packt

1ST EDITION

TLS Cryptography In-Depth

Explore the intricacies of modern cryptography and the inner workings of TLS

DR. PAUL DUPLYS
DR. ROLAND SCHMITZ

TLS Cryptography In-Depth

Explore the intricacies of modern cryptography and the inner workings of TLS

Dr. Paul Duplys

Dr. Roland Schmitz

‹packt›

BIRMINGHAM—MUMBAI

TLS Cryptography In-Depth

Group Product Manager: Pavan Ramchandani
Publishing Product Manager: Neha Sharma
Senior Editor: Arun Nadar
Technical Editor: Arjun Varma
Copy Editor: Safis Editing
Project Coordinator: Ashwini Gowda
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Production Designer: Vijay Kamble
Marketing Coordinator: Marylou De Mello

First published: December 2023
Production reference: 1291223

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK

ISBN 978-1-80461-195-1

www.packtpub.com

Contributors

About the authors

Dr. Paul Duplys is chief expert for cybersecurity at the department for technical strategies and enabling within the Mobility sector of Robert Bosch GmbH, a Tier-1 automotive supplier and manufacturer of industrial, residential, and consumer goods. Previous to this position, he spent over 12 years with Bosch Corporate Research, where he led the security and privacy research program and conducted applied research in various fields of information security. Paul's research interests include security automation, software security, security economics, software engineering, and AI. Paul holds a PhD degree in computer science from the the University of Tübingen, Germany.

Dr. Roland Schmitz has been a professor of internet security at the Stuttgart Media University (HdM) since 2001. Prior to joining HdM, from 1995 to 2001, he worked as a research engineer at Deutsche Telekom, with a focus on mobile security and digital signature standardization. At HdM, Roland teaches courses on internet security, system security, security engineering, digital rights management, theoretical computer science, discrete mathematics, and game physics. He has published numerous scientific papers in the fields of internet and multimedia security. Moreover, he has authored and co-authored several books. Roland holds a PhD degree in mathematics from Technical University Braunschweig, Germany.

Writing this book has been an amazing journey, and it is our great pleasure to thank the people who have accompanied and supported us during this time: our

project coordinators at Packt Publishing, Ashwini Gowda and Arun Nadar, and our meticulous technical reviewers, Andreas Bartelt, Simon Greiner, and Christos Grecos.

About the reviewers

Andreas Bartelt holds a diploma in computer science (bioinformatics) and has specialized in cybersecurity topics for more than 20 years. He works at Robert Bosch GmbH as an expert in the field of cryptography and secures the deployment of cryptographic protocols such as TLS, IPsec, and SSH. Andreas is also a BSD enthusiast and has acquired extensive practical experience in securing POSIX-based operating systems (e.g., Linux) as well as their secure integration with hypervisors (e.g., Xen).

Dr. Christos Grecos (SM IEEE 2006, FSPIE 2023) is the chair and professor of the CS department at Arkansas State University. He was vice dean of PG research at NCI Ireland, chair of the CS department at Central Washington University (US), and dean of FCIT at Sohar University (Oman). He also has 13 years of experience in the UK as a professor, head of school, and associate dean for research. His research interests include image/video compression standards, processing and analysis, networking, and computer vision. He is on the editorial board of many international journals and has been invited to give talks at various international conferences. He has obtained significant funding for his research from several agencies, such as the UK EPSRC, UK TSB, the EU, and the Irish HEC.

Simon Greiner has been working as an automotive security expert for Robert Bosch GmbH, one of the largest automotive suppliers, for more than five years. He works in the lead engineering team for security and supports projects on different topics regarding security engineering, such as threat and risk analysis, security concepts, testing, and implementation security. He also supports pre-development projects on security topics, mainly in the context of autonomous driving. Before joining Robert Bosch GmbH, Simon obtained his PhD in computer science with a specialization in information security from the Karlsruhe Institute of Technology, Germany.

Table of Contents

Preface

Hello and welcome to *TLS Cryptography In-Depth*!

As you perhaps know, there are already many excellent books on cryptography out there, written by renowned experts in the field. So why did we write yet another?

First of all, we wanted to make cryptography easier to grasp by showing how the theory of cryptography is used in real-world cryptographic applications. It is impossible to provide a serious introduction to cryptography without delving deeply into abstract mathematical concepts, and this book is no exception. But oftentimes, these mathematical concepts are presented in a way that is difficult for a beginner to follow, and particularly to relate theory to practice, so it takes a lot of patience and energy until you get to the seemingly far-away applications. Finally, these applications are often presented quite briefly, almost like an afterthought.

Yet applications of cryptography profoundly affect our daily lives and are not remote at all. Perhaps most importantly, practically everybody who is surfing the web today uses web addresses starting with `https`, which stands for **Hypertext Transport Protocol Secure**, and the *Secure* part is realized by a cryptographic protocol called **Transport Layer Security**, or **TLS** for short. If you are using the Firefox browser, for example, and click on the padlock icon next to the URL you are visiting, a few clicks later, you will arrive at the technical details of the *Security* tab of the page info. Here, a typical entry could be as follows:

```
TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3
```

What do these abbreviations mean? Is this really a secure connection? Providing you with the knowledge necessary to answer these questions is one of the main goals of this book.

As we will see, much of present-day cryptography comes together in TLS. We therefore use TLS not just as an application but as a leitmotif of our book. That is, all cryptographic concepts are ultimately motivated by their appearance within the TLS protocol, and advanced cryptanalytic techniques such as linear and differential cryptanalysis are discussed only if they affect TLS protocol design.

TLS is a rather old protocol: its first version dates back to 1994 (under the name **Secure Sockets Layer**, or **SSL**). In 2018, TLS underwent a major revision: not only were many old, insecure cryptographic options deprecated but also protocol messages and their sequence were changed in the latest TLS version, 1.3. The underlying internet standard, IETF RFC 8446, however, is rather complex, densely written, and provides little in the way of motivation.

Therefore, our second reason for writing this book was to show how the design of TLS 1.3 is motivated by good cryptographic practices and earlier cryptographic attacks. Very often, we also dive deeply into TLS 1.3 specification and investigate the meaning of its various data structures. Therefore, you may also read this book as a detailed introduction to the TLS protocol and its nuts and bolts, or use it as a companion to IETF RFC 8446.

Who is this book for

In this book, we address two general audience groups:

- Students of mathematics, computer science, computer engineering, and related disciplines interested in modern cryptography and its deployment within TLS
- IT professionals such as cybersecurity specialists, security engineers, cryptographers, communication engineers, software developers, and administrators interested in the inner workings of TLS

We develop most concepts from scratch. However, some previous exposure to computer networking and fundamental mathematics will certainly help you.

What this book covers

The book starts with a general introduction to cryptography in *Part 1, Getting Started*. *Part 2, Shaking Hands*, and *Part 3, Off the Record*, are loosely organized around the most important subprotocols of TLS, the *handshake protocol* and the *record protocol*. Finally, *Part 4, Bleeding Hearts and Biting Poodles*, extensively covers known attacks on previous TLS versions at the handshake, record and implementation levels.

More specifically, this is what the individual chapters are about:

- *Chapter 1, The Role of Cryptography in the Connected World*, sets the scene by providing some answers to why there are so many insecure IT systems and how cryptography helps to mitigate our security problems.
- *Chapter 2, Secure Channel and the CIA Triad*, describes the general goals and objectives you can achieve with the help of cryptography and introduces cryptography's main protagonists, **Alice** and **Bob**, and their ubiquitous opponents, **Eve** and **Mallory**.
- *Chapter 3, A Secret to Share*, teaches you what a cryptographic key – a secret shared by Alice and Bob – really is, why it is needed to establish a secure channel, and how long it has to be for Alice and Bob to communicate securely.
- *Chapter 4, Encryption and Decryption*, explains how keys are used together with cryptographic algorithms to encrypt and decrypt secret messages, and describes the prerequisites for secure encryption and decryption.
- *Chapter 5, Entity Authentication*, covers **entity authentication**, an important security objective from the CIA triad that assures Alice of the identity of Bob.
- *Chapter 6, Transport Layer Security at a Glance*, concludes *Part 1, Getting Started*, by taking a first look at **Transport Layer Security** (**TLS**) and explores the role of the World Wide Web in the development of TLS.
- *Chapter 7, Public-Key Cryptography*, explains the mathematical techniques that enable secure key transport and key agreement over an insecure channel.

- *Chapter 8, Elliptic Curves*, introduces special mathematical objects that are widely used within TLS 1.3 because they allow the use of much shorter keys compared to traditional public-key cryptography schemes.
- *Chapter 9, Digital Signatures*, covers an important application of public-key cryptography which provides message integrity and authenticity and ensures another special security objective called **non-repudiation**.
- *Chapter 10, Digital Certificates and Certification Authorities*, shows how Bob can verify the authenticity of Alice's public key by relying on a **trusted third party**.
- *Chapter 11, Hash Functions and Message Authentication Codes*, explains hash functions and message authentication codes, the main cryptographic mechanisms to ensure the authenticity of messages.
- *Chapter 12, Secrets and Keys in TLS 1.3*, examines in detail the different types of secrets and keys Alice and Bob establish during the TLS 1.3 Handshake protocol.
- *Chapter 13, TLS Handshake Protocol Revisited*, zooms out of the cryptographic details and gives a high-level description of the TLS handshake using state machines for the TLS server and the TLS client.
- *Chapter 14, Block Ciphers and Their Modes of Operation*, discusses how the TLS Record protocol uses block ciphers and their modes of operation to protect application data transmitted between Alice and Bob.
- *Chapter 15, Authenticated Encryption*, introduces a special block cipher mode of operation that combines encryption and message authentication in a single algorithm.
- *Chapter 16, The Galois Counter Mode*, gives a detailed description of the authenticated encryption algorithm that all TLS 1.3 implementations must support.
- *Chapter 17, TLS Record Protocol Revisited*, zooms out of technical and mathematical details again and revisits the TLS Record protocol by showing how the cryptographic mechanisms covered so far fit together.

- *Chapter 18, TLS Cipher Suites*, covers the combinations of ciphers and cryptographic algorithms that any TLS 1.3 endpoint must support and implement.
- *Chapter 19, Attacks on Cryptography*, describes attacks on cryptographic schemes and cryptographic protocols from a conceptual perspective.
- *Chapter 20, Attacks on the TLS Handshake Protocol*, studies actual, real-world attacks on the Handshake protocol in earlier TLS versions. These attacks either try to get hold of the key established during the handshake or to impersonate one of the communicating parties.
- *Chapter 21, Attacks on the TLS Record Protocol*, explores attacks on TLS records that aim to extract the data transmitted in the encrypted records.
- *Chapter 22, Attacks on TLS Implementations*, covers attacks that exploit implementation bugs in software stacks implementing TLS.

To get the most out of this book

Simply be curious! Since we develop most concepts from scratch, you don't need any additional literature, information sources, or tools to follow the book. If you decide to delve deeper into cryptography than what is covered in the book, you will find numerous useful references in the bibliography.

For readers interested in hands-on experiments with TLS, we describe several simple experiments that you can replicate (and extend) on any computer with the *OpenSSL* toolkit installed on it. The code listings in the corresponding chapters contain complete instructions, so there is nothing extra to download.

If you are generally interested in modern cryptography, you may skip sections describing how the individual cryptographic primitives and mechanisms are used in TLS 1.3. However, we recommend you have at least a cursory look at *Part 4, Bleeding Hearts and Biting Poodles*, to get an idea of why getting cryptography right is hard.

If you are interested specifically in TLS 1.3, we recommend reading the book side by side with RFC 8446. Moreover, as you are likely familiar with basic cryptography, you may skip *Part 1, Getting Started*.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text` indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "If the client provides parameters, for example, the set of specific cryptographic algorithms or a `key_share` value it wants to use for the TLS handshake, that are not supported by the server, the server replies with a `HelloRetryRequest` message."

A block of code is set as follows:

```
struct {
    uint16 length = Length;
    opaque label <7..255 > = "tls13" + Label;
    opaque context <0..255 > = Context;
} HkdfLabel;
```

Any command-line input or output is written as follows:

```
$ gh repo clone duplys/tls_lab
$ cd tls_lab/openssl_docker
$ docker build . -t openssl310
```

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *TLS Cryptography In-Depth*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere? Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

https://packt.link/free-ebook/9781804611951

2. Submit your proof of purchase

3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1

Getting Started

In this part, we set the scene for the **Transport Layer Security** (**TLS**) protocol. After discussing the history of the internet and TLS, we introduce the three basic security services provided by TLS, namely, *confidentiality*, *integrity* and *authenticity*, and give a first, high-level overview of TLS.

More specifically, we look at the role of cryptography in the modern connected world and highlight the reasons why **Secure Sockets Layer** (**SSL**), a predecessor of TLS, was invented in the early 1990s. Next, we explain why connectivity and complexity are the main drivers of cybersecurity and, in turn, cryptography in the modern connected world. We then introduce two cryptographic concepts: the *secure channel* and the *CIA triad*. We then show what cryptographic keys are, how the confidentiality of information transmitted between two parties can be protected using encryption and decryption, and how these parties can ensure that they are actually talking to each other rather than to an attacker. Finally, we give a high-level overview of the TLS protocol to illustrate how the theoretical concepts of cryptography are applied in TLS.

This part contains the following chapters:

- *Chapter 1, The Role of Cryptography in the Connected World*
- *Chapter 2, Secure Channel and the CIA Triad*
- *Chapter 3, A Secret to Share*
- *Chapter 4, Encryption and Decryption*

- *Chapter 5, Entity Authentication*
- *Chapter 6, Transport Layer Security at a Glance*

1

The Role of Cryptography in the Connected World

In this introductory chapter, we try to provide some answers to the following questions:

- Why are there so many insecure IT systems?
- How can cryptography help to mitigate our security problems?

Our core argument is that the simultaneous growth of connectivity and complexity of IT systems has led to an explosion of the attack surface, and that modern cryptography plays an important role in reducing that attack surface.

After a brief discussion of how the field of cryptography evolved from an exotic field appreciated by a select few to an absolutely critical skill for the design and operation of nearly every modern IT product, we will look at some recent real-world security incidents and attacks in order to illustrate our claim. This will allow you to understand why cryptography matters from a higher strategic perspective.

1.1 Evolution of cryptography

Over the past four decades or so, cryptography has evolved from an exotic field known to a select few into a fundamental skill for the design and operation of modern IT systems. Today, nearly every modern product, from the bank card in your pocket to the server farm running your favorite cloud services, requires some form of cryptography to protect it and its users against cyberattacks. Consequently, it has found its way into mainstream computer science and software engineering.

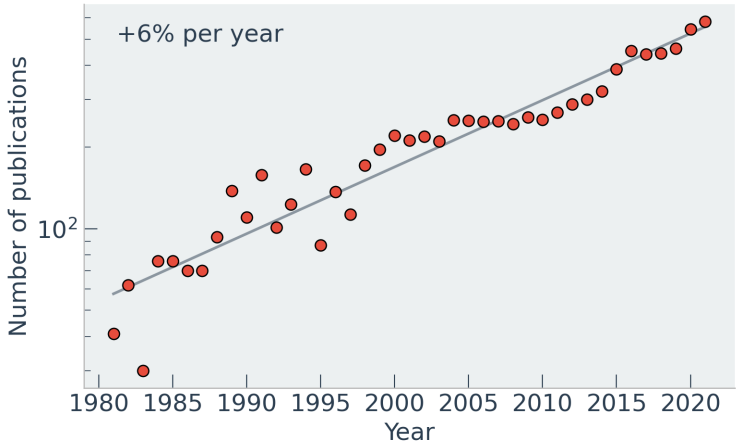

Figure 1.1: Number of publications at IACR conferences on cryptology over the years

Cryptography and its counterpart cryptanalysis were basically unknown outside of military and intelligence services until the mid 1970s. According to [172], *Cryptography is the practice and study of techniques for secure communication in the presence of adversaries*; it deals with the development and application of cryptographic mechanisms. Cryptanalysis is the study of cryptographic mechanisms' weaknesses, aimed at finding mathematical ways to render these mechanisms ineffective. Taken together, cryptography and cryptanalysis form what's called cryptology.

In 1967, David Kahn, an American historian, journalist, and writer, published a book titled *The Codebreakers – The Story of Secret Writing*, which is considered to be the first extensive treatment and a comprehensive report of the history of cryptography and military

intelligence from ancient Egypt to modern times [93]. Kahn's book introduced cryptology to a broader audience. Its content was, however, necessarily restricted to *symmetric cryptography*. In symmetric cryptography, the sender and receiver of a message share a common secret key and use it for both encrypting and decrypting. The problem of how sender and receiver should exchange the secret in a secure way was considered out of scope.

This changed in 1976, when the seminal paper *New Directions in Cryptography* by Whitfield Diffie and Martin Hellman appeared in volume IT-22 of IEEE Transactions on Information Security [49]. In that publication, Diffie and Hellman described a novel method for securely agreeing on a secret key over a public channel based on the so-called *discrete logarithm problem*. Moreover, they suggested for the first time that the sender and receiver might use different keys for encrypting (the *public key*) and decrypting (the *private key*) and thereby invented the field of *asymmetric cryptography*.

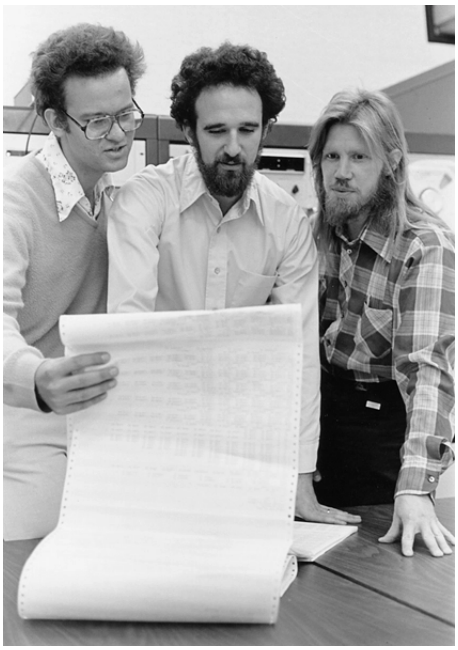

Figure 1.2: From left to right: Ralph Merkle, Martin Hellman, Whitfield Diffie [69]

While there were scientific works on cryptography dating back to the early 1970s, the publication by Diffie and Hellman is the first publicly available paper in which the use of a private key and a corresponding public key is proposed. This paper is considered to be the start of cryptography in the public domain. In 2002, Diffie and Hellman suggested their algorithm should be called Diffie-Hellman-Merkle key exchange because of Ralph Merkle's significant contribution to the invention of asymmetric cryptography [185].

In 1977, the three MIT mathematicians Ron Rivest, Adi Shamir, and Len Adleman took up the suggestion by Diffie and Hellman and published the first asymmetric encryption algorithm, the RSA algorithm [151], which is based on yet another well-known mathematical problem, the factoring problem for large integers.

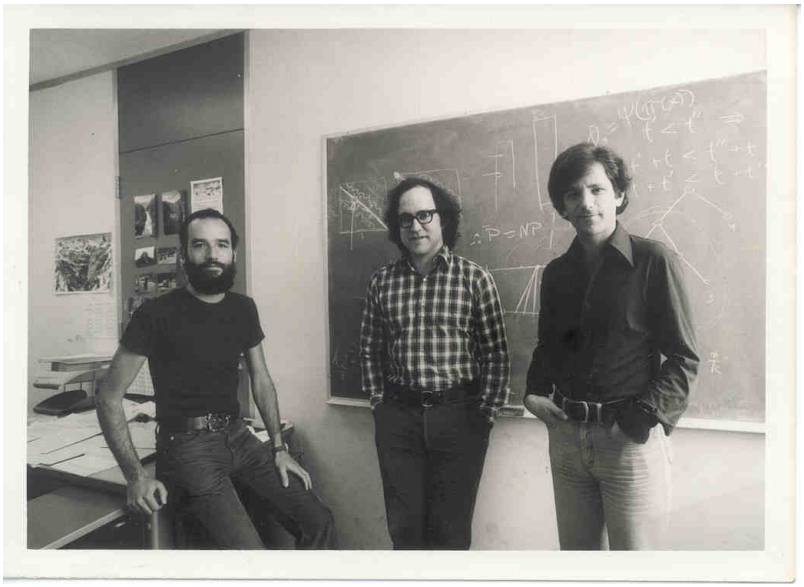

Figure 1.3: From left to right: Adi Shamir, Ron Rivest, Len Adleman [152]

The invention of asymmetric cryptography did not make symmetric cryptography obsolete. On the contrary, both fields have complementary strengths and weaknesses and can be efficiently combined in what is today called *hybrid cryptosystems*. The **Transport Layer Security** (**TLS**) protocol is a very good example of a hybrid cryptosystem.

Today, cryptography is a well-known (albeit mostly little understood in depth) topic in the IT community and an integral part of software development. As an example, as of July 2022, the OpenSSL library repository on GitHub contains over 31,500 commits by 686 contributors. Cryptography is also an integral part of numerous computer science and information security curricula, and numerous universities all over the world offer degrees in information security.

Why did this happen, and which factors led to this development and popularized cryptography within a comparably short period of time? To a large extent, this paradigm shift is a result of three—arguably still ongoing—developments in information technology that radically changed the role of cryptography in the modern connected world:

- The advent of the internet and the ever increasing need to transfer large amounts of data over untrusted channels, which also fostered the development of TLS
- The introduction of connectivity into nearly every new product, from toothbrushes to automobiles
- The ever increasing complexity of IT systems, specifically increasing hardware and software complexity

We will now discuss each of these factors in turn.

1.2 The advent of TLS and the internet

We'll now turn to the original theme of this book, TLS and the cryptographic tools it is made of. TLS is a protocol designed to protect data sent over the internet, so we'll start with a brief look into the early history of the internet.

Despite its origins as a research project financed by the **Defense Advanced Research Projects Agency** (**DARPA**), the research agency of the Department of Defence of the United States, most of the main physical components of the internet, such as cables, routers, gateways, and so on, can be (and are) accessed by untrusted third parties. In the early days of the internet, this was not considered a problem, and very few (if any) security measures were introduced into TCP and IP, the internet's main protocol workhorses, and none of

them involved cryptography. However, with more and more people using the internet, and the ever increasing available bandwidth, more and more services kept appearing on the internet, and it was quickly realized that to do real business over the internet, a certain amount of trust was needed that sensitive data such as credit card numbers or passwords did not fall into the wrong hands. Cryptography provides the answer to this problem, because it can guarantee **confidentiality** (i.e., no one can read the data in transit) and **authenticity** (i.e., you can verify that you are talking to the right party). TLS and its predecessor SSL are the protocols that implement cryptography on the internet in a secure, usable way.

Starting in 1995, SSL was shipped together with Netscape Navigator to clients. While server-side adoption of SSL was slow at first, by the end of 2021, according to the **Internet Security Research Group** (**ISRG**), 83% of web pages loaded by Firefox globally used HTTPS, that is HTTP secured via TLS [87].

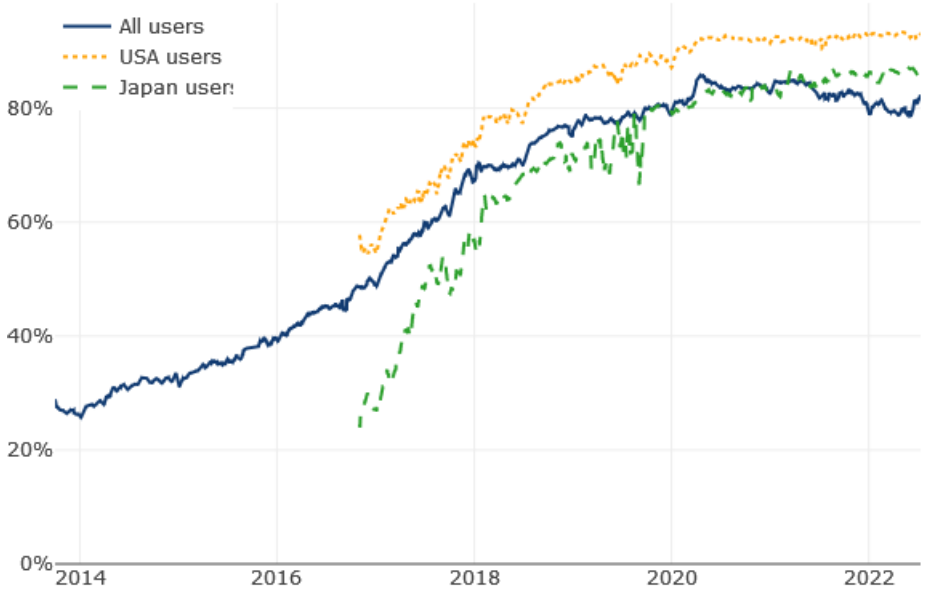

Figure 1.4: Percentage of web pages loaded by Firefox using HTTPS [88]

This is a huge success for TLS and the field of cryptography in general, but with it also comes a huge responsibility: we need to constantly monitor whether the algorithms, key lengths, modes of operations, and so on used within TLS are still secure. Moreover, we need to understand how secure algorithms work and how they can interact with each other in a secure way so that we can design secure alternatives if needed.

Maybe we should already stress at this early stage that TLS is not a remedy for all the problems mentioned here. TLS provides *channel-based security*, meaning that it can only protect data in transit between a client and a server. TLS is very successful in doing so, and how *in detail* TLS uses cryptography to achieve this goal is the main theme of this book. However, once the data leaves the secure channel, it is up to the endpoints (i.e., client and server) to protect it.

Moreover, cryptography by itself is useless in isolation. To have any practical effect, it has to be integrated into a much larger system. And to ensure that cryptography is effectively protecting that system, there must be no security holes left that would allow an attacker to circumvent its security.

There is a well-known saying among cybersecurity professionals that the security of a system is only as strong as its weakest link. Because there are so many ways to circumvent security – especially in complex systems – cryptography, or rather the cryptographic primitives a system uses, is rarely the weakest link in the chain.

There is, however, one important reason why cryptography is fundamental for the security of information systems, even if there are other security flaws and vulnerabilities. An attacker who is able to break cryptography cannot be detected because a cryptanalytic attack, that is, the breaking of a cryptographic protocol, mechanism or primitive, in most cases leaves no traces of the attack.

If the attacker's goal is to read the communication, they can simply passively listen to the communication, record the messages and decrypt them later. If the attacker's goal is to manipulate the target system, they can simply forge arbitrary messages and the system will never be able to distinguish these messages from benign ones sent by legitimate users.

While there are many other sources of insecurity (e.g., software bugs, hardware bugs, and social engineering), the first line of defense is arguably secure communication, which in itself requires a secure channel. And cryptography as a scientific discipline provides the building blocks, methods, protocols, and mechanisms needed to realise secure communication.

1.3 Increasing connectivity

Connectivity allows designers to add novel, unique features to their products and enables new business models with huge revenue potential that simply would not exist without it.

At the same time, connectivity makes it much harder to build secure systems. Similar to Ferguson and Schneier's argument on security implications of complexity, one can say that there are no connected systems that are secure. Why? Because connecting systems to large, open networks like the internet exposes them to remote attacks. Remote attacks – unlike attacks that require physical access – are much more compelling from the attacker's perspective because they *scale*.

1.3.1 Connectivity versus security – larger attack surface

While connectivity enables a multitude of desired features, it also exposes products to remote attacks carried out via the internet. Attacks that require physical access to the target device can only be executed by a limited number of attackers who actually have access to that device, for example, employees of a company in the case of devices in a corporate network. In addition, the need for physical access generally limits the attacker's window of opportunity.

Connectivity, in contrast, exposes electronic devices and IT systems to remote attacks, leading to a much higher number of potential attackers and threat actors. Moreover, remote attacks – unlike attacks that require physical access to the target – are much more compelling from the attacker's perspective because they scale.

Another aspect that makes remote attacks practical (and, to a certain extent, rather easy) is the fact that the initial targets are almost always the network-facing interfaces of the devices, which are implemented in software. As we have seen, complex software is almost

guaranteed to contain numerous implementation bugs, a number of which can be typically exploited to attack the system. Thus, the trend of increasing software and system complexity inadvertently facilitates remote attacks.

1.3.2 Connectivity versus marginal attack cost

Remote attacks are easy to launch – and hard to defend against – because their marginal cost is essentially zero. After a newly discovered security vulnerability is initially translated into a reliably working exploit, the cost of replicating the attack an additional 10, 100, or 100,000 devices is essentially the same, namely close to zero.

This is because remote attacks are implemented purely in software, and reproducing software as well as accessing devices over public networks effectively costs close to nothing. So, while businesses need to operate large – and costly – internal security organizations to protect their infrastructure, services, and products against cybersecurity attacks, any script kiddie can try to launch a remote attack on a connected product, online service, or corporate infrastructure essentially for free.

1.3.3 Connectivity versus scaling attacks

To summarize, connectivity exposes devices and IT systems to remote attacks that target network-facing software (and, thus, directly benefit from the continuously increasing software complexity), are very cheap to launch, can be launched by a large number of threat actors, and have zero marginal cost.

In addition, there exists a market for zero-day exploits [190] that allows even script kiddies to launch highly sophisticated remote attacks that infest target systems with advanced malware able to open a remote shell and completely take over the infested device.

As a result, connectivity creates an attack surface that facilitates cybersecurity attacks that scale.

1.4 Increasing complexity

While it can be argued that the problem of increasing complexity is not directly mitigated by modern cryptography (in fact, many crypto-related products and standards suffer from this problem themselves), there is no doubt that increasing complexity is in fact a major cause of security problems. We included the complexity problem in our list of crucial factors for the development of cryptography, because cryptography *can help limit the damage* caused by attacks that were in turn caused by excessive complexity.

Following Moore's law [191], a prediction made by the co-founder of Fairchild Semiconductor and Intel Gordon Moore in 1965, the number of transistors in an integrated circuit, particularly in a microprocessor, kept doubling roughly every 2 years (see *Figure 1.5*).

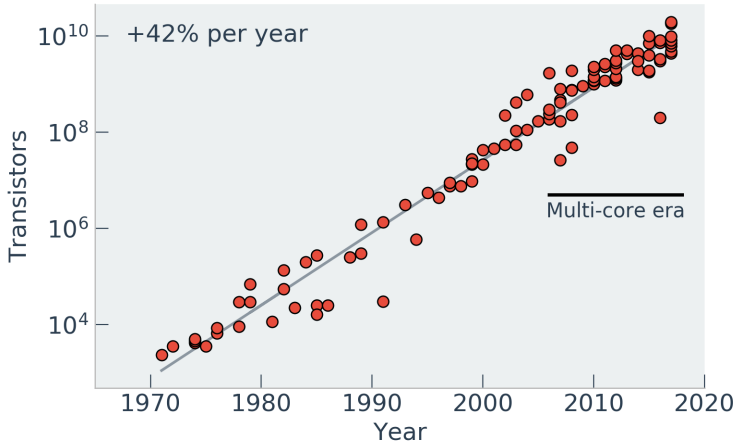

Figure 1.5: Increasing complexity of hardware: Transistors. Data is taken from https://github.com/barentsen/tech-progress-data

Semiconductor manufacturers were able to build ever bigger and ever more complex hardware with ever more features. This went so far that in the late 1990s, the Semiconductor Industry Association set off an alarm in the industry when it warned that productivity gains in **Integrated Circuit** (**IC**) manufacturing were growing faster than the capabilities of **Electronic Design Automation** (**EDA**) tools used for IC design. Entire companies in the EDA area were successfully built on this premise.

Continuously growing hardware resources paved the way for ever more complex software with ever more functionality. Operating systems became ever more powerful and feature-rich, the number of layers in software stacks kept increasing, and software libraries and frameworks used by programmers became ever more comprehensive. As predicted by a series of software evolution laws formulated by early-day computer scientists Manny Lehman and Les Belady, software exhibited continuing growth and increasing complexity [181] (see also *Figure 1.6*).

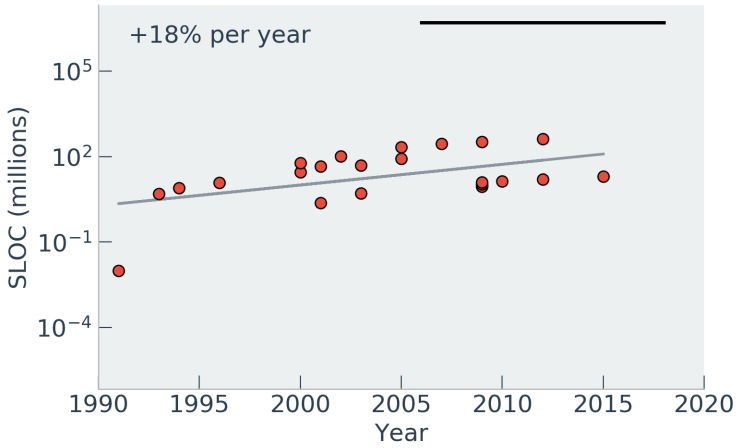

Figure 1.6: Increasing complexity of software: Source Lines of Code (SLOC) in operating systems. Data is taken from https://github.com/barentsen/tech-progress-data

Why should increasing complexity be a problem? According to leading cybersecurity experts Bruce Schneier and Niels Ferguson [65], *"Complexity is the worst enemy of security, and it almost always comes in the form of features or options"*.

While it might be argued whether complexity really is the *worst* enemy of security, it is certainly true that complex systems, whether realized in hardware or software, tend to be error-prone. Schneier and Ferguson even claim that *there are no complex systems that are secure.*

Complexity negatively affects security in several ways, including the following:

- Insufficient testability due to a combinatorial explosion given a large number of features
- Unanticipated—and unwanted—behavior that emerges from a complex interplay of individual features
- A high number of implementation bugs and, potentially, architectural flaws due to the sheer size of a system

1.4.1 Complexity versus security – features

The following thought experiment illustrates why complexity arising from the number of features or options is a major security risk. Imagine an IT system, say a small web server, whose configuration consists of 30 binary parameters (that is, each parameter has only two possible values, such as on or off). Such a system has more than a *billion* possible configurations. To guarantee that the system is secure under all configurations, its developers would need to write and run several billion tests: one test for each relevant type of attack (e.g., Denial-of-Service, cross-site scripting, and directory traversal) and each configuration. This is impossible in practice, especially because software changes over time, with new features being added and existing features being refactored. Moreover, real-world IT systems have significantly more than 30 binary parameters. As an example, the NGINX web server has nearly 800 directives for configuring how the NGINX worker processes handle connections.

1.4.2 Complexity versus security – emergent behavior

A related phenomenon that creates security risks in complex systems is the unanticipated *emergent behavior*. Complex systems tend to have properties that their parts do not have on their own, that is, properties or behaviors that emerge only when the parts interact [186]. Prime examples for security vulnerabilities arising from emergent behavior are **time-of-check-to-time-of-use** (**TOCTOU**) attacks exploiting concurrency failures, replay attacks on cryptographic protocols where an attacker reuses an out-of-date message,

and side-channel attacks exploiting unintended interplay between micro-architectural features for speculative execution.

1.4.3 Complexity versus security – bugs

Currently available software engineering processes, methods, and tools do not guarantee error-free software. Various studies on software quality indicate that, on average, 1,000 lines of source code contain 30-80 bugs [174]. In rare cases, examples of extensively tested software were reported that contain 0.5-3 bugs per 1,000 lines of code [125].

However, even a rate of 0.5-3 bugs per 1,000 lines of code is far from sufficient for most practical software systems. As an example, the Linux kernel 5.11, released in 2021, has around 30 million lines of code, roughly 14% of which are considered the "core" part (`arch`, `kernel`, and `mm` directories). Consequently, even with extensive testing and validation, the Linux 5.11 core code alone would contain approximately 2,100-12,600 bugs.

And this is only the operating system core without any applications. As of July 2022, the popular Apache HTTP server consists of about 1.5 million lines of code. So, even assuming the low rate of 0.5-3 bugs per 1,000 lines of code, adding a web server to the core system would account for another 750-4,500 bugs.

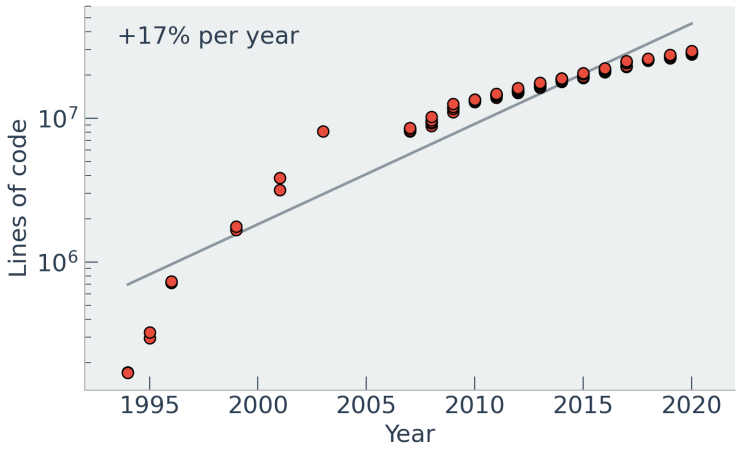

Figure 1.7: Increase of Linux kernel size over the years

What is even more concerning is the rate of bugs doesn't seem to improve significantly enough over time to cope with the increasing software size. The extensively tested software having 0.5-3 bugs per 1,000 lines of code mentioned above was reported by Myers in 1986 [125]. On the other hand, a study performed by Carnegie Mellon University's CyLab institute in 2004 identified 0.17 bugs per 1,000 lines of code in the Linux 2.6 kernel, a total of 985 bugs, of which 627 were in critical parts of the kernel. This amounts to slightly more than halving the bug rate at best – over almost 20 years.

Clearly, in that same period of time from 1986 to 2004 the size of typical software has more than doubled. As an example, Linux version 1.0, released in 1994, had about 170,000 lines of code. In comparison, Linux kernel 2.6, which was released in 2003, already had 8.1 million lines of code. This is approximately a 47-fold increase in size within less than a decade.

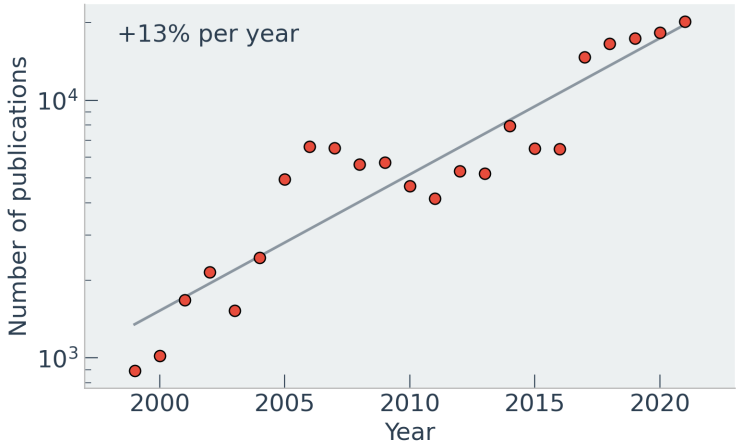

Figure 1.8: Reported security vulnerabilities per year

1.5 Example attacks

The combination of these two trends – increase in complexity and increase in connectivity – results in an attack surface explosion. The following examples shall serve to illustrate this point.

1.5.1 The Mirai botnet

In late 2016, the internet was hit by a series of massive **Distributed Denial-of-Service** (**DDoS**) attacks originating from the Mirai botnet, a large collection of infected devices (so-called *bots*) remote-controlled by attackers.

The early history of the Mirai botnet can be found in [9]: the first bootstrap scan on August 1 lasted about two hours and infected 834 devices. This initial population continued to scan for new members and within 20 hours, another 64,500 devices were added to the botnet. The infection campaign continued in September, when about 300,000 devices were infected, and reached its peak of 600,000 bots by the end of November. This corresponds to a rate of 2.2-3.4 infected devices per minute or 17.6-27.2 seconds to infect a single device.

Now contrast this with a side-channel or fault attack. Even if we assume that the actual attack – that is, the measurement and processing of the side-channel traces or the injection of a fault – can be carried out in *zero* time, an attacker would still need time to gain physical access to each target. Now suppose that, on average, the attacker needs one hour to physically access a target (actually, this is a very optimistic assumption from the attacker's perspective, given that the targets are distributed throughout the globe). In that case, attacking 200,000-300,000 devices would take approximately 22-33 *years* or 270 to 400 months (as opposed to 2 months in the case of Mirai).

Moreover, any remote attack starts at a network interface of the target system. So the first (and, oftentimes, the only) thing the attacker interacts with is software. But software is complex by nature.

1.5.2 Operation Aurora

In mid-December 2009, Google discovered a highly sophisticated, targeted attack on their corporate infrastructure that resulted in intellectual property theft [73]. During their investigation, Google discovered that at least 20 other large companies from a wide range of businesses had been targeted in a similar way [193].

This series of cyberattacks came to be known as Operation Aurora [193] and were attributed to APT groups based in China. The name was coined by McAfee Labs security researchers based on their discovery that the word *Aurora* could be found in a file on the attacker's machine that was later included in malicious binaries used in the attack as part of a file path. Typically, such a file path is inserted by the compiler into the binary to indicate where debug symbols and source code can be found on the developer's machine. McAfee Labs therefore hypothesized that Aurora could be the name of the operation used by the attackers [179].

According to McAfee, the main target of the attack was source code repositories at high-tech, security, and defense contractor companies. If these repositories were modified in a malicious way, the attack could be spread further to their client companies. Operation Aurora can therefore be considered the first major attack on software supply chains [193].

In response to Aurora, Google shut down its operations in China four months after the incident and migrated away from a purely *perimeter-based defense principle*. This means devices are not trusted by default anymore, even if they are located within a corporate LAN [198].

1.5.3 The Jeep hack

At the BlackHat 2015 conference, security researchers Charlie Miller and Chris Valasek demonstrated the first remote attack on an unaltered, factory passenger car [120]. In what later became known as the Jeep hack, the researchers demonstrated how the vehicle's infotainment system, Uconnect, which has both remote connectivity as well as the capability to communicate with other electronic control units within the vehicle, can be used for remote attacks.

Specifically, while systematically examining the vehicle's attack surface, the researchers discovered an open D-Bus over an IP port on Uconnect, which is essentially an inter-process communication and remote procedure call mechanism. The D-Bus service accessible via the open port allows anyone connected to the infotainment system to execute arbitrary code in an unauthenticated manner.

Miller and Valasek also discovered that the D-Bus port was bound to all network interfaces on the vulnerable Uconnect infotainment system and was therefore accessible remotely over the Sprint mobile network that Uconnect uses for telematics. By connecting to the Sprint network using a femtocell or simply a regular mobile phone, the researchers were able to send remote commands to the vehicle.

From that entry point, Miller and Valasek attacked a chip in the vehicle's infotainment system by re-writing its firmware to be able to send arbitrary commands over the vehicle's internal CAN communication network, effectively giving them the ability to completely take over the vehicle.

1.5.4 Commonalities

What do these examples have in common and how does it relate to cryptography? In a nutshell, these examples illustrate what happens in the absence of appropriate cryptography. In all three cases discussed, there was no mechanism in place to verify that the systems were talking to legitimate users and that the messages received were not manipulated while in transit.

In the Mirai example, anyone with knowledge of the IoT devices' IP addresses would have been able to access their login page. This information can be easily collected by scanning the public internet with tools such as `nmap`. So the designers' assumption that the users would change the default device password to a strong individual one was the only line of defense. What the security engineers should have done instead is to add a cryptographic mechanism to give access to the login procedure only to legitimate users, for example, users in possession of a digital certificate or a private key.

In the case of Operation Aurora, the perimeter defense doctrine used by the affected companies treated every device within the trusted perimeter (typically, within a corporate network) as trustworthy by default. On this premise, every device inside the perimeter had access to all resources and systems within that perimeter.

As a result, anyone able to walk inside a company building or trick an arbitrary employee into clicking on a malicious link and infect their computer with malware would have been able to access all systems within the perimeter.

As a response to Operation Aurora, Google and other companies replaced perimeter defense with a zero trust security model that establishes trust by evaluating it on a per-transaction basis instead of basing trust on the network location (the perimeter) [155]. At the core of the zero trust security model is the ability to securely authenticate users and resources in order to prevent unauthorized access to data and services. Secure authentication, in turn, is built upon cryptography.

Finally, in the Jeep hack example, the open D-Bus over IP port allowed anyone connected to the vehicle's infotainment system to execute arbitrary code in an unauthenticated manner. The possibility to access the vehicle remotely over the Sprint mobile network further increased the range of the attack. The system's designers apparently assumed that the Sprint mobile network is a secure perimeter. What they should have done instead is to add a cryptographic mechanism to ensure that only legitimate users could log in to the Uconnect system.

1.6 Summary

In this chapter, we have provided an overview of the recent history of cryptography, starting in the 1970s, and identified some global trends that explain why cryptography has become more and more important over the last few decades, to a point where it is practically around you every time you access the internet or use a connected device. In the next chapter, you will learn about the general goals and objectives you can achieve with the help of cryptography. In particular, you will get to know cryptography's main protagonists, Alice and Bob, and their ubiquitous opponents, Eve and Mallory.

2

Secure Channel and the CIA Triad

In this chapter, we discuss the fundamental objective of cryptography and computer security, namely enabling two parties to communicate securely over an insecure communication channel. As we will see shortly, this is not an easy task to accomplish because the communication needs to be secure against both passive and active attackers.

But how can we achieve security if the attacker is allowed to listen to the entire communication and even manipulate the messages sent over the channel? And what are the fundamental design principles that we must follow to build systems that can protect that communication?

To answer these questions, we will cover the most important cryptographic definitions, essential design principles, and central cryptographic goals. We will show how these goals can be achieved in principle, leaving the technical details for the chapters to follow. Finally, we will introduce the notion of a *secure channel*, which lies at the heart of TLS.

2.1 Technical requirements

This chapter introduces basic definitions, design principles, and goals and therefore requires no specific software or hardware.

2.2 Preliminaries

The fundamental objective of cryptography and computer security in general is to enable two persons, let's call them Alice and Bob, to communicate over an insecure channel so that an opponent, commonly called Eve, cannot understand or unnoticeably alter their messages [168]. Alice, Bob, and Eve can also be referred to as (communicating) *entities* or *parties* and may be people or machines. Alice and Bob are either a *sender* or *receiver*, that is, a legitimate transmitter or intended recipient, of the messages. Eve is an *adversary*, an entity that tries to compromise the information security between Alice and Bob.

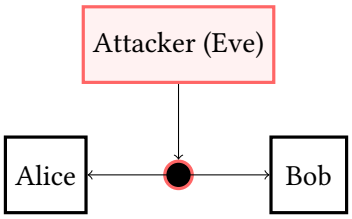

Figure 2.1: Legitimate communicating parties Alice and Bob transmit and receive information over an untrusted channel

Let's clarify terms by looking at some definitions:

- An **insecure** or **untrusted channel** is a channel an opponent, or **attacker**, can access. The capabilities of the attacker (that is, what the attacker can or cannot do on the channel) may vary and are part of the so-called **attacker model**.
- A **passive attacker** is an attacker who is only able to read information from an untrusted channel. In contrast, an **active attacker** is an attacker who can also insert their own information, or alter, replay, reorder, or delete the information on an untrusted channel.

Figure 2.2 illustrates the two fundamental attacker models in cryptography. Cryptographers use the attacker model to express the assumptions about the *power of the adversary*, that is, the actions the attacker is assumed to be able to take as well as their computational power.

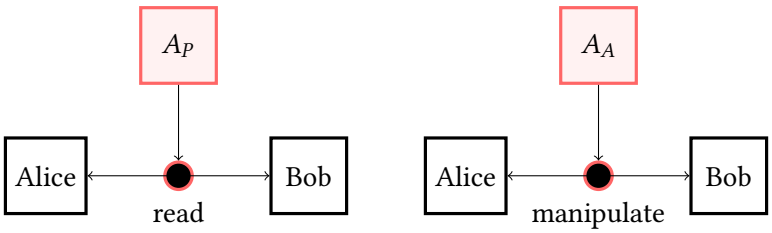

Figure 2.2: The two fundamental attacker models used in cryptography

The channel might be anything from a telephone line to a Wi-Fi network or satellite communication, and the information being sent from Alice to Bob can have an arbitrary structure – it might be numerical data, text, or simply some random blob. In computer science, blob, or binary large object, refers to a collection of binary data such as images, audio, or multimedia objects, or binary executables.

In numerous real-world applications, machine-to-machine or application-to-application communication needs to be secured. In this case, Alice could be equivalent to a web browser and Bob to a web server. Alternatively, Alice could be equivalent to an embedded device that is connected to the internet and Bob could be equivalent to a backend service the device is connected to.

Alice and Bob might even be the same person or entity, for instance, an electronic device. In this case, Alice is transmitting the information to her future self, that is, she wants to store the information securely.

Sometimes, Alice and Bob might engage in something called a *cryptographic protocol*, for example, to establish a key in a secure way, or in order to make sure they are really communicating with the correct entity. Here are precise definitions:

- A **cryptographic protocol** is a distributed algorithm defined by a sequence of steps that precisely specifies the actions required of two or more entities to achieve a

specific cryptographic goal [117]. Transport Layer Security, one of the main topics of this book, is a cryptographic protocol for establishing a secure channel between Alice and Bob.

- In contrast, a **cryptographic mechanism** is a more general term referring to protocols, algorithms – specifying the steps followed by a single entity – and non-cryptographic techniques [117] such as trusted execution environments and role-based access control to achieve specific security objectives.

So, cryptographic mechanisms (and protocols among them) are tools with which Alice and Bob seek to achieve security objectives. But what precisely are these security objectives? There are several of them, and in the next three sections, we will explore the three most important ones.

2.3 Confidentiality

A fundamental need in secure communication is to ensure the privacy of the information transmitted between the communicating parties. In cryptography, this is referred to as *confidentiality* of data.

More precisely, *confidentiality is the ability to keep the content of information from all but those authorized to have it* [117]. Confidentiality therefore guarantees that no one except the sender and the intended receiver(s) of the information can read it.

In the classical scenario illustrated in *Figure 2.1*, these are two parties (Alice and Bob). In general, cryptographic mechanisms can be used to ensure confidentiality for any number of parties, for example, using the concept of group keys. In the simplest case, there is only one party involved, for example, a user encrypting their private data stored in the cloud or on a personal device such as a smartphone.

Figure 2.3 shows the classical scenario for confidentiality. The eavesdropper Eve has the capability r to read all messages sent between Alice and Bob over an insecure channel. Eve's goal is to recover the original plaintext from the messages transmitted over the channel (within a reasonable period of time).

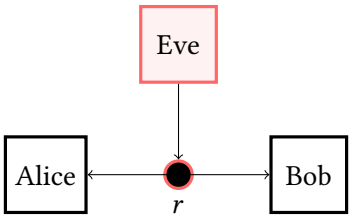

Figure 2.3: Alice and Bob must keep their communication confidential in the presence of Eve

There are many different ways to achieve confidentiality, such as physically protecting access to the media on which the private information is stored or exchanging private information in a way that both parties believe to be secure by design, such as in a private, face-to-face meeting. In contrast, cryptography focuses on mathematical algorithms – based on number-theoretic problems or the concept of pseudo-randomness – that *scramble* the message, before it is sent, in such a way that the private information becomes unintelligible for unauthorized parties.

An implicit assumption is that the communication channel between the legitimate parties is insecure, that is, that an eavesdropping adversary sees all *scrambled* messages and can store them for later processing. A cryptographic mechanism ensuring confidentiality must hold even if the messages observed by the eavesdropper are used in an offline attack.

If the goal of a cryptographic mechanism is to provide confidentiality, then a *passive* attacker is typically assumed. Recall from the last section that a passive attacker is supposed only to be able to read information from an unsecured channel [117]. While manipulating messages sent by Alice and Bob might interrupt the overall system – for example, lead to a denial-of-service attack if the legitimate parties are interpreting these messages to accomplish some kind of a computational or control task – this usually won't help the attacker to recover the original plaintext, so confidentiality will be preserved.

In cryptography, confidentiality is achieved using mathematical functions that transform the private information m, also referred to as *plaintext*, into a scrambled message $c = f_K(m)$, referred to as *ciphertext*. These functions are *bijective* meaning that for every such function f_K there is an inverse function, denoted by f_K^{-1}, which transforms the ciphertext back to

the original plaintext. In other words, an algorithm mapping plaintext to ciphertext in such a way that the plaintext cannot be recovered does *not* provide confidentiality in a cryptographic sense. Note that bijectivity also implies that there are as many plaintexts as ciphertexts.

The infamous **Wired Equivalent Privacy** (**WEP**) hack illustrates what consequences a confidentiality breach can have in practice. WEP was a cryptographic algorithm that provided confidentiality for over-the-air transmissions in **Wireless Local Area Networks** (**WLANs**) defined in the IEEE 802.11 set of standards. Released in 1997 as part of the original 802.11 standard, WEP was recommended for all 802.11b- and 802.11g-compliant devices and included in most Wi-Fi routers by the late 1990s [52].

WEP was designed as a so-called *stream cipher* based on the RC4 algorithm as a key stream generator (see *Section 4.5.1* and *Section 4.5.2*, respectively). Because interference can cause bit errors and, consequently, packet loss in a wireless channel, the WEP key stream generation was restarted each time a frame of a message had been transmitted. This restart procedure relied on a so-called **Initialization Vector** (**IV**), basically a 24-bit string. An attacker could therefore wait until an IV repeated after 2^{24} frames, which led to the repetition of the entire key stream. As is explained in *Section 4.3.2*, this is a serious security problem, especially if parts of the plaintext are already known, as is often the case with standardized network protocols.

In 2001, researchers published a complete cryptanalysis of WEP that recovered the secret key after eavesdropping on the wireless network [66]. Depending on the number of network packets available for inspection, the complete key could be recovered in as little as one minute [197].

2.4 Integrity

Integrity is the ability to **detect** data manipulation by unauthorized entities. By data manipulation, we mean unauthorized acts such as the insertion, deletion, or substitution of individual data chunks or entire messages. It is not required that manipulations as such are rendered impossible; given the multitude of possible communication channels, this

would be an impossible task. Clearly, a passively eavesdropping attacker such as Eve does not have the capability to perform data manipulation. We, therefore, assume a more active attacker named *Mallory* who also has the capability to write on the communication channel (see *Figure 2.4*).

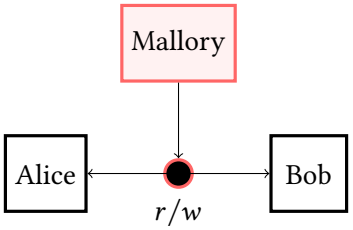

Figure 2.4: The malicious attacker Mallory has the capabilities r, w to read all messages sent between Alice and Bob and to write on the unsecured channel. Mallory's goal is to manipulate the messages in a way that is not noticed by Alice or Bob

At first glance, one might be tempted to think that a good encryption function should be able to provide both confidentiality and integrity when applied to plaintext. After all, if Mallory cannot see the plaintext and has to blindly manipulate the ciphertext c coming from Alice, this should be noticed by Bob, as decryption will fail to produce meaningful plaintext. While this is true in most cases, there are certain ways to encrypt where Mallory has a fairly good chance of getting away with her manipulations, for example, *stream ciphers* or the *Electronic Codebook Mode* (see *Chapter 4, Encryption and Decryption* and *Chapter 14, Block Ciphers and Their Mode of Operation*, respectively).

The real reason for keeping confidentiality and integrity separate from security services using different mechanisms goes deeper, however. There are many situations where integrity is needed but confidentiality is either not needed or even not wanted. A good example is provided by the header of an IP packet. Among other data fields, it contains the IP address of the recipient of the packet. While it is a good idea to provide integrity protection of the IP header (the IPSec protocol does just that), confidentiality would be counterproductive since intermediate gateways need to see the recipient's address to route the packet.

Attacks on the integrity of data can have severe consequences if they go unnoticed: important decisions might be based on wrong information, software systems might exhibit strange, inexplicable errors, and hardware-based systems could be physically damaged. A recent example of a data integrity attack is provided by the Stuxnet worm [106], allegedly composed in a joint effort by the NSA and the Israeli secret service Mossad to harm the Iranian nuclear program. Building upon a complex series of different attack vectors, Stuxnet finally managed to get access to the control software for centrifuges used to enrich uranium in Iranian nuclear facilities. By altering the control parameters, the centrifuges were made to run at a slightly higher speed than allowed and were rendered useless after a few months. Because of the long-term character of the attack, the real reason for the centrifuge failure was not detected for quite a long time.

Integrity can be achieved in a manner that is similar to detecting transmission errors in networking; basically, a cryptographic checksum h_K is computed over the data m that Alice wants to protect and appended to her message m. If Bob receives a message $(m, h_K(m))$, he can compute a checksum of his own over m. If this checksum does not equal $h_K(m)$, something has gone wrong: either there were transmission errors or someone has tampered with m.

Again, a secret parameter k has to go into the computation of the checksum. Otherwise, an intelligent attacker would be able to modify m to $\tilde{m}$ and alter the checksum so that it fits the modified $\tilde{m}$. Moreover, a checksum designed to detect transmission errors such as the **Cyclic Redundancy Check** (**CRC**) in networking will not be enough, as they can be easily forged even without knowing the secret k. Instead, one has to use secure hash functions, as discussed in *Chapter 11, Hash Functions and Message Authentication Codes*.

2.5 Authentication

Authentication is the ability to identify the source of the communication, both for the communicating parties and for the information itself. In other words, authentication refers to a cryptographic mechanism ensuring that the identity of communicating entities can be verified and that the source of a received message can be verified. Any two parties

entering into a secure communication should authenticate each other and the data received with respect to their origin. This hints at the fact that there are actually two kinds of authentication: one to verify identities (*entity authentication*) and another to verify data origin (*message authentication*).

Authentication is one of the most important security goals in cryptography. After hash functions and digital signatures were discovered, authentication and confidentiality were classified as independent information security objectives [117]. Without authentication, however, there can be no genuine confidentiality because you can never be sure who you are talking to, even if the communication is in encrypted form. Today, confidentiality and authentication are often combined in *authenticated encryption schemes* (see *Chapter 15, Authenticated Encryption*).

It might even seem superfluous to differentiate between authentication and confidentiality. In practice, however, implementing one or the other can have fundamental implications on legal matters.

As an example, export control (legislation regulating the export of goods, software, and technology) restricts the export of items considered potentially harmful to the interest of the exporting country. Such items include arms, so-called dual-use goods with military potential, radioactive materials such as uranium, and cryptography.

In the case of cryptography, it is prohibited to export hardware or software that can be used for strong encryption to export controlled or sanctioned countries, entities, and persons. Strong encryption refers to encryption algorithms (see *Chapter 4, Encryption and Decryption*) deemed to be secure by national agencies and standardization bodies such as GCHQ or NIST. If the actual goal of a security system is to authenticate individual entities, such as a sensor in a car and an electronic control unit that uses the measurement data from that sensor, it might be more practical to use a cryptographic mechanism for authentication rather than encryption.

Another example where the separation of confidentiality and authentication makes sense is provided by two communicating parties located in different countries where one or

both of the countries do not permit confidentiality in order to monitor all communications. While the legitimate parties are not allowed to use encryption, a mechanism for achieving confidentiality, they can still use a cryptographic algorithm to ensure the identity of each party as well as the origin of the information both parties receive.

2.5.1 Message authentication

Message authentication is the ability of the communicating party that receives a message to verify – through corroborative evidence – the identity of the party that originated the message [117]. This form of authentication is also referred to as **data origin authentication**.

Message authentication can be achieved by providing additional information together with the message. This information can be used by the receiving party to verify the identity of the party who sent the message – at least, this is true for asymmetric authentication protocols. In symmetric message authentication, we can only verify that a message was sent by someone in possession of the shared key (see also *Chapter 5, Entity Authentication*).

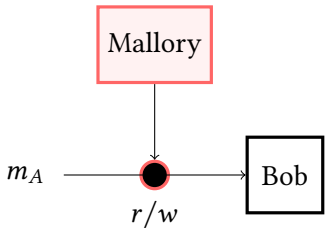

Figure 2.5: Message authentication allows Bob to verify that the message m_A he receives has indeed originated from Alice, despite malicious Mallory having the ability to read Alice's messages and write arbitrary messages to the unsecured channel

While message authentication ensures the origin of the message, it typically provides no guarantee of freshness. That is, with message authentication alone, it is not possible to say *when* the message was sent, only to verify its origin. As an example, imagine that Alice sends a message m_A to Bob today, but Mallory intercepts that message and deletes it from the unsecured channel. Mallory could then re-send m_A to Bob any time later, yet Bob would not be able to recognize that the message is delayed and is actually coming from Mallory rather than Alice.

The inability to determine whether a message is delayed or received out of order enables so-called *replay attack*, in which an authenticated message is sent a second time to the receiver, or *reflection attacks*, where an authenticated message is sent back to the sender. More details are given in *Chapter 19, Attacks on Cryptography*.

The lack of proper message authentication is the main cause for the notorious *false base station attacks*, which could originally be launched against 2G mobile networks and, because of the need for backward compatibility, also against 3G and 4G networks [143]. In these attacks, Mallory sets up a **False Base Station** (**FBS**), basically an antenna sending out a strong signal designed to lure mobile phones to connect to the FBS. The FBS then sends a standardized message to the phone to the effect that it does not support encryption. Therefore, the phone resorts to sending its speech data in plaintext to the FBS. If the phone is able to verify if the `NullEncryption` message is really coming from a legitimate network operator, the attack becomes infeasible.

Many other examples of attacks resulting from a lack of or faulty implementation of message authentication could be given. For example, email-based phishing attacks are only possible because email messages are not authenticated in most cases.

Message authentication is strongly related to integrity protection. After all, if Mallory modifies a message m_A coming from Alice to some $\tilde{m}_A$, thus breaking integrity, one might also argue that $\tilde{m}_A$ really is a new message coming from Mallory, but pretending to come from Alice. Therefore, one can say that message authenticity implies integrity.

2.5.2 Entity authentication

Entity authentication is the ability to assure one communicating party – using corroborative evidence – of both the identity of a second communicating party involved, and that the second party was actually active at the time the evidence was created or acquired [117]. This authentication type is also referred to as **identification**.

To achieve entity authentication, Alice and Bob typically engage in some kind of *authentication protocol*, which is a cryptographic protocol designed to achieve entity authentication.

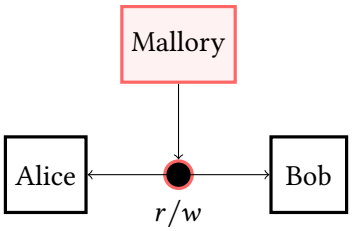

Figure 2.6: Entity authentication allows Bob to verify that the identity of the party he is communicating with is indeed Alice and that Alice is active during the identification. Malicious Mallory has the ability to read Alice's and Bob's messages and write arbitrary messages to the unsecured channel

In a typical example, Alice sends Bob a random, freshly generated challenge (for example, a random number) to which only Bob can respond correctly because Alice and Bob know a shared secret. After Bob has replied to Alice, he sends Alice a fresh, random challenge of his own and waits for the correct reply. If both replies from Alice and Bob are correct, the entity authentication is successful.

To prevent Mallory from compromising entity authentication by simply eavesdropping and replaying old messages, Alice and Bob need to verify each other's authenticity in real time, with non-repeating challenges. This is referred to as *timeliness* or *freshness*. Hence, both parties must be active in the communication.

This protocol is an instance of *mutual authentication* because Alice and Bob authenticate each other. If only Alice or only Bob needs to provide the correct answer to a random challenge, then this would be an example of *unilateral authentication*.

The hack of remote keyless entry systems deployed in VW Group vehicles built between 1995 and 2016 and the attack on the Hitag2 rolling code scheme, are prominent examples of attacks due to insufficient entity authentication [71].

A **Remote Keyless Entry** (**RKE**) system is used to lock and unlock a car without having to insert a physical key. To do this, RKE systems transmit data from the remote control embedded in the car key to the vehicle.

After a button is pressed, a radio transmitter in the car key generates a signal in a public radio frequency band (for example, the 315 MHz band in North America and the 433 MHz or 868 MHz band in Europe).

The first generation of RKEs was based on a constant secret code and is therefore an instance of *security by obscurity*, a very dangerous anti-pattern in system design where the security of a system depends solely on the secrecy of its design. When the constant code is leaked once, the security of all vehicles relying on such an RKE is instantaneously broken.

The second generation of RKE systems relies on so-called rolling codes. In rolling codes, a counter value is increased upon each button press. The counter value – together with some additional input – is used to compute the next valid rolling code message, which is encrypted in the car key and sent to the vehicle.

The vehicle decrypts the rolling code message and compares the result to the last stored counter value. The counter value is accepted and the car is locked or unlocked if the received value is larger than the stored one. If the received counter value is smaller than the stored one, the attempt to lock or unlock the car is rejected.

However, researchers [71] discovered that RKE systems designed by VW Group are based on a worldwide master key. Because the key is identical for all affected cars, it can be extracted by inspecting the firmware of a single vehicle (which is exactly what the researchers did). Knowing that key allows an attacker to lock and unlock a car after eavesdropping on a single rolling code message.

Hitag2 is another widely deployed RKE that is not specific to a single car manufacturer. The counter in the Hitag2 rolling code is not a step-wise increasing value as it is based on the output of a stream cipher (see *Chapter 4, Encryption and Decryption*). An attack reported in [71] requires Eve to eavesdrop on merely four to eight Hitag2 rolling codes and allows her to recover the cryptographic key in approximately one minute. With that, the attacker can create a clone of the original remote control.

These attacks work because the RKE systems lack a cryptographically secure entity authentication. When the car receives a rolling code, it has no means to verify that

it is indeed communicating with the right car key and that the car key is active during the communication.

2.6 Secure channels and the CIA triad

So far, we have discussed three important cryptographic goals: confidentiality, integrity, and authentication. For the purposes of this book, the term *secure system* can be defined as a system that provides a combination of those three goals. Taken together, confidentiality, integrity, and authentication are oftentimes referred to as the *CIA triad*.

Some modern-day scholars and newer books on computer security use the term *availability* instead of *authentication* for the A in CIA. In this book, we deliberately stick to the classical definition. The main reason for this is that, strictly speaking, availability belongs to the realm of security engineering, not cryptography. While cybersecurity threats such as denial-of-service attacks are sometimes discussed in cryptography-related literature, a cryptographic protocol and mechanism by itself is in principle unable to guarantee availability. As a rather simple example, any cryptographic protocol assumes that Alice and Bob can send and receive messages. On the level of cryptographic protocol design, it simply makes no sense to define a protocol if you cannot assume that the information can flow from Alice to Bob and back.

With the definition of the term *secure* via the CIA triad, we can now restate the problem defined in the *Preliminaries* section earlier in this chapter more exactly as the problem of establishing a secure communication channel between Alice and Bob (or between a client device and a server) such that the data transferred is guaranteed to be confidential, authenticated, and unmodified (which is implied by the data being authentic).

A bit less formally, you can think of a secure channel as some kind of tunnel Alice and Bob can use to transfer their data. An attacker cannot see from the outside what is going on in the tunnel: they can neither see the plaintext data nor modify the encrypted data without being detected. Moreover, the tunnel endpoints are authenticated, so Alice can be sure she is really sending her data to Bob, and that any data she receives through the tunnel really comes from Bob.

2.7 Summary

In this chapter, we introduced the most important cryptographic definitions and described the main cryptographic goals, namely confidentiality, integrity, and authentication. We showed that combining these goals into the CIA triad allows Alice and Bob to establish a secure channel in the presence of both passive and active attackers.

In the next chapter, you will learn what a cryptographic key – a secret shared by Alice and Bob – is and why it is needed to establish a secure channel. In particular, you will learn why Alice and Bob should update keys frequently, and how long the key should be so Alice and Bob can communicate securely.

3

A Secret to Share

In the last chapter, we saw that in order to communicate securely over an insecure channel, Alice and Bob need a shared secret (or possibly more than one secret) that is only known to them. Once this is given, they can use cryptography to protect their communication against both passive attackers such as Eve and active attackers such as Mallory.

In cryptography, that shared secret is called a *key*, and we have seen that you can use a secret key K to establish a secure channel between Alice and Bob. But how do you actually generate a secure cryptographic key? How long should it be, and, perhaps most importantly, how can Alice and Bob agree on a key in a secure manner? In this chapter, we will try to provide a brief overview of these issues without becoming too engrossed in the mathematical details, which will be covered later.

In this chapter, we will cover the following topics:

- What is a key and what is a keyspace?
- What is the length of a key and how long should a key be?
- What is the difference between key transport and key agreement?

- What do the terms randomness and entropy mean and what roles do they play in cryptography?

3.1 Secret keys and Kerckhoffs's principle

Let's assume a plaintext m is mapped onto a ciphertext c. Earlier, we formalized this situation in the equation $c = f_K(m)$. You may have wondered why there is a parameter K. In cryptography, we distinguish between the encryption *algorithm* f and the *key* K. We can think of the algorithm as some kind of general template for how to perform encryption. The key is a (secret) parameter that transforms the general template into some specific instantiation that can be used to encrypt the plaintext. It is very important to distinguish between the two and not to treat f_K as a single entity, because the algorithm and the key have very different security requirements. This was realized first by the 19th-century cryptographer Auguste Kerckhoffs, who in 1883 formulated his famous principle that *a cryptosystem should be secure even if everything about the system, except the key, is known to the attacker* [189]. To understand the motivation behind this principle, think of some mechanical encryption devices as they were used by the military in the 19th century. These devices had to be manufactured somewhere, and many people got to see their specifications. Moreover, they could easily get lost in battle. So, it was not wise to assume that the mechanism, that is, the encryption algorithm, could be kept secret over a long period of time.

Kerckhoffs's principle was reformulated in the 20th century by information theory pioneer Claude Shannon succinctly as *the enemy knows the system* [161]. In that form, it is called **Shannon's maxim** [189].

Today, as encryption algorithms are implemented in software, we face the same situation: it is simply not possible to reliably hide the inner workings of an algorithm within the code implementing it. One recent example where the "security by obscurity" approach failed is provided by the CSS algorithm for scrambling the contents of DVDs [78]. The specification of the CSS algorithm was not public. In contrast, it was provided only to manufacturers of playing devices who were willing to sign an agreement with the content owners to the