] E. Zio, "Reliability engineering: Old problems and new challenges," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 125–141, 2009.

[901] J. Campos, P. Sharma, E. Jantunen, D. Baglee, and L. Fumagalli, "The challenges of cybersecurity frameworks to protect data required for the development of advanced maintenance," *Procedia CIRP*, vol. 47, pp. 222–227, 2016.

[902] L. Spitzner, "Honeypots: Catching the insider threat," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE, 2003, pp. 170–179.

[903] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of internet background radiation," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 27–40.

[904] K.-K. R. Choo, "The cyber threat landscape: Challenges and future research directions," *Computers & Security*, vol. 30, no. 8, pp. 719–731, 2011.

[905] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian, "Darknet and deepnet mining for proactive cybersecurity threat intelligence," *arXiv preprint arXiv:1607.08583*, 2016.

[906] H. Haughey, G. Epiphaniou, H. Al-Khateeb, and A. Dehghantanha, "Adaptive traffic fingerprinting for darknet threat intelligence," *Cyber Threat Intelligence*, pp. 193–217, 2018.

[907] C. Miles, A. Lakhotia, C. LeDoux, A. Newsom, and V. Notani, "Virusbattle: State-of-the-art malware analysis for better cyber threat intelligence," in *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*. IEEE, 2014, pp. 1–6.

[908] S. Samtani, K. Chinn, C. Larson, and H. Chen, "Azsecure hacker assets portal: Cyber threat intelligence and malware analysis," in *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*. Ieee, 2016, pp. 19–24.

[909] S. Qamar, Z. Anwar, M. A. Rahman, E. Al-Shaer, and B.-T. Chu, "Data-driven analytics for cyber-threat intelligence and information sharing," *Computers & Security*, vol. 67, pp. 35–58, 2017.

[910] J. M. Ahrend, M. Jirotka, and K. Jones, "On the collaborative practices of cyber threat intelligence analysts to develop and utilize tacit threat and defence knowledge," in *Cyber Situational Awareness, Data Analytics And Assessment (CyberSA), 2016 International Conference On*. IEEE, 2016, pp. 1–10.

[911] C. Wagner, A. Dulaunoy, G. Wagener, and A. Iklody, "Misp: The design and implementation of a collaborative threat intelligence sharing platform," in *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. ACM, 2016, pp. 49–56.

[912] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," *Human factors*, vol. 37, no. 1, pp. 32–64, 1995.

[913] G. P. Tadda, "Measuring performance of cyber situation awareness systems," in *Information Fusion, 2008 11th International Conference*. IEEE, 2008, pp. 1–8.

[914] S. Mathew, S. Upadhyaya, M. Sudit, and A. Stotz, "Situation awareness of multistage cyber attacks by semantic event fusion," in *Military Communications Conference, 2010-MILCOM 2010*. IEEE, 2010, pp. 1286–1291.

[915] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide," *NIST Special Publication*, vol. 800, no. 61, pp. 1–147, 2012.

[916] A. Ahmad, J. Hadgkiss, and A. B. Ruighaver, "Incident response teams–challenges in supporting the organisational security function," *Computers & Security*, vol. 31, no. 5, pp. 643–652, 2012.

[917] I. A. Tøndel, M. B. Line, and M. G. Jaatun, "Information security incident management: Current practice as reported in the literature," *Computers & Security*, vol. 45, pp. 42–57, 2014.

[918] R. Baskerville, P. Spagnoletti, and J. Kim, "Incident-centered information security: Managing a strategic balance between prevention and response," *Information & management*, vol. 51, no. 1, pp. 138–151, 2014.

[919] E. Casey, *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*, 3rd ed. Academic Press, 2011, iSBN: 978-0123742681.

[920] United Kingdom Parliament. (1990) Computer misuse act. [Online]. Available: https://www.legislation.gov.uk/ukpga/1990/18/contents

[921] D. Goodstein, *Reference Manual on Scientific Evidence: Third Edition*. National Academies Press, 2011, ch. How Science Works, pp. 37–54. [Online]. Available: https://www.fjc.gov/sites/default/files/2015/SciMan3D01.pdf

[922] The Law Commission. (2011) Expert evidence in criminal proceedings in England and Wales. [Online]. Available: https://www.lawcom.gov.uk/project/expert-evidence-in-criminal-proceedings/

[923] K. Kent, S. Chevalier, T. Grance, and H. Dang, "Guide to integrating forensic techniques into incident response," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-86, 2006. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf

[924] Comprehensive Crime Control Act of 1984. [Online]. Available: https://www.ncjrs.gov/App/publications/Abstract.aspx?id=123365

[925] 18 US Code § 1030 - Fraud and related activity in connection with computers. [Online]. Available: https://www.law.cornell.edu/uscode/text/18/1030

[926] D. Goodstein, *Reference Manual on Scientific Evidence: Third Edition*. National Academies Press, 2011, ch. How Science Works, pp. 37–54. [Online]. Available: https://www.fjc.gov/sites/default/files/2015/SciMan3D01.pdf

[927] Forensic Science Regulator. (2014) Codes of practice and conduct, appendix: Digital forensic services FSR-C-107. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/351220/2014.08.28_FSR-C-107_Digital_forensics.pdf

[928] ISO/IEC. (2017) ISO/IEC 17025 General requirements for the competence of testing and calibration laboratories. [Online]. Available: https://webstore.iec.ch/preview/info_isoiec17025%7Bed3.0%7Den.pdf

[929] G. Palmer. (2001) Report from the first digital forensic research workshop (DFRWS). [Online]. Available: https://www.dfrws.org/sites/default/files/session-files/a_road_map_for_digital_forensic_research.pdf

[930] V. Roussev, "Hashing and data fingerprinting in digital forensics," *IEEE Security Privacy*, vol. 7, no. 2, pp. 49–55, March 2009, DOI: 10.1109/MSP.2009.40.

[931] S. Garfinkel, A. J. Nelson, and J. Young, "A general strategy for differential forensic analysis," in *The Proceedings of the Twelfth Annual Digital Forensic Researcg Conference (DFRWS)*, 2012, pp. S50–S59, DOI: 10.1016/j.diin.2012.05.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S174228761200028X

[932] P. Pirolli and S. Card, "Sensemaking processes of intelligence analysts and possible leverage points as identified through cognitive task analysis," in *Proceedings of the 2005 International Conference on Intelligence Analysis*, 2005. [Online]. Available: http://researchgate.net/publication/215439203

[933] J. J. Thomas and K. A. Cook, Eds., *Illuminating the path: the research and development agenda for visual analytics*. US Department of Homeland Security, National Visualization and Analytics Center (NVAC), 2005.

[934] E. Patterson, E. Roth, and D. Woods, "Predicting vulnerabilities in computer-supported inferential analysis under data overload," *Cognition, Technology and Work*, vol. 3, no. 4, pp. 224–237, 2001, DOI: 10.1007/s10111-001-8004-y.

[935] P. Pirolli, *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, 2009, iSBN: 978-0195387797.

[936] G. Klein, B. Moon, and R. Hoffman, "Making sense of sensemaking 1: Alternative perspectives," *IEEE Intelligent Systems*, vol. 21, no. 4, pp. 70–73, 2006, DOI: 10.1109/MIS.2006.75.

[937] V. Roussev, C. Quates, and R. Martell, "Real-time digital forensics and triage," *Digital Investigation*, vol. 10, no. 2, pp. 158–167, 2013, DOI: 10.1016/j.diin.2013.02.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287613000091

[938] D. Farmer and W. Venema, *Forensic Discovery*, 1st ed. Addison-Wesley Professional, 2005, iSBN: 978-0201634976.

[939] B. Carrier, *File System Forensic Analysis*, 1st ed. Addison-Wesley Professional, 2005, ISBN: 978-0321268174.

[940] J. von Neumann, "First draft of a report on the EDVAC," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993, DOI: 10.1109/85.238389.

[941] S. Willassen, "Forensic analysis of mobile phone internal memory," in *Advances in Digital Forensics*. Springer, 2005, pp. 191–204, DOI: 10.1007/0-387-31163-7_16. [Online]. Available: http://dl.ifip.org/db/conf/ifip11-9/df2005/Willassen05.pdf

[942] IEEE. (1990) IEEE 1149.1-1990 - IEEE standard test access port and boundary-scan architecture. [Online]. Available: https://standards.ieee.org/standard/1149_1-1990.html

[943] NVM Express. (2019) NVM express base specification, revision 1.4. [Online]. Available: https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4-2019.06.10-Ratified.pdf

[944] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltsidas, "Implementation and implications of a stealth hard-drive backdoor," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13, 2013, pp. 279–288, DOI: 10.1145/2523649.2523661. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523661

[945] NIST. (2015) Computer forensic tool testing program. [Online]. Available: http://www.cftt.nist.gov/

[946] C. Stevens. (2011) Formatting, cloning and duplicating advanced format media. Technical Paper, Revision 1.0. [Online]. Available: http://idema.org/wp-content/plugins/download-monitor/download.php?id=1220

[947] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, Jun 1994, DOI: 10.1145/176979.176981. [Online]. Available: http://doi.acm.org/10.1145/176979.176981

[948] C. King and T. Vidas, "Empirical analysis of solid state disk data retention when used with contemporary operating systems," in *Proceedings of the 11th Annual DFRWS Conference. DFRWS'11.*, 2011, pp. S111–S117, DOI: 10.1016/j.diin.2011.05.013. [Online]. Available: http://dfrws.org/2011/proceedings/17-349.pdf

[949] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*, 1st ed. Wiley, 2014, iSBN: 978-1118825099.

[950] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, "Understanding data lifetime via whole system simulation," in *Proceedings of the USENIX Security Symposium*, 2004, pp. 321–336. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/chow/chow_html/

[951] J. Solomon, E. Huebner, D. Bem, and M. Szeżynska, "User data persistence in physical memory," *Journal of Digital Investigation*, vol. 4, no. 2, pp. 68–72, 2007, DOI: 10.1016/j.diin.2007.03.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S174228760700028X

[952] S. Jeon, J. Bang, K. Byun, and S. Lee, "A recovery method of deleted record for SQLite database," *Personal and Ubiquitous Computing*, vol. 16, no. 6, pp. 707–715, 2012, DOI: 10.1007/s00779-011-0428-7.

[953] B. Wu, M. Xu, H. Zhang, J. Xu, Y. Ren, and N. Zheng, *A Recovery Approach for SQLite History Recorders from YAFFS2*. Springer Berlin Heidelberg, 2013, pp. 295–299, DOI: 10.1007/978-3-642-36818-9_30.

[954] NIST. (2015) SHA-3 standard: Permutation-based hash and extendable-output functions. FIPS Publication 202, DOI: 10.6028/NIST.FIPS.202. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[955] ——. (2016) National software reference library. [Online]. Available: http://www.nsrl.nist.gov/

[956] S. Garfinkel, A. Nelson, D. White, and V. Roussev, "Using purpose-built functions and block hashes to enable small block and sub-file forensics," in *Proceedings of the Tenth Annual DFRWS Conference. DFRWS'10.*, 2010, DOI: 10.1016/j.diin.2010.05.003. [Online]. Available: http://dfrws.org/2010/proceedings/2010-302.pdf

[957] F. Breitinger, B. Guttman, M. McCarrin, V. Roussev, and D. White, "Approximate matching: Definition and terminology," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-168, 2014. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-168.pdf

[958] V. Roussev and S. McCulley, "Forensic analysis of cloud-native artifacts," in *Proceedings of the Third Annual DFRWS Europe (DFRWS-EU)*, 2016, pp. S104–S113, DOI: 10.1016/j.diin.2016.01.013.

[959] A. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences*, Jun 1997, pp. 21–29, DOI: 10.1109/SEQUEN.1997.666900.

[960] eCrypt-CSA Consortium, "Algorithms, key size and protocols report (2018)," http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf, 2018.

[961] D. Kahn, *The Codebreakers*. Simon and Schuster, 1996.

[962] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[963] N. P. Smart, *Cryptography Made Simple*, ser. Information Security and Cryptography. Springer, 2016.

[964] S. D. Galbraith, *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. [Online]. Available: https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html

[965] O. Goldreich, *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[966] ——, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[967] A. K. Lenstra and H. W. L. Jr., *The Development of the Number Field Sieve*, ser. Lecture Notes in Mathematics. Springer, 1993.

[968] P. Q. Nguyen and B. Vallée, Eds., *The LLL Algorithm - Survey and Applications*, ser. Information Security and Cryptography. Springer, 2010.

[969] D. Welsh, *Codes and Cryptography*. Oxford University Press, 1988.

[970] H. M. Heys, "A tutorial on linear and differential cryptanalysis," https://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf.

[971] E. BIham and O. Dunkelman, *Techniques for Cryptanalysis of Block Ciphers*, ser. Information Security and Cryptography. Springer, 2018.

[972] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, ser. Information Security and Cryptography. Springer, 2002.

[973] eCrypt-II Consortium, "eSTREAM: The ECRYPT Stream Cipher Project," http://www.ecrypt.eu.org/stream/, 2012.

[974] K. W. Site, "Team Keccak," https://keccak.team/, 2018.

[975] NIST-CSRC, "Block cipher techniques: Block cipher modes," https://csrc.nist.gov/Projects/Block-Cipher-Techniques/BCM, 2018.

[976] ——, "Post-quantum cryptography: Post-quantum cryptography standardization," https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2018.

[977] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, ser. Information Security and Cryptography. Springer, 2003. [Online]. Available: https://doi.org/10.1007/978-3-662-09527-0

[978] R. Cramer, I. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. [Online]. Available: http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB&isbn=9781107043053

[979] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols - Techniques and Constructions*, ser. Information Security and Cryptography. Springer, 2010. [Online]. Available: https://doi.org/10.1007/978-3-642-14303-8

[980] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[981] M. Joye and M. Tunstall, Eds., *Fault Analysis in Cryptography*, ser. Information Security and Cryptography. Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-29656-7

[982] D. Sgandurra and E. Lupu, "Evolution of attacks, threat models, and solutions for virtualized systems," *ACM Computing Surveys*, vol. 48, no. 3, pp. 46:1–46:38, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2856126

[983] G. C. Hunt and J. R. Larus, "Singularity: Rethinking the software stack," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 2, pp. 37–49, Apr. 2007. [Online]. Available: http://doi.acm.org/10.1145/1243418.1243424

[984] E. Perla and M. Oldani, *A Guide to Kernel Exploitation: Attacking the Core*. Syngress Publishing, 2010.

[985] A. S. Tanenbaum and H. Bos, *Modern operating systems*. Pearson, 2015.

[986] V. van der Veen, N. Dutt-Sharma, L. Cavallaro, and H. Bos, "Memory errors: The past, the present, and the future," in *RAID*, Oct. 2012. [Online]. Available: http://www.few.vu.nl/~herbertb/papers/memerrors_raid12.pdf

[987] S. Boyd-Wickizer and N. Zeldovich, "Tolerating malicious device drivers in linux," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855849

[988] B. Grill, A. Bacs, C. Platzer, and H. Bos, "Nice boots—a large-scale analysis of bootkits and new ways to stop them," in *DIMVA*, Sep. 2015. [Online]. Available: http://www.cs.vu.nl/%7Eherbertb/papers/bootkits_dimva2015.pdf

[989] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 361–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665726

[990] P. Wang, J. Krinke, K. Lu, G. Li, and S. Dodier-Lazaro, "How double-fetch situations turn into double-fetch vulnerabilities: A study of double fetches in the linux kernel," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 1–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=3241189.3241191

[991] R. N. M. Watson, "Exploiting concurrency vulnerabilities in system call wrappers," in *Proceedings of the First USENIX Workshop on Offensive Technologies*, ser. WOOT '07. Berkeley, CA, USA: USENIX Association, 2007, pp. 2:1–2:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1323276.1323278

[992] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 605–622.

[993] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. Berkeley, CA, USA: USENIX Association, 2018, pp. 973–990. [Online]. Available: http://dl.acm.org/citation.cfm?id=3277203.3277276

[994] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *ArXiv Preprint ArXiv:1801.01203*, 2018.

[995] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. Berkeley, CA, USA: USENIX Association, 2018, pp. 991–1008. [Online]. Available: http://dl.acm.org/citation.cfm?id=3277203.3277277

[996] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: Rogue In-flight Data Load," in *S&P*, May 2019. [Online]. Available: https://mdsattacks.com/files/ridl.pdf

[997] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *USENIX Security*, Aug. 2018. [Online]. Available: https://www.vusec.net/download/?t=papers/tlbleed_sec18.pdf

[998] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup est machina: Memory deduplication as an advanced exploitation vector," in *IEEE Security & Privacy*, May 2016. [Online]. Available: https://www.vusec.net/download/?t=papers/dedup-est-machina_sp16.pdf

[999] P. Larsen and A.-R. Sadeghi, Eds., *The Continuing Arms Race: Code-Reuse Attacks and Defenses*. New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2018.

[1000] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schröder-Preikschat, D. Lohmann, and R. Kapitza, "Attack surface metrics and automated compile-time OS kernel tailoring," in *NDSS*. The Internet Society, 2013. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2013.html#KurmusTDHRRSLK13

[1001] T. Jaeger, *Operating System Security*, 1st ed. Morgan and Claypool Publishers, 2008.

[1002] D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Communications of the ACM*, vol. 17, no. 7, pp. 365–375, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361061

[1003] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A new kernel foundation for UNIX development," Computer Science Department Carnegie Mellon University, Tech. Rep., 1986.

[1004] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "MINIX 3: A highly reliable, self-repairing operating system," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 3, pp. 80–89, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151374.1151391

[1005] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An operating system architecture for application-level resource management," in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '95. New York, NY, USA: ACM, 1995, pp. 251–266. [Online]. Available: http://doi.acm.org/10.1145/224056.224076

[1006] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE J.Sel. A. Commun.*, vol. 14, no. 7, pp. 1280–1297, Sep. 96. [Online]. Available: http://dx.doi.org/10.1109/49.536480

[1007] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," *Queue*, vol. 11, no. 11, pp. 30:30–30:44, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2557963.2566628

[1008] A. S. Tanenbaum, *Operating Systems: Design and Implementation*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[1009] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: A new OS architecture for scalable multicore systems," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 29–44. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629579

[1010] Version 7 Unix, "chroot," Modern chroot man page http://man7.org/linux/man-pages/man2/chroot.2.html, 1979.

[1011] P.-H. Kamp and R. N. M. Watson, "Jails: confining the omnipotent root," in *Proceedings of SANE*, Maastricht, The Netherlands, May 2000.

[1012] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[1013] R. Anderson, *Security Engineering: a guide to building dependable distributed systems*. Wiley, 2008.

[1014] United States Department of Defense, *Department of Defense, Trusted Computer System Evaluation Criteria*, ser. Rainbow Series. Dept. of Defense, 1985, no. 5200.28-STD. [Online]. Available: https://books.google.nl/books?id=-KBPAAAAMAAJ

[1015] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," The MITRE Corporation, Bedford MA, Tech. Rep. ESD-TR-73-278, Nov. 1973.

[1016] K. J. Biba, "Integrity consideration for secure computer systems," The MITRE Corporation, Bedford, MA, Tech. Rep. ESD-TR-76-372, MTR-3153, April 1977.

[1017] P. G. Neumann, *New Solutions for Cybersecurity*. MIT Press, 2017, ch. Fundamental Security Principles.

[1018] D. Ferraiolo and D. Kuhn, "Role-based access controls," in *Proceedings of the 15th Annual National Computer Security Conference*. NSA/NIST, 1992, pp. 554–563.

[1019] H. M. Levy, *Capability-based computer systems*. Digital Press, 1984.

[1020] P. A. Karger and R. R. Schell, "Thirty years later: Lessons from the multics security evaluation," in *Proceedings of the 18th Annual Computer Security Applications Conference*, ser. ACSAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 119–. [Online]. Available: http://dl.acm.org/citation.cfm?id=784592.784794

[1021] J. H. Saltzer, "Protection and the control of information sharing in multics," *Communications of the ACM*, vol. 17, no. 7, pp. 388–402, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361067

[1022] R. C. Daley and P. G. Neumann, "A general-purpose file system for secondary storage," in *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I*, ser. AFIPS '65 (Fall, part I). New York, NY, USA: ACM, 1965, pp. 213–229. [Online]. Available: http://doi.acm.org/10.1145/1463891.1463915

[1023] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a linux security module," *NAI Labs Report*, vol. 1, no. 43, p. 139, 2001.

[1024] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The flask security architecture: System support for diverse security policies," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99.

Berkeley, CA, USA: USENIX Association, 1999, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251432

[1025] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazieres, F. Kaashoek, and R. Morris, "Labels and event processes in the asbestos operating system," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 17–30.

[1026] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières, "Making information flow explicit in HiStar," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 263–278.

[1027] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information flow control for standard OS abstractions," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 321–334.

[1028] J. B. Dennis and E. C. Van Horn, "Programming semantics for multiprogrammed computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, 1966.

[1029] S. J. Mullender and A. S. Tanenbaum, "The design of a capability-based distributed operating system," *The Computer Journal*, vol. 29, no. 4, pp. 289–299, 1986.

[1030] W. B. Ackerman and W. W. Plummer, "An implementation of a multiprocessing computer system," in *Proceedings of the First ACM Symposium on Operating System Principles*, ser. SOSP '67. New York, NY, USA: ACM, 1967, pp. 5.1–5.10. [Online]. Available: http://doi.acm.org/10.1145/800001.811666

[1031] R. Fabry, "A user's view of capabilities," ICR Quarterly Report. U. of Chicago Institute for Computer Research, pp. pages C1–C8, Nov. 1967.

[1032] R. M. Needham and R. D. Walker, "The cambridge CAP computer and its protection system," in *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, ser. SOSP '77. New York, NY, USA: ACM, 1977, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/800214.806541

[1033] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: the kernel of a multiprocessor operating system," *Communications of the ACM*, vol. 17, no. 6, pp. 337–345, Jun. 1974. [Online]. Available: http://doi.acm.org/10.1145/355616.364017

[1034] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 207–220. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629596

[1035] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for UNIX," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929820.1929824

[1036] F. Güntsch, "Logischer entwurf eines digitalen rechnergerätes mit mehreren asynchron laufenden trommeln und automatischem schnellspeicherbetrieb Logical Design of a

Digital Computer with Multiple Asynchronous Rotating Drums and Automatic High Speed Memory Operation," Ph.D. dissertation, Technische Universität Berlin, 1957.

[1037] T. Killburn, "One-level storage system," *IRE Transactions on Electronic Computers*, vol. EC-ll, no. 2, Apr. 1962.

[1038] R. C. Daley and J. B. Dennis, "Virtual memory, processes, and sharing in MULTICS," *Communications of the ACM*, vol. 11, no. 5, pp. 306–312, May 1968. [Online]. Available: http://doi.acm.org/10.1145/363095.363139

[1039] Oracle, "M7: Next generation SPARC." [Online]. Available: https://www.hotchips.org/wp-content/uploads/hc_archives/hc26/HC26-12-day2-epub/HC26.12-8-Big-Iron-Servers-epub/HC26.12.820-Next_Gen_SPARC_Phillips-Oracle-FinalPub.pdf

[1040] ARM, "Arm a-profile architecture developments 2018: Armv8.5-a." [Online]. Available: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-a-profile-architecture-2018-developments-armv85a

[1041] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel SGX," in *Proceedings of the 10th European Workshop on Systems Security*, ser. EuroSec'17. New York, NY, USA: ACM, 2017, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/3065913.3065915

[1042] U. Frisk, "Direct memory attack the kernel," Aug. 2016. [Online]. Available: https://archive.org/details/youtube-fXthwl6ShOg

[1043] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, and R. N. M. Watson, "Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals," in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, Feb. 2019.

[1044] A. Milburn, H. Bos, and C. Giuffrida, "SafeInit: Comprehensive and Practical Mitigation of Uninitialized Read Vulnerabilities," in *NDSS*, Feb. 2017. [Online]. Available: https://www.vusec.net/download/?t=papers/safeinit_ndss17.pdf

[1045] E. Bosman and H. Bos, "Framing signals-a return to portable shellcode," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 243–258.

[1046] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the internet of things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2013, pp. 79–80.

[1047] PaX Team, "Design & implementation of PAGEEXEC," 2000.

[1048] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 340–353. [Online]. Available: http://doi.acm.org/10.1145/1102120.1102165

[1049] PaX Team, "Address space layout randomization," https://pax.grsecurity.net/docs/aslr.txt (Patch originally released in 2001), 2001.

[1050] E. Goktas, E. Athanasopoulos, H. Bos, and G. Portokalidis, "Out of control: Overcoming control-flow integrity," in *IEEE Security & Privacy*, Dec. 2014. [Online]. Available: http://www.cs.vu.nl/%7Eherbertb/papers/outofcontrol_sp14.pdf

[1051] M. Castro, M. Costa, and T. Harris, "Securing software by enforcing data-flow integrity," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 147–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298470

[1052] E. Bauman, G. Ayoade, and Z. Lin, "A survey on hypervisor-based monitoring: Approaches, applications, and evolutions," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 10:1–10:33, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2775111

[1053] R. Natan, *Implementing Database Security and Auditing*. Elsevier Science, 2005. [Online]. Available: https://books.google.nl/books?id=5WIP24cbtSEC

[1054] J. Forristal, "NT web technology vulnerabilities," *Phrack Magazine*, vol. 8, no. 4, Dec. 1998, published as `rain.forest.puppy`.

[1055] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Trans. Storage*, vol. 2, no. 2, pp. 107–138, May 2006. [Online]. Available: http://doi.acm.org/10.1145/1149976.1149977

[1056] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distrbuted Programming*. Springer, 2011.

[1057] K. Birman, *Reliable Distributed Systems*. Springer, 2005.

[1058] P. Verissimo and L. Rodriques, *Distributed Systems for System Architects*. Kluwer, 2001.

[1059] A. Tannenbaum and M. Steen, *Distributed Systems: Principles & Paradigms*. Prentice Hall, 2007.

[1060] M. Steen and A. Tannenbaum, *Distributed Systems*. Prentice Hall, 2017.

[1061] B. Hartman, D. Flinn, and K. Beznosov, *Enterprise Security with EJB and CORBA*. Wiley, 2001.

[1062] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1996.

[1063] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[1064] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing*, 2001, pp. 99–100.

[1065] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P systems scalable," in *Proc. of Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM. ACM, 2003, pp. 407–418.

[1066] I. Stoica et al., "Chord: A scalable peer-to-peer lookup service for internet applications," *In Proc. SIGCOMM*, pp. 149 – 160, 2001.

[1067] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Proc. Middleware*, pp. 329–350, 2001.

[1068] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan 2004.

[1069] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," *In Proc. IPTPS*, pp. 53 – 65, 2002.

[1070] B. Cohen, "BitTorrent protocol specification," BitTorrent, Tech. Rep., 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.htm

[1071] B. Yang and H. Garcia-Molina, "Comparing hybrid peer-to-peer systems," *Proc. of VLDB*, pp. 561–570, 2001.

[1072] L. Garcés-Erice, E. W. Biersack, K. W. Ross, P. Felber, and G. Urvoy-Keller, "Hierarchical peer-to-peer systems," *Parallel Processing Letters*, vol. 13, no. 4, pp. 643–657, 2003.

[1073] F. Swiderski and W. Snyder, *Threat Modeling*. Springer, 2003.

[1074] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[1075] C. Esposito and M. Ciampi, "On security in publish/subscribe services: A survey," *IEEE Communication Surveys and Tutorials*, vol. 17, no. 2, 2015.

[1076] A. Uzunov, "A survey of security solutions for distributed publish/subscribe systems," *Computers and Security*, vol. 61, pp. 94–129, 2016.

[1077] A. Walters, D. Zage, and C. Rotaru, "A framework for mitigating attacks against measurement-based adaptation mechanisms in unstructured multicast overlay networks," *IEEE/ACM Trans on Networking*, vol. 16, no. 6, pp. 1434–1446, Dec 2008.

[1078] T. Isdal, M. Piatek, and A. Krishnamurthy, "Privacy preserving P2P data sharing with Oneswarm," *SIGCOMM*, 2011.

[1079] J. Seibert, X. Sun, C. Nita-Rotaru, and S. Rao, "Towards securing data delivery in peer-to-peer streaming," in *Proc. Communication Systems and Networks (COMSNETS)*, 2010, pp. 1–10.

[1080] R. Barra de Almeida, J. Miranda Natif, A. Couto da Silva, and A. Borges Vieira, "Pollution and whitewashing attacks in a P2P live streaming system: Analysis and counter-attack," in *IEEE Intl. Conf on Communications (ICC)*, June 2013, pp. 2006–2010.

[1081] J. Liang, N. Naoumov, and K. Ross, "The Index Poisoning Attack in P2P File Sharing Systems," in *INFOCOM*, 2006, pp. 1–12.

[1082] N. Naoumov and K. Ross, "Exploiting P2P systems for DDoS attacks," in *ACM Proceedings of Scalable Information Systems*, 2006.

[1083] D. Li, J. Wu, and Y. Cui, "Defending against buffer map cheating in DONet-like P2P streaming," *IEEE Trans. Multimedia*, vol. 11, no. 3, pp. 535–542, April 2009.

[1084] J. R. Douceur, "The sybil attack," in *Intl. Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 251–260.

[1085] A. Singh et al., "Eclipse attacks on overlay networks: Threats and defenses," *Proc. INFOCOM*, pp. 1–12, 2006.

[1086] F. DePaoli and L. Mariani, "Dependability in peer-to-peer systems," *IEEE Internet Computing*, vol. 8, no. 4, pp. 54–61, July 2004.

[1087] G. Gheorghe, R. Lo Cigno, and A. Montresor, "Security and privacy issues in P2P streaming systems: A survey," *Peer-to-Peer Networking and Applications*, vol. 4, no. 2, pp. 75–91, 2011.

[1088] G. Urdaneta, G. Pierre, and M. V. Steen, "A survey of DHT security techniques," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 8:1–8:49, 2011.

[1089] Y.-K. Kwok, "Autonomic peer-to-peer systems: Incentive and security issues," in *Autonomic Computing and Networking*, Y. Zhang, L. T. Yang, and M. K. Denko, Eds. Springer, 2009, pp. 205–236.

[1090] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, Dec. 2004.

[1091] D. Wallach, "A survey of peer-to-peer security issues," in *Software Security - Theories and Systems*, ser. Lecture Notes in Computer Science. Springer, 2003, vol. 2609, pp. 42–57.

[1092] B. Hartman, D. Flinn, and K. Beznosov, *Mastering Web Services Security*. Wiley, 2003.

[1093] M. Reiter, "How to securely replicate servers," *ACM Trans. Programming Language Systems*, pp. vol. 16, 3, 986–1009, 1994.

[1094] M. Vukolic, "Quorum systems: Applications to storage & consensus," *Morgan Claypool*, 2012.

[1095] P. Viotti and M. Vukolic, "Consistency in non-transactional distributed storage systems," *ACM Computing Surveys*, vol. 49, no. 1, 2016.

[1096] P. DuBois and M. Foreword By-Widenius, *MySQL*. New riders publishing, 1999.

[1097] https://www.microsoft.com/en-us/sql-server.

[1098] https://www.mongodb.com.

[1099] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 335–350.

[1100] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 42–81, 2005.

[1101] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[1102] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[1103] F. Schneider, "Implementing fault tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, p. 23(4), 1990.

[1104] E. Brewer, "CAP: Twelve years later: How the rules have changed," *IEEE Computer*, pp. 23–29, Feb 2012.

[1105] L. Lamport, "Paxos made simple," *ACM SIGACT News*, 2001.

[1106] T. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," *Proc. of ACM PODC*, 2007.

[1107] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference (USENIX ATC)*, 2014, pp. 305–319.

[1108] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, p. 4(3), 1982.

[1109] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," Yale Univ, Dept of CS, Tech. Rep., 1982.

[1110] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 45–58, 2007.

[1111] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[1112] Y. Zhang, Z. Zheng, and M. R. Lyu, "BFTCloud: A byzantine fault tolerance framework for voluntary-resource cloud computing," in *IEEE Conf. on Cloud Computing*, 2011, pp. 444–451.

[1113] M. Castro and B. Liskov, "Practical byzantine faut tolerance," *Proc. of OSDI*, pp. 23–29, 1999.

[1114] M. Platania, D. Obenshain, T. Tantillo, Y. Amir, and N. Suri, "On choosing server- or client-side solutions for BFT," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 61:1–61:30, 2016.

[1115] P. Manadhata and J. Wing, "An attack surface metric," *IEEE Trans Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.

[1116] G. Hogben and M. Dekker, "Survey and analysis of security parameters in cloud SLAs across the european public sector," European Network and Information Security Agency, Tech. Rep., 2014.

[1117] Technical Committee ISO/IEC JTC 1/SC 27, "Information technology — security techniques — code of practice for information security controls based on iso/iec 27002 for cloud services," International Organization for Standardization, Tech. Rep. ISO/IEC 27017:2015, December 2015.

[1118] A. Bessani et al, "Secure storage in a cloud-of-clouds," *ACM Eurosys*, pp. 31–46, 2011.

[1119] G. Karame et al, "Reconciling security and functional requirements in multi-tenacy clouds," *Proc. of SCC*, 2017.

[1120] B. Lampson, "Protection," *Operating Systems Review*, pp. 8, 1, 18–24, 1974.

[1121] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *Proceedings of Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016*, 2016, pp. 118–140.

[1122] E. Androulaki et al, "Hyperledger fabric: A distributed operating system for permissioned bockchains," *ACM Eurosys*, 2018.

[1123] A. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. Sirer, "Decentralization in Bitcoin and Ethereum networks," *Financial Cryptography and Data Security Conference*, 2018.

[1124] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," *USENIX Security Symposium*, pp. 129–144, 2015.

[1125] W. Conradie and V. Goranko, *Logic and Discrete Mathematics - A Concise Introduction*. Wiley, 2015. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118751272.html

[1126] J. Harrison, *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[1127] T. Nipkow and G. Klein, *Concrete Semantics: With Isabelle/HOL*. Springer Publishing Company, Incorporated, 2014.

[1128] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "SLAM and static driver verifier: Technology transfer of formal methods inside microsoft," in *Integrated Formal Methods*, E. A. Boiten, J. Derrick, and G. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–20.

[1129] A. Chudnov, N. Collins, B. Cook, J. Dodds, B. Huffman, C. MacCárthaigh, S. Magill, E. Mertens, E. Mullen, S. Tasiran, A. Tomb, and E. Westbrook, "Continuous formal verification of Amazon s2n," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, pp. 430–446.

[1130] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How Amazon Web Services uses formal methods," *Commun. ACM*, vol. 58, no. 4, p. 66–73, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2699417

[1131] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at Google," *Commun. ACM*, vol. 61, no. 4, p. 58–66, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3188720

[1132] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, vol. 39, no. 12, p. 92–106, Dec. 2004. [Online]. Available: https://doi.org/10.1145/1052883.1052895

[1133] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: Using static analysis to find bugs in the real world," *Commun. ACM*, vol. 53, no. 2, p. 66–75, Feb. 2010. [Online]. Available: https://doi.org/10.1145/1646353.1646374

[1134] G. Winskel, *The Formal Semantics of Programming Languages: An Introduction*. Cambridge, MA, USA: MIT Press, 1993.

[1135] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *J. Comput. Secur.*, vol. 18, no. 6, p. 1157–1210, Sep. 2010.

[1136] F. Schneider, "Blueprint for a science of cybersecurity," *The Next Wave*, vol. 19, no. 2, pp. 47–57, 2012.

[1137] C. Herley and P. C. Van Oorschot, "SoK: Science, security and the elusive goal of security as a scientific pursuit," in *2017 IEEE Symposium on Security and Privacy*, 2017, pp. 99–120.

[1138] D. Basin and S. Capkun, "The research value of publishing attacks," *Commun. ACM*, vol. 55, no. 11, pp. 22–24, November 2012. [Online]. Available: http://doi.acm.org/10.1145/2366316.2366324

[1139] M. T. Dashti and D. A. Basin, "Security testing beyond functional tests," in *Engineering Secure Software and Systems - 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*, 2016, pp. 1–19. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-30806-7_1

[1140] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distrib. Comput.*, vol. 2, no. 3, p. 117–126, Sep. 1987. [Online]. Available: https://doi.org/10.1007/BF01782772

[1141] J. Goguen and J. Meseguer, "Security policies and security models," in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, 1982, pp. 11–20.

[1142] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*. Cham: Springer International Publishing, 2018, pp. 305–343. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8-11

[1143] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '77. New York, NY, USA: Association for Computing Machinery, 1977, p. 238–252. [Online]. Available: https://doi.org/10.1145/512950.512973

[1144] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293 – 303, 2009, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1567832608000775

[1145] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 2, pp. 125–143, 1977.

[1146] D. E. Bell and L. J. L. Padula, "Secure computer system: Unified exposition and multics interpretation," 1976.

[1147] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, p. 236–243, May 1976. [Online]. Available: https://doi.org/10.1145/360051.360056

[1148] D. Hedin and A. Sabelfeld, "A perspective on information-flow control," in *Software Safety and Security - Tools for Analysis and Verification*, ser. NATO Science for Peace and Security Series - D: Information and Communication Security, T. Nipkow, O. Grumberg, and B. Hauptmann, Eds. IOS Press, 2012, vol. 33, pp. 319–347. [Online]. Available: https://doi.org/10.3233/978-1-61499-028-4-319

[1149] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," *CoRR*, vol. abs/1401.4492, 2014. [Online]. Available: http://arxiv.org/abs/1401.4492

[1150] B. Finkbeiner, M. N. Rabe, and C. Sánchez, "Algorithms for model checking HyperLTL and HyperCTL*," in *Computer Aided Verification*, D. Kroening and C. S. Păsăreanu, Eds. Cham: Springer International Publishing, 2015, pp. 30–48.

[1151] S. Zdancewic and A. C. Myers, "Observational determinism for concurrent program security," in *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings.*, 2003, pp. 29–43.

[1152] D. McCullough, "Noninterference and the composability of security properties," in *2012 IEEE Symposium on Security and Privacy*. Los Alamitos, CA, USA: IEEE Computer Society, apr 1988, p. 177. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SECPRI.1988.8110

[1153] J. McLean, "A general theory of composition for a class of "possibilistic" properties," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 53–67, 1996.

[1154] D. Park, "Concurrency and automata on infinite sequences," in *Proceedings of the 5th GI-Conference on Theoretical Computer Science*. Berlin, Heidelberg: Springer-Verlag, 1981, p. 167–183.

[1155] R. Milner, *Communication and Concurrency*. USA: Prentice-Hall, Inc., 1989.

[1156] R. Gorrieri and C. Versari, *Introduction to Concurrency Theory: Transition Systems and CCS*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[1157] M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols: The spi calculus," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, ser. CCS '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 36–47. [Online]. Available: https://doi.org/10.1145/266420.266432

[1158] ——, "A bisimulation method for cryptographic protocols," *Nordic J. of Computing*, vol. 5, no. 4, p. 267–303, Dec. 1998.

[1159] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, 2001, pp. 104–115. [Online]. Available: https://doi.org/10.1145/360204.360213

[1160] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Found. Trends Priv. Secur.*, vol. 1, no. 1–2, p. 1–135, Oct. 2016. [Online]. Available: https://doi.org/10.1561/3300000004

[1161] B. Blanchet, "Automatic proof of strong secrecy for security protocols," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, 2004, pp. 86–100.

[1162] V. Cheval, "Automatic verification of cryptographic protocols: privacy-type properties," PhD Thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2012.

[1163] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Adv. Comput.*, vol. 58, pp. 117–148, 2003. [Online]. Available: https://doi.org/10.1016/S0065-2458(03)58003-2

[1164] M. Kaufmann and J. S. Moore, "An industrial strength theorem prover for a logic based on Common Lisp," *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 203–213, 1997.

[1165] J. S. Moore, "Milestones from the pure lisp theorem prover to ACL2," *Formal Aspects Comput.*, vol. 31, no. 6, pp. 699–732, 2019. [Online]. Available: https://doi.org/10.1007/s00165-019-00490-3

[1166] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2283. [Online]. Available: https://doi.org/10.1007/3-540-45949-9

[1167] J. Harrison, "HOL Light: An overview," in *Theorem Proving in Higher Order Logics*, S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 60–66.

[1168] Y. Bertot and P. Castran, *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[1169] G. Huet, G. Kahn, and C. Paulin-Mohring, *The* Coq *Proof Assistant - A tutorial - Version 7.1*, Oct. 2001, http://coq.inria.fr.

[1170] L. C. Paulson, *Logic and Computation: Interactive Proof with Cambridge LCF*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1987.

[1171] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 530–535. [Online]. Available: https://doi.org/10.1145/378239.379017

[1172] J. P. Marques Silva and K. A. Sakallah, *Grasp—A New Search Algorithm for Satisfiability*. Boston, MA: Springer US, 2003, pp. 73–89. [Online]. Available: https://doi.org/10.1007/978-1-4615-0292-0_7

[1173] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518.

[1174] F. Corzilius, U. Loup, S. Junges, and E. Ábrahám, "SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox," in *Theory and Applications of Satisfiability Testing – SAT 2012*, A. Cimatti and R. Sebastiani, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 442–448.

[1175] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, p. 69–77, Sep. 2011. [Online]. Available: https://doi.org/10.1145/1995376.1995394

[1176] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[1177] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "Cvc4," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–177.

[1178] B. Dutertre, "Yices 2.2," in *Computer Aided Verification*, A. Biere and R. Bloem, Eds. Cham: Springer International Publishing, 2014, pp. 737–744.

[1179] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 2000.

[1180] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[1181] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, p. 293–318, Sep. 1992. [Online]. Available: https://doi.org/10.1145/136035.136043

[1182] D. Peled, "Ten years of partial order reduction," in *Computer Aided Verification*, A. J. Hu and M. Y. Vardi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 17–28.

[1183] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, p. 279–295, May 1997. [Online]. Available: https://doi.org/10.1109/32.588521

[1184] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, E. Brinksma and K. G. Larsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 359–364.

[1185] P. Gardiner, M. Goldsmith, J. Hulance, D. Jackson, A. Roscoe, B. Scattergood, and P. Armstrong, "FDR2 user manual," 2000.

[1186] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 585–591.

[1187] A. Chudnov, N. Collins, B. Cook, J. Dodds, B. Huffman, C. MacCárthaigh, S. Magill, E. Mertens, E. Mullen, S. Tasiran, A. Tomb, and E. Westbrook, "Continuous formal verification of Amazon s2n," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, pp. 430–446.

[1188] B. Cook, "Formal reasoning about the security of Amazon Web Services," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, pp. 38–47.

[1189] Ú. Erlingsson and F. B. Schneider, "IRM enforcement of Java stack inspection," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 2000, pp. 246–255.

[1190] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "Efficient monitoring of hyperproperties using prefix trees," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 6, pp. 729–740, 2020. [Online]. Available: https://doi.org/10.1007/s10009-020-00552-5

[1191] K. Havelund and G. Roşu, "An overview of the runtime verification tool Java PathExplorer," *Formal Methods in System Design*, vol. 24, no. 2, pp. 189–215, Mar 2004.

[1192] D. Basin, F. Klaedtke, and S. Müller, "Monitoring security policies with metric first-order temporal logic," in *15th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM Press, 2010, pp. 23–33.

[1193] E. Arfelt, D. Basin, and S. Debois, "Monitoring the GDPR," in *Computer Security – ESORICS 2019*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds. Cham: Springer International Publishing, 2019, pp. 681–699.

[1194] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu, "Monitoring metric first-order temporal properties," *J. ACM*, vol. 62, no. 2, pp. 15:1–15:45, May 2015. [Online]. Available: http://doi.acm.org/10.1145/2699444

[1195] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Theory in practice for system design and verification," *ACM SIGLOG News*, vol. 2, no. 1, p. 46–51, Jan. 2015. [Online]. Available: https://doi.org/10.1145/2728816.2728827

[1196] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461.

[1197] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke, "CacheAudit: A tool for the static analysis of cache side channels," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, Jun. 2015. [Online]. Available: https://doi.org/10.1145/2756550

[1198] M. Bond and R. Anderson, "API-level attacks on embedded systems," *Computer*, vol. 34, no. 10, pp. 67–75, 2001.

[1199] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: A survey," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 2, p. 123–193, Apr. 1999. [Online]. Available: https://doi.org/10.1145/307988.307989

[1200] T. Kropf, *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 1999.

[1201] E. Clarke and D. Kroening, "Hardware verification using ANSI-C programs as a reference," in *Proceedings of the ASP-DAC Asia and South Pacific Design Automation Conference, 2003.*, 2003, pp. 308–311.

[1202] J. Sawada and W. A. Hunt Jr., "Verification of FM9801: an out-of-order microprocessor model with speculative execution, exceptions, and program-modifying capability," *Formal Methods Syst. Des.*, vol. 20, no. 2, pp. 187–222, 2002. [Online]. Available: https://doi.org/10.1023/A:1014122630277

[1203] M. M. Wilding, D. A. Greve, R. J. Richards, and D. S. Hardin, *Formal Verification of Partition Management for the AAMP7G Microprocessor*. Boston, MA: Springer US, 2010, pp. 175–191. [Online]. Available: https://doi.org/10.1007/978-1-4419-1539-9_6

[1204] A. Sabelfeld and D. Sands, "Dimensions and principles of declassification," in *18th IEEE Computer Security Foundations Workshop (CSFW'05)*, 2005, pp. 255–269.

[1205] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '99. Berlin, Heidelberg: Springer-Verlag, 1999, p. 398–412.

[1206] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," *J. Comput. Secur.*, vol. 8, no. 2,3, p. 141–158, Aug. 2000.

[1207] S. Micali and L. Reyzin, "Physically observable cryptography," in *Theory of Cryptography*, M. Naor, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 278–296.

[1208] B. Köpf and D. A. Basin, "Automatically deriving information-theoretic bounds for adaptive side-channel attacks," *Journal of Computer Security*, vol. 19, no. 1, pp. 1–31, 2011.

[1209] S. Cauligi, G. Soeller, B. Johannesmeyer, F. Brown, R. S. Wahby, J. Renner, B. Grégoire, G. Barthe, R. Jhala, and D. Stefan, "FaCT: A DSL for timing-sensitive computation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for

Computing Machinery, 2019, p. 174–189. [Online]. Available: https://doi.org/10.1145/3314221.3314605

[1210] M. Wu, S. Guo, P. Schaumont, and C. Wang, "Eliminating timing side-channel leaks using program repair," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 15–26. [Online]. Available: https://doi.org/10.1145/3213846.3213851

[1211] G. Barthe, B. Grégoire, and V. Laporte, "Secure compilation of side-channel countermeasures: The case of cryptographic "constant-time"," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018, pp. 328–343.

[1212] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," *SIGPLAN Not.*, vol. 50, no. 4, p. 503–516, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2775054.2694372

[1213] M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel, "Attacking and fixing PKCS#11 security tokens," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 260–269. [Online]. Available: https://doi.org/10.1145/1866307.1866337

[1214] B. Blanchet and D. Pointcheval, "Automated security proofs with sequences of games," in *CRYPTO'06*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Santa Barbara, CA: Springer, Aug. 2006, pp. 537–554.

[1215] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001.*, 2001, pp. 82–96.

[1216] L. Paulson, "The inductive approach to verifying cryptographic protocols," *J. Computer Security*, vol. 6, pp. 85–128, 1998. [Online]. Available: http://www.cl.cam.ac.uk/users/lcp/papers/Auth/jcs.pdf

[1217] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1996, pp. 147–166.

[1218] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A. Roscoe, *The Modelling and Analysis of Security Protocols: The CSP Approach*, 1st ed. Addison-Wesley Professional, 2000.

[1219] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, june 2012, pp. 78 –94.

[1220] D. Basin, C. Cremers, and C. Meadows, *Model Checking Security Protocols*. Springer, 2018, ch. 24, pp. 727–762.

[1221] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," Cryptology ePrint Archive, Report 2019/1393, 2019, https://eprint.iacr.org/2019/1393.

[1222] D. A. Basin, C. J. F. Cremers, K. Miyazaki, S. Radomirovic, and D. Watanabe, "Improving

the security of cryptographic protocol standards," *IEEE Security & Privacy*, vol. 13, no. 3, pp. 24–31, 2015. [Online]. Available: http://dx.doi.org/10.1109/MSP.2013.162

[1223] D. Basin, C. Cremers, and S. Meier, "Provably repairing the ISO/IEC 9798 standard for entity authentication," in *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*, ser. Lecture Notes in Computer Science, P. Degano and J. D. Guttman, Eds., vol. 710.1007/978-3-319-40667-1_185. Springer, 2012, pp. 129–148.

[1224] K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hritcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. R. Lorch, K. Maillard, J. Pan, B. Parno, J. Protzenko, T. Ramananandro, A. Rane, A. Rastogi, N. Swamy, L. Thompson, P. Wang, S. Z. Béguelin, and J. K. Zinzindohoue, "Everest: Towards a verified, drop-in replacement of HTTPS," in *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA*, ser. LIPIcs, B. S. Lerner, R. Bodík, and S. Krishnamurthi, Eds., vol. 71. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 1:1–1:12. [Online]. Available: https://doi.org/10.4230/LIPIcs.SNAPL.2017.1

[1225] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue, "A messy state of the union: Taming the composite state machines of TLS," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 535–552.

[1226] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1773–1788. [Online]. Available: https://doi.org/10.1145/3133956.3134063

[1227] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5G authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1383–1396. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243846

[1228] D. A. Basin, R. Sasse, and J. Toro-Pozo, "The EMV standard: Break, Fix, Verify," in *42nd IEEE Symposium on Security and Privacy (Oakland S&P)*, 2021, to appear.

[1229] ——, "Card brand mixup attack: Bypassing the PIN in non-Visa cards by using them for Visa transactions," in *30th USENIX Security Symposium*, 2021, to appear.

[1230] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983. [Online]. Available: https://doi.org/10.1109/TIT.1983.1056650

[1231] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 1997, pp. 31–43.

[1232] G. Bella, F. Massacci, and L. C. Paulson, "Verifying the SET registration protocols," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 77–87, 2003.

[1233] S. Meier, C. Cremers, and D. Basin, "Strong invariants for the efficient construction of machine-checked protocol security proofs," in *23rd IEEE Computer Security*

Foundations Symposium. Los Alamitos, USA: IEEE Computer Society, 7 2010, pp. 231–245.

[1234] S. Meier, C. Cremers, and D. A. Basin, "Efficient construction of machine-checked symbolic protocol security proofs," *Journal of Computer Security*, vol. 21, no. 1, pp. 41–87, 2013.

[1235] M. Abadi and L. Lamport, "The existence of refinement mappings," *Theor. Comput. Sci.*, vol. 82, no. 2, pp. 253–284, 1991. [Online]. Available: https://doi.org/10.1016/0304-3975(91)90224-P

[1236] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[1237] J. Lallemand, D. A. Basin, and C. Sprenger, "Refining authenticated key agreement with strong adversaries," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, 2017, pp. 92–107.

[1238] C. Sprenger and D. A. Basin, "Refining security protocols," *Journal of Computer Security*, vol. 26, no. 1, pp. 71–120, 2018. [Online]. Available: https://doi.org/10.3233/JCS-16814

[1239] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov, "Multiset rewriting and the complexity of bounded security protocols," *Journal of Computer Security*, vol. 12, no. 2, pp. 247–311, Apr. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1017273.1017276

[1240] J. K. Millen, S. C. Clark, and S. B. Freedman, "The Interrogator: Protocol security analysis," *IEEE Trans. Software Eng.*, vol. 13, no. 2, pp. 274–288, 1987.

[1241] D. Longley and S. Rigby, "An automatic search for security flaws in key management schemes," *Computers & Security*, vol. 11, no. 1, pp. 75–89, 1992.

[1242] C. Meadows, "The NRL Protocol Analyzer: An overview," *J. Log. Program.*, vol. 26, no. 2, pp. 113–131, 1996.

[1243] G. Lowe, "Casper: A compiler for the analysis of security protocols," *J. Comput. Secur.*, vol. 6, no. 1–2, p. 53–84, Jan. 1998.

[1244] A. Armando, R. Carbone, and L. Compagna, "SATMC: A SAT-based model checker for security protocols, business processes, and security apis," *Int. J. Softw. Tools Technol. Transf.*, vol. 18, no. 2, p. 187–204, Apr. 2016. [Online]. Available: https://doi.org/10.1007/s10009-015-0385-y

[1245] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Proceedings of CAV'2005*, ser. LNCS 3576. Springer-Verlag, 2005, pp. 281–285.

[1246] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *25th International Conference on Computer Aided Verification (CAV 2013)*, ser. LNCS, N. Sharygina and H. Veith, Eds., vol. 8044. Saint Petersburg, Russia: Springer, July 2013, pp. 696–701.

[1247] B. Blanchet, "Automatic verification of correspondences for security protocols," *J. Comput. Secur.*, vol. 17, no. 4, p. 363–434, Dec. 2009.

[1248] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," in *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, 2005, pp. 331–340.

[1249] K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse, "Verified interoperable implementations of security protocols," in *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006, pp. 14 pp.–152.

[1250] D. Basin and C. Cremers, "Know your enemy: Compromising adversaries in protocol analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 2, pp. 7:1–7:31, Nov. 2014. [Online]. Available: http://doi.acm.org/10.1145/2658996

[1251] M. D. Ryan and B. Smyth, "Applied pi calculus," in *Formal Models and Techniques for Analyzing Security Protocols*, V. Cortier and S. Kremer, Eds. IOS Press, 2011, ch. 6.

[1252] V. Cortier and S. Delaune, "A method for proving observational equivalence," in *2009 22nd IEEE Computer Security Foundations Symposium*, 2009, pp. 266–276.

[1253] D. Basin, J. Dreier, and R. Sasse, "Automated symbolic proofs of observational equivalence," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 1144–1155. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813662

[1254] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using maude-npa," in *Security and Trust Management*, S. Mauw and C. D. Jensen, Eds. Cham: Springer International Publishing, 2014, pp. 162–177.

[1255] M. Baudet, V. Cortier, and S. Delaune, "YAPA: A generic tool for computing intruder knowledge," *ACM Trans. Comput. Logic*, vol. 14, no. 1, Feb. 2013. [Online]. Available: https://doi.org/10.1145/2422085.2422089

[1256] c. Ciobâcă, S. Delaune, and S. Kremer, "Computing knowledge in security protocols under convergent equational theories," in *Proceedings of the 22nd International Conference on Automated Deduction*, ser. CADE-22. Berlin, Heidelberg: Springer-Verlag, 2009, p. 355–370. [Online]. Available: https://doi.org/10.1007/978-3-642-02959-2_27

[1257] B. Conchinha, D. Basin, and C. Caleiro, "Efficient decision procedures for message deducibility and static equivalence," in *Formal Aspects of Security and Trust*, P. Degano, S. Etalle, and J. Guttman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 34–49.

[1258] M. Baudet, "Deciding security of protocols against off-line guessing attacks," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 16–25. [Online]. Available: https://doi.org/10.1145/1102120.1102125

[1259] R. Corin, J. Doumen, and S. Etalle, "Analysing password protocol security against off-line dictionary attacks," *Electronic Notes in Theoretical Computer Science*, vol. 121, pp. 47–63, 2005, proceedings of the 2nd International Workshop on Security Issues

with Petri Nets and other Computational Models (WISP 2004). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066105000241

[1260] M. Stoelinga, "An introduction to probabilistic automata," *Bull. EATCS*, vol. 78, pp. 176–198, 2002.

[1261] C. Baier, L. de Alfaro, V. Forejt, and M. Kwiatkowska, *Model Checking Probabilistic Systems*. Cham: Springer International Publishing, 2018, pp. 963–999. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8-28

[1262] M. Kwiatkowska, G. Norman, and D. Parker, "Quantitative analysis with the probabilistic model checker PRISM," *Electron. Notes Theor. Comput. Sci.*, vol. 153, no. 2, p. 5–31, May 2006. [Online]. Available: https://doi.org/10.1016/j.entcs.2005.10.030

[1263] S. Basagiannis, P. Katsaros, A. Pombortsis, and N. Alexiou, "Probabilistic model checking for the quantification of DoS security threats," *Comput. Secur.*, vol. 28, no. 6, p. 450–465, Sep. 2009. [Online]. Available: https://doi.org/10.1016/j.cose.2009.01.002

[1264] V. Shmatikov, "Probabilistic analysis of an anonymity system," *J. Comput. Secur.*, vol. 12, no. 3,4, p. 355–377, May 2004.

[1265] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, p. 66–92, Nov. 1998. [Online]. Available: https://doi.org/10.1145/290163.290168

[1266] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, "Automatic analysis of a non-repudiation protocol," in *Proc. 2nd International Workshop on Quantitative Aspects of Programming Languages (QAPL'04)*, 2004.

[1267] G. Norman and V. Shmatikov, "Analysis of probabilistic contract signing," in *Formal Aspects of Security*, A. E. Abdallah, P. Ryan, and S. Schneider, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 81–96.

[1268] V. Galpin, "Formal modelling of software defined networking," in *Integrated Formal Methods - 14th International Conference, IFM 2018, Maynooth, Ireland, September 5-7, 2018, Proceedings*, ser. Lecture Notes in Computer Science, C. A. Furia and K. Winter, Eds., vol. 11023. Springer, 2018, pp. 172–193. [Online]. Available: https://doi.org/10.1007/978-3-319-98938-9_11

[1269] R. Dimitrova, B. Finkbeiner, and H. Torfah, "Probabilistic hyperproperties of Markov decision processes," in *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, ser. Lecture Notes in Computer Science, D. V. Hung and O. Sokolsky, Eds., vol. 12302. Springer, 2020, pp. 484–500. [Online]. Available: https://doi.org/10.1007/978-3-030-59152-6_27

[1270] E. Ábrahám, E. Bartocci, B. Bonakdarpour, and O. Dobe, "Probabilistic hyperproperties with nondeterminism," in *Automated Technology for Verification and Analysis*, D. V. Hung and O. Sokolsky, Eds. Cham: Springer International Publishing, 2020, pp. 518–534.

[1271] N. Koblitz and A. J. Menezes, "Another look at "provable security"," *J. Cryptol.*, vol. 20, no. 1, p. 3–37, Jan. 2007. [Online]. Available: https://doi.org/10.1007/s00145-005-0432-z

[1272] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *EUROCRYPT 2006*, ser. LNCS, vol. 4004. Springer, 2006, pp. 409–426.

[1273] V. Shoup, "Sequences of games: A tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004. [Online]. Available: http://eprint.iacr.org/2004/332

[1274] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 2001, pp. 136–145.

[1275] U. Maurer, "Constructive cryptography – a new paradigm for security definitions and proofs," in *Joint Workshop on Theory of Security and Applications*. Springer, 2011, pp. 33–56.

[1276] S. Halevi, "A plausible approach to computer-aided cryptographic proofs," 2005, shaih@alum.mit.edu 12949 received 15 Jun 2005. [Online]. Available: http://eprint.iacr.org/2005/181

[1277] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (the computational soundness of formal encryption)," *J. Cryptol.*, vol. 20, no. 3, p. 395, Jul. 2007.

[1278] D. Micciancio and B. Warinschi, "Soundness of formal encryption in the presence of active adversaries," in *Theory of Cryptography*, M. Naor, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 133–151.

[1279] M. Backes, D. Hofheinz, and D. Unruh, "CoSP: A general framework for computational soundness proofs," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 66–78. [Online]. Available: https://doi.org/10.1145/1653662.1653672

[1280] V. Cortier and B. Warinschi, "Computationally sound, automated proofs for security protocols," in *Programming Languages and Systems*, M. Sagiv, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 157–171.

[1281] V. Cortier, S. Kremer, and B. Warinschi, "A survey of symbolic methods in computational analysis of cryptographic systems," *J. Autom. Reason.*, vol. 46, no. 3-4, pp. 225–259, 2011. [Online]. Available: https://doi.org/10.1007/s10817-010-9187-9

[1282] B. Blanchet, "A computationally sound mechanized prover for security protocols," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 04, pp. 193–207, oct 2008.

[1283] G. Barthe, B. Grégoire, and S. Zanella Béguelin, "Formal certification of code-based cryptographic proofs," in *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 90–101. [Online]. Available: https://doi.org/10.1145/1480881.1480894

[1284] A. Petcher and G. Morrisett, "The foundational cryptography framework," in *POST 2015*, ser. LNCS, vol. 9036. Springer, 2015, pp. 53–72.

[1285] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-based proofs in higher-order logic," *Journal of Cryptology*, pp. 1–73, 2020.

[1286] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Proceedings of the 31st Annual Conference on Advances in Cryptology*, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 71–90.

[1287] Y. Lindell, *How to Simulate It – A Tutorial on the Simulation Proof Technique*. Cham: Springer International Publishing, 2017, pp. 277–346. [Online]. Available: https://doi.org/10.1007/978-3-319-57048-8_6

[1288] A. Datta, R. Küsters, J. C. Mitchell, and A. Ramanathan, "On the relationships between notions of simulation-based security," in *Proceedings of the Second International Conference on Theory of Cryptography*, ser. TCC'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 476–494. [Online]. Available: https://doi.org/10.1007/978-3-540-30576-7_26

[1289] M. Backes, B. Pfitzmann, and M. Waidner, "A universally composable cryptographic library," Cryptology ePrint Archive, Report 2003/015, 2003, https://eprint.iacr.org/2003/015.

[1290] R. Kusters, "Simulation-based security with inexhaustible interactive turing machines," in *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, ser. CSFW '06. USA: IEEE Computer Society, 2006, p. 309–320. [Online]. Available: https://doi.org/10.1109/CSFW.2006.30

[1291] A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague, "Probabilistic bisimulation and equivalence for security analysis of network protocols," in *Foundations of Software Science and Computation Structures*, I. Walukiewicz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 468–483.

[1292] S. Delaune, S. Kremer, and O. Pereira, "Simulation based security in the applied pi calculus," in *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, ser. Leibniz International Proceedings in Informatics (LIPIcs), R. Kannan and K. N. Kumar, Eds., vol. 4. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009, pp. 169–180. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2009/2316

[1293] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner, "Cryptographically sound theorem proving," in *19th IEEE Computer Security Foundations Workshop, Venice, Italy*. IEEE Computer Society, July 2006, pp. 153–166.

[1294] A. Lochbihler, S. R. Sefidgar, D. Basin, and U. Maurer, "Formalizing constructive cryptography using CryptHOL," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, June 2019, pp. 152–15 214.

[1295] D. Butler, D. Aspinall, and A. Gascón, "How to simulate it in Isabelle: Towards formal proof for secure multi-party computation," in *Interactive Theorem Proving*, M. Ayala-Rincón and C. A. Muñoz, Eds. Cham: Springer International Publishing, 2017, pp. 114–130.

[1296] R. Canetti, A. Stoughton, and M. Varia, "EasyUC: Using EasyCrypt to mechanize proofs of universally composable security," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, 2019, pp. 167–16 716.

[1297] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE J.Sel. A. Commun.*, vol. 21, no. 1, pp. 5–19, Sep. 2006.

[1298] A. C. Myers and B. Liskov, "Protecting privacy using the decentralized label model," *ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, p. 410–442, Oct. 2000. [Online]. Available: https://doi.org/10.1145/363516.363526

[1299] M. Eilers, P. Müller, and S. Hitz, "Modular product programs," *ACM Trans. Program. Lang. Syst.*, vol. 42, no. 1, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3324783

[1300] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "HACL*: A verified modern cryptographic library," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1789–1806. [Online]. Available: https://doi.org/10.1145/3133956.3134043

[1301] A. Chlipala, "Mostly-automated verification of low-level programs in computational separation logic," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 234–245. [Online]. Available: https://doi.org/10.1145/1993498.1993526

[1302] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: formal verification of an OS kernel," in *Symposium on Operating Systems Principles (SOSP)*, J. N. Matthews and T. E. Anderson, Eds. ACM, 2009, pp. 207–220.

[1303] W. R. Bevier, W. A. H. Jr., J. S. Moore, and W. D. Young, "An approach to systems verification," *J. Autom. Reason.*, vol. 5, no. 4, pp. 411–428, 1989. [Online]. Available: https://doi.org/10.1007/BF00243131

[1304] H. Garavel and S. Graf, *Formal Methods for Safe and Secure Computers Systems - BSI Study 875*. BSI German Federal Office for Information Security, 2013.

[1305] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

[1306] S. Chong, J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, and N. Zeldovich, "Report on the NSF workshop on formal methods for security," NSF, USA, Tech. Rep., 2016.

[1307] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. ACM*, vol. 20, no. 7, p. 504–513, Jul. 1977. [Online]. Available: https://doi.org/10.1145/359636.359712

[1308] D. Volpano, C. Irvine, and G. Smith, "A sound type system for secure flow analysis," *J. Comput. Secur.*, vol. 4, no. 2–3, p. 167–187, Jan. 1996.

[1309] G. Smith, "Principles of secure information flow analysis," in *Malware Detection*, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. Boston, MA: Springer US, 2007, pp. 291–307.

[1310] A. C. Myers and B. Liskov, "A decentralized model for information flow control," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 129–142. [Online]. Available: https://doi.org/10.1145/268998.266669

[1311] V. Simonet, "Flow Caml in a nutshell," in *Proceedings of the first APPSEM-II workshop*, G. Hutton, Ed., Nottingham, United Kingdom, Mar. 2003, pp. 152–165.

[1312] F. Pottier and V. Simonet, "Information flow inference for ML," *ACM Transactions on Programming Languages and Systems*, vol. 25, no. 1, pp. 117–158, Jan. 2003, ©ACM.

[1313] R. Chapman and A. Hilton, "Enforcing security and safety models with an information flow analysis tool," in *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time and Distributed Systems Using Ada and Related Technologies*, ser. SIGAda '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 39–46. [Online]. Available: https://doi.org/10.1145/1032297.1032305

[1314] J.-F. Bergeretti and B. A. Carré, "Information-flow and data-flow analysis of while-programs," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, p. 37–61, Jan. 1985. [Online]. Available: https://doi.org/10.1145/2363.2366

[1315] D. Giffhorn and G. Snelting, "A new algorithm for low-deterministic security," *International Journal of Information Security*, no. 14, pp. 263––287, 2015.

[1316] F. Pottier, "A simple view of type-secure information flow in the spl pi-calculus," in *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, 2002, pp. 320–330.

[1317] G. Barthe, J. M. Crespo, and C. Kunz, "Beyond 2-safety: Asymmetric product programs for relational program verification," in *Logical Foundations of Computer Science*, S. Artemov and A. Nerode, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 29–43.

[1318] T. Terauchi and A. Aiken, "Secure information flow as a safety problem," in *Proceedings of the 12th International Conference on Static Analysis*, ser. SAS'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 352–367. [Online]. Available: https://doi.org/10.1007/11547662_24

[1319] R. Joshi and K. R. M. Leino, "A semantic approach to secure information flow," *Sci. Comput. Program.*, vol. 37, no. 1–3, p. 113–138, May 2000. [Online]. Available: https://doi.org/10.1016/S0167-6423(99)00024-6

[1320] G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," in *Proceedings. 17th IEEE Computer Security Foundations Workshop*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2004, p. 100. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSFW.2004.1310735

[1321] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," *SIGPLAN Not.*, vol. 39, no. 1, p. 14–25, Jan. 2004. [Online]. Available: https://doi.org/10.1145/982962.964003

[1322] G. Barthe, J. M. Crespo, and C. Kunz, "Relational verification using product programs," in *Proceedings of the 17th International Conference on Formal Methods*, ser. FM'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 200–214.

[1323] P. Müller, M. Schwerhoff, and A. J. Summers, "Viper: A verification infrastructure for permission-based reasoning," in *Verification, Model Checking, and Abstract Interpretation*, B. Jobstmann and K. R. M. Leino, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 41–62.

[1324] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P. Strub, M. Kohlweiss, J. K. Zinzindohoue, and S. Z. Béguelin, "Dependent types and multi-monadic effects in F*," in *Principles of Programming Languages (POPL)*, R. Bodík and R. Majumdar, Eds. ACM, 2016, pp. 256–270.

[1325] B. Bond, C. Hawblitzel, M. Kapritsos, K. R. M. Leino, J. R. Lorch, B. Parno, A. Rane, S. Setty, and L. Thompson, "Vale: Verifying high-performance cryptographic assembly code," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, p. 917–934.

[1326] K. R. M. Leino, "Dafny: An automatic program verifier for functional correctness," in *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, ser. Lecture Notes in Computer Science, E. M. Clarke and A. Voronkov, Eds., vol. 6355. Springer, 2010, pp. 348–370.

[1327] J. B. Almeida, M. Barbosa, G. Barthe, A. Blot, B. Grégoire, V. Laporte, T. Oliveira, H. Pacheco, B. Schmidt, and P.-Y. Strub, "Jasmin: High-assurance and high-speed cryptography," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1807–1823. [Online]. Available: https://doi.org/10.1145/3133956.3134078

[1328] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, and M. Emmi, "Verifying constant-time implementations," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 53–70. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida

[1329] M. Patrignani, A. Ahmed, and D. Clarke, "Formal approaches to secure compilation: A survey of fully abstract compilation and related work," *ACM Comput. Surv.*, vol. 51, no. 6, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3280984

[1330] C. Abate, R. Blanco, D. Garg, C. Hritcu, M. Patrignani, and J. Thibault, "Journey beyond full abstraction: Exploring robust property preservation for secure compilation," in *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 256–271. [Online]. Available: https://doi.org/10.1109/CSF.2019.00025

[1331] J. B. Jensen, N. Benton, and A. Kennedy, "High-level separation logic for low-level code," *SIGPLAN Not.*, vol. 48, no. 1, p. 301–314, Jan. 2013. [Online]. Available: https://doi.org/10.1145/2480359.2429105

[1332] J. C. Reynolds, "Separation logic: A logic for shared mutable data structures," in *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, ser. LICS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 55–74.

[1333] A. Chlipala, "From network interface to multithreaded web applications: A case study in modular program verification," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 609–622. [Online]. Available: https://doi.org/10.1145/2676726.2677003

[1334] P. Wang, S. Cuellar, and A. Chlipala, "Compiler verification meets cross-language linking via data abstraction," in *Proceedings of the 2014 ACM International Conference*

on Object Oriented Programming Systems Languages & Applications, OOPSLA 2014, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014, A. P. Black and T. D. Millstein, Eds. ACM, 2014, pp. 675–690. [Online]. Available: https://doi.org/10.1145/2660193.2660201

[1335] C. Sprenger, T. Klenze, M. Eilers, F. A. Wolf, P. Müller, M. Clochard, and D. Basin, "Igloo: Soundly linking compositional refinement and separation logic for distributed system verification," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3428220

[1336] S. Maus, M. Moskal, and W. Schulte, "Vx86: x86 assembler simulated in c powered by automated theorem proving," in *Algebraic Methodology and Software Technology*, J. Meseguer and G. Roşu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 284–298.

[1337] E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, and S. Tobies, "VCC: A practical system for verifying concurrent C," in *Theorem Proving in Higher Order Logics*, S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 23–42.

[1338] T. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "seL4: From general purpose to a proof of information flow enforcement," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 415–429.

[1339] Y. Zhao, D. Sanán, F. Zhang, and Y. Liu, "High-assurance separation kernels: A survey on formal methods," *CoRR*, vol. abs/1701.01535, 2017. [Online]. Available: http://arxiv.org/abs/1701.01535

[1340] R. Gu, Z. Shao, H. Chen, X. N. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "CertiKOS: An extensible architecture for building certified concurrent OS kernels," in *Operating Systems Design and Implementation (OSDI)*, K. Keeton and T. Roscoe, Eds. USENIX Association, 2016, pp. 653–669.

[1341] H. Mai, E. Pek, H. Xue, S. T. King, and P. Madhusudan, "Verifying security invariants in ExpressOS," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, p. 293–304, Mar. 2013. [Online]. Available: https://doi.org/10.1145/2490301.2451148

[1342] X. Wang, D. Lazar, N. Zeldovich, A. Chlipala, and Z. Tatlock, "Jitk: A trustworthy in-kernel interpreter infrastructure," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. USA: USENIX Association, 2014, p. 33–47.

[1343] X. Leroy, "Formal verification of a realistic compiler," *Commun. ACM*, vol. 52, no. 7, p. 107–115, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1538788.1538814

[1344] H. Chen, D. Ziegler, T. Chajed, A. Chlipala, M. F. Kaashoek, and N. Zeldovich, "Using crash hoare logic for certifying the FSCQ file system," in *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, E. L. Miller and S. Hand, Eds. ACM, 2015, pp. 18–37. [Online]. Available: https://doi.org/10.1145/2815400.2815402

[1345] S. Amani, A. Hixon, Z. Chen, C. Rizkallah, P. Chubb, L. O'Connor, J. Beeren, Y. Nagashima, J. Lim, T. Sewell, J. Tuong, G. Keller, T. Murray, G. Klein, and G. Heiser, "Cogent: Verifying high-assurance file system implementations," *SIGPLAN Not.*, vol. 51,

no. 4, p. 175–188, Mar. 2016. [Online]. Available: https://doi.org/10.1145/2954679.2872404

[1346] J. Yang and C. Hawblitzel, "Safe to the last instruction: Automated verification of a type-safe operating system," in *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 99–110. [Online]. Available: https://doi.org/10.1145/1806596.1806610

[1347] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill, "Ironclad apps: End-to-end security via automated full-system verification," in *Operating Systems Design and Implementation (OSDI)*, J. Flinn and H. Levy, Eds. USENIX Association, 2014, pp. 165–181.

[1348] C. Tice, T. Roeder, P. Collingbourne, S. Checkoway, Ú. Erlingsson, L. Lozano, and G. Pike, "Enforcing forward-edge control-flow integrity in GCC & LLVM," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 941–955. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/tice

[1349] J. Criswell, A. Lenharth, D. Dhurjati, and V. Adve, "Secure virtual architecture: A safe execution environment for commodity operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 351–366, Oct. 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294295

[1350] J. Criswell, N. Dautenhahn, and V. Adve, "KCoFI: Complete control-flow integrity for commodity operating system kernels," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. USA: IEEE Computer Society, 2014. [Online]. Available: https://doi.org/10.1109/SP.2014.26

[1351] M. Bugliesi, S. Calzavara, and R. Focardi, "Formal methods for web security," *Journal of Logical and Algebraic Methods in Programming*, vol. 87, pp. 110 – 126, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352220816301055

[1352] A. Guha, C. Saftoiu, and S. Krishnamurthi, "The essence of JavaScript," in *ECOOP 2010 – Object-Oriented Programming*, T. D'Hondt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 126–150.

[1353] M. Bodin, A. Charguéraud, D. Filaretti, P. Gardner, S. Maffeis, D. Naudziuniene, A. Schmitt, and G. Smith, "A trusted mechanised javascript specification," in *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 87–100. [Online]. Available: https://doi.org/10.1145/2535838.2535876

[1354] C. Fournet, N. Swamy, J. Chen, P.-E. Dagand, P.-Y. Strub, and B. Livshits, "Fully abstract compilation to JavaScript," *SIGPLAN Not.*, vol. 48, no. 1, p. 371–384, Jan. 2013. [Online]. Available: https://doi.org/10.1145/2480359.2429114

[1355] D. Hedin, A. Birgisson, L. Bello, and A. Sabelfeld, "JSFlow: Tracking information flow in javascript and its APIs," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1663–1671. [Online]. Available: https://doi.org/10.1145/2554850.2554909

[1356] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," *SIGPLAN Not.*, vol. 52, no. 6, p. 185–200, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140587.3062363

[1357] C. Watt, J. Renner, N. Popescu, S. Cauligi, and D. Stefan, "CT-Wasm: Type-driven secure cryptography for the web ecosystem," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290390

[1358] A. Bohannon and B. C. Pierce, "Featherweight Firefox: Formalizing the core of a web browser," in *USENIX Conference on Web Application Development (WebApps 10)*. USENIX Association, Jun. 2010. [Online]. Available: https://www.usenix.org/conference/webapps-10/featherweight-firefox-formalizing-core-web-browser

[1359] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google apps," in *Proceedings of the 6th ACM workshop on Formal methods in security engineering*. ACM, 2008, pp. 1–10.

[1360] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security," in *2010 23rd IEEE Computer Security Foundations Symposium*, 2010, pp. 290–304.

[1361] D. Fett, R. Küsters, and G. Schmitz, "An expressive model for the web infrastructure: Definition and application to the browser ID SSO system," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 673–688.

[1362] B. C. Pierce, "The science of deep specification (keynote)," in *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, SPLASH 2016, Amsterdam, Netherlands, October 30 - November 4, 2016*, E. Visser, Ed. ACM, 2016, p. 1. [Online]. Available: https://doi.org/10.1145/2984043.2998388

[1363] A. Erbsen, J. Philipoom, J. Gross, R. Sloan, and A. Chlipala, "Simple high-level code for cryptographic arithmetic - with proofs, without compromises," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1202–1219.

[1364] W. Mansky, A. W. Appel, and A. Nogin, "A verified messaging system," *PACMPL*, vol. 1, no. OOPSLA, pp. 87:1–87:28, 2017. [Online]. Available: https://doi.org/10.1145/3133911

[1365] N. Koh, Y. Li, Y. Li, L. Xia, L. Beringer, W. Honoré, W. Mansky, B. C. Pierce, and S. Zdancewic, "From C to interaction trees: specifying, verifying, and testing a networked server," in *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, A. Mahboubi and M. O. Myreen, Eds. ACM, 2019, pp. 234–248. [Online]. Available: https://doi.org/10.1145/3293880.3294106

[1366] Q. Cao, L. Beringer, S. Gruetter, J. Dodds, and A. W. Appel, "VST-Floyd: A separation logic tool to verify correctness of C programs," *J. Autom. Reasoning*, vol. 61, no. 1-4, pp. 367–422, 2018. [Online]. Available: https://doi.org/10.1007/s10817-018-9457-5

[1367] C. Cotrini, T. Weghorn, D. Basin, and M. Clavel, "Analyzing first-order role based access control," in *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*. IEEE, 2015, pp. 3–17.

[1368] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 196–205. [Online]. Available: https://doi.org/10.1145/1062455.1062502

[1369] N. Zhang, M. Ryan, and D. P. Guelev, "Synthesising verified access control systems in XACML," in *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, ser. FMSE '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 56–65. [Online]. Available: https://doi.org/10.1145/1029133.1029141

[1370] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 4, p. 391–420, Nov. 2006. [Online]. Available: https://doi.org/10.1145/1187441.1187442

[1371] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards formal verification of role-based access control policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 242–255, 2008.

[1372] T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "The Margrave tool for firewall analysis," in *Proceedings of the 24th International Conference on Large Installation System Administration*, ser. LISA'10. USA: USENIX Association, 2010, p. 1–8.

[1373] C. Bodei, P. Degano, L. Galletta, R. Focardi, M. Tempesta, and L. Veronese, "Language-independent synthesis of firewall policies," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, pp. 92–106.

[1374] C. Bertolissi, D. R. dos Santos, and S. Ranise, "Automated synthesis of run-time monitors to enforce authorization policies in business processes," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 297–308. [Online]. Available: https://doi.org/10.1145/2714576.2714633

[1375] D. Basin, S. Burri, and G. Karjoth, "Dynamic enforcement of abstract separation of duty constraints," in *14th European Symposium on Research in Computer Security (ESORICS)*, ser. 5789, M. Backes and P. N. (Eds.), Eds., vol. 5789. Saint Malo, France: Springer-Verlag, 09 2009, pp. 250–267. [Online]. Available: http://www.springer.com/computer/security+and+cryptology/book/978-3-642-04443-4

[1376] L. Lessig, *Code: And other laws of cyberspace*. ReadHowYouWant.com, 2009.

[1377] D. F. Sterne, "On the buzzword 'security policy'," in *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1991, pp. 219–230.

[1378] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems*, vol. 10, no. 4, pp. 265–310, November 1992.

[1379] J. Park and R. Sandhu, "The UCON ABC usage control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 128–174, 2004.

[1380] R. S. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role based access control:

Toward a unified standard," in *Proceedings of the 5th ACM Workshop on Role Based Access Control*, July 2000, pp. 47–63.

[1381] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information Systems Security*, vol. 3, no. 1, pp. 30–50, Feb. 2000.

[1382] M. C. Libicki, "Cyberspace is not a warfighting domain," *ISJLP*, vol. 8, p. 321, 2012.

[1383] R. Kissel, *Revision 2: Glossary of key information security terms*. Diane Publishing, 2013.

[1384] J. Crampton and J. Sellwood, "Path conditions and principal matching: a new approach to access control," in *Proceedings of the 19th ACM symposium on Access control models and technologies*. ACM, 2014, pp. 187–198.

[1385] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the linux operating system." in *USENIX Annual Technical Conference, FREENIX Track*, 2001, pp. 29–42.

[1386] F. Mayer, D. Caplan, and K. MacMillan, *SELinux by example: using security enhanced Linux*. Pearson Education, 2006.

[1387] S. Smalley and R. Craig, "Security enhanced (SE) Android: Bringing flexible MAC to Android," in *NDSS*, vol. 310, 2013, pp. 20–38.

[1388] N. Condori-Fernández and, V. N. L. Franqueira, and R. Wieringa, "Report on the survey of role-based access control (RBAC) in practice," Centre for Telematics and Information Technology, University of Twente, Tech. Rep. TR-CTIT-12-06, 2012.

[1389] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (ABAC) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, 2013.

[1390] L. Gong, M. Dageforde, and G. W. Ellison, *Inside Java 2 Platform Security*, 2nd ed. Reading, MA: Addison-Wesley, 2003.

[1391] B. A. La Macchia, S. Lange, M. Lyons, R. Martin, and K. T. Price, *.NET Framework Security*. Boston, MA: Addison-Wesley Professional, 2002.

[1392] N. Hardy, "The confused deputy," *Operating Systems Reviews*, vol. 22, no. 4, pp. 36–38, 1988.

[1393] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:14. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335360

[1394] R. Moan and I. Kawahara, "Superdistribution: An electronic infrastructure for the economy of the future," *Transactions of Information Processing Society of Japan*, vol. 38, no. 7, pp. 1465–1472, 1997.

[1395] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services," in *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.* IEEE, 2003, pp. 377–382.

[1396] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 132–145.

[1397] M. Salah, R. Philpott, S. Srinivas, J. Kemp, and J. Hodges, "FIDO UAF architectural overview," FIDO Alliance, Tech. Rep., February 2018, draft 20.

[1398] E. Rissanen, "eXtensible access control markup language (XACML)," Oasis, Tech. Rep. xacml-3.0-core-spec-os-en, 2013, version 3.0.

[1399] E. Rissanen, H. Lockhart, and T. Moses, "Xacml v3. 0 administration and delegation profile version 1.0," *Committee Draft*, vol. 1, 2009.

[1400] "DoD Trusted Computer System Evaluation Criteria," U.S. Department of Defense, 1985, DOD 5200.28-STD.

[1401] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, "The seaview security model," *IEEE Transactions on software engineering*, vol. 16, no. 6, pp. 593–607, 1990.

[1402] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 27-29, 1987*, 1987, pp. 184–195. [Online]. Available: https://doi.org/10.1109/SP.1987.10001

[1403] D. F. C. Brewer and M. J. Nash, "The Chinese Wall security policy," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.

[1404] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, August 1976.

[1405] N. Li, B. N. Grosof, and J. Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 1, pp. 128–171, 2003.

[1406] M. Abadi, M. Burrows, B. W. Lampson, and G. D. Plotkin, "A calculus for access control in distributed systems," *ACM Transactions Programming Language Systems*, vol. 15, no. 4, pp. 706–734, 1993. [Online]. Available: https://doi.org/10.1145/155183.155225

[1407] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 164–173.

[1408] J. DeTreville, "Binder, a logic-based security language," in *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 105–113.

[1409] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 921–930.

[1410] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 89–98.

[1411] ITU-T, "X509 (2014) ISO/IEC 9594-8:2014/Cor 2:2016, Directory: Public-key and attribute certificate framework [TECHNICAL CORRIGENDUM 2]," 2016.

[1412] M. Wong and W. Schlitt, "Sender policy framework (SPF) for authorizing use of domains in e-mail, version 1," RFC 4408, Tech. Rep., 2006.

[1413] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, "CSP is dead, long live CSP! on the insecurity of whitelists and the future of content security policy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1376–1387.

[1414] A. Van Kesteren, "Cross-origin resource sharing," *W3C Working Draft, http://www.w3.org/TR/2014/REC-cors-20140116/*, 2014.

[1415] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.

[1416] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee, "On the practicality of cryptographically enforcing dynamic access control policies in the cloud," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 819–838.

[1417] C. M. Ellison, B. Frantz, B. W. Lampson, R. Rivest, B. Thomas, and T. Ylönen, "SPKI certificate theory," *RFC*, vol. 2693, pp. 1–43, 1999. [Online]. Available: https://doi.org/10.17487/RFC2693

[1418] O. Bandmann, M. Dam, and B. S. Firozabadi, "Constrained delegation," in *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 131–140.

[1419] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[1420] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems." in *Winter 1988 Usenix Conference*. Citeseer, 1988, pp. 191–202.

[1421] H. Lockhart and B. Campbell, "Security assertion markup language (SAML) v2. 0 technical overview," *OASIS Committee Draft*, vol. 2, pp. 94–106, 2008.

[1422] D. Hardt, "The OAuth 2.0 authorization framework," Internet Requests for Comments, RFC Editor, RFC 6749, October 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt

[1423] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Digital identity guidelines," *NIST special publication*, vol. 800, pp. 63–3, 2017.

[1424] S. K. Modi, S. J. Elliott, J. Whetsone, and H. Kim, "Impact of age groups on fingerprint recognition performance," in *2007 IEEE Workshop on Automatic Identification Advanced Technologies*, June 2007, pp. 19–23.

[1425] L. Ghiani, D. Yambay, V. Mura, S. Tocco, G. L. Marcialis, F. Roli, and S. Schuckcrs, "Livdet 2013 fingerprint liveness detection competition 2013," in *2013 International Conference on Biometrics (ICB)*. IEEE, 2013, pp. 1–6.

[1426] R. D. Labati, A. Genovese, E. Muñoz, V. Piuri, F. Scotti, and G. Sforza, "Biometric recognition in automated border control: a survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 24, 2016.

[1427] L. M. Mayron, "Biometric authentication on mobile devices." *IEEE Security & Privacy*, vol. 13, no. 3, pp. 70–73, 2015.

[1428] M. S. Obaidat, I. Traore, and I. Woungang, *Biometric-Based Physical and Cybersecurity Systems*. Springer, 2019.

[1429] R. Joyce and G. Gupta, "Identity authentication based on keystroke latencies," *Communications of the ACM*, vol. 33, no. 2, pp. 168–176, 1990.

[1430] F. Monrose and A. D. Rubin, "Keystroke dynamics as a biometric for authentication," *Future Generation Computer Systems*, vol. 16, no. 4, pp. 351–359, 2000.

[1431] M. Ammar, Y. Yoshida, and T. Fukumura, "A new effective approach for on-line verification of signatures by using pressure features," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-6, no. 3, pp. 39–47, 1986.

[1432] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID connect core 1.0 incorporating errata set 1," *The OpenID Foundation, specification*, 2014.

[1433] W. Li and C. J. Mitchell, "Security issues in OAuth 2.0 SSO implementations," in *International Conference on Information Security*. Springer, 2014, pp. 529–541.

[1434] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of OAuth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1204–1215.

[1435] R. Seggelmann, M. Tüxen, and M. G. Williams, "Transport layer security (TLS) and datagram transport layer security (DTLS) heartbeat extension," Internet Engineering Task Force (IETF), Tech. Rep. RFC 6250, 2012. [Online]. Available: https://doi.org/10.17487/RFC6520

[1436] T. Y. Woo and S. S. Lam, "Verifying authentication protocols: Methodology and example," in *1993 International Conference on Network Protocols*. IEEE, 1993, pp. 36–45.

[1437] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

[1438] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[1439] M. Abadi, "Two facets of authentication," in *Proceedings. 11th IEEE Computer Security Foundations Workshop (Cat. No. 98TB100238)*. IEEE, 1998, pp. 27–32.

[1440] M. Bishop, *Computer security: Art and science. 2003*. Westford, MA: Addison Wesley Professional, 2003.

[1441] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice*. Upper Saddle River, NJ, USA: Pearson Education, 2012.

[1442] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transactionsoin Information and System Security (TISSEC)*, vol. 2, no. 2, pp. 159–176, 1999.

[1443] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage (TOS)*, vol. 5, no. 1, p. 2, 2009.

[1444] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.

[1445] S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh, "Certificate transparency with privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 329–344, 2017.

[1446] S. Sinclair and S. W. Smith, "What's wrong with access control in the real world?" *IEEE Security & Privacy*, vol. 8, no. 4, pp. 74–77, 2010.

[1447] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 48–62. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.13

[1448] W. Du, *Computer Security: A hands-on Approach*, 2017.

[1449] B. Chess and J. West, *Secure Programming with Static Analysis*, 1st ed. Addison-Wesley Professional, 2007.

[1450] M. Dowd, J. McDonald, and J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.

[1451] B. Goetz, J. Bloch, J. Bowbeer, D. Lea, D. Holmes, and T. Peierls, *Java Concurrency in Practice*. Addison-Wesley Longman, Amsterdam, 2006.

[1452] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. ACM, 2013, pp. 73–84.

[1453] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. Springer-Verlag, 1996, pp. 104–113.

[1454] M. Abadi, "Protection in programming-language translations," in *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, ser. ICALP '98. London, UK, UK: Springer-Verlag, 1998, pp. 868–883.

[1455] B. C. Pierce, *Types and Programming Languages*, 1st ed. The MIT Press, 2002.

[1456] L. Cardelli, "Type systems," in *The Computer Science and Engineering Handbook*, 1997, pp. 2208–2236.

[1457] Software Engineering Institute – Carnegie Mellon University, "SEI CERT C coding standard: Rules for developing safe, reliable, and secure systems," 2016.

[1458] IEEE Computer Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK®)*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[1459] "SPARK 2014," http://www.spark-2014.org/about, accessed: 2018-04-17.

[1460] H. Hosoya, J. Vouillon, and B. C. Pierce, "Regular expression types for xml," *ACM Trans. Program. Lang. Syst.*, vol. 27, no. 1, pp. 46–90, Jan. 2005.

[1461] M. S. Miller, "Robust composition: Towards a unified approach to access control and concurrency control," Ph.D. dissertation, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[1462] F. Long, D. Mohindra, R. C. Seacord, D. F. Sutherland, and D. Svoboda, *The CERT Oracle Secure Coding Standard for Java*, 1st ed. Addison-Wesley Professional, 2011.

[1463] MISRA Ltd, *MISRA-C:2012 Guidelines for the use of the C language in Critical Systems*, Motor Industry Software Reliability Association Std., Oct. 2013. [Online]. Available: www.misra.org.uk

[1464] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 317–331.

[1465] B. Livshits, M. Sridharan, Y. Smaragdakis, O. Lhoták, J. N. Amaral, B.-Y. E. Chang, S. Z. Guyer, U. P. Khedker, A. Møller, and D. Vardoulakis, "In defense of soundiness: A manifesto," *Commun. ACM*, vol. 58, no. 2, pp. 44–46, Jan. 2015.

[1466] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "SoK: Automated software diversity," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 276–291.

[1467] P. Holzinger, S. Triller, A. Bartel, and E. Bodden, "An in-depth study of more than ten years of java exploitation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 779–790.

[1468] B. Parno, J. M. McCune, and A. Perrig, "Bootstrapping trust in commodity computers," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 414–429.

[1469] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley Professional, 2002.

[1470] M. Howard, D. LeBlanc, and J. Viega, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

[1471] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, 1st ed. Addison-Wesley Professional, 2009.

[1472] Google, "Manage flash in your users' Chrome browsers," 2019. [Online]. Available: https://support.google.com/chrome/a/answer/7084871

[1473] OWASP, "OWASP cheat sheet series," 2019. [Online]. Available: https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series

[1474] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Requests for Comments, Network Working Group, RFC 2616, June 1999. [Online]. Available: https://www.ietf.org/rfc/rfc2616.txt

[1475] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith, "SoK: Lessons learned from Android security research for appified software platforms," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 433–451.

[1476] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," Internet Requests for Comments, Network Working Group, RFC 1738, December 1994. [Online]. Available: https://www.ietf.org/rfc/rfc1738.txt

[1477] W3C, "HTML 5.2," Dec 2017. [Online]. Available: https://www.w3.org/TR/html52/

[1478] "Ecmascript language specification," August 2019. [Online]. Available: https://tc39.es/ecma262/

[1479] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[1480] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446 (Proposed Standard), RFC Editor, Fremont, CA, USA, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446.txt

[1481] ——, "HTTP over TLS," Internet Requests for Comments, RFC Editor, RFC 2818, May 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2818.txt

[1482] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *IEEE Symposium on Security and Privacy*. IEEE, May 2012. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-quest-to-replace-passwords-a-framework-for-comparative-evaluation-of-web-authentication-schemes/

[1483] E. Enge, "Mobile VS. Desktop Usage in 2019," 2019. [Online]. Available: https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study

[1484] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The rise of the citizen developer: Assessing the security impact of online app generators," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, May 2018, pp. 634–647. [Online]. Available: https://doi.org/10.1109/SP.2018.00005

[1485] M. Zalewski, *The Tangled Web: A Guide to Securing Modern Web Applications*, 1st ed. San Francisco, CA, USA: No Starch Press, 2011.

[1486] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7540, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[1487] I. Fette and A. Melnikov, "The websocket protocol," Internet Requests for Comments, RFC Editor, RFC 6455, December 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6455.txt

[1488] E. Leung, "Learn to style HTML using CSS," Aug 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS

[1489] Node.js Foundation, "Node.js," 2019. [Online]. Available: https://nodejs.org/en/

[1490] "Webassembly 1.0," 2019. [Online]. Available: https://webassembly.org/

[1491] Google, "Android Developer Documentation - WebView," 2019. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView

[1492] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A Large-Scale Study of Mobile Web App Security," in *Proceedings of the Mobile Security Technologies Workshop (MoST)*, May 2015.

[1493] M. Georgiev, S. Jana, and V. Shmatikov, "Breaking and fixing origin-based access control in hybrid web/mobile application frameworks," *NDSS symposium*, vol. 2014, pp. 1–15, 2014.

[1494] ——, "Rethinking security of web-based system applications," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2015, pp. 366–376. [Online]. Available: https://doi.org/10.1145/2736277.2741663

[1495] H. Lockheimer, "Android and security," 2012. [Online]. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html

[1496] Apple Inc., "App store review guidelines," 2019. [Online]. Available: https://developer.apple.com/app-store/review/guidelines/

[1497] Android Developers, "Sign your app | Android Developers," 2019. [Online]. Available: https://developer.android.com/studio/publish/app-signing

[1498] D. C. Nguyen, E. Derr, M. Backes, and S. Bugiel, "Short text, large effect: Measuring the impact of user reviews on Android app security & privacy," in *Proceedings of the IEEE Symposium on Security & Privacy, May 2019*. IEEE, May 2019. [Online]. Available: https://publications.cispa.saarland/2815/

[1499] A. Barth, C. Reis, C. Jackson, and Google Chrome Team, "The security architecture of the chromium browser," Jan. 2008. [Online]. Available: http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

[1500] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The multi-principal OS construction of the Gazelle web browser," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 417–432. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855794

[1501] Google, "Chrome sandbox," 2019. [Online]. Available: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

[1502] ——, "Android application sandbox," 2019. [Online]. Available: https://source.android.com/security/app-sandbox

[1503] A. Barth, "The web origin concept," Internet Requests for Comments, RFC Editor, RFC 6454, December 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6454.txt

[1504] T. C. Projects, "Site Isolation Design Document." [Online]. Available: https://www.chromium.org/developers/design-documents/site-isolation

[1505] M. West, "Initial assignment for the content security policy directives registry," Internet Requests for Comments, Google, Inc., RFC 7762, January 2016.

[1506] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi, and P. Tabriz, "Where the wild warnings are: Root causes of Chrome HTTPS certificate errors," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and*

Communications Security, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 1407–1420. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134007

[1507] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes, "Improving SSL Warnings: Comprehension and Adherence," in *Conference on Human Factors and Computing Systems*, 2015.

[1508] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," Internet Requests for Comments, RFC Editor, RFC 6797, November 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6797.txt

[1509] "HTTPS everywhere," Mar 2018. [Online]. Available: https://www.eff.org/https-everywhere

[1510] Access, "The weakest link in the chain: Vulnerabilities in the SSL certificate authority system and what should be done about them," 2011. [Online]. Available: https://www.accessnow.org/cms/assets/uploads/archive/docs/Weakest_Link_in_the_Chain.pdf

[1511] Google - Certificate Transparency Team, "Certificate transparency," 2019. [Online]. Available: https://www.certificate-transparency.org/

[1512] J. Reschke, "The 'Basic' HTTP Authentication Scheme," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7617, September 2015. [Online]. Available: https://tools.ietf.org/html/rfc7617

[1513] M. Harbach, E. Von Zezschwitz, A. Fichtner, A. De Luca, and M. Smith, "It's a hard lock life: A field study of smartphone (un)locking behavior and risk perception," in *Proceedings of the Tenth USENIX Conference on Usable Privacy and Security*, ser. SOUPS '14. Berkeley, CA, USA: USENIX Association, 2014, pp. 213–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=3235838.3235857

[1514] A. De Luca, E. von Zezschwitz, N. D. H. Nguyen, M.-E. Maurer, E. Rubegni, M. P. Scipioni, and M. Langheinrich, "Back-of-device authentication on smartphones," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: ACM, 2013, pp. 2389–2398. [Online]. Available: http://doi.acm.org/10.1145/2470654.2481330

[1515] A. Barth, "Http state management mechanism," Internet Requests for Comments, RFC Editor, RFC 6265, April 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6265.txt

[1516] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta, "Surviving the web: A journey into web session security," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 13:1–13:34, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/3038923

[1517] National Institute of Standards and Technology (NIST), U.S., "NIST Special Publication 800-63B – Digital Identity Guidelines," 2019. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63b.html

[1518] National Cyber Security Center (NCSC), UK, "Password administration for system owners," 2019. [Online]. Available: https://www.ncsc.gov.uk/collection/passwords/updating-your-approach

[1519] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: Password use in the wild," in *Proceedings of the SIGCHI Conference on Human*

Factors in Computing Systems, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 383–392. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753384

[1520] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of passwords and people: Measuring the effect of password-composition policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2595–2604. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979321

[1521] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. Emami Naeini, H. Habib, N. Johnson, and W. Melicher, "Design and evaluation of a data-driven password meter," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 3775–3786. [Online]. Available: http://doi.acm.org/10.1145/3025453.3026050

[1522] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley, "Does my password go up to eleven?: The impact of password meters on password selection," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: ACM, 2013, pp. 2379–2388. [Online]. Available: http://doi.acm.org/10.1145/2470654.2481329

[1523] S. G. Lyastani, M. Schilling, S. Fahl, M. Backes, and S. Bugiel, "Better managed than memorized? Studying the impact of managers on password strength and reuse," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 203–220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lyastani

[1524] M. View, J. Rydell, M. Pei, and S. Machani, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, May 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6238.txt

[1525] D. Balfanz, A. Czeskis, J. Hodges, J. J.C. Jones, M. B. Jones, A. Kumar, A. Liao, R. Lindemann, and E. Lundberg, "Web authentication: An API for accessing public key credentials," https://www.w3.org/TR/webauthn-1/, 2019.

[1526] D. Hardt, "The OAuth 2.0 Authorization Framework," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 6749, October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749

[1527] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "Openid connect core 1.0," 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html

[1528] W. Li and C. J. Mitchell, "Analysing the security of Google's implementation of OpenID connect," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, ser. DIMVA 2016. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 357–376. [Online]. Available: https://doi.org/10.1007/978-3-319-40667-1_18

[1529] S. Fahl, S. Dechand, H. Perl, F. Fischer, J. Smrcek, and M. Smith, "Hey, NSA: Stay away from my market! Future proofing app markets against powerful attackers." in *ACM Conference on Computer and Communications Security*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 1143–1155. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2014.html#FahlDPFSS14

[1530] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on Android," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 2187–2200. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134059

[1531] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated JavaScript libraries on the web," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/thou-shalt-not-depend-me-analysing-use-outdated-javascript-libraries-web/

[1532] P. Kumaraguru, S. Sheng, A. Acquisti, L. F. Cranor, and J. Hong, "Teaching Johnny not to fall for phish," *ACM Trans. Internet Technol.*, vol. 10, no. 2, pp. 7:1–7:31, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1754393.1754396

[1533] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham, "School of phish: A real-world evaluation of anti-phishing training," in *Proceedings of the 5th Symposium on Usable Privacy and Security*, ser. SOUPS '09. New York, NY, USA: ACM, 2009, pp. 3:1–3:12. [Online]. Available: http://doi.acm.org/10.1145/1572532.1572536

[1534] Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha, and M. Guizani, "SoK: A systematic review of software-based web phishing detection," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2797–2819, Fourthquarter 2017.

[1535] F. Callegati and M. Ramilli, "Frightened by links," *IEEE Security Privacy*, vol. 7, no. 6, pp. 72–76, Nov 2009.

[1536] T. Espiner, "Adobe addresses flash player 'clickjacking' flaw," Oct 2008. [Online]. Available: https://www.cnet.com/news/adobe-addresses-flash-player-clickjacking-flaw/

[1537] G. Maone, "NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! - what is it? - InformAction," 2019. [Online]. Available: https://noscript.net/

[1538] S. Tang, N. Dautenhahn, and S. T. King, "Fortifying web-based applications automatically," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 615–626. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046777

[1539] Mozilla and individual contributors, "Web Storage API," 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

[1540] ——, "IndexedDB API," 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[1541] J. Manico, D. Righetto, and P. Ionescu, "JSON web token for Java. OWASP cheat sheet series," 2017. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.html

[1542] Mozilla and individual contributors, "Window.sessionStorage," 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[1543] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proceedings of the 4th USENIX Conference on Offensive Technologies*, ser. WOOT'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1925004.1925009

[1544] M. Eiband, M. Khamis, E. von Zezschwitz, H. Hussmann, and F. Alt, "Understanding shoulder surfing in the wild: Stories from users and observers," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 4254–4265. [Online]. Available: http://doi.acm.org/10.1145/3025453.3025636

[1545] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Communications of the ACM*, vol. 45, no. 2, pp. 128–, Feb. 2002. [Online]. Available: http://doi.acm.org/10.1145/503124.503156

[1546] F. Quinkert, T. Lauinger, W. K. Robertson, E. Kirda, and T. Holz, "It's not what it looks like: Measuring attacks and defensive registrations of homograph domains," in *7th IEEE Conference on Communications and Network Security, CNS 2019, Washington, DC, USA, June 10-12, 2019*, 2019, pp. 259–267. [Online]. Available: https://doi.org/10.1109/CNS.2019.8802671

[1547] S. Aonzo, A. Merlo, G. Tavella, and Y. Fratantonio, "Phishing attacks on modern Android," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1788–1801. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243778

[1548] Y. Fratantonio, C. Qian, S. Chung, and W. Lee, "Cloak and dagger: From two permissions to complete control of the UI feedback loop," in *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2017.

[1549] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 603–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon

[1550] OWASP, "Top 10-2017 A1-Injection," 2017. [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_A1-Injection

[1551] ——, "Blind SQL Injection," 2013. [Online]. Available: https://www.owasp.org/index.php/Blind_SQL_Injection

[1552] Oracle Corporation, "Prepared SQL Statement Syntax," 2019. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-prepared-statements.html

[1553] The PostgreSQL Global Development Group, "PostgreSQL: Documentation: 11: PREPARE," 2019. [Online]. Available: https://www.postgresql.org/docs/current/sql-prepare.html

[1554] DHS and CISA, "CVE - common vulnerabilities and exposures," 2019. [Online]. Available: https://cve.mitre.org/

[1555] J. Reschke, "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)," Internet Request for Comments, Internet Engineering Task Force (IETF), RFC, June 2011. [Online]. Available: https://tools.ietf.org/html/rfc6266

[1556] "Cross-site scripting," April 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting

[1557] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: ACM, 2008, pp. 75–88. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455782

[1558] I. Synopsis, "The heartbleed bug," http://heartbleed.com/, 2014.

[1559] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of Heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 475–488. [Online]. Available: http://doi.acm.org/10.1145/2663716.2663755

[1560] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," in *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '06. New York, NY, USA: ACM, 2006, pp. 372–382. [Online]. Available: http://doi.acm.org/10.1145/1111037.1111070

[1561] M. Van Gundy and H. Chen, "Noncespaces: Using randomization to defeat cross-site scripting attacks," *Comput. Secur.*, vol. 31, no. 4, pp. 612–628, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2011.12.004

[1562] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 427–444. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068856

[1563] S. Fahl, Y. Acar, H. Perl, and M. Smith, "Why Eve and Mallory (also) love webmasters: A study on the root causes of SSL misconfigurations," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 507–512. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590341

[1564] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, ""I have no idea what I'm doing" - on the usability of deploying HTTPS," in *26th USENIX Security Symposium (USENIX Security 2017)*, August 2017, p. 1338. [Online]. Available: https://publications.cispa.saarland/2654/

[1565] K. Krombholz, K. Busse, K. Pfeffer, M. Smith, and E. von Zezschwitz, ""If HTTPS were secure, i wouldn't need 2FA" - end user and administrator mental models of HTTPS," in *S&P 2019*, May 2019. [Online]. Available: https://publications.cispa.saarland/2788/

[1566] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function," Internet Engineering Task Force, Internet-Draft draft-irtf-cfrg-argon2-08, Oct. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-argon2-08

[1567] S. Josefsson, "PKCS #5: Password-Based Key Derivation Function 2 (PBKDF2) Test Vectors," RFC 6070, Jan. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6070.txt

[1568] OWASP, "OWASP - password storage cheat sheet," 2019. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[1569] Wikipedia contributors, "List of data breaches — Wikipedia, the free encyclopedia," 2019. [Online]. Available: https://en.wikipedia.org/wiki/List_of_data_breaches

[1570] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein, "Protecting accounts from credential stuffing with password breach alerting," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1556–1571. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/thomas

[1571] "Bill gates: Trustworthy computing," https://www.wired.com/2002/01/bill-gates-trustworthy-computing/, 2002.

[1572] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.

[1573] Poneman Institute, "2018 cost of a data breach study: Global overview," July 2018, online. [Online]. Available: https://securityintelligence.com/series/ponemon-institute-cost-of-a-data-breach-2018/

[1574] G. McGraw, "Testing for security during development: why we should scrap penetrate-and-patch," *IEEE Aerospace and Electronic Systems Magazine*, vol. 13, no. 4, pp. 13–15, April 1998.

[1575] T. Greene, "That Heartbleed problem may be more pervasive than you think," January 2017, online. [Online]. Available: https://www.networkworld.com/article/3162232/security/that-hearbleed-problem-may-be-more-pervasive-than-you-think.html

[1576] eWeek editors, "Microsoft trustworthy computing timeline," October 2005, online. [Online]. Available: https://www.eweek.com/security/microsoft-trustworthy-computing-timeline

[1577] M. Howard and D. E. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2003.

[1578] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[1579] Microsoft, "The security development lifecycle," https://www.microsoft.com/en-us/securityengineering/sdl/, 2019.

[1580] "Owasp secure software development lifecycle project," https://www.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Project, 2018.

[1581] P. Morrison, D. Moye, R. Pandita, and L. Williams, "Mapping the field of software life cycle security metrics," *Information and Software Technology*, vol. 102, pp. 146 – 159, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491830096X

[1582] M. Howard, "Fending off future attacks by reducing attack surface," *MSDN Magazine*, February 4, 2003. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms972812.aspx

[1583] I. Arce, K. Clark-Fisher, N. Daswani, J. DelGrosso, D. Dhillon, C. Kern, T. Kohno, C. Landwehr, G. McGraw, B. Schoenfield *et al.*, "Avoiding the top 10 software security design flaws," IEEE Computer Society Center for Secure Design (CSD), Tech. Rep., 2014.

[1584] Synopsys, "2018 open source security and risk analysis," Synopsys Center for Open Source Research and Innovation, Tech. Rep., 2018. [Online]. Available: https://www.blackducksoftware.com/open-source-security-risk-analysis-2018

[1585] A. Austin, C. Holmgreen, and L. Williams, "A comparison of the efficiency and effectiveness of vulnerability discovery techniques," *Information and Software Technology*, vol. 55, no. 7, pp. 1279 – 1288, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584912002339

[1586] P. Hope, G. McGraw, and A. I. Anton, "Misuse and abuse cases: getting past the positive," *IEEE Security and Privacy*, vol. 2, no. 3, pp. 90–92, May 2004.

[1587] G. Sindre and A. L. Opdahl, "Eliciting security requirements by misuse cases," in *Proceedings 37th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS-Pacific 2000*, Nov 2000, pp. 120–131.

[1588] K. Tuma, G. Calikli, and R. Scandariato, "Threat analysis of software systems: A systematic literature review," *Journal of Systems and Software*, vol. 144, 06 2018.

[1589] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 371–386, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.60

[1590] C. Theisen, N. Munaiah, M. Al-Zyoud, J. C. Carver, A. Meneely, and L. Williams, "Attack surface definitions: A systematic literature review," *Information and Software Technology*, vol. 104, pp. 94 – 103, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584918301514

[1591] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," *Comput. Stand. Interfaces*, vol. 50, no. C, pp. 107–115, Feb. 2017. [Online]. Available: https://doi.org/10.1016/j.csi.2016.10.001

[1592] J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, *Software Security Engineering: A Guide for Project Managers (The SEI Series in Software Engineering)*, 1st ed. Addison-Wesley Professional, 2008.

[1593] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 148–157. [Online]. Available: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=998675.999421

[1594] G. Elahi and E. Yu, "A goal oriented approach for modeling and analyzing security trade-offs," in *Proceedings of the 26th International Conference on Conceptual Modeling*, ser. ER'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 375–390. [Online]. Available: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=1784489.1784524

[1595] V. Saini, Q. Duan, and V. Paruchuri, "Threat modeling using attack trees," *J. Comput.*

Sci. Coll., vol. 23, no. 4, pp. 124–131, Apr. 2008. [Online]. Available: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=1352079.1352100

[1596] L. Williams, A. Meneely, and G. Shipley, "Protection poker: The new software security "game";," *IEEE Security Privacy*, vol. 8, no. 3, pp. 14–20, May 2010.

[1597] G. McGraw, "The new killer app for security: Software inventory," *Computer*, vol. 51, no. 2, pp. 60–62, February 2018.

[1598] R. Kuhn, M. Raunak, and R. Kacker, "What proportion of vulnerabilities can be attributed to ordinary coding errors?: Poster," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS '18. New York, NY, USA: ACM, 2018, pp. 30:1–30:1. [Online]. Available: http://doi.acm.org/10.1145/3190619.3191686

[1599] NIST Computer Security, "Guide for conducting risk assessments," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-30 Revision 1, 2012. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final

[1600] SAFECode, "Fundamental practices for secure software development: Essential elements of a secure development lifecycle program," SAFECode, Tech. Rep. Third Edition, March 2018. [Online]. Available: https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

[1601] Federal Office for Information Security, "Guidelines for developer documentation according to common criteria version 3.1," Federal Office for Information Security, Tech. Rep. Version 1.0, 2007. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CommonCriteriaDevelopersGuide_1_0.pdf

[1602] B. D. Win, R. Scandariato, K. Buyens, J. Grégoire, and W. Joosen, "On the secure software development process: Clasp, sdl and touchpoints compared," *Information and Software Technology*, vol. 51, no. 7, pp. 1152 – 1171, 2009, detailed data analysis of practices available online. [Online]. Available: lirias.kuleuven.be/1655460

[1603] SAFECode, "Practical security stories and security tasks for agile development environments," SAFECode, Tech. Rep., July 2012. [Online]. Available: http://safecode.org/wp-content/uploads/2018/01/SAFECode_Agile_Dev_Security0712.pdf

[1604] Microsoft, "Secure devops," https://www.microsoft.com/en-us/securityengineering/devsecops, 2019.

[1605] "Owasp mobile security project," https://www.owasp.org/index.php/OWASP_Mobile_Security_Project, 2017.

[1606] T. Eston, "OWASP mobile security project - mobile threat model," 2013. [Online]. Available: https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Mobile_Threat_Model

[1607] B. Sullivan, S. Tabet, E. Bonver, J. Furlong, S. Orrin, and P. Uhley, "Practices for secure development of cloud applications," SAFECode, Tech. Rep., December 2013. [Online]. Available: https://safecode.org/publication/SAFECode_CSA_Cloud_Final1213.pdf

[1608] J. Voas, R. Kuhn, P. Laplante, and S. Applebaum, "Inernet of things (iot) trust concerns," National Institute of Standards and Technology, Tech. Rep. Draft, 2018. [Online].

Available: https://csrc.nist.gov/publications/detail/white-paper/2018/10/17/iot-trust-concerns/draft

[1609] ENISA, "Cloud computing: Benefits, risks and recommendations for information security," ENISA, Tech. Rep., 2009. [Online]. Available: https://www.enisa.europa.eu/publications/cloud-computing-risk-assessment/at_download/fullReport

[1610] Y. Vardi, "Where automotive cybersecurity is headed in 2019," 2019. [Online]. Available: https://thenextweb.com/contributors/2019/02/10/where-automotive-cybersecurity-is-headed-in-2019/

[1611] G. McGraw, "From mainframes to connected cars: How software drives the automotive industry," *Security Ledger*, vol. August 15, August 2018.

[1612] G. McGraw, S. Migues, and J. West, "Building security in maturity model," https://www.bsimm.com/, 2009.

[1613] L. Williams, G. McGraw, and S. Migues, "Engineering security vulnerability prevention, detection, and response," *IEEE Software*, vol. 35, no. 5, pp. 76–80, Sep. 2018.

[1614] G. A. Moore, *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers*. Harper Collins, 2002.

[1615] R. Anderson, "Why information security is hard - an economic perspective," in *Seventeenth Annual Computer Security Applications Conference*, Dec 2001, pp. 358–365.

[1616] N. R. Mead and C. Woody, *Cyber Security Engineering: A Practical Approach for Systems and Software Assurance*, 1st ed. Addison-Wesley Professional, 2016.

[1617] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing cyber resilient systems: a systems security engineering approach," National Institute of Standards and Technology, Tech. Rep. Draft (FPD) SP 800-160 Volume 2, 2019. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-160/vol-2/draft

[1618] W. Newhouse, S. Keith, B. Scribner, and G. Witte, "National Initiative for Cybersecurity Education (NICE) cybersecurity workforce framework," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-181, 2017. [Online]. Available: https://www.nist.gov/itl/applied-cybersecurity/nice/resources/nice-cybersecurity-workforce-framework

[1619] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 0.5 ed., 2020. [Online]. Available: http://toc.cryptobook.us/

[1620] K. Martin, *Everyday Cryptography: Fundamental Principles and Applications*, 2nd ed. Oxford University Press, 2017.

[1621] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková, "Collective information security in large-scale urban protests: the case of Hong Kong," in *30th USENIX Security Symposium (USENIX Security '21)*. USENIX Association, 2021.

[1622] A. Greenberg, "Monero, the drug dealer's cryptocurrency of choice, is on fire," 2017. [Online]. Available: https://www.wired.com/2017/01/monero-drug-dealers-cryptocurrency-choice-fire/

[1623] C. Dion-Schwarz, D. Manheim, and P. B. Johnston, "Terrorist use of cryptocurrencies: Technical and organizational barriers and future threats," 2019. [Online]. Available:

https://www.rand.org/content/dam/rand/pubs/research_reports/RR3000/RR3026/RAND_RR3026.pdf

[1624] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, M. Green, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, M. A. Specter, and D. J. Weitzner, "Keys under doormats: mandating insecurity by requiring government access to all data and communications," *Journal of Cybersecurity*, vol. 1, no. 1, pp. 69–79, 11 2015. [Online]. Available: https://doi.org/10.1093/cybsec/tyv009

[1625] K. Martin, *Cryptography: The Key to Digital Security, How It Works, and Why It Matters*. W. W. Norton & Company, 2020.

[1626] NIST, "Secure Hash Standard (SHS)," August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[1627] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. D. Jr., "Advanced Encryption Standard (AES)," November 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[1628] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," November 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[1629] J. Mason, K. Watkins, J. Eisner, and A. Stubblefield, "A natural language approach to automated cryptanalysis of two-time pads," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 235–244. [Online]. Available: https://doi.org/10.1145/1180405.1180435

[1630] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997, updated by RFC 6151. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[1631] P. Rogaway, "Nonce-based symmetric encryption," in *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, ser. Lecture Notes in Computer Science, B. K. Roy and W. Meier, Eds., vol. 3017. Springer, 2004, pp. 348–359. [Online]. Available: https://doi.org/10.1007/978-3-540-25937-4_22

[1632] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, "Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS," in *10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, USA, August 8-9, 2016*, N. Silvanovich and P. Traynor, Eds. USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/bock

[1633] S. Gueron, A. Langley, and Y. Lindell, "AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption," RFC 8452 (Informational), RFC Editor, Fremont, CA, USA, Apr. 2019. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8452.txt

[1634] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, V. Atluri, Ed. ACM, 2002, pp. 98–107. [Online]. Available: https://doi.org/10.1145/586110.586125

[1635] P. Rogaway and T. Shrimpton, "A provable-security treatment of the key-wrap problem,"

in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, 2006, pp. 373–390. [Online]. Available: https://doi.org/10.1007/11761679_23

[1636] S. Vaudenay, "Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ..." in *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 2332. Springer, 2002, pp. 534–546. [Online]. Available: https://doi.org/10.1007/3-540-46035-7_35

[1637] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 526–540. [Online]. Available: https://doi.org/10.1109/SP.2013.42

[1638] M. R. Albrecht, K. G. Paterson, and G. J. Watson, "Plaintext recovery attacks against SSH," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009, pp. 16–26. [Online]. Available: https://doi.org/10.1109/SP.2009.5

[1639] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed., vol. 1976. Springer, 2000, pp. 531–545. [Online]. Available: https://doi.org/10.1007/3-540-44448-3_41

[1640] C. Namprempre, P. Rogaway, and T. Shrimpton, "Reconsidering generic composition," in *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, ser. Lecture Notes in Computer Science, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Springer, 2014, pp. 257–274. [Online]. Available: https://doi.org/10.1007/978-3-642-55220-5_15

[1641] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439 (Informational), RFC Editor, Fremont, CA, USA, Jun. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8439.txt

[1642] M. Abdalla, M. Bellare, and G. Neven, "Robust encryption," *J. Cryptol.*, vol. 31, no. 2, pp. 307–350, 2018. [Online]. Available: https://doi.org/10.1007/s00145-017-9258-8

[1643] K. Moriarty (Ed.), B. Kaliski, J. Jonsson, and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017 (Informational), RFC Editor, Fremont, CA, USA, Nov. 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8017.txt

[1644] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed., vol. 1462. Springer, 1998, pp. 1–12. [Online]. Available: https://doi.org/10.1007/BFb0055716

[1645] H. Böck, J. Somorovsky, and C. Young, "Return of Bleichenbacher's oracle threat (ROBOT)," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 817–849. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[1646] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," *J. Cryptol.*, vol. 17, no. 2, pp. 81–104, 2004. [Online]. Available: https://doi.org/10.1007/s00145-002-0204-y

[1647] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055638

[1648] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for Authentication and Key Establishment*, 2nd ed., ser. Information Security and Cryptography. Springer, 2020.

[1649] M. R. Albrecht, J. Massimo, K. G. Paterson, and J. Somorovsky, "Prime and prejudice: Primality testing under adversarial conditions," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 281–298. [Online]. Available: https://doi.org/10.1145/3243734.3243787

[1650] S. D. Galbraith, J. Massimo, and K. G. Paterson, "Safety in numbers: On the need for robust Diffie-Hellman parameter validation," in *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, D. Lin and K. Sako, Eds., vol. 11443. Springer, 2019, pp. 379–407. [Online]. Available: https://doi.org/10.1007/978-3-030-17259-6_13

[1651] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985. [Online]. Available: https://doi.org/10.1109/TIT.1985.1057074

[1652] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle Diffie-Hellman assumptions and an analysis of DHIES," in *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, ser. Lecture Notes in Computer Science, D. Naccache, Ed., vol. 2020. Springer, 2001, pp. 143–158. [Online]. Available: https://doi.org/10.1007/3-540-45353-9_12

[1653] A. Menezes and N. P. Smart, "Security of signature schemes in a multi-user setting," *Des. Codes Cryptogr.*, vol. 33, no. 3, pp. 261–274, 2004. [Online]. Available: https://doi.org/10.1023/B:DESI.0000036250.18062.3f

[1654] M. Bellare and P. Rogaway, "The exact security of digital signatures - how to sign with RSA and Rabin," in *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Springer, 1996, pp. 399–416. [Online]. Available: https://doi.org/10.1007/3-540-68339-9_34

[1655] NIST, "Digital Signature Standard (DSS)," July 2013. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[1656] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032 (Informational), RFC Editor, Fremont, CA, USA, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8032.txt

[1657] L. Lamport, "Constructing digital signatures from a one-way function," SRI International Computer Science Laboratory, Tech. Rep. SRI-CSL-98, October 1979.

[1658] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Plenum Press, New York, 1982, pp. 199–203. [Online]. Available: https://doi.org/10.1007/978-1-4757-0602-4_18

[1659] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 552–565. [Online]. Available: https://doi.org/10.1007/3-540-45682-1_32

[1660] N. Smart, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Cryptography, version 1.0.1. [Online]. Available: https://www.cybok.org/

[1661] K. G. Paterson and T. van der Merwe, "Reactive and proactive standardisation of TLS," in *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*, ser. Lecture Notes in Computer Science, L. Chen, D. A. McGrew, and C. J. Mitchell, Eds., vol. 10074. Springer, 2016, pp. 160–186. [Online]. Available: https://doi.org/10.1007/978-3-319-49100-4_7

[1662] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, 2006, pp. 409–426. [Online]. Available: https://doi.org/10.1007/11761679_25

[1663] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 332, 2004. [Online]. Available: http://eprint.iacr.org/2004/332

[1664] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1393, 2019. [Online]. Available: https://eprint.iacr.org/2019/1393

[1665] N. Koblitz and A. Menezes, "Critical perspectives on provable security: Fifteen years of "another look" papers," *Adv. Math. Commun.*, vol. 13, no. 4, pp. 517–558, 2019. [Online]. Available: https://doi.org/10.3934/amc.2019034

[1666] E. Barker, "Recommendation for Key Management: Part 1 – General," May 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[1667] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: breaking TLS using SSLv2," in *25th USENIX Security Symposium,*

USENIX Security 16, Austin, TX, USA, August 10-12, 2016, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 689–706. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram

[1668] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue, "A messy state of the union: taming the composite state machines of TLS," *Commun. ACM*, vol. 60, no. 2, pp. 99–107, 2017. [Online]. Available: https://doi.org/10.1145/3023357

[1669] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/

[1670] D. Ott and C. Peikert, "Identifying research challenges in post quantum cryptography migration and cryptographic agility," Computing Community Consortium Technical Report, 2019. [Online]. Available: https://cra.org/ccc/wp-content/uploads/sites/2/2018/11/CCC-Identifying-Research-Challenges-in-PQC-Workshop-Report.pdf

[1671] T. R. Johnson, "American cryptology during the cold war, 1945-1989. book III: Retrenchment and reform, 1972-1980," 1998. [Online]. Available: https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/cryptologic-histories/cold_war_iii.pdf

[1672] E. B. Barker and J. M. Kelsey, "Recommendation for random number generation using deterministic random bit generators," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90A, 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[1673] VCAT, "NIST cryptographic standards and guidelines development process — report and recommendations of the Visiting Committee on Advanced Technology of the National Institute of Standards and Technology," July 2014. [Online]. Available: https://www.nist.gov/system/files/documents/2017/05/09/VCAT-Report-on-NIST-Cryptographic-Standards-and-Guidelines-Process.pdf

[1674] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700

[1675] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, G. L. Miller, Ed. ACM, 1996, pp. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[1676] M. Bellare, T. Kohno, and C. Namprempre, "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, pp. 206–241, 2004. [Online]. Available: https://doi.org/10.1145/996943.996945

[1677] S. Ruhault, "SoK: Security models for pseudo-random number generators," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 506–544, 2017. [Online]. Available: https://doi.org/10.13154/tosc.v2017.i1.506-544

[1678] P. L. Gorski, Y. Acar, L. L. Iacono, and S. Fahl, "Listen to developers! A participatory design study on security warnings for cryptographic APIs," in *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, R. Bernhaupt, F. F. Mueller, D. Verweij, J. Andres, J. McGrenere, A. Cockburn, I. Avellino, A. Goguey, P. Bjøn, S. Zhao, B. P. Samson, and R. Kocielnik, Eds. ACM, 2020, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3313831.3376142

[1679] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical

[1680] J. Jancar, V. Sedlacek, P. Svenda, and M. Sýs, "Minerva: The curse of ECDSA nonces systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 281–308, 2020. [Online]. Available: https://doi.org/10.13154/tches.v2020.i4.281-308

[1681] S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer, "Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 1767–1784. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/weiser

[1682] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password interception in a SSL/TLS channel," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 583–599. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_34

[1683] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[1684] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," *IACR Cryptol. ePrint Arch.*, vol. 2002, p. 169, 2002. [Online]. Available: http://eprint.iacr.org/2002/169

[1685] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers' Track at the RSA Conference*. Springer, 2006, pp. 1–20.

[1686] F. Tramèr, D. Boneh, and K. Paterson, "Remote side-channel attacks on anonymous transactions," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 2739–2756. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/tramer

[1687] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. A. Smolin, "Experimental quantum cryptography," in *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, ser. Lecture Notes in Computer Science, I. Damgård, Ed., vol. 473.

Springer, 1990, pp. 253–265. [Online]. Available: https://doi.org/10.1007/3-540-46877-3_23

[1688] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 37–51. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_4

[1689] F. Piessens, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Software Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[1690] I. Verbauwhede, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Hardware Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[1691] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., vol. 4833. Springer, 2007, pp. 29–50. [Online]. Available: https://doi.org/10.1007/978-3-540-76900-2_3

[1692] T. Pornin, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979 (Informational), RFC Editor, Fremont, CA, USA, Aug. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6979.txt

[1693] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your ps and qs: Detection of widespread weak keys in network devices," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, T. Kohno, Ed. USENIX Association, 2012, pp. 205–220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

[1694] K. Mowery, M. Y. C. Wei, D. Kohlbrenner, H. Shacham, and S. Swanson, "Welcome to the entropics: Boot-time entropy in embedded devices," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 589–603. [Online]. Available: https://doi.org/10.1109/SP.2013.46

[1695] T. Ristenpart and S. Yilek, "When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010. [Online]. Available: https://www.ndss-symposium.org/ndss2010/when-good-randomness-goes-bad-virtual-machine-reset-vulnerabilities-and-hedging-deployed

[1696] E. Barker and W. C. Barker, "Recommendation for Key Management: Part 2 – Best Practices for Key Management Organizations," May 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt2r1.pdf

[1697] E. Barker and Q. Dang, "Recommendation for Key Management: Part 3 – Application-Specific Key Management Guidance," January 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

[1698] T. Jager, K. G. Paterson, and J. Somorovsky, "One bad apple: Backwards compatibility attacks on state-of-the-art cryptography," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013. [Online]. Available: https://www.ndss-symposium.org/ndss2013/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography

[1699] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869 (Informational), Internet Engineering Task Force, May 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5869.txt

[1700] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," RFC 7914 (Informational), RFC Editor, Fremont, CA, USA, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7914.txt

[1701] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 33–62. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_2

[1702] S. Katzenbeisser, Ü. Koçabas, V. Rozic, A. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, ser. Lecture Notes in Computer Science, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, 2012, pp. 283–301. [Online]. Available: https://doi.org/10.1007/978-3-642-33027-8_17

[1703] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Public keys," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 626–642. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_37

[1704] M. Nemec, M. Sýs, P. Svenda, D. Klinec, and V. Matyas, "The return of Coppersmith's attack: Practical factorization of widely used RSA moduli," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1631–1648. [Online]. Available: https://doi.org/10.1145/3133956.3133969

[1705] S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot, "White-box cryptography and an AES implementation," in *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, ser. Lecture Notes in Computer Science, K. Nyberg and H. M. Heys, Eds., vol. 2595. Springer, 2002, pp. 250–270. [Online]. Available: https://doi.org/10.1007/3-540-36492-7_17

[1706] K. Cohn-Gordon, C. J. F. Cremers, and L. Garratt, "On post-compromise security," in *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal,*

June 27 - July 1, 2016. IEEE Computer Society, 2016, pp. 164–178. [Online]. Available: https://doi.org/10.1109/CSF.2016.19

[1707] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), RFC Editor, Fremont, CA, USA, May 2008, updated by RFCs 6818, 8398, 8399. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5280.txt

[1708] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962 (Experimental), RFC Editor, Fremont, CA, USA, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6962.txt

[1709] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 6960 (Proposed Standard), RFC Editor, Fremont, CA, USA, Jun. 2013, updated by RFC 8954. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6960.txt

[1710] J. Larisch, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 539–556. [Online]. Available: https://doi.org/10.1109/SP.2017.17

[1711] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 17–36. [Online]. Available: https://doi.org/10.1007/11535218_2

[1712] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10401. Springer, 2017, pp. 570–596. [Online]. Available: https://doi.org/10.1007/978-3-319-63688-7_19

[1713] G. Leurent and T. Peyrin, "From collisions to chosen-prefix collisions application to full SHA-1," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11478. Springer, 2019, pp. 527–555. [Online]. Available: https://doi.org/10.1007/978-3-030-17659-4_18

[1714] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, ser. Lecture Notes in Computer Science, G. R. Blakley and D. Chaum, Eds., vol. 196. Springer, 1984, pp. 47–53. [Online]. Available: https://doi.org/10.1007/3-540-39568-7_5

[1715] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, ser. Lecture Notes in Computer

Science, C. Laih, Ed., vol. 2894. Springer, 2003, pp. 452–473. [Online]. Available: https://doi.org/10.1007/978-3-540-40061-5_29

[1716] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 289–305. [Online]. Available: https://doi.org/10.1109/SP.2016.25

[1717] ETSI, "Security architecture and procedures for 5G System (3GPP TS 33.501 version 15.4.0 Release 15)." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf

[1718] C. Troncoso, M. Payer, J. Hubaux, M. Salathé, J. R. Larus, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, L. Barman, S. Chatel, K. G. Paterson, S. Capkun, D. A. Basin, J. Beutel, D. Jackson, M. Roeschlin, P. Leu, B. Preneel, N. P. Smart, A. Abidin, S. Gurses, M. Veale, C. Cremers, M. Backes, N. O. Tippenhauer, R. Binns, C. Cattuto, A. Barrat, D. Fiore, M. Barbosa, R. Oliveira, and J. Pereira, "Decentralized privacy-preserving proximity tracing," *IEEE Data Eng. Bull.*, vol. 43, no. 2, pp. 36–66, 2020. [Online]. Available: http://sites.computer.org/debull/A20june/p36.pdf

[1719] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 483–502. [Online]. Available: https://doi.org/10.1109/SP.2017.26

[1720] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1773–1788. [Online]. Available: https://doi.org/10.1145/3133956.3134063

[1721] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1044, 2020. [Online]. Available: https://eprint.iacr.org/2020/1044

[1722] N. Drucker and S. Gueron, "Selfie: reflections on TLS 1.3 with PSK," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 347, 2019. [Online]. Available: https://eprint.iacr.org/2019/347

[1723] F. Günther, B. Hale, T. Jager, and S. Lauer, "0-RTT key exchange with full forward secrecy," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 519–548. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_18

[1724] N. Aviram, K. Gellert, and T. Jager, "Session resumption protocols and efficient forward security for TLS 1.3 0-RTT," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11477. Springer, 2019, pp. 117–150. [Online]. Available: https://doi.org/10.1007/978-3-030-17656-3_5

[1725] D. Derler, K. Gellert, T. Jager, D. Slamanig, and C. Striecks, "Bloom filter encryption and

applications to efficient forward-secret 0-RTT key exchange," *J. Cryptol.*, vol. 34, no. 2, p. 13, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09374-3

[1726] F. Dallmeier, J. P. Drees, K. Gellert, T. Handirk, T. Jager, J. Klauke, S. Nachtigall, T. Renzelmann, and R. Wolf, "Forward-secure 0-RTT goes live: Implementation and performance analysis in QUIC," in *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, S. Krenn, H. Shulman, and S. Vaudenay, Eds., vol. 12579. Springer, 2020, pp. 211–231. [Online]. Available: https://doi.org/10.1007/978-3-030-65411-5_11

[1727] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan, "Dancing on the lip of the volcano: Chosen ciphertext attacks on Apple imessage," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 655–672. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/garman

[1728] A. Viand, P. Jattke, and A. Hithnawi, "SoK: Fully homomorphic encryption compilers," *CoRR*, vol. abs/2101.07078, 2021. [Online]. Available: https://arxiv.org/abs/2101.07078

[1729] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 370–389. [Online]. Available: https://doi.org/10.1109/EuroSP48549.2020.00031

[1730] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome," in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 644–660. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00048

[1731] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (7th Edition)*, 7th ed. Pearson, 2017.

[1732] A. S. Tanenbaum and D. Wetherall, *Computer Networks*, 5th ed. Prentice Hall, 2011.

[1733] W. Stallings, *Network Security Essentials, Applications and Standards (6th Edition)*, 6th ed. Pearson, 2016.

[1734] C. W. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*. Englewood Cliffs, New Jersey: Prentice-Hall, March 1995.

[1735] C. P. Pfleeger, S. L. Pfleeger, and J. Margulies, *Security in Computing (5th Edition)*, 5th ed. USA: Prentice Hall Press, 2015.

[1736] "Modbus," https://www.modbus.org/specs.php, accessed: 2021-03-23.

[1737] "Knx," https://www.knx.org, accessed: 2021-03-23.

[1738] I. O. for Standardization, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," International Organization for Standardization, Geneva, CH, Standard, Dec. 2015.

[1739] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 232–249. [Online]. Available: https://doi.org/10.1109/SP.2015.22

[1740] S. Bortzmeyer, "DNS Query Name Minimisation to Improve Privacy," RFC 7816, Mar. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7816.txt

[1741] "Internet society — dnssec deployment maps," https://www.internetsociety.org/deploy360/dnssec/maps/, accessed: 2021-02-19.

[1742] S. Singanamalla, S. Chunhapanya, J. Hoyland, M. Vavruša, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. Wood, "Oblivious dns over https (odoh): A practical privacy enhancement to dns," in *DNS Privacy Workshop 2021*, 2021.

[1743] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth 2013.

[1744] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, February 2014.

[1745] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks," in *Proceedings of the 23rd USENIX Security Symposium*, August 2014.

[1746] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the network time protocol," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1020, 2015.

[1747] T. Rytilahti, D. Tatang, J. Köpper, and T. Holz, "Masters of Time: An Overview of the NTP Ecosystem," in *IEEE EuroS&P*, 2018.

[1748] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (network) time travel with chronos," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[1749] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, 1999, pp. 173–186.

[1750] R. A. Bazzi and G. Konjevod, "On the establishment of distinct identities in overlay networks," *Distrib. Comput.*, vol. 19, no. 4, p. 267–287, Mar. 2007. [Online]. Available: https://doi.org/10.1007/s00446-006-0012-y

[1751] N. Borisov, "Computational puzzles as sybil defenses," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, 2006, pp. 171–176.

[1752] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson, "Sybil-resistant dht routing," in *Computer Security – ESORICS 2005*, S. d. C. di Vimercati, P. Syverson, and D. Gollmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 305–318.

[1753] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The secondgeneration onion router," in *Usenix Security*, 2004, pp. 303–320.

[1754] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *2005 IEEE Symposium on Security and Privacy (S P'05)*, 2005.

[1755] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 332–346.

[1756] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Proceedings of the 11th European Conference on Research in Computer Security*, ser. ESORICS'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 18–33.

[1757] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of tls 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1773–1788.

[1758] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962, Jun. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6962.txt

[1759] N. Weaver, R. Sommer, and V. Paxson, "Detecting forged tcp reset packets," in *in Proceedings of the Network and Distributed System Security Symposium, NDSS 2009*. The Internet Society, 09 2009.

[1760] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[1761] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704, Mar. 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3704.txt

[1762] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org," in *2013 IEEE Conference on Communications and Network Security (CNS)*, 2013, pp. 224–232.

[1763] "Openvpn," https://openvpn.net/, accessed: 2021-03-23.

[1764] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6071.txt

[1765] C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 7296, Oct. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7296.txt

[1766] J. Turner, M. J. Schertler, M. S. Schneider, and D. Maughan, "Internet Security Association and Key Management Protocol (ISAKMP)," RFC 2408, Nov. 1998. [Online]. Available: https://rfc-editor.org/rfc/rfc2408.txt

[1767] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 605–620.

[1768] C. Grundemann, "Ipv6 security myth – no ipv6 nat means less security," https://www.internetsociety.org/blog/2015/01/ipv6-security-myth-3-no-ipv6-nat-means-less-security/, accessed: 2021-03-23.

[1769] "Border Gateway Protocol (BGP)," RFC 1163, Jun. 1990. [Online]. Available: https://rfc-editor.org/rfc/rfc1163.txt

[1770] "Border Gateway Protocol 3 (BGP-3)," RFC 1267, Oct. 1991. [Online]. Available: https://rfc-editor.org/rfc/rfc1267.txt

[1771] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of bgp security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[1772] M. Lad, R. Oliveira, B. Zhang, and L. Zhang, "Understanding resiliency of internet topology against prefix hijack attacks," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 368–377.

[1773] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," RFC 6480, Feb. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6480.txt

[1774] M. Lepinski and K. Sriram, "BGPsec Protocol Specification," RFC 8205, Sep. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8205.txt

[1775] C. Hall, R. Anderson, R. Clayton, E. Ouzounis, and P. Trimintzios, "Resilience of the internet interconnection ecosystem," in *Economics of Information Security and Privacy III*. Springer, 2013, pp. 119–148.

[1776] Q. Li, Y.-C. Hu, and X. Zhang, "Even rockets cannot make pigs fly sustainably: Can bgp be secured with bgpsec?" in *Proceedings of the NDSS Workshop on Security of Emerging Network Technologies (SENT'14)*, Feb. 2014.

[1777] X. Zhang, H. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: scalability, control, and isolation on next-generation networks," in *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011, pp. 212–227. [Online]. Available: https://doi.org/10.1109/SP.2011.45

[1778] A. Azimov, E. Bogomazov, R. Bush, K. Patel, and J. Snijders, "Verification of AS_PATH Using the Resource Certificate Public Key Infrastructure and Autonomous System Provider Authorization," Internet Engineering Task Force, Internet-Draft draft-ietf-sidrops-aspa-verification-07, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-aspa-verification-07

[1779] D. Simon, R. Hurst, and D. B. D. Aboba, "The EAP-TLS Authentication Protocol," RFC 5216, Mar. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5216.txt

[1780] "arpwatch - keep track of ethernet/ip address pairings," https://linux.die.net/man/8/arpwatch, accessed: 2021-02-25.

[1781] T. Kiravuo, M. Sarela, and J. Manner, "A survey of Ethernet LAN security," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1477–1491, Third 2013.

[1782] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348, Aug. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7348.txt

[1783] A. Stubblefield, J. Ioannidis, and A. D. Rubin, "Using the Fluhrer, Mantin, and Shamir attack to break WEP," in *NDSS*, 2002.

[1784] S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," in *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, ser. Lecture Notes in Computer Science, S. Vaudenay and A. M. Youssef, Eds., vol. 2259. Springer, 2001, pp. 1–24.

[1785] E. Tews and M. Beck, "Practical attacks against WEP and WPA," in *Proceedings of the Second ACM Conference on Wireless Network Security*, ser. WiSec '09. New York, NY, USA: ACM, 2009, pp. 79–86.

[1786] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[1787] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell, "A modular correctness proof of IEEE 802.11i and TLS," in *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, V. Atluri, C. A. Meadows, and A. Juels, Eds. ACM, 2005, pp. 2–15.

[1788] J. Jonsson, "On the security of CTR + CBC-MAC," in *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, ser. Lecture Notes in Computer Science, K. Nyberg and H. M. Heys, Eds., vol. 2595. Springer, 2002, pp. 76–93.

[1789] D. Harkins and W. A. Kumari, "Opportunistic Wireless Encryption," RFC 8110, Mar. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8110.txt

[1790] "Internet society — dnssec deployment maps," https://www.autosar.org/standards/, accessed: 2021-03-20.

[1791] S. Nürnberger and C. Rossow, "–vatiCAN–vetted, authenticated CAN bus," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 106–124.

[1792] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "Vulcan: Efficient component authentication and software isolation for automotive control networks," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 225–237.

[1793] R. Bhatia, V. Kumar, K. Serag, Z. B. Celik, M. Payer, and D. Xu, "Evading voltage-based intrusion detection on automotive can," in *Proceedings of the 2021 Network and Distributed System Security (NDSS) Symposium*, February 2021.

[1794] "Modbussecure," https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf, accessed: 2021-05-20.

[1795] M. G. Gouda and A. X. Liu, "Structured firewall design," *Computer networks*, vol. 51, no. 4, pp. 1106–1120, 2007.

[1796] "Firewall builder," https://github.com/fwbuilder/fwbuilder, accessed: 2021-03-23.

[1797] "Capirca," https://github.com/google/capirca, accessed: 2021-03-23.

[1798] V. Paxson, "Bro: a System for Detecting Network Intruders in Real-Time," *Computer*

Networks, vol. 31, no. 23-24, pp. 2435–2463, 1999. [Online]. Available: http://www.icir.org/vern/papers/bro-CN99.pdf

[1799] "Suricata," https://suricata-ids.org, accessed: 2021-03-23.

[1800] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov, "Botzilla: detecting the "phoning home" of malicious software," in *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C. Hung, Eds. ACM, 2010, pp. 1978–1984.

[1801] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, P. C. van Oorschot, Ed. USENIX Association, 2008, pp. 139–154.

[1802] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008*. The Internet Society, 2008.

[1803] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 63–68.

[1804] G. Choudhary, V. Sharma, I. You, K. Yim, R. Chen, and J.-H. Cho, "Intrusion detection systems for networked unmanned aerial vehicles: a survey," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2018, pp. 560–565.

[1805] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[1806] A. M. Lonea, D. E. Popescu, and H. Tianfield, "Detecting ddos attacks in cloud computing environment," *International Journal of Computers Communications & Control*, vol. 8, no. 1, pp. 70–78, 2012.

[1807] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 275–283.

[1808] A. Orebaugh, S. Biles, and J. Babbin, *Snort cookbook - solutions and examples for Snort administrators*. O'Reilly, 2005. [Online]. Available: http://www.oreilly.de/catalog/snortckbk/index.html

[1809] K. Cox and C. Gerg, *Managing security with Snort and IDS tools - intrusion detection with open source tools*. O'Reilly, 2004. [Online]. Available: http://www.oreilly.de/catalog/snortids/index.html

[1810] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.

[1811] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Oct. 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3954.txt

[1812] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011, Sep. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc7011.txt

[1813] Z. Jadidi, V. Muthukkumarasamy, E. Sithirasenan, and M. Sheikhan, "Flow-based anomaly detection using neural network optimized with gsa algorithm," in *2013 IEEE 33rd international conference on distributed computing systems workshops*. IEEE, 2013, pp. 76–81.

[1814] "Networkminer," https://www.netresec.com/?page=networkminer, accessed: 2021-03-23.

[1815] "Xplico," https://www.xplico.org, accessed: 2021-03-23.

[1816] "Nmap: the network mapper," https://nmap.org, accessed: 2021-03-23.

[1817] D. Moore, C. Shannon, G. Voelker, S. Savage *et al.*, "Network telescopes: Technical report," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., 2004.

[1818] G. Yao, J. Bi, and A. V. Vasilakos, "Passive ip traceback: Disclosing the locations of ip spoofers from path backscatter," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 3, pp. 471–484, 2014.

[1819] C. Seifert, I. Welch, P. Komisarczuk *et al.*, "Honeyc-the low-interaction client honeypot," in *NZCSRCS*, vol. 6. Citeseer, 2007.

[1820] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in network security: a survey," in *Proceedings of the 2011 international conference on communication, computing & security*, 2011, pp. 600–605.

[1821] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the Rise of IoT Compromises," in *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, August 2015.

[1822] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow, "AmpPot: Monitoring and Defending Amplification DDoS Attacks," in *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions and Defenses*, November 2015.

[1823] S. Taha Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Transactions on Reliability*, vol. 64, pp. 1–12, 09 2015.

[1824] A. Shaghaghi, M. A. Kaafar, and S. Jha, "Wedgetail: An intrusion prevention system for the data plane of software defined networks," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 849–861.

[1825] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, 2015, pp. 77–81.

[1826] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015.

[1827] S. Liu, M. K. Reiter, and V. Sekar, "Flow reconnaissance via timing attacks on SDN switches," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 196–206.

[1828] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2160–2173, 2016.

[1829] S. Zerkane, D. Espes, P. Le Parc, and F. Cuppens, "Vulnerability analysis of software defined networking," in *Foundations and Practice of Security*, F. Cuppens, L. Wang, N. Cuppens-Boulahia, N. Tawbi, and J. Garcia-Alfaro, Eds. Cham: Springer International Publishing, 2017, pp. 97–116.

[1830] Z. G. A. Gember, P. Prabhu and A. Akella, "Toward software defined middlebox networking," in *In Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets)*, Redmond, WA, 10 2012, pp. 7–12.

[1831] S. Lal, T. Taleb, and A. Dutta, "NFV: Security threats and best practices," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 211–217, Aug 2017.

[1832] W. Yang and C. Fung, "A survey on security in network functions virtualization," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 15–19.

[1833] T. C. Group, "TNC Architecture for Interoperability," Trusted Computing Group, Specification, Oct. 2017. [Online]. Available: https://trustedcomputinggroup.org/resource/tnc-architecture-for-interoperability-specification/

[1834] F. Schwarz and C. Rossow, "SENG, the SGX-Enforcing Network Gateway: Authorizing Communication from Shielded Clients," in *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA: USENIX Association, Aug. 2020.

[1835] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 743–754.

[1836] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[1837] "Zero trust principles," https://www.ncsc.gov.uk/blog-post/zero-trust-principles-beta-release, accessed: 2021-05-21.

[1838] V. M. Escobedo, F. Zyzniewski, B. A. E. Beyer, and M. Saltonstall, "Beyondcorp: The user experience," *Login*, vol. tbd, p. tbd, 2017.

[1839] A. Studer and A. Perrig, "The coremelt attack," in *European Symposium on Research in Computer Security*. Springer, 2009, pp. 37–52.

[1840] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 127–141.

[1841] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 38–54.

[1842] W. D. Ivancic, "Security analysis of dtn architecture and bundle protocol specification for space-based networks," in *2010 IEEE Aerospace Conference*. IEEE, 2010, pp. 1–12.

[1843] S. H. Sellke, C.-C. Wang, S. Bagchi, and N. Shroff, "Tcp/ip timing channels: Theory to implementation," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 2204–2212.

[1844] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver, "Practical comprehensive bounds on surreptitious communication over {DNS}," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 17–32.

[1845] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert channels in ipv6," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2005, pp. 147–166.

[1846] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–26, 2015.

[1847] F. Massacci and N. C. Ngo, "Distributed financial exchanges: Security challenges and design principles," *IEEE Security & Privacy*, 2020.

[1848] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams, "Futuresmex: secure, distributed futures market exchange," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 335–353.

[1849] H. Kaislin, *Top-Down Digital VLSI Design: From Architectures to Gate-Level Circuits and FPGAs*. Morgan Kaufmann, 2015.

[1850] D. Grawrock, *Dynamics of a Trusted Platform: A building block approach*. Intel Press, 2008.

[1851] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," *CoRR*, vol. abs/1811.05441, 2018. [Online]. Available: http://arxiv.org/abs/1811.05441

[1852] National Institute of Standards and Technology, "FIPS 140-2 security requirements for cryptographic modules," may 25, 2001 (Change Notice 2, 12/3/2002). [Online]. Available: https://csrc.nist.gov/publications/detail/fips/140/2/final

[1853] "NIST glossary." [Online]. Available: https://csrc.nist.gov/glossary/term/tampering

[1854] "Common Criteria certified products." [Online]. Available: https://www.commoncriteriaportal.org/products/

[1855] V. Lomne, "Common criteria certification of a smartcard: a technical overview," CHES 2016 Tutorial 1. [Online]. Available: https://iacr.org/workshops/ches/ches2016/presentations/CHES16-Tutorial1.pdf

[1856] W. Slegers, *Security Evaluation Scheme for IoT Platforms, version 1.2*, TrustCB, 2019. [Online]. Available: https://www.trustcb.com/iot/sesip/

[1857] Trusted Computing Group. [Online]. Available: https://trustedcomputinggroup.org

[1858] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design - The*

Hardware / Software Interface (5th Edition), ser. The Morgan Kaufmann Series in Computer Architecture and Design. Academic Press, 2014.

[1859] P. Maene, J. Goetzfried, R. D. Clercq, T. Mueller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 361–374, 2017.

[1860] A. P. Martin, "The ten page introduction to trusted computing," University of Oxford, Tech. Rep. CS-RR-08-11, 2008.

[1861] Global Platform Device Committee, "EE protection profile," version 1.2, Public Release, November 2014, Document Reference: GPD_SPE_021. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/140/2/final

[1862] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://eprint.iacr.org/2016/086.

[1863] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 4, pp. 315–328, 2008.

[1864] J. Noorman, J. V. Bulck, J. T. Muehlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Goetzfried, T. Mueller, and F. Freiling, "Sancus 2.0: A low-cost security architecture for IoT devices," *ACM Transactions on Privacy and Security*, vol. 20, no. 3, p. 33, 2017.

[1865] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "SMART: secure and minimal architecture for (establishing dynamic) root of trust," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[1866] R. Avanzi, "The qarma block cipher family – almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," Cryptology ePrint Archive, Report 2016/444, 2016, https://eprint.iacr.org/2016/444.

[1867] NIST Lightweight Cryptography. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[1868] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. [Online]. Available: https://competitions.cr.yp.to/caesar.html

[1869] Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration bound analysis and throughput optimum architecture of SHA-256 (384, 512) for hardware implementations," in *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, 2007, pp. 102–114. [Online]. Available: https://doi.org/10.1007/978-3-540-77535-5_8

[1870] Y. K. Lee, L. Batina, K. Sakiyama, and I. Verbauwhede, "Elliptic curve based security processor for RFID," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1514–1527, 2008.

[1871] F. Turan and I. Verbauwhede, "Compact and flexible FPGA implementation of Ed25519 and X25519," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 3, p. 21, 2019.

[1872] NIST Post Quantum Cryptography. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[1873] Homomorphic Encryption Standardization. [Online]. Available: http://homomorphicencryption.org/introduction/

[1874] Link to NATO Tempest website. [Online]. Available: https://www.ia.nato.int/niapc/tempest

[1875] D. Bernstein, "Cache-timing attacks on AES," 2005. [Online]. Available: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[1876] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology – CT-RSA 2006*. Springer Berlin Heidelberg, 2006, pp. 1–20.

[1877] S. Chari, J. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*. Springer Berlin Heidelberg, 2003, pp. 13–28.

[1878] B. W. Lampson, "A note on the confinement problem," *Communications ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[1879] D. Page, "MASCAB: a Micro-Architectural Side-Channel Attack Bibliography." [Online]. Available: http://www.github.com/danpage/mascab

[1880] D. Karaklajic, J.-M. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12, pp. 2295 – 2306, 2013.

[1881] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of eliminating errors in cryptographic computations," *J. Cryptology*, vol. 14, no. 2, pp. 101–119, 2001.

[1882] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Design, Automation and Test in Europe Conference, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, 2017, pp. 1116–1121.

[1883] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2002, pp. 2–12.

[1884] H. Lohrke, S. Tajik, T. Krachenfels, C. Boit, and J. Seifert, "Key extraction using thermal laser stimulation: Case study on Xilinx ultrascale FPGAs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 573–595, 2018.

[1885] J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: Attacks, countermeasures and cost," in *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, ser. Lecture Notes in Computer Science, D. Naccache, Ed., vol. 6805. Louvain-la-Neuve,Belgium: Springer-Verlag, 2012, pp. 265–282.

[1886] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *Information and Communications Security, 8th International Conference, ICICS 2006*, ser. Lecture Notes in Computer Science, N. Li, P. Ning, and S. Qing, Eds., vol. 4307. Raleigh, NC, USA: Springer-Verlag, 2006, pp. 529–545.

[1887] D. Bernstein, “Implementing "practical leakage-resilient symmetric cryptography",” 2012, presented at CHES 2012 rumpsession. [Online]. Available: https://www.cosic.esat.kuleuven.be/ches2012/ches_rump/rs9.pdf

[1888] S. Belaïd, V. Grosso, and F. Standaert, “Masking and leakage-resilient primitives: One, the other(s) or both?” *Cryptography and Communications*, vol. 7, no. 1, pp. 163–184, 2015.

[1889] M. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, “Recommendation for the entropy sources used for random bit generation,” National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90B, 2018. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf

[1890] E. B. Barker and J. M. Kelsey, “Recommendation for random bit generator (RGB) constructions,” National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90C, Second Draft, 2016. [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf

[1891] “Functionality classes and evaluation methodology for deterministic random number generators, version 3,” 2013, (In German: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren). [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_pdf.pdf

[1892] “Functionality classes and evaluation methodology for physical random number generators, version 3,” 2013, (In German: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren). [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_pdf.pdf

[1893] W. Killmann and W. Schindler, “A proposal for functionality classes for random number generators,” 2011, aIS 20 / AIS 31, version 2.0.

[1894] D. Johnston, *Random Number Generators - Principles and practices: A guide for engineers and programmers*. De Gruyter, 2018.

[1895] J. Balasch, F. Bernard, V. Fischer, M. Grujic, M. Laban, O. Petura, V. Rozic, G. V. Battum, I. Verbauwhede, M. Wakker, and B. Yang, “Design and testing methodologies for true random number generators towards industry certification,” in *International IEEE European Test Symposium - ETS 2018*, ser. IEEE Computer Society, Bremen,DE, 2018, p. 10.

[1896] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2013.

[1897] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, “FPGA intrinsic PUFs and their use for IP protection,” in *Cryptographic Hardware and Embedded Systems - CHES 2007*. Springer Berlin Heidelberg, 2007, pp. 63–80.

[1898] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, 2008.

[1899] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, “Silicon physical random functions,”

in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. ACM, 2002, pp. 148–160.

[1900] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A survey on lightweight entity authentication with strong PUFs," *ACM Computing Surveys*, vol. 48, no. 2, p. 42, 2015.

[1901] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[1902] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[1903] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. ACM, 2013, pp. 709–720.

[1904] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1601–1618.

[1905] T. J. Williams, "The Purdue enterprise reference architecture," *Computers in industry*, vol. 24, no. 2, pp. 141–158, 1994.

[1906] E. A. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.

[1907] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, and Y. Halgand, "A survey of approaches combining safety and security for industrial control systems," *Reliability engineering & system safety*, vol. 139, pp. 156–178, 2015.

[1908] A. A. Cárdenas, S. Amin, and S. Sastry, "Research challenges for the security of control systems." in *Proceedings of the 3rd Conference on Hot Topics in Security*. USENIX Association, 2008, pp. 1–6.

[1909] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops*. IEEE, 2008, pp. 495–500.

[1910] F. Mueller, "Challenges for cyber-physical systems: Security, timing analysis and soft error protection," in *High-Confidence Software Platforms for Cyber-Physical Systems (HCSP-CPS) Workshop, Alexandria, Virginia*, 2006, p. 4.

[1911] M. Sun, S. Mohan, L. Sha, and C. Gunter, "Addressing safety and security contradictions in cyber-physical systems," in *Proceedings of the 1st Workshop on Future Directions in Cyber-Physical Systems Security (CPSSW'09)*, 2009.

[1912] E. A. Lee, "Cyber-physical systems-are computing foundations adequate," in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, vol. 2. Citeseer, 2006.

[1913] M. Anand, E. Cronin, M. Sherr, M. Blaze, Z. Ives, and I. Lee, "Security challenges in next generation cyber physical systems," in *Proceedings of Beyond SCADA: Networked Embedded Control for Cyber Physical Systems*. Academic Press, 2006, pp. 1–4.

[1914] H. Tang and B. M. McMillin, "Security property violation in CPS through timing," in *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*. IEEE, 2008, pp. 519–524.

[1915] C. Neuman, "Challenges in security for cyber-physical systems," in *DHS Workshop on Future Directions in Cyber-Physical Systems Security*. CPS-VO, 2009, pp. 22–24.

[1916] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry, "Challenges for securing cyber physical systems," in *Workshop on future directions in cyber-physical systems security*, 2009, p. 5.

[1917] P. Oman, E. Schweitzer, and D. Frincke, "Concerns about intrusions into remotely accessible substation controllers and scada systems," in *Proceedings of the Twenty-Seventh Annual Western Protective Relay Conference*, vol. 160, 2000.

[1918] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[1919] J. A. Stankovic and R. Rajkumar, "Real-time operating systems," *Real-Time Systems*, vol. 28, no. 2-3, pp. 237–253, 2004.

[1920] M. Felser, "Real-time ethernet-industry prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118–1129, 2005.

[1921] C. Alcaraz and S. Zeadally, "Critical control system protection in the 21st century," *Computer*, vol. 46, no. 10, pp. 74–83, 2013.

[1922] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *IEEE real-time and embedded technology and applications symposium*. IEEE, 2008, pp. 377–386.

[1923] V. C. Gungor, G. P. Hancke *et al.*, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches." *IEEE Trans. Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[1924] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[1925] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of Supercomputing*, vol. 68, no. 1, pp. 1–48, 2014.

[1926] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[1927] K. Ogata, *Discrete-time control systems*. Prentice Hall Englewood Cliffs, NJ, 1995, vol. 2.

[1928] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[1929] R. Goebel, R. G. Sanfelice, and A. R. Teel, "Hybrid dynamical systems," *IEEE Control Systems*, vol. 29, no. 2, pp. 28–93, 2009.

[1930] A. E. Summers, "Introduction to layers of protection analysis," *Journal of Hazardous Materials*, vol. 104, no. 1-3, pp. 163–168, 2003.

[1931] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE transactions on control systems technology*, vol. 18, pp. 636–653, 2009.

[1932] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice hall Upper Saddle River, NJ, 1998, vol. 104.

[1933] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proceedings of the conference on Computer and communications security (CCS)*. ACM, 2009, pp. 21–32.

[1934] A. A. Cardenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *Proceedings of the ACM symposium on information, computer and communications security*, 2011, pp. 355–366.

[1935] B. Krebs. (2008, June) Cyber incident blamed for nuclear power plant shutdown. http://www.washingtonpost.com/wp-dyn/content/article/2008/06/05/AR2008060501958.html.

[1936] Lloyds, "Business blackout: The insurance implications of a cyber attack on the us power grid," Lloyd's, Tech. Rep., 2015. [Online]. Available: http://www.lloyds.com/news-and-insight/risk-insight/library/society-and-security/business-blackout

[1937] A. Greenberg, "Hackers remotely kill a jeep on the highway?with me in it," *Wired*, vol. 7, p. 21, 2015.

[1938] K. C. Zeng, S. Liu, Y. Shu, D. Wang, H. Li, Y. Dou, G. Wang, and Y. Yang, "All your GPS are belong to us: Towards stealthy manipulation of road navigation systems," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1527–1544.

[1939] J. Valente and A. A. Cardenas, "Understanding security threats in consumer drones through the lens of the discovery quadcopter family," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*. ACM, 2017, pp. 31–36.

[1940] Y.-L. Huang, A. A. Cárdenas, S. Amin, Z.-S. Lin, H.-Y. Tsai, and S. Sastry, "Understanding the physical and economic consequences of attacks on control systems," *International Journal of Critical Infrastructure Protection*, vol. 2, no. 3, pp. 73–83, 2009.

[1941] S. Amin, A. A. Cárdenas, and S. S. Sastry, "Safe and secure networked control systems under denial-of-service attacks," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2009, pp. 31–45.

[1942] M. Krotofil, A. Cardenas, J. Larsen, and D. Gollmann, "Vulnerabilities of cyber-physical systems to stale data?determining the optimal time to launch attacks," *International journal of critical infrastructure protection*, vol. 7, no. 4, pp. 213–232, 2014.

[1943] S. McLaughlin, "CPS: Stateful policy enforcement for control system device usage," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. New York, NY, USA: ACM, 2013, pp. 109–118.

[1944] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks

in control systems," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, Oct 2012, pp. 1806–1813.

[1945] S. Amin, X. Litrico, S. Sastry, and A. M. Bayen, "Cyber security of water SCADA systems—Part I: analysis and experimentation of stealthy deception attacks," *Control Systems Technology, IEEE Transactions on*, vol. 21, no. 5, pp. 1963–1970, 2013.

[1946] A. Jakaria, W. Yang, B. Rashidi, C. Fung, and M. A. Rahman, "VFence: A defense against distributed denial of service attacks using network function virtualization," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 2. IEEE, 2016, pp. 431–436.

[1947] K. Zetter. (2016, mar) Inside the cunning, unprecedented hack of ukraine's power grid. WIRED magazine. [Online]. Available: http://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/

[1948] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *IEEE Symposium on Security and Privacy*, 2008.

[1949] K. Fu and W. Xu, "Risks of trusting the physics of sensors," *Communications of the ACM*, vol. 61, no. 2, pp. 20–23, 2018.

[1950] I. Giechaskiel and K. B. Rasmussen, "SoK: Taxonomy and challenges of out-of-band signal injection attacks and defenses," *CoRR*, vol. abs/1901.06935, 2019. [Online]. Available: http://arxiv.org/abs/1901.06935

[1951] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.

[1952] J. Selvaraj, G. Y. Dayanıklı, N. P. Gaunkar, D. Ware, R. M. Gerdes, M. Mina *et al.*, "Electromagnetic induction attacks against embedded systems," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 499–510.

[1953] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 103–117.

[1954] R. J. Turk, "Cyber incidents involving control systems," Idao National Laboratory, Tech. Rep. INL/EXT-05-00671, October 2005.

[1955] R. Esposito. (2006, October) Hackers penetrate water system computers. [Online]. Available: https://www.computerworld.com/article/2547938/hackers-break-into-water-system-network.html

[1956] K. Hemsley and R. Fisher, "A history of cyber incidents and threats involving industrial control systems," in *International Conference on Critical Infrastructure Protection*. Springer, 2018, pp. 215–242.

[1957] T. Reed, *At the Abyss: An Insider's History of the Cold War*. Presidio Press, March 2004.

[1958] M. Abrams and J. Weiss, "Malicious control system cyber security attack case study–maroochy water services, australia," The MITRE Corporation, Tech. Rep., 2008.

[1959] N. Sayfayn and S. Madnick, "Cybersafety analysis of the Maroochy shire sewage spill," Cybersecurity Interdisciplinary Systems Laboratory (CISL), Sloan School of Management, Massachusetts Institute of Technology, Working Paper CISL#2017-09, 2017.

[1960] AP, "Revenge hacker: 34 months, must repay Georgia-Pacific $1m," February 2017. [Online]. Available: https://www.usnews.com/news/louisiana/articles/2017-02-16/revenge-hacker-34-months-must-repay-georgia-pacific-1m

[1961] D. Bilefsky. (2017, January) Hackers use new tactic at Austrian hotel: Locking the doors. The New York Times.

[1962] B. Krebs, "FBI: smart meter hacks likely to spread," April 2012. [Online]. Available: http://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/

[1963] M. Dalli, *Enemalta employees suspended over 1,000 tampered smart meters*. https://www.maltatoday.com.mt/news/national/35650/enemalta-employees-suspended-over-1-000-tampered-smart-meters-20140211: Malta Today, February 2014.

[1964] R. M. Lee, M. J. Assante, and T. Conway, "German steel mill cyber attack," *Industrial Control Systems*, vol. 30, p. 62, 2014.

[1965] "United States Attorney, Eastern District of California. Willows man arrested for hacking into Tehama Colusa Canal Authority computer system," November 2007. [Online]. Available: https://www.computerworld.com/article/2540235/insider-charged-with-hacking-california-canal-system.html

[1966] "United States Attorney, Eastern District of California. Sacramento man pleads guilty to attempting ot shut down california's power grid," November 2007. [Online]. Available: http://www.usdoj.gov/usao/cae/press_releases/docs/2007/12-14-07DenisonPlea.pdf

[1967] D. Kravets. (2009, March) Feds: Hacker disabled offshore oil platform leak-detection system. http://www.wired.com/threatlevel/2009/03/feds-hacker-dis/.

[1968] J. Leyden. (2008, 11th Jan) Polish teen derails tram after hacking train network. http://www.theregister.co.uk/2008/01/11/tram_hack/.

[1969] Booz Allen Hamilton, "When the lights went out: Ukraine cybersecurity threat briefing," p. 20, 2016. [Online]. Available: http://www.boozallen.com/content/dam/boozallen/documents/2016/09/ukraine-report-when-the-lights-wentout

[1970] A. Greenberg, "'Crash override': The malware that took down a power grid," *Wired Magazine*, pp. 09–20, 2017.

[1971] A. Cherepanov, "Win32/industroyer, a new threat for industrial control systems," *White Paper. ESET*, 2017.

[1972] M. Giles, "Triton is the world's most murderous malware, and it's spreading," 2019. [Online]. Available: https://www.technologyreview.com/s/613054/cybersecurity-critical-infrastructure-triton-malware/

[1973] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. Megas, E. Nadeau, B. Piccarreta, D. G. O'Rourke, and K. Scarfone, "Considerations for managing internet of things (IoT) cybersecurity and privacy risks," *National Institute of Standards and Technology, NISTIR 8228*, 2018.

[1974] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 76, 2018.

[1975] R. Langner, *Robust Control System Networks: How to Achieve Reliable Control After Stuxnet*. Momentum Press, 2011.

[1976] R. Antrobus, B. Green, S. Frey, and A. Rashid, "The forgotten I in IIoT: A vulnerability scanner for industrial internet of things," in *Living in the Internet of Things 2019*, 4 2019.

[1977] P. Ralston, J. Graham, and J. Hieb, "Cyber security risk assessment for SCADA and DCS networks," *ISA transactions*, vol. 46, no. 4, pp. 583–594, 2007.

[1978] P. Craig, J. Mortensen, and J. Dagle, "Metrics for the National SCADA Test Bed Program," PNNL-18031, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), Tech. Rep., 2008.

[1979] G. Hamoud, R. Chen, and I. Bradley, "Risk assessment of power systems SCADA," in *IEEE Power Engineering Society General Meeting, 2003*, vol. 2, 2003.

[1980] Y. Cherdantseva, P. Burnap, A. Blyth, P. Eden, K. Jones, H. Soulsby, and K. Stoddart, "A review of cyber security risk assessment methods for scada systems," *Computers & security*, vol. 56, pp. 1–27, 2016.

[1981] J. H. Castellanos, M. Ochoa, and J. Zhou, "Finding dependencies between cyber-physical domains for security testing of industrial control systems," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018, pp. 582–594.

[1982] H. Sandberg, A. Teixeira, and K. H. Johansson, "On security indices for state estimators in power networks," in *Preprints of the First Workshop on Secure Control Systems, CPSWEEK 2010, Stockholm, Sweden*, 2010.

[1983] U. Vaidya and M. Fardad, "On optimal sensor placement for mitigation of vulnerabilities to cyber attacks in large-scale networks," in *Proceedings of the 2013 European Control Conference (ECC)*, July 2013, pp. 3548–3553.

[1984] O. Vukovic, K. C. Sou, G. Dan, and H. Sandberg, "Network-aware mitigation of data integrity attacks on power system state estimation," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 6, pp. 1108–1118, July 2012.

[1985] H. Okhravi and F. T. Sheldon, "Data diodes in support of trustworthy cyber infrastructure," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2010, p. 23.

[1986] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, "Key management systems for sensor networks in the context of the internet of things," *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, 2011.

[1987] R. Anderson and S. Fuloria, "Security economics and critical national infrastructure," in *Economics of Information Security and Privacy*. Springer, 2010, pp. 55–66.

[1988] J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, and M. Ochoa, "Legacy-compliant data authentication for industrial control system traffic," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 665–685.

[1989] P. P. Tsang and S. W. Smith, "YASIR: A low-latency high-integrity security retrofit for lecacy SCADA systems," in *23rd International Information Security Conference (IFIC SEC)*, September 2008, pp. 445–459.

[1990] S. Gollakota, H. Hassanieh, B. Ransford, D. Katabi, and K. Fu, "They can hear your heartbeats: Non-invasive security for implantable medical devices," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 2–13.

[1991] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of BLE device users." in *USENIX Security Symposium*, 2016, pp. 1205–1221.

[1992] R. Roman, C. Alcaraz, and J. Lopez, "A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 231–244, 2007.

[1993] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Nistir 8114: Draft report on lightweight cryptography," *Available on the NIST website: http://csrc. nist. gov/publications/drafts/nistir-8114/nistir_8114_draft. pdf*, 2016.

[1994] S. Koteshwara and A. Das, "Comparative study of authenticated encryption targeting lightweight IoT applications," *IEEE Design & Test*, vol. 34, no. 4, pp. 26–33, 2017.

[1995] K.-A. Shim, "A survey of public-key cryptographic primitives in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 577–601, 2016.

[1996] A. Abbasi, J. Wetzels, T. Holz, and S. Etalle, "Challenges in designing exploit mitigations for deeply embedded systems," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy*. IEEE, 2019.

[1997] K. Fisher, J. Launchbury, and R. Richards, "The HACMS program: using formal methods to eliminate exploitable bugs," *Phil. Trans. R. Soc. A*, vol. 375, no. 2104, p. 20150401, 2017.

[1998] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks," in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 2017, pp. 3–18.

[1999] D. F. Kune, J. Backes, S. S. Clark, D. Kramer, M. Reynolds, K. Fu, Y. Kim, and W. Xu, "Ghost talk: Mitigating emi signal injection attacks against analog sensors," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 145–159.

[2000] X. Carpent, K. Eldefrawy, N. Rattanavipanon, A.-R. Sadeghi, and G. Tsudik, "Reconciling remote attestation and safety-critical operation on simple iot devices," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[2001] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: secure and scalable aggregate network attestation," in *Proceedings of the 2016 ACM*

SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 731–742.

[2002] V. P. Illiano, R. V. Steiner, and E. C. Lupu, "Unity is strength!: Combining attestation and measurements inspection to handle malicious data injections in WSNs," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 134–144.

[2003] R. V. Steiner and E. Lupu, "Attestation in wireless sensor networks: A survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, p. 51, 2016.

[2004] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "Hydra: hybrid design for remote attestation (using a formally verified microkernel)," in *Proceedings of the 10th ACM Conference on Security and Privacy in wireless and Mobile Networks*. ACM, 2017, pp. 99–110.

[2005] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for SCADA networks," in *Proceedings of the SCADA Security Scientific Symposium*, Miami Beach, FL, USA, 2007.

[2006] R. Berthier and W. H. Sanders, "Specification-based intrusion detection for advanced metering infrastructures," in *Proceedings of Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2011, pp. 184–193.

[2007] N. Goldenberg and A. Wool, "Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems," *International Journal of Critical Infrastructure Protection*, vol. 6, pp. 63–75, 2013.

[2008] C. Markman, A. Wool, and A. A. Cardenas, "Temporal phase shifts in SCADA networks," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*. ACM, 2018, pp. 84–89.

[2009] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 13–24.

[2010] M. Faisal, A. A. Cardenas, and A. Wool, "Modeling Modbus TCP for intrusion detection," in *Communications and Network Security (CNS), 2016 IEEE Conference on*. IEEE, 2016, pp. 386–390.

[2011] W. Aoudi, M. Iturbe, and M. Almgren, "Truth will out: Departure-based process-level detection of stealthy attacks on control systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 817–831.

[2012] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel, "Through the eye of the PLC: semantic security monitoring for industrial processes," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 126–135.

[2013] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," *IEEE Symposium on Security and Privacy*, 2018.

[2014] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on

industrial control systems," in *Proceedings of the Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 1092–1105.

[2015] K. Paridari, N. O'Mahony, A. E.-D. Mady, R. Chabukswar, M. Boubekeur, and H. Sandberg, "A framework for attack-resilient industrial control systems: Attack detection and controller reconfiguration," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 113–128, 2018.

[2016] Q. Gu, D. Formby, S. Ji, H. Cam, and R. Beyah, "Fingerprinting for cyber-physical system security: Device physics matters too," *IEEE Security & Privacy*, vol. 16, no. 5, pp. 49–59, 2018.

[2017] A. Petropulu, K. I. Diamantaras, Z. Han, D. Niyato, and S. Zonouz, "Contactless monitoring of critical infrastructure [from the guest editors]," *IEEE Signal Processing Magazine*, vol. 36, no. 2, pp. 19–21, 2019.

[2018] T. Shekari, C. Bayens, M. Cohen, L. Graber, and R. Beyah, "RFDIDS: Radio frequency-based distributed intrusion detection system for the power grid." in *NDSS*, 2019.

[2019] W. Jardine, S. Frey, B. Green, and A. Rashid, "Senami: Selective non-invasive active monitoring for ics intrusion detection," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, 2016, pp. 23–34.

[2020] T. Roth and B. McMillin, "Physical attestation of cyber processes in the smart grid," in *International Workshop on Critical Information Infrastructures Security*. Springer, 2013, pp. 96–107.

[2021] J. Valente, C. Barreto, and A. A. Cárdenas, "Cyber-physical systems attestation," in *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 354–357.

[2022] R. B. Bobba, K. M. Rogers, Q. Wang, H. Khurana, K. Nahrstedt, and T. J. Overbye, "Detecting false data injection attacks on DC state estimation," in *Proceedings of Workshop on Secure Control Systems*, 2010.

[2023] J. Valente and A. A. Cárdenas, "Using visual challenges to verify the integrity of security cameras," in *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM, 2015, pp. 141–150.

[2024] Y. Mo, S. Weerakkody, and B. Sinopoli, "Physical authentication of control systems: designing watermarked control inputs to detect counterfeit sensor outputs," *Control Systems, IEEE*, vol. 35, no. 1, pp. 93–109, 2015.

[2025] M. A. Rahman, E. Al-Shaer, and R. B. Bobba, "Moving target defense for hardening the security of the power system state estimation," in *Proceedings of the First ACM Workshop on Moving Target Defense*. ACM, 2014, pp. 59–68.

[2026] J. Rowe, K. N. Levitt, T. Demir, and R. Erbacher, "Artificial diversity as maneuvers in a control theoretic moving target defense," in *National Symposium on Moving Target Research*, 2012.

[2027] J. Giraldo and A. A. Cardenas, "Moving target defense for attack mitigation in multi-vehicle systems," in *Proactive and Dynamic Network Defense*. Springer, 2019, pp. 163–190.

[2028] J. Giraldo, A. Cardenas, and R. G. Sanfelice, "A moving target defense to reveal cyber-attacks in CPS and minimize their impact," in *American Control Conference*, 2019.

[2029] C. G. Rieger, D. I. Gertman, and M. A. McQueen, "Resilient control systems: Next generation design research," in *2009 2nd Conference on Human System Interactions*. IEEE, 2009, pp. 632–636.

[2030] Y. Mo and B. Sinopoli, "Secure estimation in the presence of integrity attacks," *Automatic Control, IEEE Transactions on*, vol. 60, no. 4, pp. 1145–1151, April 2015.

[2031] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1454–1467, 2014.

[2032] D. Wijayasekara, O. Linda, M. Manic, and C. Rieger, "FN-DFE: Fuzzy-neural data fusion engine for enhanced resilient state-awareness of hybrid energy systems," *IEEE transactions on cybernetics*, vol. 44, no. 11, pp. 2065–2075, 2014.

[2033] E. P. Blasch, D. A. Lambert, P. Valin, M. M. Kokar, J. Llinas, S. Das, C. Chong, and E. Shahbazian, "High level information fusion (HLIF): Survey of models, issues, and grand challenges," *IEEE Aerospace and Electronic Systems Magazine*, vol. 27, no. 9, pp. 4–20, 2012.

[2034] E. Blasch, I. Kadar, J. Salerno, M. M. Kokar, S. Das, G. M. Powell, D. D. Corkill, and E. H. Ruspini, "Issues and challenges of knowledge representation and reasoning methods in situation assessment (level 2 fusion)," in *Signal Processing, Sensor Fusion, and Target Recognition XV*, vol. 6235. International Society for Optics and Photonics, 2006, p. 623510.

[2035] A. F. M. Piedrahita, V. Gaur, J. Giraldo, A. A. Cardenas, and S. J. Rueda, "Virtual incident response functions in control systems," *Computer Networks*, vol. 135, pp. 147–159, 2018.

[2036] S. H. Kafash, J. Giraldo, C. Murguia, A. A. Cardenas, and J. Ruths, "Constraining attacker capabilities through actuator saturation," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 986–991.

[2037] M. Arroyo, H. Kobayashi, S. Sethumadhavan, and J. Yang, "FIRED: frequent inertial resets with diversification for emerging commodity cyber-physical systems," *arXiv preprint arXiv:1702.06595*, 2017.

[2038] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press, 2018, pp. 10–21.

[2039] L. F. Combita, J. A. Giraldo, A. A. Cardenas, and N. Quijano, "Dddas for attack detection and isolation of control systems," in *Handbook of Dynamic Data Driven Applications Systems*. Springer, 2018, pp. 407–422.

[2040] A. Farraj, E. Hammad, A. A. Daoud, and D. Kundur, "A game-theoretic control approach to mitigate cyber switching attacks in smart grid systems," in *Proceedings of the IEEE Smart Grid Communications*, Venice, Italy, 2014, pp. 958–963.

[2041] C. Barreto, A. A. Cárdenas, and N. Quijano, "Controllability of dynamical systems: Threat models and reactive security," in *Decision and Game Theory for Security*. Springer, 2013, pp. 45–64.

[2042] P. Hu, H. Li, H. Fu, D. Cansever, and P. Mohapatra, "Dynamic defense strategy against advanced persistent threat with insiders," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 747–755.

[2043] Y. Yuan, F. Sun, and H. Liu, "Resilient control of cyber-physical systems against intelligent attacker: a hierarchal stackelberg game approach," *To appear on International Journal of Systems Science*, 2015.

[2044] A. Barth, B. Rubinstein, M. Sundararajan, J. Mitchell, D. Song, and P. Bartlett, "A learning-based approach to reactive security," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 482–493, July 2012.

[2045] D. Shelar and S. Amin, "Analyzing vulnerability of electricity distribution networks to der disruptions," in *American Control Conference (ACC), 2015*, 2015, pp. 2461–2468.

[2046] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, Jul 2001.

[2047] M. Rostami, A. Juels, and F. Koushanfar, "Heart-to-heart (H2H): authentication for implanted medical devices," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1099–1112.

[2048] K. Stouffer, S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn, "Guide to industrial control systems (ICS) security: Supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and other control system configurations such as programmable logic controllers (plc)," NIST, Tech. Rep. Special Publication 800-82: Revision 2, May 2015.

[2049] T. Koppel, *Lights out: a cyberattack, a nation unprepared, surviving the aftermath*. Broadway Books, 2016.

[2050] NERC-CIP, *Critical Infrastructure Protection*. North American Electric Reliability Corporation, 2008. [Online]. Available: https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx

[2051] F. Sakiz and S. Sen, "A Survey of Attacks and Detection Mechanisms on Intelligent Transportation Systems: VANETs and IoV," *Ad Hoc Networks*, vol. 61, pp. 33–50, 2017.

[2052] B. Nassi, A. Shabtai, R. Masuoka, and Y. Elovici, "Sok-security and privacy in the age of drones: Threats, challenges, solution mechanisms, and scientific gaps," *arXiv preprint arXiv:1903.05155*, 2019.

[2053] L. J. Wells, J. A. Camelio, C. B. Williams, and J. White, "Cyber-physical security challenges in manufacturing systems," *Manufacturing Letters*, vol. 2, no. 2, pp. 74–77, 2014.

[2054] M. Rushanan, A. D. Rubin, D. F. Kune, and C. M. Swanson, "SoK: Security and privacy in implantable medical devices and body area networks," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 524–539.

[2055] C. Alcaraz, G. Fernandez, and F. Carvajal, "Security aspects of SCADA and DCS environments," in *Critical Infrastructure Protection*. Springer, 2012, pp. 120–149.

[2056] M. Iturbe, I. Garitano, U. Zurutuza, and R. Uribeetxeberria, "Towards large-scale, heterogeneous anomaly detection systems in industrial networks: A survey of current trends," *Security and Communication Networks*, vol. 2017, 2017.

[2057] I. Garitano, C. Siaterlis, B. Genge, R. Uribeetxeberria, and U. Zurutuza, "A method to construct network traffic models for process control systems," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE, 2012, pp. 1–8.

[2058] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, "A systematic framework to generate invariants for anomaly detection in industrial control systems." in *NDSS*, 2019.

[2059] C. M. Ahmed, J. Zhou, and A. P. Mathur, "Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in CPS," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018, pp. 566–581.

[2060] J. Giraldo, D. Urbina, A. A. Cardenas, and N. O. Tippenhauer, "Hide and seek: An architecture for improving attack-visibility in industrial control systems," in *International Conference on Applied Cryptography and Network Security*. Springer, 2019, pp. 175–195.

[2061] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the Ukrainian power grid," SANS Industrial Control Systems, Tech. Rep., 03 2016. [Online]. Available: https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf

[2062] R. Langner, "To kill a centrifuge: A technical analysis of what stuxnet's creators tried to achieve," *Arlington, VA: Langner Group*, vol. 7, p. 21, 2013. [Online]. Available: http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf

[2063] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks in control systems," in *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2012, pp. 1806–1813.

[2064] M. Parvania, G. Koutsandria, V. Muthukumary, S. Peisert, C. McParland, and A. Scaglione, "Hybrid control network intrusion detection systems for automated power distribution systems," in *Proceedings of Conference on Dependable Systems and Networks (DSN)*, June 2014, pp. 774–779.

[2065] G. Koutsandria, V. Muthukumar, M. Parvania, S. Peisert, C. McParland, and A. Scaglione, "A hybrid network IDS for protective digital relays in the power transmission grid," in *Proceedings of Conference on Smart Grid Communications (SmartGridComm)*, 2014.

[2066] H. Lin, A. Slagell, Z. Kalbarczyk, P. W. Sauer, and R. K. Iyer, "Semantic security analysis of SCADA networks to detect malicious control commands in power grids," in *Proceedings of the first ACM workshop on Smart energy grid security*. ACM, 2013, pp. 29–34.

[2067] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. N. Fovino, and A. Trombetta, "A multidimensional critical state analysis for detecting intrusions in SCADA systems," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 2, pp. 179–186, 2011.

[2068] A. Le, U. Roedig, and A. Rashid, "Lasarus: Lightweight attack surface reduction for

legacy industrial control systems," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2017, pp. 36–52.

[2069] A. Keliris and M. Maniatakos, "ICSREF: A framework for automated reverse engineering of industrial control systems binaries," *NDSS*, 2018.

[2070] S. McLaughlin, S. Zonouz, D. Pohly, and P. McDaniel, "A trusted safety verifier for process controller code," in *Proceedings of the ISOC Network and Distributed Systems Security Symposium (NDSS)*, 2014.

[2071] M. Krotofil and D. Gollmann, "Industrial control systems security: What is happening?" in *2013 11th IEEE International Conference on Informatics (INDIN)*, July 2013, pp. 670–675.

[2072] W. Knowles, D. Prince, D. Hutchison, J. F. P. Disso, and K. Jones, "A survey of cyber security management in industrial control systems," *International journal of critical infrastructure protection*, vol. 9, pp. 52–80, 2015.

[2073] B. Green, A. Lee, R. Antrobus, U. Roedig, D. Hutchison, and A. Rashid, "Pains, gains and PLCs: ten lessons from building an industrial control systems testbed for security research," in *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*, 2017.

[2074] G. Constable and B. Somerville, Eds., *A Century of Innovation: Twenty Engineering Achievements that Transformed our Lives*. Washington, DC: The National Academies Press, 2003. [Online]. Available: https://www.nap.edu/catalog/10726/a-century-of-innovation-twenty-engineering-achievements-that-transformed-our

[2075] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 13, 2011.

[2076] A. Teixeira, G. Dán, H. Sandberg, and K. H. Johansson, "A cyber security study of a SCADA energy management system: Stealthy deception attacks on the state estimator," in *Proceedings of IFAC World Congress*, vol. 18, no. 1, 2011, pp. 11 271–11 277.

[2077] G. Dán and H. Sandberg, "Stealth attacks and protection schemes for state estimators in power systems," in *First IEEE Smart Grid Commnunications Conference (SmartGridComm)*, October 2010.

[2078] O. Kosut, L. Jia, R. Thomas, and L. Tong, "Malicious Data Attacks on Smart Grid State Estimation: Attack Strategies and Countermeasures," in *First IEEE Smart Grid Commnunications Conference (SmartGridComm)*, October 2010.

[2079] A. Teixeira, S. Amin, H. Sandberg, K. H. Johansson, and S. S. Sastry, "Cyber security analysis of state estimators in electric power systems," in *Proceedings of Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5991–5998.

[2080] A. Giani, E. Bitar, M. Garcia, M. McQueen, P. Khargonekar, and K. Poolla, "Smart grid data integrity attacks: characterizations and countermeasures$^{\pi}$," in *Proceedings of Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2011, pp. 232–237.

[2081] M. A. Rahman, E. Al-Shaer, M. Rahman *et al.*, "A formal model for verifying stealthy

attacks on state estimation in power grids," in *Proceedings of Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2013, pp. 414–419.

[2082] A. A. Cárdenas and R. Safavi-Naini, "Security and privacy in the smart grid," *Handbook on Securing Cyber-Physical Critical Infrastructure.*, pp. 637–654, 2012.

[2083] Government Accountability Office, "Electricity grid modernization. progress being made on cybersecurity guidelines, but key challenges remain to be addressed," January 2011. [Online]. Available: https://www.gao.gov/new.items/d11117.pdf

[2084] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker, "Inferring personal information from demand-response systems," *IEEE Security & Privacy Magazine*, vol. 8, no. 1, pp. 11–20, January/February 2010.

[2085] X. Liao, D. Formby, C. Day, and R. A. Beyah, "Towards secure metering data analysis via distributed differential privacy," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, 2014, pp. 780–785. [Online]. Available: http://dx.doi.org/10.1109/DSN.2014.82

[2086] R. Tan, V. Badrinath Krishna, D. K. Yau, and Z. Kalbarczyk, "Impact of integrity attacks on real-time pricing in smart grids," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 439–450.

[2087] C. Barreto, A. A. Cárdenas, N. Quijano, and E. Mojica-Nava, "CPS: Market analysis of attacks against demand response in the smart grid," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 136–145.

[2088] S. Amini, F. Pasqualetti, and H. Mohsenian-Rad, "Dynamic load altering attacks against power system stability: Attack models and protection schemes," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2862–2872, 2018.

[2089] J. Giraldo, A. Cárdenas, and N. Quijano, "Integrity attacks on real-time pricing in smart grids: impact and countermeasures," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2249–2257, 2016.

[2090] A.-H. Mohsenian-Rad and A. Leon-Garcia, "Distributed internet-based load altering attacks against smart power grids," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 667–674, 2011.

[2091] B. Chen, N. Pattanaik, A. Goulart, K. L. Butler-Purry, and D. Kundur, "Implementing attacks for Modbus/TCP protocol in a real-time cyber physical system test bed," in *Communications Quality and Reliability (CQR), 2015 IEEE International Workshop Technical Committee on*. IEEE, 2015, pp. 1–6.

[2092] A. Laszka, A. Dubey, M. Walker, and D. Schmidt, "Providing privacy, safety, and security in IoT-based transactive energy systems using distributed ledgers," in *Proceedings of the Seventh International Conference on the Internet of Things*. ACM, 2017, p. 13.

[2093] S. Soltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 15–32.

[2094] B. Huang, A. A. Cardenas, and R. Baldick, "Not everything is dark and gloomy: Power grid protections against IoT demand attacks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

[2095] R. Herring, A. Hofleitner, S. Amin, T. Nasr, A. Khalek, and A. Bayen, "Using mobile phones to forecast arterial traffic through statistical learning," in *Proc. of the 89th Annual Meeting of the Transportation Research Board (TRB)*, 2010, pp. 1–22.

[2096] Texas Transportation Institute, "Annual Urban Mobility Report." 2010, http://mobility.tamu.edu/ums.

[2097] E. De Cristofaro and C. Soriente, "Participatory privacy: Enabling privacy in participatory sensing," *IEEE Network*, vol. 27, no. 1, pp. 32–36, 2013.

[2098] C.-L. Huang, Y. Fallah, R. Sengupta, and H. Krishnan, "Intervehicle transmission rate control for cooperative active safety system," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 3, pp. 645 –658, sept. 2011.

[2099] B. Ghena, W. Beyer, A. Hillaker, J. Pevarnek, and J. A. Halderman, "Green lights forever: analyzing the security of traffic infrastructure," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[2100] (2014, October 28) Sensys networks traffic sensor vulnerabilities (Update A). https://ics-cert.us-cert.gov/advisories/ICSA-14-247-01A.

[2101] (2014, March 31) Israeli students spoof waze app with fake traffic jam. http://www.popsci.com/article/gadgets/israeli-students-spoof-waze-app-fake-traffic-jam.

[2102] M. T. Garip, M. E. Gursoy, P. Reiher, and M. Gerla, "Congestion attacks to autonomous cars using vehicular botnets," in *NDSS Workshop on Security of Emerging Networking Technologies (SENT), San Diego, CA*, 2015.

[2103] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao, "Defending against sybil devices in crowdsourced mapping services," in *Accepted in the 14th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*, 2016.

[2104] (2016, June 5) Traffic-weary homeowners and waze are at war, again. guess who?s winning? https://www.washingtonpost.com/local/traffic-weary-homeowners-and-waze-are-at-war-again-guess-whos-winning/2016/06/05/c466df46-299d-11e6-b989-4e5479715b54_story.html.

[2105] K. Sampigethaya, R. Poovendran, S. Shetty, T. Davis, and C. Royalty, "Future e-enabled aircraft communications and security: The next 20 years and beyond," *Proceedings of the IEEE*, vol. 99, no. 11, pp. 2040–2055, 2011.

[2106] A. Costin and A. Francillon, "Ghost in the air (traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices," *Black Hat USA*, pp. 1–12, 2012.

[2107] E. J. Weyuker, "Testing component-based software: A cautionary tale," *IEEE software*, vol. 15, no. 5, pp. 54–59, 1998.

[2108] D. Jenkins and B. Vasigh, *The economic impact of unmanned aircraft systems integration in the United States*. Association for Unmanned Vehicle Systems International (AUVSI), 2013.

[2109] (2017) Gartner predicts 3 million drones to be shipped in 2017. [Online]. Available: https://dronelife.com/2017/02/10/gartner-predicts-3-million-drones-shipped-2017/

[2110] R. Altawy and A. M. Youssef, "Security, privacy, and safety aspects of civilian drones: A survey," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 2, p. 7, 2016.

[2111] R. J. Rosen, "So this is how it begins: Guy refuses to stop drone-spying on seattle woman," 2013. [Online]. Available: https://www.theatlantic.com/technology/archive/2013/05/so-this-is-how-it-begins-guy-refuses-to-stop-drone-spying-on-seattle-woman/275769/

[2112] B. Schneier, "Is it OK to shoot down a drone over your backyard?" 2015. [Online]. Available: https://www.cnn.com/2015/09/09/opinions/schneier-shoot-down-drones/

[2113] O. B. Waxman, "World's most embarrassing dad has drone follow daughter to school," 2015. [Online]. Available: https://time.com/3831431/dad-daughter-drone-school/

[2114] D. Davidson, H. Wu, R. Jellinek, T. Ristenpart, and V. Singh, "Controlling UAVs with sensor input spoofing attacks," in *Proceedings of the 10th USENIX Conference on Offensive Technologies*, ser. WOOT'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 221–231.

[2115] M. Diulio, C. Savage, B. Finley, and E. Schneider, "Taking the integrated condition assessment system to the year 2010," in *13th Int. Ship Control Systems Symposium, Orlando, FL*, 2003.

[2116] M. Diulio, R. Halpin, M. Monaco, H. Chin, T. Hekman, and F. Dugie, "Advancements in equipment remote monitoring programs–providing optimal fleet support in a cyber-safe environment," *Naval Engineers Journal*, vol. 127, no. 3, pp. 109–118, 2015.

[2117] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[2118] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1109–1123.

[2119] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, "Cloaking the clock: emulating clock skew in controller area networks," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press, 2018, pp. 32–42.

[2120] S. Dadras, R. M. Gerdes, and R. Sharma, "Vehicular platooning in an adversarial environment," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, 2015, pp. 167–178.

[2121] K. Poulsen. (2010, March) Hacker disables more than 100 cars remotely. WIRED. [Online]. Available: https://www.wired.com/2010/03/hacker-bricks-cars/

[2122] Y. Pan, J. White, D. C. Schmidt, A. Elhabashy, L. Sturm, J. Camelio, and C. Williams, "Taxonomies for reasoning about cyber-physical attacks in IoT-based manufacturing systems," *International Journal of Interactive Multimedia and Artificial Inteligence*, vol. 4, no. Special Issue on Advances and Applications in the Internet of Things and Cloud Computing, 2017.

[2123] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "An experimental security analysis of an industrial robot controller," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 268–286.

[2124] P. W. Singer, *Wired for war: The robotics revolution and conflict in the 21st century*. Penguin, 2009.

[2125] N. G. Leveson and C. S. Turner, "An investigation of the Therac-25 accidents," *IEEE computer*, vol. 26, no. 7, pp. 18–41, 1993.

[2126] C. Camara, P. Peris-Lopez, and J. E. Tapiador, "Security and privacy issues in implantable medical devices: A comprehensive survey," *Journal of biomedical informatics*, vol. 55, pp. 272–289, June 2015.

[2127] E. Marin, D. Singelée, B. Yang, V. Volski, G. A. Vandenbosch, B. Nuttin, and B. Preneel, "Securing wireless neurostimulators," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. ACM, 2018, pp. 287–298.

[2128] M.-Z. Poh, D. J. McDuff, and R. W. Picard, "Advancements in noncontact, multiparameter physiological measurements using a webcam," *IEEE transactions on biomedical engineering*, vol. 58, no. 1, pp. 7–11, 2010.

[2129] D. Hambling, "The Pentagon has a laser that can identify people from a distance by their heartbeat," *MIT Technology Review*, 2019.

[2130] C. S. Kruse, B. Frederick, T. Jacobson, and D. K. Monticone, "Cybersecurity in healthcare: A systematic review of modern threats and trends," *Technology and Health Care*, vol. 25, no. 1, pp. 1–10, 2017.

[2131] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[2132] L. Mathews, "Criminals Hacked A Fish Tank To Steal Data From A Casino," 2017. [Online]. Available: https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino/#5dba8c4632b9

[2133] K. Albrecht and L. Mcintyre, "Privacy nightmare: When baby monitors go bad [opinion]," *IEEE Technology and Society Magazine*, vol. 34, no. 3, pp. 14–19, 2015.

[2134] D. Goldman, "Shodan: The scariest search engine," Apr. 2013. [Online]. Available: https://money.cnn.com/2013/04/08/technology/security/shodan/

[2135] J. Valente, M. A. Wynn, and A. A. Cardenas, "Stealing, spying, and abusing: Consequences of attacks on internet of things devices," *IEEE Security Privacy*, vol. 17, no. 5, pp. 10–21, Sep. 2019.

[2136] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems," in *Dangerous Skills: Understanding and Mitigating Security Risks of Voice-Controlled Third-Party Functions on Virtual Personal Assistant Systems*. IEEE, 2019, p. 0.

[2137] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home IoT devices with an in-hub security manager," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2017, pp. 551–556.

[2138] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1074–1088.

[2139] J. Wilson, R. S. Wahby, H. Corrigan-Gibbs, D. Boneh, P. Levis, and K. Winstein, "Trust but verify: Auditing the secure internet of things," in *Proceedings of the 15th Annual*

International Conference on Mobile Systems, Applications, and Services. ACM, 2017, pp. 464–474.

[2140] B. Schneier, *Click Here to Kill Everybody: Security and Survival in a Hyper-connected World*. WW. Norton & Company, September 2018.

[2141] M. Schmitt, "International law in cyberspace: The Koh Speech and the Tallinn manual juxtaposed," *Harvard International Law Journal Online*, vol. 13, 2012.

[2142] Energy Sector Control Systems Working Group, "Roadmap to achieve energy delivery systems cybersecurity," U.S. Department of Energy, Tech. Rep., 2011. [Online]. Available: https://www.energy.gov/ceser/downloads/roadmap-achieve-energy-delivery-systems-cybersecurity-2011

[2143] B. Schneier, "The internet of things will upend our industry," *IEEE Security and Privacy*, vol. 15, no. 2, pp. 108–108, 2017.

[2144] K. Fu. (2016) Infrastructure disruption: Internet of things security. U.S. House of Representatives. [Online]. Available: http://docs.house.gov/meetings/IF/IF17/20161116/105418/HHRG-114-IF17-Wstate-FuK-20161116.pdf

[2145] M. Y. Vardi, "Cyber insecurity and cyber libertarianism," *Communications of the ACM*, vol. 60, no. 5, pp. 5–5, 2017.

[2146] M. Schallbruch, "The european network and information security directive–a cornerstone of the digital single market," in *Digital Marketplaces Unleashed*. Springer, 2018, pp. 287–295.

[2147] "Security assessment principles for the civil nuclear industry," 2017.

[2148] M. Daniel. (2013) Incentives to support adoption of the cybersecurity framework. https://obamawhitehouse.archives.gov/blog/2013/08/06/incentives-support-adoption-cybersecurity-framework.

[2149] Department of Homeland Security Integrated Task Force, "Executive order 13636: Improving critical infrastructure cybersecurity," https://www.dhs.gov/sites/default/files/publications/dhs-eo13636-analytic-report-cybersecurity-incentives-study.pdf, Department of Homeland Security, Tech. Rep., 2013.

[2150] (2014) Protection of critical infrastructure. European Commission. [Online]. Available: https://ec.europa.eu/home-affairs/what-we-do/policies/crisis-and-terrorism/critical-infrastructure_en

[2151] T. Augustinos, L. Bauer, A. Cappelletti, J. Chaudhery, I. Goddijn, L. Heslault, N. Kalfigkopoulos, V. Katos, N. Kitching, M. Krotofil *et al.*, "Cyber insurance: recent advances, good practices & challenges," *European Union Agency For Network and Information Security (ENISA)*, 2016.

[2152] I. Ehrlich and G. S. Becker, "Market insurance, self-insurance, and self-protection," *Journal of political Economy*, vol. 80, no. 4, pp. 623–648, 1972.

[2153] P. J. Denning and D. E. Denning, "Discussing cyber attack," *Communications of the ACM*, vol. 53, no. 9, pp. 29–31, 2010.

[2154] A. K. Cebrowski and J. J. Garstka, "Network-centric warfare: Its origin and future," in *US Naval Institute Proceedings*, vol. 124, no. 1, 1998, pp. 28–35.

[2155] M. Robinson, K. Jones, and H. Janicke, "Cyber warfare: Issues and challenges," *Computers & security*, vol. 49, pp. 70–94, 2015.

[2156] A. Greenberg, "The untold story of NotPetya, the most devastating cyberattack in history, august 2018," *From the book Sandworm published on Security wired website.*, 2019. [Online]. Available: https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/

[2157] R. Piggin, "Development of industrial cyber security standards: IEC 62443 for SCADA and industrial control system security," in *IET Conference on Control and Automation 2013: Uniting Problems and Solutions*. IET, 2013, pp. 1–6.

[2158] NIST, US, "Guidelines for smart grid cyber security (vol. 1 to 3)," *NIST IR-7628, Aug*, 2010.

[2159] B. Obama, "Executive order 13636: Improving critical infrastructure cybersecurity," *Federal Register*, vol. 78, no. 33, p. 11739, 2013.

[2160] L. Tanczer, I. Brass, M. Elsden, M. Carr, and J. J. Blackstock, "The United Kingdom's emerging internet of things (IoT) policy landscape," *Tanczer, LM, Brass, I., Elsden, M., Carr, M., & Blackstock, J.(2019). The United Kingdom's Emerging Internet of Things (IoT) Policy Landscape. In R. Ellis & V. Mohan (Eds.), Rewired: Cybersecurity Governance*, pp. 37–56, 2019.

[2161] E. Lear, R. Droms, and D. Romascanu, "Manufacturer usage description specification," IETF Network Working Group, Tech. Rep. draft-ietf-opsawg-mud-11, 2018.

[2162] H. Tschofenig and E. Baccelli, "Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security," *IEEE Security Privacy*, vol. 17, no. 5, pp. 47–57, Sep. 2019.

[2163] S. K. Das, K. Kant, and N. Zhang, *Handbook on securing cyber-physical critical infrastructure*. Elsevier, 2012.

[2164] R. Liu and W. Trappe, *Securing Wireless Communications at the Physical Layer*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[2165] C. Ye, S. Mathur, A. Reznik, Y. Shah, W. Trappe, and N. B. Mandayam, "Information-theoretically secret key generation for fading wireless channels," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 2, pp. 240–254, 2010.

[2166] S. Eberz, M. Strohmeier, M. Wilhelm, and I. Martinovic, "A practical man-in-the-middle attack on signal-based key generation protocols," in *Computer Security – ESORICS 2012*, S. Foresti, M. Yung, and F. Martinelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 235–252.

[2167] N. Anand, S.-J. Lee, and E. W. Knightly, "Strobe: Actively securing wireless communications using zero-forcing beamforming," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 720–728.

[2168] S. Čapkun, M. Čagalj, R. Rengaswamy, I. Tsigkogiannis, J.-P. Hubaux, and M. Srivastava, "Integrity codes: Message integrity protection and authentication over insecure channels," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 208–223, Oct 2008.

[2169] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[2170] A. D. Wyner, "The wire-tap channel," *Bell system technical journal*, vol. 54, no. 8, pp. 1355–1387, 1975.

[2171] I. Csiszár and J. Korner, "Broadcast channels with confidential messages," *IEEE transactions on information theory*, vol. 24, no. 3, pp. 339–348, 1978.

[2172] M. Bloch, J. Barros, M. R. Rodrigues, and S. W. McLaughlin, "Wireless information-theoretic security," *IEEE Transactions on Information Theory*, vol. 54, no. 6, pp. 2515–2534, 2008.

[2173] D. Adamy, *EW 101: a first course in electronic warfare*. Artech House, 2001.

[2174] C. Popper, "On secure wireless communication under adversarial interference," PhD thesis, ETH Zurich, 2011.

[2175] C. Pöpper, N. O. Tippenhauer, B. Danev, and S. Čapkun, "Investigation of signal and message manipulations on the wireless channel," in *Proceedings of the European Symposium on Research in Computer Security*, 2011.

[2176] H. Yang, S. Bae, M. Son, H. Kim, S. M. Kim, and Y. Kim, "Hiding in plain signal: Physical signal overshadowing attack on LTE," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 55–72. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/yang-hojoon

[2177] B. Danev, D. Zanetti, and S. Capkun, "On physical-layer identification of wireless devices," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 6:1–6:29, Dec. 2012.

[2178] G. Avoine, M. A. Bingöl, I. Boureanu, S. čapkun, G. Hancke, S. Kardaş, C. H. Kim, C. Lauradoux, B. Martin, J. Munilla, A. Peinado, K. B. Rasmussen, D. Singelée, A. Tchamkerten, R. Trujillo-Rasua, and S. Vaudenay, "Security of distance-bounding: A survey," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 94:1–94:33, Sep. 2018.

[2179] T. Beth and Y. Desmedt, "Identification tokens—or: Solving the chess grandmaster problem," in *Conference on the Theory and Application of Cryptography*. Springer, 1990, pp. 169–176.

[2180] S. Brands and D. Chaum, "Distance-bounding protocols," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1993, pp. 344–359.

[2181] J. Clulow, G. P. Hancke, M. G. Kuhn, and T. Moore, "So near and yet so far: Distance-bounding attacks in wireless networks," in *Security and Privacy in Ad-Hoc and Sensor Networks*, L. Buttyán, V. D. Gligor, and D. Westhoff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 83–97.

[2182] A. Ranganathan and S. Capkun, "Are we really close? Verifying proximity in wireless systems," *IEEE Security & Privacy*, vol. 15, no. 3, pp. 52–58, 2017.

[2183] M. Singh, P. Leu, and S. Capkun, "UWB with pulse reordering: Securing ranging against relay and physical layer attacks." *IACR Cryptology ePrint Archive*, vol. 2017, p. 1240, 2017.

[2184] S. Capkun and J.-P. Hubaux, "Secure positioning in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 221–232, Feb 2006.

[2185] P. Leu, M. Singh, M. Roeschlin, K. G. Paterson, and S. Capkun, "Message time of arrival codes: A fundamental primitive for secure distance measurement," *IEEE Symposium on Security and Privacy*, 2020.

[2186] G. P. Hancke and M. G. Kuhn, "An RFID distance bounding protocol," in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. IEEE, 2005, pp. 67–73.

[2187] M. Poturalski, M. Flury, P. Papadimitratos, J.-P. Hubaux, and J.-Y. Le Boudec, "Distance bounding with IEEE 802.15. 4a: Attacks and countermeasures," *IEEE Transactions on Wireless Communications*, vol. 10, no. 4, pp. 1334–1344, 2011.

[2188] M. G. Kuhn and C. M. G. Kuhn, "Compromising emanations: Eavesdropping risks of computer displays," 2003.

[2189] M. G. Kuhn, "Electromagnetic eavesdropping risks of flat-panel displays," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2004, pp. 88–107.

[2190] M. Backes, T. Chen, M. Duermuth, H. P. A. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *2009 30th IEEE Symposium on Security and Privacy*, May 2009, pp. 315–327.

[2191] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 444–461.

[2192] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 551–562.

[2193] Y. Zhang and K. Rasmussen, "Detection of electromagnetic interference attacks on sensor systems," in *IEEE Symposium on Security and Privacy (S&P)*, May 2020.

[2194] W. van Eck, "Electromagnetic radiation from video display units: An eavesdropping risk?" *Computers & Security*, vol. 4, no. 4, pp. 269 – 286, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016740488590046X

[2195] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi, and M. Srivastava, "Pycra: Physical challenge-response authentication for active sensors under spoofing attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1004–1015.

[2196] L. Francis, G. P. Hancke, K. Mayes, and K. Markantonakis, "Practical relay attack on contactless transactions by using NFC mobile phones." *IACR Cryptology ePrint Archive*, vol. 2011, p. 618, 2011.

[2197] M. Strohmeier, "Security in next generation air traffic communication networks," Ph.D. dissertation, University of Oxford, 2016.

[2198] D. Forsberg, G. Horn, W.-D. Moeller, and V. Niemi, *LTE Security*, 2nd ed. Wiley Publishing, 2012.

[2199] A. Ranganathan, "Physical-layer techniques for secure proximity verification and localization," PhD thesis, ETH Zurich, 2016.

[2200] S. N. Premnath, S. Jana, J. Croft, P. L. Gowda, M. Clark, S. K. Kasera, N. Patwari, and S. V. Krishnamurthy, "Secret key extraction from wireless signal strength in real environments," *IEEE Transactions on mobile Computing*, vol. 12, no. 5, pp. 917–930, 2012.

[2201] S. Mathur, R. Miller, A. Varshavsky, W. Trappe, and N. Mandayam, "Proximate: proximity-based secure pairing using ambient wireless signals," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 211–224.

[2202] J. Zhang, T. Q. Duong, A. Marshall, and R. Woods, "Key generation from wireless channels: A review," *Ieee access*, vol. 4, pp. 614–626, 2016.

[2203] J. Zhang, A. Marshall, R. Woods, and T. Q. Duong, "Efficient key generation by exploiting randomness from channel responses of individual OFDM subcarriers," *IEEE Transactions on Communications*, vol. 64, no. 6, pp. 2578–2588, 2016.

[2204] B. Azimi-Sadjadi, A. Kiayias, A. Mercado, and B. Yener, "Robust key generation from signal envelopes in wireless networks," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 401–410.

[2205] M. Strasser, C. Popper, S. Capkun, and M. Cagalj, "Jamming-resistant key establishment using uncoordinated frequency hopping," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 64–78.

[2206] D. W. K. Ng, E. S. Lo, and R. Schober, "Robust beamforming for secure communication in systems with wireless information and power transfer," *IEEE Transactions on Wireless Communications*, vol. 13, no. 8, pp. 4599–4615, 2014.

[2207] Y. Zheng, M. Schulz, W. Lou, Y. T. Hou, and M. Hollick, "Profiling the strength of physical-layer security: A study in orthogonal blinding," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2016, pp. 21–30.

[2208] M. Schulz, A. Loch, and M. Hollick, "Practical known-plaintext attacks against physical layer security in wireless mimo systems." in *The Network and Distributed System Security Symposium (NDSS)*, 2014.

[2209] P. Robyns, P. Quax, and W. Lamotte, "PHY-layer security is no alternative to cryptography," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 160–162.

[2210] H. Mahdavifar and A. Vardy, "Achieving the secrecy capacity of wiretap channels using polar codes," *arXiv preprint arXiv:1007.3568*, 2010.

[2211] P. Parada and R. Blahut, "Secrecy capacity of SIMO and slow fading channels," in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.* IEEE, 2005, pp. 2152–2155.

[2212] A. Mukherjee, S. A. A. Fakoorian, J. Huang, and A. L. Swindlehurst, "Principles of physical layer security in multiuser wireless networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1550–1573, 2014.

[2213] P. K. Gopala, L. Lai, and H. El Gamal, "On the secrecy capacity of fading channels," in *2007 IEEE International Symposium on Information Theory*. IEEE, 2007, pp. 1306–1310.

[2214] W. Shen, P. Ning, X. He, and H. Dai, "Ally friendly jamming: How to jam your enemy and maintain your own wireless connectivity at the same time," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 174–188.

[2215] D. S. Berger, F. Gringoli, N. Facchi, I. Martinovic, and J. Schmitt, "Gaining insight on friendly jamming in a real-world IEEE 802.11 network," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 105–116.

[2216] J. P. Vilela, M. Bloch, J. Barros, and S. W. McLaughlin, "Wireless secrecy regions with friendly jamming," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 256–266, 2011.

[2217] N. O. Tippenhauer, L. Malisa, A. Ranganathan, and S. Capkun, "On limitations of friendly jamming for confidentiality," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 160–173.

[2218] S. Capkun, M. Cagalj, G. Karame, and N. O. Tippenhauer, "Integrity regions: Authentication through presence in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 11, pp. 1608–1621, 2010.

[2219] A. Polydoros and K. Woo, "LPI detection of frequency-hopping signals using autocorrelation techniques," *IEEE journal on selected areas in communications*, vol. 3, no. 5, pp. 714–726, 1985.

[2220] R. F. Mills and G. E. Prescott, "Waveform design and analysis of frequency hopping LPI networks," in *Proceedings of MILCOM'95*, vol. 2. IEEE, 1995, pp. 778–782.

[2221] L. Frikha, Z. Trabelsi, and W. El-Hajj, "Implementation of a covert channel in the 802.11 header," in *2008 International Wireless Communications and Mobile Computing Conference*. IEEE, 2008, pp. 594–599.

[2222] C. Popper, M. Strasser, and S. Capkun, "Anti-jamming broadcast communication using uncoordinated spread spectrum techniques," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 5, pp. 703–715, June 2010.

[2223] W. Xu, W. Trappe, and Y. Zhang, "Anti-jamming timing channels for wireless networks," in *Proceedings of the First ACM Conference on Wireless Network Security*, ser. WiSec '08. New York, NY, USA: ACM, 2008, pp. 203–213.

[2224] M. Strasser, C. Pöpper, and S. Čapkun, "Efficient uncoordinated FHSS anti-jamming communication," in *Proceedings of the Tenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '09. New York, NY, USA: ACM, 2009, pp. 207–218.

[2225] K. Grover, A. Lim, and Q. Yang, "Jamming and anti-jamming techniques in wireless networks: a survey," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 17, no. 4, pp. 197–215, 2014.

[2226] W. Wang, Z. Sun, S. Piao, B. Zhu, and K. Ren, "Wireless physical-layer identification: Modeling and validation," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2091–2106, 2016.

[2227] T. J. Bihl, K. W. Bauer, and M. A. Temple, "Feature selection for RF fingerprinting with multiple discriminant analysis and using zigbee device emissions," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1862–1874, 2016.

[2228] T. D. Vo-Huu, T. D. Vo-Huu, and G. Noubir, "Fingerprinting Wi-Fi devices using software defined radios," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2016, pp. 3–14.

[2229] S. Capkun, K. El Defrawy, and G. Tsudik, "Group distance bounding protocols," in *International Conference on Trust and Trustworthy Computing*. Springer, 2011, pp. 302–312.

[2230] N. O. Tippenhauer and S. Čapkun, "Id-based secure distance bounding and localization," in *European Symposium on Research in Computer Security*. Springer, 2009, pp. 621–636.

[2231] M. Kuhn, H. Luecken, and N. O. Tippenhauer, "UWB impulse radio based distance bounding," in *2010 7th Workshop on Positioning, Navigation and Communication*. IEEE, 2010, pp. 28–37.

[2232] L. Bussard and W. Bagga, "Distance-bounding proof of knowledge to avoid real-time attacks," in *IFIP International Information Security Conference*. Springer, 2005, pp. 223–238.

[2233] D. Singelée and B. Preneel, "Distance bounding in noisy environments," in *European Workshop on Security in Ad-hoc and Sensor Networks*. Springer, 2007, pp. 101–115.

[2234] K. B. Rasmussen and S. Capkun, "Realization of RF distance bounding." in *USENIX Security Symposium*, 2010, pp. 389–402.

[2235] A. Ranganathan, N. O. Tippenhauer, B. Škorić, D. Singelée, and S. Čapkun, "Design and implementation of a terrorist fraud resilient distance bounding system," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 415–432.

[2236] N. O. Tippenhauer, H. Luecken, M. Kuhn, and S. Capkun, "UWB rapid-bit-exchange system for distance bounding," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2015, p. 2.

[2237] S. Drimer, S. J. Murdoch *et al.*, "Keep your enemies close: Distance bounding against smartcard relay attacks." in *USENIX security symposium*, vol. 312, 2007.

[2238] C. Cremers, K. B. Rasmussen, B. Schmidt, and S. Capkun, "Distance hijacking attacks on distance bounding protocols," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 113–127.

[2239] G. P. Hancke and M. G. Kuhn, "Attacks on time-of-flight distance bounding channels," in *Proceedings of the first ACM conference on Wireless network security*. ACM, 2008, pp. 194–202.

[2240] K. B. Rasmussen and S. Čapkun, "Location privacy of distance bounding protocols," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 149–160.

[2241] M. Flury, M. Poturalski, P. Papadimitratos, J.-P. Hubaux, and J.-Y. Le Boudec, "Effectiveness of distance-decreasing attacks against impulse radio ranging," in *Proceedings of the third ACM conference on Wireless network security*. ACM, 2010, pp. 117–128.

[2242] R. Shokri, M. Poturalski, G. Ravot, P. Papadimitratos, and J.-P. Hubaux, "A practical

secure neighbor verification protocol for wireless sensor networks," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 193–200.

[2243] S. Čapkun and J.-P. Hubaux, "Secure positioning of wireless devices with application to sensor networks," in *IEEE infocom*, no. CONF, 2005.

[2244] J. T. Chiang, J. J. Haas, and Y.-C. Hu, "Secure and precise location verification using distance bounding and simultaneous multilateration," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 181–192.

[2245] N. Basilico, N. Gatti, M. Monga, and S. Sicari, "Security games for node localization through verifiable multilateration," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 1, pp. 72–85, 2013.

[2246] L. Lazos, R. Poovendran, and S. Čapkun, "Rope: robust position estimation in wireless sensor networks," in *Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press, 2005, p. 43.

[2247] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers." in *USENIX Security symposium*, 2010, pp. 307–322.

[2248] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 170–183.

[2249] M. Backes, M. Dürmuth, and D. Unruh, "Compromising reflections-or-how to read lcd monitors around the corner," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 158–169.

[2250] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 527–536.

[2251] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1273–1285.

[2252] C. Kasmi and J. L. Esteves, "IEMI threats for information security: Remote command injection on modern smartphones," *IEEE Transactions on Electromagnetic Compatibility*, vol. 57, no. 6, pp. 1752–1755, 2015.

[2253] Y. Park, Y. Son, H. Shin, D. Kim, and Y. Kim, "This ain't your dose: Sensor spoofing attack on medical infusion pump," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[2254] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun, "Proximity-based access control for implantable medical devices," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 410–419.

[2255] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger, "NFC devices: Security and privacy," in *2008 Third International Conference on Availability, Reliability and Security*. IEEE, 2008, pp. 642–647.

[2256] S. Burkard, "Near field communication in smartphones," *Dep. of Telecommunication Systems, Service-centric Networking, Berlin Institute of Technology, Germany*, 2012.

[2257] N. Alexiou, S. Basagiannis, and S. Petridou, "Security analysis of NFC relay attacks using probabilistic model checking," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2014, pp. 524–529.

[2258] M. Schäfer, V. Lenders, and I. Martinovic, "Experimental analysis of attacks on next generation air traffic communication," in *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 253–271.

[2259] M. Smith, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "Economy class crypto: Exploring weak cipher usage in avionic communications via ACARS," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 285–301.

[2260] M. Strohmeier, V. Lenders, and I. Martinovic, "Intrusion detection for airborne communication using phy-layer information," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 67–77.

[2261] A. Shaik, R. Borgaonkar, N. Asokan, V. Niemi, and J.-P. Seifert, "Practical attacks against privacy and availability in 4G/LTE mobile communication systems," *arXiv preprint arXiv:1510.07563*, 2015.

[2262] A. N. Bikos and N. Sklavos, "LTE/SAE security issues on 4G wireless networks," *IEEE Security & Privacy*, vol. 11, no. 2, pp. 55–62, 2012.

[2263] J.-G. Remy and C. Letamendia, *LTE standards*. Wiley Online Library, 2014.

[2264] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful GPS spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 75–86.

[2265] A. Ranganathan, H. Ólafsdóttir, and S. Capkun, "SPREE: A spoofing resistant GPS receiver," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 348–360.

[2266] C. Bonebrake and L. R. O'Neil, "Attacks on GPS time reliability," *IEEE Security & Privacy*, vol. 12, no. 3, pp. 82–84, 2014.

[2267] T. Nighswander, B. Ledvina, J. Diamond, R. Brumley, and D. Brumley, "GPS software attacks," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 450–461.

[2268] J. V. Carroll, "Vulnerability assessment of the US transportation infrastructure that relies on the global positioning system," *The Journal of Navigation*, vol. 56, no. 2, pp. 185–193, 2003.

[2269] Techopedia. [Online]. Available: https://www.techopedia.com/dictionary

[2270] Cyber-physical systems. [Online]. Available: https://www.nsf.gov/pubs/2019/nsf19553/nsf19553.htm

[2271] V. R. Segovia and A. Theorin, "History of control history of PLC and DCS," *University of Lund*, 2012.

[2272] Industrial Internet Consortium, "The Industrial Internet Vocabulary Technical Report V 2.1 ," IIconsortium, Tech. Rep., May 2018.

[2273] W. Wahlster, "From industry 1.0 to industry 4.0: towards the 4th industrial revolution (forum business meets research)," *3rd European Summit on Future Internet Towards Future Internet International Collaborations Espoo, Finland*, vol. 31, 2012.

[2274] K. Stouffer, T. Zimmerman, C. Tang, J. Lubell, J. Cichonski, and J. McCarthy, "Cybersecurity framework manufacturing profile," National Institute of Standards and Technology, Tech. Rep. NISTIR 8183, 2017.

[2275] A. Siemens, F. O. L. TID, A. S. Segura, V. Toubiana, J. W. Walewski, and S. Haller, "Internet-of-things architecture IoT-a project deliverable D1.2—initial architectural reference model for IoT," EU Commission Seventh Framework Programme, Tech. Rep., 2011.

[2276] J. A. Simpson and E. S. C. Weiner, Eds., *Oxford English Dictionary*. Oxford University Press, 1989.

Acronyms

2FA Two Factor Authentication.

2PC Two-Phase Commit.

3GPP Third Generation Partnership Plan.

ABAC Attribute-Based Access Control.

ABCs Attribute-Based Credentials.

ABE Attribute-Based Encryption.

ACID Atomic, Consistent, Isolated and Durable.

ACL Access Control List.

ACL2 A Computational Logic for Applicative Common Lisp.

ACN Anonymous Communication Network.

AD Associated Data.

ADC Analogue-to-Digital Converter.

ADI Application Data Integrity.

ADS-B Automatic Dependent Surveillance-Broadcast.

ADS-B Automatic Dependent Surveillance-Broadcast.

AE Authenticated Encryption.

AEAD Authenticated Encryption with Associated Data.

AES Advanced Encryption Standard.

AG Application Gateway.

AGC Automatic Gain Control.

AH Authentication Header.

AI Artificial Intelligence.

AJAX Asynchronous JavaScript And XML.

AKA Authentication and Key Agreement.

AKE Authenticated Key Exchange.

ALU Arithmetic Logic Unit.

AM Approximate Matching.

AMQP Advanced Message Queuing Protocol.

AOL America Online.

AP Access Point.

API Application Programing Interface.

app application.

APT Advanced Persistent Threat.

ARP Address Resolution Protocol.

AS Autonomous System.

AS Authorisation Server.

ASC Application Security Controls.

ASIC Application Specific Integrated Circuit.

ASLR Address Space Layout Randomization.

ASN Autonmous System.

ASPA Autonomous System Provider Authorization.

AST Abstract Syntax Tree.

ASVS Application Security Verfication Standard.

ATA Advanced Technology Attachment.

ATC Air Traffic Control.

ATM Automated Teller Machine..

ATT&CK Adversarial Tactics, Techniques & Common Knowledge.

AuC Authentication Centre.

AuthS Authentication Server.

AUTOSAR AUTomotive Open System ARchitecture.

AV AntiVirus.

AVA Actual Vulnerability Assessment.

AWS Amazon Web Services.

BAN Burrows-Abadi-Needham.

BFT Byzantine Fault Tolerance.

BGP Border Gateway Protocol.

BIOS Basic Input/Output System.

BLE Bluetooth Low Energy.

BLP Bell-LaPadula.

BPH Bullet Proof Hosting.

BSIMM Building Security In Maturity Model.

BYOD Bring Your Own Device.

C&C Command and Control.

CA Certification Authority.

CaaS Cryptography-as-a-Service.

CADF Cloud Auditing Data Federation.

CAM Content Addressable Memory.

CAN Controller Area Network.

CAP Consistency, Availability and Partition.

CAPEC Common Attack Pattern Enumeration and Classification.

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart.

CB Certification/Validation Body.

CBAC Code-Based Access Control.

CBC Cipher Block Chaining.

CC Common Criteria.

CC Common Criteria.

CCA Chosen Ciphertext Attack.

CCMP Cipher Block Chaining Message Authentication Code Protocol.

CCS Calculus of Communicating Systems.

ccTLD Country Code Top-Level Domain.

CCTV Closed Circuit Television.

CDA Cyber Defence Alliance.

CDHP Computational Diffie-Hellman Problem.

CDS Cross-Domain Solutions.

CEE Common Event Expression.

CEF Common Event Format.

CEO Chief Executive Officer.

CERT Computer Emergency Response Team.

CET Control-Flow Enforcement Technology.

CFB Cipher Feedback.

CFG Control Flow Graph.

CFI Control-Flow Integrity.

CFRG Crypto Forum Research Group.

CFTT Computer Forensic Tool Testing.

CG Circuit-level Gateway.

CGA Cryptographically Generated Address.

CHAP Challenge Handshake Authentication Protocol.

CHERI Capability Hardware Enhanced RISC Instructions.

CI/CD Continuous Integration/Continuous Delivery.

CIA Central Intelligence Agency.

CIA Confidentiality, Integrity and Availability.

CIM Common Information Model.

CIR Channel Impulse Response.

CISO Chief Information Security Officer.

CISP Cyber Information Sharing Partnership.

CLASP Comprehensive, Lightweight Application Security Process.

CLF Common Log Format.

CMA Chosen Message Attack.

CMOS Complementary Metal-Oxide-Semiconductor.

CNI Critical National Infrastructure.

CORS Cross-Origin Resource Sharing.

COTS Common Off The Shelf.

CP-ABE Ciphertext-Policy Attribute-Based Encryption.

CPA Chosen Plaintext Attack.

CPS Cyber-Physical System.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

CRL Certificate Revocation List.

CRS Common Reference String.

CRT Cathode Ray Tube.

CRT Chinese Remainder Theorem.

CRT Chinese Remainder Theorem.

CRTM Core Root of Trust for Measurements.

CSA Cloud Security Alliance.

CSD Circuit-Switched Data Service.

CSIRT Computer Security Incident Response Team.

CSL Continuous Stochastic Logic.

CSP Cloud Service Provider.

CSP Content Security Policy.

CSP Communicating Sequential Processes.

CSR certificate signing request.

CSRF Cross Site Request Forgery.

CSS Chirp-Spread Spectrum.

CSS Cascading Style Sheets.

CT Certificate Transparency.

CTA Cognitive Task Analysis.

CTF Capture The Flag.

CTI Cyber-Threat Intelligence.

CTL Computational Tree Logic.

CTR Counter Mode.

CVC Cooperating Validity Checker.

CVE Common Vulnerabilities and Exposures.

CVP Closest Vector Problem.

CVSS Common Vulnerability Scoring System.

CWE Common Weakness Enumeration.

CyberSA Cyber-Situational Awareness.

CyBOK Cyber Security Body of Knowledge.

DAA Direct Anonymous Attestation.

DAC Discretionary Access Control.

DARPA Defence Advanced Research Projects Agency.

DAST Dynamic Analysis Security Testing.

DBI Dynamic Binary Instrumentation.

DCS Distributed Control System.

DDH Decision Diffie–Hellman.

DDoS Distributed Denial of Service.

DEK Data Encryption Key.

DEM Data Encryption Mechanism.

DEP Data Execution Prevention.

DES Data Encryption Standard.

DFI Data-Flow Integrity.

DFRWS Digital Forensics Research Workshop.

DFS Depth-First Search.

DGA Domain-name Generation Algorithm.

DH Diffie-Hellman.

DHIES Diffie-Hellman Integrated Encryption Scheme.

DHKE Diffie-Hellman Key Exchange.

DHP Diffie–Hellman Problem.

DHT Distributed Hash Table.

DICE Device Identifier Composition Engine.

DLP Discrete Logarithm Problem.

DLR Device Level Ring.

DMA Direct Memory Access.

DMARC Domain-based Message Authentication Reporting and Conformance.

DMTF Distributed Management Task Force.

DMZ Demilitarised Zone.

DNS Domain Name System.

DNSSEC DNS Security Extensions.

DoE Department of Energy.

DoH DNS over HTTPS.

DOM Domain Object Model.

DOM Document Object Model.

DoS Denial of Service.

DoT DNS over TLS.

DPA Differential and Higher Order Power Analysis.

DPI Deep Packet Inspection.

DRAM Dynamic Random Access Memory.

DRBG Deterministic Random Bit Generator.

DRM Digital Rights Management.

DSA Digital Signature Algorithm.

DSKS Duplicate Signature Key Selection.

DSSS Direct-Sequence Spread Spectrum.

DTLS Datagram TLS.

DVI Digital Serial Interface.

DWT Discrete Wavelet Transform.

E2E End-to-End.

E2EE End-to-end encryption.

EAL Evaluation Assurance Level.

EAP Extensible Authentication Protocol.

EAP-PEAP EAP Protected Authentication Protocol.

EAP-SIM EAP for GSM Subscriber Identity.

EAP-TLS EAP-Transport Layer Security.

EAPoL Extensible Authentication Protocol over LAN.

ECB Electronic Code Book.

ECC Elliptic Curve Cryptography.

ECIES Elliptic Curve Integrated Encryption Scheme.

ECLF Extended Common Log Format.

ECU Electrical Control Unit.

EDA Electronic Design Automation.

EEA European Economic Area.

EIGRP Enhanced Interior Gateway Routing Protocol.

ELK ElasticSearch-Kibana-Logstash.

EMI Electromagnetic Interference.

ENISA European Union Agency for Cybersecurity.

ESP Encapsulation Security Payload.

ETSI European Telecommunications Standards Institute.

FCF Foundational Cryptography Framework.

FCN Field Communication Network.

FCS Frame Check Sequence.

FDH Full Domain Hash.

FDIR Fault Detection, Isolation, and Reconfiguration.

FDR Failures-Divergences Refinement.

FFT Fast Fourier Transform.

FHE Fully Homomorphic Encryption.

FHSS Frequency Hopping Spread Spectrum.

FIB Focused Ion Beam.

FIDO Fast Identity Online.

FIDO2 Fast IDentity Online 2.

FIRST Forum of Incident Response and Security Teams.

FLASK Flux Advanced Security Kernel.

FMS Fluhrer, Martin and Shamir.

FORBAC First-Order Role-Based Access Control.

FPGA Field Programmable Gate Array.

FS File System.

FUD Fear Uncertainty and Doubt.

FUSE File System in User Space.

GAEN Google-Apple Exposure Notification.

GCC GNU Compiler Collection.

GCI Global Cybersecurity Index.

GCM Galois Counter Mode.

GDPR General Data Protection Regulation [105].

GMAC Galois Message Authentication Code.

GNS Global Navigation Systems.

GNSS Global Navigation Satellite Systems.

GOMS Goals, Operators, Methods.

GPG GNU Privacy Guard.

GPRS General Packet Radio Service.

GPS Global Positioning System.

GPU Graphics Processing Unit.

GRE Generic Routing Encapsulation.

GSM Global System for Mobile Communications.

HAC High Assurance Controller.

HACMS High Assurance Cyber Military Systems.

HBA Host Bus Adapter.

HDD Hard Disk Drive.

HDL Hardware Description Language.

HDLC High-Level Data Link Control.

HIDS Host Intrusion Detection System.

HIPAA Health Insurance Portability and Accountability Act.

HLR Home Location Register.

HMAC Hash MAC.

HOL Higher Order Logic.

HPC High Performance Controller.

HR Human Resources.

HRL Hyper-V Replica Log.

HRU Harrison, Ruzzo, and Ullman model.

HSM Hardware Security Module.

HSS Home Subscriber Server.

HSTS HTTP Strict Transport Security.

HTML Hypertext Markup Language.

HTSA Highway Traffic Safety Administration.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

I^2C Inter-Integrated Circuit.

IaaS Infrastructure as a Service.

IBAC Identity-Based Access Control.

IBC Identity-Based Cryptography.

IBE Identity Based Encryption.

IC Integrated Circuit.

ICAS Integrated Condition Assessment System.

ICMP Internet Control Message Protocol.

ICS Industrial Control System.

ICT Information and Communication Technologies.

ICV Integrity Check Value.

ICV Intelligent and Connected Vehicle.

IDE Integrated Development Environment.

IDMEF Intrusion Detection Message Exchange Format.

IDN Internationalised Domain Name.

IdP Identity Provider.

IDPS Intrusion Detection and Prevention System.

IDS Intrusion Detection System.

IDXP Intrusion Detection eXchange Protocol.

IEC International Electrotechnical Commission.

IETF Internet Engineering Task Force.

IFC Information Flow Control.

IFP Integer Factorisation Problem.

IGO International Governmental Organisation.

IGP Interior Gateway Protocol.

IGRP Interior Gateway Routing Protocol.

IIoT Industrial IOT.

IIS Internet Information Services.

IKE Internet Key Exchange.

IMAP Internet Mail Access Protocol.

IMAP Internet Message Access Protocol.

IMD Implantable Medical Device.

IMSI International Mobile Subscriber Identity.

IND Indistinguishable.

IND-CCA Indistinguishability under Chosen Ciphertext Attack.

IND-CPA Indistinguishability under Chosen Plaintext Attack.

INT-CTXT Integrity of Ciphertexts.

IoC Indicator Of Compromise.

IODEF Incident Object Description Exchange Format.

IOMMU Input-Output Memory Management Unit.

IoT Internet of Things.

IP Internet Protocol.

IPC Inter-Process Communication.

IPS Intrusion Prevention System.

IPsec Internet Protocol Security.

IR Intermediate Representation.

IR-UWB Impulse-Radio Ultra Wideband.

IRGC International Risk Governance Council.

IRP Incident Response Plan.

IRTF Internet Research Task Force.

ISA Instruction Set Architecture.

ISAC Information Sharing and Analysis Center.

ISAKMP Internet Security Association and Key Management Protocol.

ISF Information Security Forum.

ISI Information Security Indicators.

ISN Initial Sequence Number.

ISO International Organization for Standardization.

ISP Internet Service Provider.

IV Initialisation Vector.

JIT Just-In-Time.

JOANA Java Object-sensitive ANAlysis.

JSON JavaScript Object Notation.

KA Knowledge Area.

KAOS Keep All Objectives Satisfied.

KAS Kerberos Authentication Server.

KASLR Kernel ASLR.

KDF Key Derivation Function.

KEK Key Encryption Key.

KEM Key Encapsulation Mechanism.

KISS Knowledge In Security protocolS.

KMAC Keccak MAC.

KNX Konnex Bus.

KP-ABE Key-Policy Attribute-Based Encryption.

KPI Key Performance Indicator.

KSA Knowledge, Skills and Abilities.

KVS Key Value Store.

L2TP Layer 2 Tunneling Protocol.

LAN Local Area Network.

LEA Localised Eclipse Attacks.

LED Light-Emmiting Diode.

LEEF Log Event Enhanced Format.

LINQ Language Integrated Query.

LOIC Low Orbit Ion Cannon.

LPC Low Pin Count.

LPI Low Probability of Intercept.

LTE Long Term Evolution.

LTL Linear Temporal Logic.

LTM Long Term Memory.

LTM-EM Episodic Memory.

LTM-SM Semantic Memory.

LTS Labelled Transition System.

LWE Learning With Errors.

MAC Message Authentication Code.

MAC Mandatory Access Control.

MAC Media Access Control.

MAPE-K Monitor Analyze Plan Execute-Knowledge.

MASVS OWASP Mobile Application Security Verification Standard.

ME Management Engine.

MEMS Microelectromechanical Systems.

MILE Managed Lightweight Incident Exchange.

MIME Multipurpose Internet Mail Extensions.

MIMO Multi-Antenna, Multiple Input Multiple Output.

MISP Malware Information Sharing Platform.

MITM Man In the Middle (attack).

MitM Man-in-the-Middle.

MK Master Key.

ML Machine Learning.

MLAT Multilateration.

MME Mobility Management Engine.

MMOG Massive Multiplayer Online Games.

MMU Memory Management Unit.

MPC Multi-Party Computation.

MPC Model Predictive Control.

MPK Memory Protection Key.

MPLS Multiprotocol Label Switching.

MPU Memory Protection Unit.

MPX Memory Protection Extensions.

MSK Master Session Key.

MSSP Managed Security Services Provider.

MSTG Mobile Security Testing Guide.

MTAC Message Time of Arrival Code.

MTE Memory Tagging Extensions.

MTTC Mean Time To Contain.

MTTI Mean Time To Identify.

MTU Maximum Transmission Unit.

MUD Manufacturer Usage Description.

NASA National Aeronautics and Space Administration.

NAT Network Address Translation.

NATO North Atlantic Treaty Organization.

NCA National Crime Agency.

NCSC National Cyber Security Centre.

NDP Neighbor Discovery Protocol.

NERC North American Electric Reliability Corporation.

NFC Near-Field Communication.

NFV Network Functions Virtualisation.

NICE National Institute for Cybersecurity Education.

NIS Network and Information Systems.

NIST National Institute of Standards and Technology.

NRBG Non-Deterministic Random Bit Generator.

NS Non-Secure.

NS name server.

NSA National Security Agency.

NSF National Science Foundation.

NSRL National Software Reference Library.

NTP Network Time Protocol.

NVD National Vulnerability Database.

NX No Execute.

O-DM Open Dependency Modelling.

OAEP Optimized Asymmetric Encryption Padding.

OAG Online Application Generator.

OAuth Open Authorisation.

OCSP Online Certificate Status Protocol.

ODB Outsourced Database.

OEA Outgoing Eclipse Attacks.

OFB Output Feedback.

ONR Office for Nuclear Regulation.

OR Onion Router.

ORAM Oblivious Random Access Memory.

ORM Object Relational Mapping.

OS Operating System.

OSI Open Systems Interconnection.

OSPF Open Shortest Path First.

OT Oblivious Transfer.

OT Operational Technology.

OTP One Time Password.

OTR Off-the-Record messaging.

OW One-Way.

OWASP Open Web Application Security Project.

OWE Opportunistic Wireless Encryption.

P-DS P2P Data Structures.

P-OP P2P Operations.

P2P Peer to Peer.

P3P Privacy Preferences Project.

PaaS Platform as a Service.

PAC Pointer Authentication Code.

PAN Privileged Access Never.

PAP Password Authentication Protocol.

PASS Passive Attack.

PBKDF Password-based Key Derivation Function.

PCB Printer Circuit Board.

PCCP Persuasive Cued Click Points.

PCI Payment Card Industry.

PCI DSS Payment Card Industry Data Security Standard.

PCR Platform Configuration Register.

PCS Process Control System.

PCTL Probabilistic Computation Tree Logic.

PDF Portable Document Format.

PDG Program Dependence Graph.

PEAP Protected Extensible Authentication Protocol.

PGP Pretty Good Privacy.

PII Personally Identifiable Information.

PIN Personal Identification Number.

PIR Private Information Retrieval.

PITM Person In The Middle.

PKC Public Key Crytography.

PKC Public Key Certificate.

PKCS Public Key Cryptography Standards.

PKE Public Key Encryption.

PKI Public-Key Infrastructure.

PLC Programmable Logic Controller.

PMA Protected Model Architecture.

PMK Pairwise Master Key.

PMS Pre-Master Secret.

POLA Principle Of Least Authority.

POP Post Office Protocol.

PoS Proof-of-Stake.

PoW Proof-of-Work.

PPI Pay Per Install service.

PPP Point-to-Point Protocol.

PPPoE PPP over Ethernet.

PPTP Point-to-Point Tunneling Protocol.

PQC Post-Quantum Cryptography.

PRF Pseudo-Random Function.

PRNG Pseudo Random Number Generator.

PRP Pseudo-Random Permutation.

PSD2 European Payment Services Directive 2.

PSK Pre-Shared Key.

PSM Password Strength Meter.

PSPACE Polynomial Space.

PSS Probabilistic Signature Scheme.

PSTN Public Switched Telephone Network.

PTE Page Table Entry.

PTK Pairwise Transient Key.

PUF Physically Unclonable Function.

PUP Potentially Unwanted Program.

PXN Privileged Execute Never.

QKD Quantum Key Distribution.

QPSK Quadrature Phase Shift Keying.

QR Quick Response.

QUIC Quick UDP Internet Connections.

RAID Redundant Array of Inexpensive Disks.

RAII Resource Acquisition Is Initialisation.

RAM Random Access Memory.

RARP Reverse Address Resolution Protocol.

RAT Remote Access Trojan.

RBAC Role-Based Access Control.

REE Rich Execution Environment.

RF Radio Frequency.

RFC Request For Comments.

RFID Radio-Frequency Identification.

RIDL Rogue In-Flight Data.

RIP Routing Information Protocol.

RIR Regional Internet Registry.

RNG Random Number Generator.

ROA Route Origin Authorization.

ROC Receiver Operating Characteristic.

ROM Read-Only Memory.

ROT Root of Trust.

ROV Route Origin Validation.

RPC Remote Procedure Call.

RPKI Resource Public Key Infrastructure.

RPL Routing Protocol for Low-Power and Lossy Networks.

RSA Rivest-Shamir-Adleman.

RSN Robust Secure Networking.

RSS Received Signal Strength.

RSSI Received Signal Strength Indicator.

RTL Register Transfer Level.

RTM Root of Trust for Measurement.

RTOS Real-Time Operating System.

RTP Routing Table Poisoning.

RTR Root of Trust for Reporting.

RTS Root of Trust for Storage.

RTT Round-Trip Time.

RTU Remote Terminal Unit.

S-SDLC Secure-Software Development Lifecycle Project.

SA Security Association.

SaaP Software as a Product.

SaaS Software as a Service.

SAD Security Association Database.

SAE Simultaneous Authentication of Equals.

SAE Society for Automotive Engineering.

SAML Security Assertion Markup Language.

SAMM Software Assurance Maturity Model.

SAST Static Analysis Security Testing.

SAT Satisfiability.

SATA Serial ATA.

SBC Session Border Controller.

SCA Software Composition Analysis.

SCADA Supervisory Control and Data Acquisition.

SCN Supervisory Control Network.

SCSI Small Computer System Interface.

SD Secure Digital.

SDL Security Development Lifecycle.

SDN Software Defined Networking.

SDSI Simple Distributed Security Infrastructure.

SEI Software Engineering Institute.

SEM Scanning Electron Microscope.

SEO Search Engine Optimization.

SET Secure Electronic Transactions.

SGSN Serving GPRS Support Node.

SGX Software Guard Extension.

SHE Somewhat Homomorphic Encryption.

SIEM Security Information and Event Management.

SIM Subscriber Identity Module.

SIMO Single Input, Multiple Output.

SIS Safety Instrumented System.

SLA Service Level Agreement.

SMAP Supervisor Mode Access Protection.

SME Small and Medium-sized Enterprise.

SMEP Supervisor Mode Execution Protection.

SMIME Secure Multipurpose Internet Mail Extensions.

SMM System Management Mode.

SMS Short Message Service.

SMT Satisfiability Modulo Theories.

SMTP Simple Mail Transfer Protocol.

SNARK Succinct Non-Interactive Arguments of Knowledge.

SNMP Simple Network Management Protocol.

SOAR Security Orchestration, Automation and Response.

SOC Security Operating Center.

SoC System on a Chip.

SoD Separation of Duties.

SOIM Security Operations and Incident Management.

SOP Same-Origin-Policy.

SoS Systems-of-Systems.

SP Service Provider.

SPA Simple Power Analysis.

SPF Sender Policy Framework.

SPKI Simple Public-Key Infrastructure.

SPTA Spanning Tree Algorithm.

SQL Structured Query Language.

SQUARE Security Quality Requirements Engineering.

SRAM Static Random Access Memory.

SRE Site Reliability Engineering.

SROP Sigreturn-Oriented Programming.

SSD Solid State Drive.

SSH Secure Shell.

SSL Secure Sockets Layer.

SSO Single Sign On.

SSTP Secure Socket Tunneling Protocol.

STAMP Systems-Theoretic Accident Model and Process.

STIX Structured Thread Information eXchange.

STM Short Term Memory.

STS Station-to-Station.

SUF-CMA Strong Unforgeability under Chosen Message Attack.

SVA Secure Virtual Architecture.

SVP Shortest Vector Problem.

SWEBOK Software Engineering Body of Knowledge.

SWIFT Society for Worldwide Interbank Financial Telecommunication.

TA Transmitter MAC Address.

TA Trusted Authority.

TAL Typed Assembly Language.

taLEA Topology Aware Localised Eclipse Attacks.

TAN Transaction Authentication Number.

TCB Trusted Computing Base.

TCG Trusted Computing Group.

TCP Transmission Control Protocol.

TCSEC Trusted Computer System Evaluation Criteria.

TDOA Time-Difference Of Arrival.

TEE Trusted Execution Environment.

TF-CSIRT Computer Security Incident Response Teams.

TFC Traffic Flow Confidentiality.

TGS Ticket Granting Server.

TGT Ticket Granting Ticket.

TK Temporal Key.

TKIP Temporal Key Integrity Protocol.

TLA Temporal Logic of Actions.

TLB Transaction Lookaside Buffer.

TLD Top-level Domain.

TLS Transport Layer Security.

TLX Task Load Index.

TMAC Temporal MAC.

TNC Trusted Network Connect.

ToA Time of Arrival.

TOCTOU Time Of Check Time Of Use.

ToF Time of Flight.

TOGAF The Open Group Architectural Framework.

TPM Trusted Platform Module.

TRBG True Random Bit Generator.

TRNG True Random Number Generator.

TTL Time To Live.

TTP Trusted Third Party.

TXT Trusted Execution Technology.

U2F Universal 2nd Factor.

UAF Universal Authentication Framework.

UAV Unmanned Aerial Vehicle.

UC Universal Composability.

UCON Usage Control.

UDP User Datagram Protocol.

UDSSS Uncoordinated Direct-Sequence Spread Spectrum.

UE User Equipment.

UEFI Unified Extensible Firmware Interface.

UF Universal Forgery.

UFH Uncoordinated Frequency Hopping.

UFLS Under Frequency Load Shedding.

UI User Interface.

UMDF User Mode Driver Framework.

UML Unified Modelling Language.

UMTS Universal Mobile Telecommunications Systems.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

uRPF Unicast Reverse Path Forwarding.

USB Universal Serial Bus.

UTC Coordinated Universal Time.

UV Unmanned Vehicle.

UWB Ultra-Wideband.

VCC Verifying C Compiler.

VHF Very High Frequency.

VLAN Virtual LAN.

VLSI Very Large Scale Integration.

VM Virtual Machine.

VMI Virtual Machine Introspection.

VNF Virtual Network Function.

VPN Virtual Private Network.

VST Verified Software Toolchain.

VXLAN Virtual eXtensible LAN.

W3C World Wide Web Consortium.

WAF Web Application Firewall.

WAN Wide Area Network.

WEP Wired Equivalent Privacy.

WLAN Wireless LAN.

WORM Write-Once, Read-Many.

WPA Wi-Fi Protected Access.

XACML eXtensible Access Control Markup Language.

XD Execute Disable.

XDAS Distributed Audit Service.

XML Extensible Markup Language.

XN Execute Never.

XOF Extendable Output Function.

XSS Cross-Site Scripting.

ZK Zero Knowledge.

Glossary

419 scam a particular version of advance fee fraud specific to Nigeria.

access control the process of denying or granting access requests.

Actuator An actuator is a device that moves or controls some mechanism. An actuator turns a control signal into mechanical action such as an electric motor. Actuators may be based on hydraulic, pneumatic, electric, thermal or mechanical means, but are increasingly being driven by software. An actuator ties a control system to its environment [2269].

advance fee fraud a crime in which the victim is promised a reward, but in order to obtain it has to first pay a small fee to the fraudster.

advanced persistent threat An attack to an organization that continues its activities and yet remains undetected for an extended period of time.

affiliate programme a scheme where main organisation provides a “brand” and all the means required to carry out orders, shipments and payments.

ALARA A method to reduce risk to levels As Low As Reasonably Allowable.

ALARP A method to reduce risk to levels As Low As Reasonably Possible.

alert Notification that a specific attack has been directed at an organisation’s information systems (Source = NIST IR 7298r2). In the SOIM context, an alert should refer to an event, or group of events, of interest from a security perspective, representing either an attack symptom or consequence. An alert is necessarily the outcome of an analysis process performed by an Intrusion Detection System sensor on event traces.

anonymity The state of being not identifiable within a set of subjects, the anonymity set.

appification The replacement of websites with applications that run on mobile devices..

ASIC Application Specific Integrated Circuit is one class on integrated circuits, where the circuit is tuned to a specific application or set of applications. E.g. a TPM is a dedicated ASIC for security applications .

attack An attempt to gain unauthorised access to an Information System services, resources, or information, or an attempt to compromise system integrity. (Source = NIST IR 7298r2).

attack surface The set of entry points where an attacker can attempt unauthorised access. Security approaches endeavor to keep the attack surface as small as possible.

authentication verifying a claimed attribute value.

authentication The process of verifying the identity of an individual or entity.

authorisation a) deciding whether to grant an access request (to a subject) or b) assigning access rights to a principal.

botnet A network of compromised computers (or, bots) that is controlled by an attacker to launch coordinated malicious activities.

bulletproof hosting service providers providers that are well known not to comply with law enforcement takedown requests. This is made possible by either being located in countries with lax cybercrime legislation, or by the service provider operators actively bribing local law enforcement.

Byzantine Anomalous behavior when an entity/attacker sends different (albeit valid) information to different recipients. The reader is referred to [1108] for the technical definition.

card skimming the practice of installing devices on ATM that allow for the cloning of the cards that are being inserted.

carving (File/data content carving) The process of recovering and reconstructing file content directly from block storage without using the filesystem metadata. More generally, data (structure) carving is the process of reconstructing logical objects (such as files and database records) from a bulk data capture (disk/RAM image) without using metadata that describes the location and layout of the artifacts. Data carvers use knowledge of the data formats to identify and validate the extracted content..

certificate a digitally signed data structure binding an entity (called subject) to some attribute.

click fraud the practice of using malware to generate fake clicks on websites.

CMOS Complementary Metal Oxide Semiconductor technology is the most popular silicon technology to make integrated circuits. It consitst of complementary PMOS and NMOS transistors. Its main advantages are that it has a very low static power consumption and relative robust operation. Hence it made it possible to integrate a large number of transistors (millions to billions) into one integrated circuit.

compromise Disclosure of information to unauthorised persons, or a violation of the security policy of a system in which unauthorised intentional or unintentional disclosure,

modification, destruction, or loss of an object may have occurred. (Source = NIST IR 7298r2).

Computer Security Incident Response Team A capability set up for the purpose of assisting in responding to computer security-related incidents; also called a Computer Incident Response Team (CIRT) or a CIRC (Computer Incident Response Center, Computer Incident Response Capability). (Source = NIST IR 7298r2).

confidentiality The property that ensures that information is not made available or disclosed to unauthorised individuals, entities, or processes.

consensus Consensus (and similarly for Consistency) refers to mechanisms and the property of achieving varied types of agreement on values or coordination of state/entities, typically in the presence of specified failures. As there are precise technical specifications involved for consensus and different types of consistency, the reader is referred to the section on Coordination Properties and to the references [1056, 1057, 1062].

consumer In the context of a given transaction, a natural person who enters into a transaction other than for business or professional purposes. A given person may act as a consumer in some transactional contexts, and a non-consumer in others. N.B. This definition is far from universal. Some laws adopt definitions of 'consumer' that vary from this.

coordination schema The mechanisms that help orchestrate the actions of the involved entities.

countermeasure Actions, devices, procedures, or techniques that meet or oppose (i.e., counters) a threat, a vulnerability, or an attack by eliminating or preventing it, by minimising the harm it can cause, or by discovering and reporting it so that corrective action can be taken. (Source = NIST IR 7298r2).

Covert Channel Attack An attack that results in the unauthorised capability to glean or transfer information between entities that are not specified to be able to communicate as per the security policy.

CPU Central Processing Unit is a general purpose integrated circuit made to execute a program. It typically consists of an arithmetic unit, a program control unit, a bus structure and storage for code and data. Many types and variations exists. One SOC could contain one or more CPU cores with peripherals, extra memory, etc.

credential an input presented for authentication.

Critical National Infrastructure Facilities, systems, sites, information, people, networks and processes, necessary for a country to function and upon which daily life depend.

cryptocurrency mining the practice of generating cryptocurrencies by solving cryptographic tasks.

cyber-dependent crime crime that can only be committed with the use of computers or technology devices.

cyber-enabled crime crime that has an increased reach compared to its offline counterpart through the use of technology.

Cyber-Physical System Engineered systems that are built from, and depend upon, the seamless integration of computation, and physical components [2270].

cyberbullying sending or posting harmful material or engaging in other forms of social aggression using the Internet or other digital technologies.

cyberspace A global domain within the information environment consisting of an interdependent network of information system infrastructures including the Internet, telecommunications networks, computer systems, and embedded processors and controllers [343].

cyberstalking the practice of using electronic means to stalk another person.

delegation the act of granting access rights one holds to another principal.

Denial of Service The prevention of authorised access to resources or the delaying of time-critical operations. (Time-critical may be milliseconds or hours, depending on the service provided.) (Source = NIST IR 7298r2).

digital (forensic) trace An explicit, or implicit, record that testifies to the execution of specific computations, or the communication and/or storage of specific data..

digital forensics The process of identifying and reconstructing the relevant sequence of events that have led to the currently observable state of a target IT system or (digital) artifacts.

Distributed Control System A control system that combines supervised control of several individual computer-based controllers in different control-loop throughout a process. In contrast to SCADA systems, the supervision of these systems tends to be onsite rather than remote [2271].

Distributed Denial of Service A Denial of Service technique that uses numerous hosts to perform the attack. (Source = NIST IR 7298r2).

doxing an attack where the victim's private information is publicly released online.

DRAM DRAM is Dynamic Random Access Memory. Very popular because of its high density. It requires only one transistor and one small capacitance to store one bit of data. It requires regular refreshing. It looses its value when the power supply is turned off.

drive-by download attack an attack in which a webpage tries to exploit a software vulnerability in the victim's browser with the goal of installing malware.

dumpz stolen credit card records.

email spam unsolicited bulk email.

encryption The process of transforming information (commonly referred to as plaintext/data) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, commonly referred to as a cryptographic key.

event Any observable occurrence in a network or system. (Source = NIST IR 7298r2). Trace of activity provided by a computing environment. In the SOIM context, this is a piece of evidence logged that an activity was performed in the monitored system. Events are

acquired sequentially by sensors to obtain a trace of the activity on a computer or network, to find indicator of compromise.

exploit Software or data that takes advantage of a vulnerability in a system to cause unintended consequences. (Source = NCSC Glossary).

exploit kit a tool that collects a large number of vulnerabilities and are sold on the black market for other criminals to use.

file A named (opaque) sequence of bytes that is stored persistently.

file system (filesystem) An operating system subsystem that is responsible for the persistent storage and organisation of user and system files on a partition/volume.

firewall A gateway that limits access between networks in accordance with local security policy. (Source = NIST IR 7298r2).

forensics The practice of gathering, retaining, and analysing computer-related data for investigative purposes in a manner that maintains the integrity of the data. (Source = NIST IR 7298r2).

FPGA A Field Programmable Gate Array or FPGA is a specialized integrated circuit that contains configurable logic, which can still be programmed after fabrication. Programming is done by loading a bitstream which configures each of the programmable logic gates individually.

fullz stolen credit card records that also contain billing information.

GPU Graphics Processing Unit is a specialized programmable integrated circuit. Its components (arithmetic units, instruction set, memory configuration, bus structure) are all optimized to accelerate graphics, video and image processing applications.

hacktivism the act of computer crime motivated by a political goal.

HDL A Hardware Description Language is a special language to describe digital hardware at the register transfer level. Most well known languages are VHDL and Verilog.

homomorphic encryption A form of encryption that when computing on ciphertexts, generates an encrypted result which, when decrypted, matches the result of the computation as if it had been performed on the plaintext.

honeypot A system (e.g., a Web server) or system resource (e.g., a file on a server, an email address, a table or row or column in a database) that is designed to be attractive to potential crackers and intruders and with no authorised users other than its administrators (Source = NIST IR 7298r2). In the context of SOIM, honeypots can be operated locally as an additional detection method supplementing IDS sensors, or by an external CTI service provider.

IC An Integrated Circuit is an electronic device that contains a large amount of electronic components, mostly transistors integrated into one piece of semiconductor material, usually CMOS silicon. A common name is a ‘chip’ or a ‘silicon chip’ .

identity management the process of creating, using, and terminating electronic identities.

Impact The result of a threat exploiting a vulnerability.

impact The magnitude of harm that can be expected to result from the consequences of unauthorised disclosure of information, unauthorised modification of information, unauthorised destruction of information, or loss of information or information system availability (Source = NIST IR 7298r2). In the context of SOIM, this is the extent of damage caused by the attack to either the ICT infrastructure, or to business processes.

incident Actions taken through using computer networks that result in an actual or potentially adverse effect on an information system and/or the information residing therein. (Source = NIST IR 7298r2). In the SOIM context, an incident is described as a set of alerts that are considered evidence of a cybersecurity breach, generally a successful attack (although serious attempts, or attempts against critical systems, may also be considered incidents.

indicator of compromise Recognised action, specific, generalized, or theoretical, that an adversary might be expected to take in preparation for an attack. (Source = NIST IR 7298).

Industrial Control Systems General term that encompasses several types of control systems, including supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC) often found in the industrial sectors and critical infrastructures. An ICS consists of combinations of control components (e.g., electrical, mechanical, hydraulic, pneumatic) that act together to achieve an industrial objective (e.g., manufacturing, transportation of matter or energy) [2048].

Industrial Internet of Things System that connects and integrates industrial control systems with enterprise systems, business processes, and analytics [2272].

Industry 4.0 Industry 4.0 refers to the modernization of manufacturing with Internet of Things services, which provide the basis for the fourth industrial revolution. The first industrial revolution was enabled by the introduction of mechanical production facilities powered by water and steam, the second revolution was enabled by mass production powered by electrical energy, and the third revolution was enabled by the introduction of electronics and information technology [2273].

Information System A discrete set of information resources organised for the collection, processing, maintenance, use, sharing, dissemination, or disposition of information being monitored (Source = NIST IT 7298r2). In the SOIM context, it designs the ICT infrastructure to detect possible attacks.

integrity The property that ensures that data is real, accurate and safeguarded from unauthorised user modification.

international governmental organisation A legal person established and recognised as such by more than one state pursuant to treaty (e.g., the United Nations, INTERPOL, the International Maritime Organization, etc.). In practice, often simplified as 'International Organisation' or 'Treaty Organisation'.

Internet The Internet is the single, interconnected, worldwide system of commercial, governmental, educational, and other computer networks that share (a) the protocol suite specified by the Internet Architecture Board (IAB), and (b) the name and address spaces managed by the Internet Corporation for Assigned Names and Numbers (ICANN).(Source = NIST IR 7298r2).

Internet of Things Network of physical objects or "things" embedded with electronics, software, sensors, and connectivity to enable objects to exchange data with the manufacturer, operator and/or other connected devices. The IoT refers to devices, that are often constrained in communication and computation capabilities, now becoming more commonly connected to the Internet, and to various services that are built on top of the capabilities these devices jointly provide[1].

Intrusion Detection System (IDS) Hardware or software product that gathers and analyses information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside organisations) and misuse (attacks from inside the organisations.) See also sensor. (Source = NIST IR 7298r2).

Intrusion Prevention System (IDPS) Intrusion Detection System with the additional capability to take immediate and local action to block the detected attack. This implies two differences, the positioning of the device as an interceptor through which all requests, malicious or benign, will pass, and the ability to diagnose the malicious behaviour with certainty. See also Intrusion Detection System and sensor.

jurisdiction See the discussion in Section 3.2.

key-logger A virus or physical device that logs keystrokes to secretly capture private information such as passwords or credit card details.(Source = BSI Glossary).

leader election Following the replacement of an existing leader, on failure of a leader or for fairness or load balancing, the process of electing a new entity to perform the leadership activities of coordination.

legal action The process by which a person brings a legal claim to a tribunal for adjudication or to enforce the results of a prior adjudication. This is the method used to enforce a right of action.

legal person An entity vested with sufficient characteristics of personhood to merit a legal identity separate from its constituent members. These characteristics include: the right to commence or respond to legal action in the entity's name; the right to own assets in the entity's name; and the right to enter into obligations in the entity's name. Legal persons generally include: states; international governmental organisations; public or private entities incorporated pursuant to the law of a state and vested by that state with the characteristics of personhood, such as an English public limited company (PLC), a Delaware limited liability partnership (LLP), a French société anonyme (S.A.), a German gesellschaft mit beschränkter hafting (GmbH), the City of New York, etc..

Likelihood A measure capturing the degree of possibility that a threat will exploit a vulnerability, and therefore produce an undesirable outcome.

logical acquisition The process of obtaining the data relies on one or more software layers as intermediaries to acquire the data from the storage device.

logical volume A collection of physical volumes presented and managed as a single unit.

[1] https://www.ietf.org/topics/iot/

macro virus A virus that attaches itself to documents and uses the macro programming capabilities of the document's application to execute and propagate.(Source = NIST IR 7298).

malware A program inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications or operating system, or of otherwise annoying or disrupting the victim. Synonym = malicious code. (Source = NIST IR 7298r2).

malware analysis The process of analyzing malware code and understanding its intended functionalities.

malware detection The process of detecting the presence of malware in a system.

metadata Information about data or sent along with data, e.g., the IP address, the location, or the operative system a message is sent from.

metamorphic malware Malware of which each iteration or instance has different code from the preceding one. The code changes make it difficult to recognize the different iterations are the same malware (contrast with polymorphic malware).

meterpreter A tool that allows an attacker to control a victim's computer by running an invisible shell and establishing a communication channel back to the attacking machine.

middleware A software layer between the Operating System and the Application Layer designed to facilitate the interconnection and interaction between distributed components. Often referred to as the "software glue" that binds components together.

money mule a person who is recruited by a criminal to perform money laundering.

natural person A human being, living or deceased.

object the entity accessed by an access operation.

obligation operation to be performed in conjunction with an access request that had been granted.

Operational Technology Components and systems, also known as Industrial Control Systems (ICS) that underpin Critical National Infrastructure (CNI) such as energy provision, transportation, and water treatment. They also underpin complex manufacturing systems where processes are too heavy-duty, monotonous, or dangerous for human involvement.

Operational Technology Hardware and software that detects or causes a change through the direct monitoring and/or control of physical devices, processes and events in the enterprise [2274].

overlay Refers to the overlay network in peer-to-peer systems that is a virtual network linking a specified set of nodes as built on top of the nodes of the physical network.

packed malware Packed malware is obfuscated malware in which the malicious program is compressed and cannot be analysed statically.

packing A technique to obfuscate malware (see packed malware).

partition (physical volume) A contiguous allocation of blocks for a specific purpose, such as the organisation of a file system.

passive dns A mechanism to collect large amounts of DNS data by storing DNS responses from servers. (Source = RFC7719.

PCB A Printed Circuit Board is a specialized board which holds the different integrated circuits. It is made of an insulated material with copper wiring to connect the pins of different integrated circuits with each other and the outside.

permission synonym for access right.

person A natural person or legal person.

phishing a fraud that lures users into giving away access credentials to online services to a criminal.

phishing kit a programme that can be installed on a server and will produce an appropriately-looking web page for many popular services.

physical acquisition The process of obtaining the data directly from hardware media, without the mediation of any (untrusted) third-party software.

polymorphic malware Malware that changes each instance to avoid detection. It typically has two parts: the decryptor and the encrypted program body. Each instance can encrypt the malware program differently and hence has a different decryptor; however, once decrypted, the same malware code is executed. (contrast with metamorphic malware).

polymorphism See polymorphic malware.

potentially unwanted program A program that may not be wanted by a user and is often downloaded along with a program that the user wants. Examples include adware, spyware, etc.

principal in policies, the active entity in an access request.

privilege an access right to a system resource.

privilege a synonym for access right.

Programmable Logic Controller (PLC) An industrially hardened computer-based unit that performs discrete or continuous control functions in a variety of processing plant and factory environments. It was originally intended as a relay replacement equipment for the automotive industry. As opposed to DCS, they can be sold as stand-alone equipment (instead of an integrated system as DCS) [2271].

proof See the discussion in Section 3.1.4.

prove See the discussion in Section 3.1.4.

RAM RAM is Random Access Memory. It is memory on an integrated circuit to store values (data or code).

ransomware Malicious software that makes data or systems unusable until the victim makes a payment. (Source = NIST IR 7298).

reference monitor the abstract component that mediates all accesses to objects.

replication The aspect of adding physical or logical copies of a resource.

reshipping mule a person who is recruited by a criminal to send goods purchased with stolen credit cards abroad.

right of action A right arising in law for one person to take legal action against another.

Root of Trust A root of trust is a component used to realize a security function, upon which a designer relies but of which the trustworthiness can not be explicitly verified [1013] .

safety In the context of malware analysis, a requirement that malware should be prevented from causing damage to the connected systems and networks while it runs in the analysis environment.

security model high-level specifications of a system designed to enforce certain security policies.

Sensor A device that perceives certain characteristics of the real world and transfers them into a digital representation [2275].

sensor Equipment (software and/or hardware) aimed at detecting and alerting cyberattacks, also referred to as the Intrusion Detection System (IDS).

sextortion a crime in which a miscreant lures victims to perform sexual acts in front of a camera (e.g., a webcam in a chatroom), records those acts, and later asks for a monetary payment in order not to release the footage.

Side Channel Attack Side Channel Attack An attack based on information gained from the implementation of a system (e.g., that of a cryptographic algorithm) rather than weaknesses in the algorithm (e.g., those discovered via cryptanalysis). Side Channel attacks can be mounted based on monitoring data or key dependent variations in execution time, power consumption or electromagnetic radiation of integrated circuits.

signature A characteristic byte pattern used in malicious code or an indicator, or set of indicators, that allows the identification of malicious network activities. (Source = NIST IR 7298r2). A more current definition is indicator of compromise.

sinkholing A technique used by a DNS server to give out false information to prevent the use of a domain name.

slack space The difference between the *allocated* storage for a data object, such as file, or a volume, and the storage in actual use.

SOC System-on-chip is a very large integrated circuit that combines multiple large components, which in previous generations might have consisted of multiple chips on one circuit board.

spam The abuse of electronic messaging systems to indiscriminately send unsolicited bulk messages. (Source = NIST IR 7298).

spyware Software that is secretly or surreptitiously installed into an information system to gather information on individuals or organizations without their knowledge; a type of malicious code. (Source = NIST IR 7298).

SRAM SRAM is Static Random Access Memory, a type of memory that makes it easy to address each individual bit, requiring typically 6 transistors per bit. SRAM looses its values when the power supply is turned off.

state A legal person that normally possesses the following qualifications: a permanent population; a defined territory; a government; and a legal capacity to enter into relations with other states. In the context of pubic international law and diplomacy, confirming the status of an entity as a 'state' is a decision normally made individually by other states through proclamation, exchange of ambassadors, etc. In the context of a federation (e.g., States of Australia, Provinces of Canada, Länder of Germany, States of the US), recognition normally takes place in accordance with the constitutional procedures of that federation.

subject an entity in an IT system that speaks for a principal (sometimes used as a synonym for principal).

Supervisory Control and Data Acquisition A supervisory control system that integrates remote data acquisition systems with data transmission systems and Human-Machine Interface (HMI) software to provide a centralised monitoring and control system for numerous process inputs and outputs. SCADA systems are designed to collect field information, transfer it to a central computer facility, and display the information to the operator graphically or textually, thereby allowing the operator to monitor or control an entire system from a central location in near real time. SCADA systems and Distributed Control Systems (DCS) are often networked together. This is the case for electric power control, although the electric power generation facility is controlled by a DCS, the DCS must communicate with the SCADA system to coordinate production output with transmission and distribution demands [2048].

territorial Of, or related to, territory.

territory A delimited region of geographic space (i.e., real space, including air and water). Often used in law to describe boundaries of a state (e.g., the territory of the Republic of Italy).

threat An individual, event, or action that has the capability to exploit a vulnerability.

token a device used for authentication.

token a data structure encoding the result of an access decision.

trace Ordered set of events, generally of the same type, gathered in a container for easy sequential access. A trace is, for example, a packet capture or a log file. The order is not necessarily chronological, but is fixed at the time of writing the trace.

Transducer A device that converts variations in a physical quantity, such as pressure or brightness, into an electrical signal, or vice versa [2276].

triage Triage is a partial forensic examination conducted under (significant) time and resource constraints..

trojan A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the program. (Source = NIST IR 7298).

Trusted Computing Base The Trusted Computing Base (TCB) is the typical root of trust for a computer system. It contains all hardware and software components, that need to be trusted and of which the trustworthiness can not be explicitly verified. If security

vulnerabilities occur in the TCB, then the security of the entire computer system might be at risk.

Trusted Platform Module A Trusted Platform Module is a functional component that can perform cryptographic operations, manage keys, and provide remote attestation services. When implemented as a cryptographic co-processor and embedded on a personal computer platform, it provides roots of trust so that the platform can identify itself, its current configuration, and running software..

unlinkability The property of two (or more) items in a system that ensures that these items are no more and no less related than they are related concerning the a-priori knowledge of the adversary.

virus A hidden, self-replicating section of computer software, usually malicious logic, that propagates by infecting - i.e., inserting a copy of itself into and becoming part of - another program. A virus cannot run by itself; it requires that its host program be run to make the virus active. (Source = SANS security glossary).

VLSI Very Large Scale Integration is a collection of electronic design automation techniques to translate a HDL description into the actual polygons required for the maskmaking of an integrated circuit. The VLSI tools made it possible to manage the complexity of designing large integrated circuits.

vulnerability Something open to attack or misuse that could lead to an undesirable outcome.

webification The process of using web technologies to display and transfer content on the web and mobile devices..

WiFi A family of radio technologies that is used for the wireless local area networking (WLAN).

worm A computer program that can run independently, can propagate a complete working version of itself onto other hosts on a network, and may consume computer resources destructively. (Source = SANS security glossary).

YARA YARA is a tool primarily used in malware analysis. It describes malware families using textual or binary patterns. (Source = Wikipedia).

Index