Chapter 1

Introduction

In this chapter, we pitch the field and introduce the topic of the book, namely cryptography, at a high operating altitude and level of abstraction. More specifically, we elaborate on cryptology (including cryptography) in Section 1.1, address cryptographic systems (or cryptosystems for short) in Section 1.2, provide some historical background information in Section 1.3, and outline the rest of the book in Section 1.4. The aim is to lay the foundations to understand and put into perspective the different book contents.

1.1 CRYPTOLOGY

The term *cryptology* is derived from the Greek words “kryptós,” meaning “hidden,” and “lógos,” meaning “word.” Consequently, the term cryptology can be paraphrased as “hidden word.” This refers to the original intent of cryptology, namely to hide the meaning of words and to protect the confidentiality and secrecy of the respective data accordingly. As will (hopefully) become clear throughout the book, this viewpoint is too narrow, and the term cryptology is currently used for many other security-related purposes and applications in addition to the protection of the confidentiality and secrecy of data.

More specifically, cryptology refers to the mathematical science and field of study that comprises cryptography and cryptanalysis.

- The term *cryptography* is derived from the Greek words “kryptós” (see above) and “gráphein,” meaning “to write.” Consequently, the meaning of the term cryptography can be paraphrased as “hidden writing.” According to the Internet security glossary provided in Request for Comments (RFC) 4949 [1], cryptography also refers to the “mathematical science that deals with

transforming data to render its meaning unintelligible (i.e., to hide its semantic content), prevent its undetected alteration, or prevent its unauthorized use. If the transformation is reversible, cryptography also deals with restoring encrypted data to intelligible form." Consequently, cryptography refers to the process of protecting data in a very broad sense.

- The term *cryptanalysis* is derived from the Greek words "kryptós" (see above) and "analýein," meaning "to loosen." Consequently, the meaning of the term can be paraphrased as "to loosen the hidden word." This paraphrase refers to the process of destroying the cryptographic protection, or—more generally—to study the security properties and possibilities to break cryptographic techniques and systems. Again referring to [1], the term cryptanalysis refers to the "mathematical science that deals with analysis of a cryptographic system in order to gain knowledge needed to break or circumvent[1] the protection that the system is designed to provide." As such, the cryptanalyst is the antagonist of the cryptographer, meaning that his or her job is to break or—more likely—circumvent the protection that the cryptographer has designed and implemented in the first place. Quite naturally, there is an arms race going on between the cryptographers and the cryptanalysts (but note that an individual person may have both skills, cryptographic and cryptanalytical ones).

Many other definitions for the terms cryptology, cryptography, and cryptanalysis exist and can be found in the literature (or on the Internet, respectively). For example, the term cryptography is sometimes said to more broadly refer to the study of mathematical techniques related to all aspects of information security (e.g., [2]). These aspects include (but are not restricted to) data confidentiality, data integrity, entity authentication, data origin authentication, nonrepudiation, and/or many more. Again, this definition is broad and comprises anything that is directly or indirectly related to information security.

In some literature, the term cryptology is even said to include steganography (in addition to cryptography and cryptanalysis).

- The term *steganography* is derived from the Greek words "steganos," meaning "impenetrable," and "gráphein" (see above). Consequently, the meaning of the term can be paraphrased as "impenetrable writing." According to [1], the

1 In practice, circumventing (bypassing) the protection is much more common than breaking it. In his 2002 ACM Turing Award Lecture (https://amturing.acm.org/vp/shamir_2327856.cfm or https://www.youtube.com/watch?v=KUHaLQFJ6Cc), for example, Adi Shamir—a coinventor of the RSA public key cryptosystem—made the point that "cryptography is typically bypassed, not penetrated," and this point was so important to him that he put it as a third law of security (in addition to "absolutely secure systems do not exist" and "to halve your vulnerability you have to double your expenditure").

term refers to "methods of hiding the existence of a message or other data. This is different than cryptography, which hides the meaning of a message but does not hide the message itself." Let us consider an analogy to clarify the difference between steganography and cryptography: if we have money to protect or safeguard, then we can either hide its existence (by putting it, for example, under a mattress), or we can put it in a safe that is as burglarproof as possible. In the first case, we are referring to steganographic methods, whereas in the second case, we are referring to cryptographic ones. An example of a formerly widely used steganographic method is invisible ink. A message remains invisible, unless the ink is subject to some chemical reaction that has the message reappear and become visible again. Currently deployed steganographic methods are much more sophisticated, and can, for example, be used to hide information in files. In general, this information is arbitrary, but it is typically used to identify the owner or the recipient of a file. In the first case, one refers to *digital watermarking*, whereas in the second case one refers to *digital fingerprinting*. Both types of watermarks are subject to research and development.

The relationship between cryptology, cryptography, cryptanalysis, and steganography is illustrated in Figure 1.1. In this book, we only focus on cryptography in a narrow sense (this is symbolized with the shaded box in Figure 1.1). This can also be stated as a disclaimer: we elaborate on cryptanalysis only where necessary and appropriate, and we do not address steganography at all. There are other books that address cryptanalysis (e.g., [3–5]) or provide useful information about steganography in general (e.g., [6, 7]) and digital watermarking and fingerprinting in particular (e.g., [8, 9]).

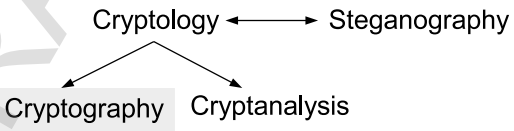


Figure 1.1 The relationship among cryptology, cryptography, cryptanalysis, and steganography.

Interestingly, cryptographic and steganographic technologies and techniques are not mutually exclusive, and it may make a lot of sense to combine them. For

example, the open source disk encryption software VeraCrypt[2] employs steganographic techniques to hide the existence of an encrypted disk volume (known as a "hidden volume"). It is possible and likely that we will see more combinations of cryptographic and steganographic technologies and techniques in future products.

1.2 CRYPTOGRAPHIC SYSTEMS

According to [1], the term *cryptographic system*[3] (or *cryptosystem*) refers to "a set of cryptographic algorithms together with the key management processes that support use of the algorithms in some application context." Again, this definition is fairly broad and comprises all kinds of cryptographic algorithms and—as introduced below—protocols. Hence, the notion of an algorithm[4] captured in Definition 1.1 is key for cryptographic systems.

Definition 1.1 (Algorithm) *An* algorithm *is a well-defined computational procedure that takes a value as input and turns it into another value that represents the output.*

In addition to being well-defined, it is sometimes required that the algorithm halts within a reasonable amount of time (for any meaningful definition of "reasonable"). Also, Definition 1.1 is rather vague and not mathematically precise. It neither states the computational model for the algorithm, nor does it say anything about the problem the algorithm is supposed to solve, such as, for example, computing a mathematical function. Consequently, from a theoretical viewpoint, an algorithm can be more precisely defined as a well-defined computational procedure for a well-defined computational model for solving a well-defined problem. This definition, however, is a little bit clumsy and therefore not widely used in field. Instead, people usually prefer the simpler (and more intuitive) definition stated above.

In practice, a major distinction is made between algorithms that are deterministic and algorithms that are not (in which case they are probabilistic or randomized).

2 VeraCrypt (https://www.veracrypt.fr) is a fork of the formerly very popular but discontinued TrueCrypt project.

3 In some literature, the term *cryptographic scheme* is used to refer to a cryptographic system. Unfortunately, it is seldom explained what the difference is between a (cryptographic) scheme and a system. So for the purpose of this book, we don't make a distinction, and we use the term cryptographic system to refer to either of them. We hope that this simplification is not too confusing. In the realm of digital signatures, for example, people often use the term digital signature scheme that is not used in this book. Instead, we consistently use the term digital signature system to refer to the same construct.

4 The term *algorithm* is derived from the name of the mathematician Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and lived from about 780 to 850.

- An algorithm is *deterministic* if its behavior is completely determined by the input. This also means that the algorithm always generates the same output for the same input (if executed multiple times).
- An algorithm is *probabilistic* (or *randomized*) if its behavior is not completely determined by the input, meaning that the algorithm internally employs some (pseudo)random values.[5] Consequently, a probabilistic (or randomized) algorithm may generate different outputs each time it is executed with the same input.

Today, probabilistic algorithms play a much more important role in cryptography than they used to play in the past. Anyway, an algorithm may be implemented by a computer program that is written in a specific programming language, such as Pascal, C, or Java. Whenever we describe algorithms in this book, we don't use a specific programming language, but we use a more formal and simpler notation that looks as follows:

(input parameters)
---
computational step
...
computational step

(output parameters)

The input and output parameters are written in brackets at the beginning and at the end of the algorithm description, whereas the body of the algorithm consists of a sequence of computational steps that are executed in the specified order (or sometimes in parallel). Throughout the book, we introduce some cryptosystems as sets of algorithms that are each written in this notation.

If more than one entity is involved in the execution of an algorithm (or the computational procedure it defines, respectively), then one is in the realm of protocols—a term that originates from diplomacy. Definition 1.2 captures the notion of a protocol.

Definition 1.2 (Protocol) *A* protocol *is a distributed algorithm in which two or more entities take part.*

Alternatively, one can define a protocol as a distributed algorithm in which a set of more than one entity (instead of two or more entities) takes part. In this case, it is intuitively clear that an algorithm also represents a protocol, namely one

5 A value is random (pseudorandom) if it is randomly (pseudorandomly) generated.

that is degenerated in the sense that the set consists of just one entity. Hence, an algorithm can also be seen as a special case of a protocol. The major distinction between an algorithm and a protocol is that only one entity is involved in the former, whereas two or more entities are involved in the latter. This distinguishing fact is important and must be kept in mind when one talks about algorithms and protocols—not only cryptographic ones. It means that in a protocol the different entities may have to send messages to each other, and hence that a protocol may also comprise communication steps (in addition to computational steps). As such, protocols tend to be more involved than algorithms and this also affects their analysis.

Similar to an algorithm, a protocol may be deterministic or probabilistic, depending on whether the protocol internally employs random values. For the purpose of this book, we are mainly interested in cryptographic algorithms and protocols as suggested in Definitions 1.3 and 1.4.

Definition 1.3 (Cryptographic algorithm) *A* cryptographic algorithm *is an algorithm that employs and makes use of cryptographic techniques and mechanisms.*

Definition 1.4 (Cryptographic protocol) *A* cryptographic protocol *is a protocol that employs and makes use of cryptographic techniques and mechanisms.*

Remember the definition for a cryptographic system (or cryptosystem) given at the beginning of this section. According to this definition, a cryptosystem may comprise more than one algorithm, and the algorithms need not be executed by the same entity, that is, they may be executed by multiple entities in a distributed way. Consequently, this notion of a cryptosystem also captures the notion of a cryptographic protocol as suggested in Definition 1.4. Hence, another way to look at cryptographic algorithms and protocols is to say that a cryptographic algorithm is a *single-entity cryptosystem*, whereas a cryptographic protocol is a *multi-entity* or *multiple entities cryptosystem*. These terms, however, are not used in the literature, and hence we don't use them in this book either.

At this point in time, it is important to note that a typical cryptographic application may consist of multiple (cryptographic) protocols, that these protocols and their concurrent execution may interact in subtle ways, and that the respective interdependencies may be susceptible to *multi-protocol attacks*.[6] As its name suggests, more than one protocol is involved in such an attack, and the adversary may employ messages from one protocol execution to construct valid-looking messages for other protocols and executions thereof. If, for example, one protocol uses digital signatures for random-looking data and another protocol is an authentication protocol in which

6 The notion of a *chosen protocol* or *multi-protocol attack* first appeared in a 1997 paper [10], but the problem had certainly preexisted before that.

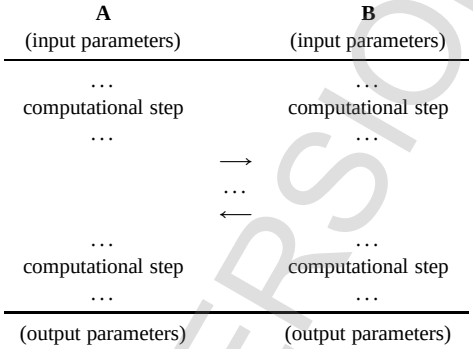
an entity must digitally sign a nonce[7] to authenticate itself, then an adversary can use the first protocol as an oracle to digitally sign a nonce from an execution of the second protocol. This is a simple and devastating attack that can be mitigated easily (for example, by using two distinct keys). However, more involved interactions and interdependencies are possible and likely exist, and hence multi-protocol attacks tend to be powerful and difficult to mitigate. Fortunately, many such attacks have been described in scientific papers, but only few have been mounted in the field so far—at least as far as we know today.

In the cryptographic literature, it is common to use human names for entities that participate in cryptographic protocols, such as a Diffie-Hellman key exchange. For example, in a two-party protocol the participating entities are usually called *Alice* and *Bob*. This is a convenient way of making things unambiguous with relatively few words, since the pronoun *she* can be used for Alice, and *he* can be used for Bob. The disadvantage of this naming scheme is that people automatically assume that the entities refer to human beings. This need not be the case though, and Alice, Bob, and all other entities are rather computer systems, cryptographic devices, hardware modules, smartcards, or anything along these lines. In this book, we don't follow the tradition of using Alice, Bob, and the rest of the gang. Instead, we use single-letter characters, such as A, B, and so on, to refer to the entities that take part and participate in a cryptographic protocol. This is admittedly less fun, but more appropriate and in line with the traditional notation used in information theory (see, for example, [11] for a more comprehensive reasoning about this issue). In reality, the entities refer to social-technical systems that may have a user interface, and the question of how to properly design and implement such an interface is key to the overall security of the system. If this interface is not appropriate, then phishing and many other types of social engineering attacks become trivial to mount.

The cryptographic literature provides many examples of more or less useful cryptographic protocols. Some of these protocols are overviewed, discussed, and put into perspective in this book. To formally describe a (cryptographic) protocol in which A and B take part, we use the notation given at the top of the next page. Some input parameters may be required on either side of the protocol (note that the input parameters need not be the same). The protocol then includes a sequence of computational and communication steps. Each computational step may occur only on one side of the protocol, whereas each communication step requires data to be transferred from one side to the other. In this case, the direction of the data flow is indicated by an arrow. The set of all data that is communicated this way refers to a *protocol transcript*. Finally, some parameters may be output on either side of the protocol. These output parameters actually represent the result of the protocol

7 The term *nonce* refers to a "number that is used only once." More specifically, it is a random-looking number that is fresh and unique.

execution. Similar to the input parameters, the output parameters need not be the same on either side. In many cases, however, the output parameters are the same. In the case of the Diffie-Hellman key exchange, for example, the output is the session key that can subsequently be used to secure communications. But we will explain this in detail in Section refDiffieHellmanKeyExchange.



1.2.1 Classes of Cryptographic Systems

Cryptographic systems may or may not use secret parameters (e.g., cryptographic keys), and if such parameters are used, then they may or may not be shared among the participating entities. Consequently, there are three classes of cryptographic systems that can be distinguished.[8] They are captured in Definitions 1.5–1.7.

Definition 1.5 (Unkeyed cryptosystem) *An* unkeyed cryptosystem *is a cryptographic system that uses no secret parameter.*

Definition 1.6 (Secret key cryptosystem) *A* secret key cryptosystem *is a cryptographic system that uses secret parameters that are shared among the participating entities.*

Definition 1.7 (Public key cryptosystem) *A* public key cryptosystem *is a cryptographic system that uses secret parameters that are not shared among the participating entities.*

In Chapter 2, we informally introduce and briefly overview the most important representatives of these classes. These representatives are then formally addressed

8 The classification scheme was created by Ueli M. Maurer.

in Part I (unkeyed cryptosystems), Part II (secret key cryptosystems), and Part III (public key cryptosystems) of the book. In these parts, we also provide more formal definitions of both the cryptosystems and their security properties. In the rest of this section, we continue to argue informally about the security of cryptographic systems and the different perspectives one may take to look at them.

1.2.2 Secure Cryptographic Systems

The goal of cryptography is to design, implement, and employ cryptographic systems that are "secure." But to make precise statements about the security of a particular cryptosystem, one must formally define the term security. Unfortunately, reality looks different, and the literature is full of cryptographic systems that are claimed to be secure without providing an appropriate definition for it. This is dissatisfactory, mainly because anything can be claimed to be secure, unless the meaning of the word is precisely nailed down.

Instead of properly defining the term security and analyzing whether a cryptographic system meets this definition, people often like to argue about key lengths. This is because the key length is a simple and very intuitive security parameter. So people frequently use it to characterize the cryptographic strength of a system. This is clearly an oversimplification, because the key length is a suitable (and meaningful) measure of security if and only if an exhaustive key search is the most efficient way to break it. In practice, however, this is seldom the case, and there are often simpler ways to break the security of a system (e.g., by reading out some keying material from memory). In this book, we avoid discussions about key lengths. Instead, we refer to the recommendations[9] compiled and hosted by a Belgian company named BlueKrypt.[10] They provide practical advice to decide what key lengths are appropriate for any given cryptosystem (even using different methods that may lead to different conclusions and recommendations).

In order to discuss the security of a cryptosystem, there are two perspectives one may take: a theoretical one and a practical one. Unfortunately, the two perspectives are fundamentally different, and one may have a cryptosystem that is theoretically secure but practically insecure (e.g., due to a poor implementation), or—vice versa—a cryptosystem that provides a sufficient level of security in practice but is not very sophisticated from a theoretical viewpoint. Let us have a closer look at both perspectives before we focus on some general principles for the design of cryptographic systems.

9 https://www.keylength.com.
10 https://www.bluekrypt.com.

1.2.2.1 Theoretical Perspective

According to what has been said above, one has to start with a precise definition for the term "security" before one can scientifically argue about the security of a particular cryptosystem. What does it mean for such a system to be "secure"? What properties does it have to fulfill? In general, there are two questions that need to be answered here:

1. Who is the adversary; that is, what are his or her capabilities and how powerful is he or she?
2. What is the task the adversary has to solve in order to be successful; that is, to break the security of the system?

An answer to the first question must comprise a *threats model*, i.e., a model of the adversary one has in mind and against whom one wants to protect oneself. This adversary must be described (and hence modeled) in terms of computational resources, memory, time, and a priori information that are available, as well as the types of attacks that are feasible to mount (including, for example, whether the adversary has physical access to the system under attack). Such a threats model is subtle and highly relevant. If, for example, an adversary is tasked to remove the shell from an egg but has no physical access to the egg, then it is fairly simple to argue that the shell cannot be removed. However, the resulting (security) argument is also irrelevant, mainly because anybody tasked to remove an egg shell will also be given physical access to the egg.

Finding an appropriate answer to the second question is even more tricky. In general, the adversary's task is to find (i.e., compute, guess, or otherwise determine) one or several pieces of information that he or she should not be able to know. If, for example, the adversary is able to determine the cryptographic key used to encrypt a message, then he or she must clearly be considered to be successful. But what if he or she is able to determine only half of the key, or—maybe even more controversial—a single bit of the key? Similar difficulties in defining an appropriate task for the adversary occur in other cryptosystems that are used for other purposes than confidentiality protection.

The preferred way to deal with these difficulties is to define a *security game* in the so-called *ideal/real simulation paradigm* that is illustrated in Figure 1.2. On the left side, there is a system that is either *ideal* or *real*, meaning that it is either a theoretically perfect system for the task one has in mind (ideal system) or it is a real-world implementation thereof (real system). If, for example, the real system is a block cipher used for symmetric encryption, then the ideal system can be thought of as a pseudorandom permutation. The adversary on the right side can interact

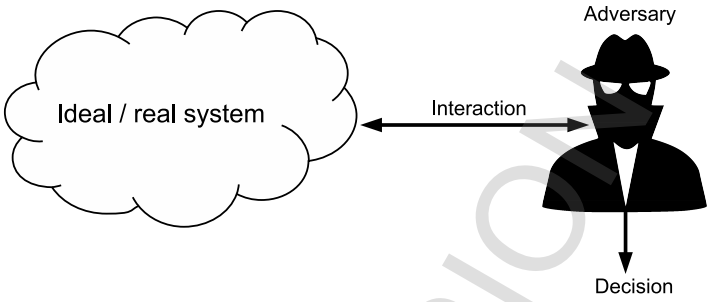


Figure 1.2 Security game in the ideal/real simulation paradigm.

with the (ideal or real) system in some predefined way, and his or her task is to tell the two cases apart. Again referring to a block cipher, it may be the case that the adversary can have arbitrary plaintext messages be encrypted or arbitrary ciphertext messages be decrypted. The type of interaction matters and must therefore be part of the specification of the security game. In either case, the adversary wins the game if he or she is able to tell whether he or she is interacting with an ideal or a real system with a probability that is better than guessing. If the adversary is not able to do so, then the real system is indistinguishable from an ideal one, and hence it has all the relevant properties of the ideal system, including its security. It is therefore as secure as the ideal system. Most security proofs in cryptography follow this line of argumentation and define the task the adversary needs to solve (in order to be successful) in such a game-theoretic way. We will see many examples throughout the book.

As captured in Definition 1.8, a definition for a secure cryptographic system must start with the two questions itemized above and their respective answers.

Definition 1.8 (Secure cryptographic system) *A cryptographic system is* secure *if a well-defined adversary cannot break it, meaning that he or she cannot solve a well-defined task.*

This definition gives room for several notions of security. In principle, there is a distinct notion for every possible adversary combined with every possible task. As a general rule of thumb, we say that strong security definitions assume an adversary that is as powerful as possible and a task to solve that is as simple as possible. If a system can be shown to be secure in this setting, then there is a security margin. In reality, the adversary is likely less powerful and the task he or she must solve is

likely more difficult, and this, in turn, means that it is very unlikely that the security of the system gets broken.

Let us consider a provocative question (or mind experiment) to clarify this point: If we have two safes S_1 and S_2, where S_1 cannot be cracked by a schoolboy within one minute and S_2 cannot be cracked by a group of world-leading safe crackers within one year, and we ask the question whether S_1 or S_2 can be considered to be more secure, then we would unanimously opt for S_2. The argument would go as follows: If even a group of world-leading safe crackers cannot crack S_2 within one year, then it very unlikely that a real-world criminal is able to crack it in a shorter but more realistic amount of time, such as one hour. In contrast, the security level S_1 provides is much less convincing. The fact that even a schoolboy can crack S_1 within one minute makes us believe that the criminal is likely to be successful in cracking the safe within one hour. This gives us a much better feeling about the security of S_2. In cryptography, we use a similar line of argumentation when we talk about the strength of a security definition.

If we say that an adversary cannot solve a task, then we can still distinguish the two cases in which he or she is simply not able solve it (independent from his or her computational resources) or he or she can in principle solve it but doesn't have the computational resources to do so, meaning that it is computationally too expensive. This distinction brings us to the following two notions of security.

Unconditional security: If an adversary with infinite computational resources is not able to solve the task within a finite amount of time, then we are talking about *unconditional* or *information-theoretic security*. The mathematical theories behind this notion of security are probability theory and information theory, as briefly introduced in Appendixes B and C.

Conditional security: If an adversary is in principle able to solve the task within a finite amount of time but doesn't have the computational resources to do so,[11] then we are talking about *conditional* or *computational security*. The mathematical theory behind this notion of security is computational complexity theory, as briefly introduced in Appendix D.

The distinction between unconditional and conditional security is at the core of modern cryptography. Interestingly, there are cryptosystems known to be secure in the strong sense (i.e., unconditionally or information-theoretically secure), whereas there are no cryptosystems known to be secure in the weak sense (i.e., conditionally or computationally secure). There are many cryptosystems that are assumed to be computationally secure, but no proof is available for any of these systems. In fact, not even the existence of a conditionally or computationally secure cryptosystem

11 It is usually assumed that the adversary can run algorithms that have a polynomial running time.

has been proven so far. The underlying problem is that it is generally impossible to prove a lower bound for the computational complexity of solving a problem. To some extent, this is an inherent weakness or limitation of complexity theory as it stands today.

In addition to the unconditional and conditional notions of security, people often use the term *provable security* to refer to another—arguably strong—notion of security. This goes back to the early days of public key cryptography, when Whitfield Diffie and Martin E. Hellman proposed a key exchange protocol [12][12] (Section 12.3) and a respective security proof. In fact, they showed that their protocol is secure unless somebody is able to solve a hard mathematical problem (that is strongly related to the discrete logarithm problem). The security of the protocol is thus reduced to a hard mathematical problem (i.e., if somebody is able to solve the mathematical problem, then he or she is also able to break the security of the protocol). This is conceptually similar to proving that squaring a circle with compass and straightedge is impossible. This well-known fact from geometry can be proven by reducing the problem of squaring a circle to the problem of finding a non-zero polynomial $f(x) = a_n x^n + \ldots + a_1 x + a_0$ with rational coefficients a_i for $i = 0, 1, \ldots, n$, such that π is a root; that is, $f(\pi) = 0$. Because we know that π is not algebraic (it is transcendental), we know that such a polynomial does not exist and cannot be found in the first place. Conversely, this implies that a circle cannot be squared with a compass and straightedge.

More specifically, the basic concept of a reduction proof is to show that breaking a cryptosystem is computationally equivalent to solving a hard mathematical problem. This means that one must prove the following two directions:

- If the hard problem can be solved, then the cryptosystem can be broken.
- If the cryptosystem can be broken, then the hard problem can be solved.

Diffie and Hellman only proved the first direction, and they did not prove the second direction.[13] This is unfortunate, because the second direction is equally important for security. If we can prove that an adversary who is able to break a cryptosystem is also able to solve the hard problem, then we can reasonably argue that it is unlikely that such an adversary exists, and hence that the cryptosystem in question is likely to be secure. Michael O. Rabin[14] was the first researcher who

12 This paper is the one that officially gave birth to public key cryptography. There is a companion paper entitled "Multiuser Cryptographic Techniques" that was presented by the same authors at the National Computer Conference on June 7–10, 1976.

13 This was changed in [13] and some follow-up publications.

14 In 1976, Michael O. Rabin and Dana S. Scott jointly received the ACM Turing Award for their work on nondeterministic machines that is key to theoretical computer science and complexity theory. This work was not yet directly related to cryptography.

proposed a cryptosystem [14] that can be proven to be computationally equivalent to a mathematically hard problem (Section 13.3.2).

The notion of (provable) security has fueled a lot of research since the late 1970s. In fact, there are many (public key) cryptosystems proven secure in this sense. It is, however, important to note that a complexity-based proof is not absolute, and that it is only relative to the (assumed) intractability of the underlying mathematical problem(s). This is a similar situation to proving that a problem is **NP**-hard. It proves that the problem is at least as difficult as other problems in **NP**, but it does not provide an absolute proof of its computational difficulty.[15]

There are situations in which a security proof requires an additional assumption, namely that a cryptographic primitive–typically a cryptographic hash function—behaves like a random function. This leads to a new paradigm and methodology to design cryptographic systems that are "provably secure" in the *random oracle model* (in contrast to the *standard model*) [15]. The methodology employs the idea of an ideal system (as introduced above) and consists of the following three steps:

- First, one designs an ideal system that uses random functions[16] (also known as random oracles), most notably cryptographic hash functions.
- Second, one proves the security of the ideal system.
- Third, one replaces the random functions with real ones.

As a result, one obtains an implementation of the ideal system in the real world (where random functions do not exist). Due to the use of random oracles, this methodology is known as *random oracle methodology*, and—as mentioned above—it yields cryptosystems that are secure in the random oracle model. As further addressed in Section 8.3, such cryptosystems and their respective "security proofs" are widely used in the field, but they must be taken with a grain of salt. In fact, it has been shown that it is possible to construct cryptographic systems that are provably secure in the random oracle model, but become totally insecure whenever the cryptographic hash function used in the protocol (to replace the random oracle) is instantiated. This theoretical result is worrisome, and since its publication many researchers have started to think controversially about the random oracle methodology and the usefulness of the random oracle model per se. At least it must be noted that formal analyses in the random oracle model are not security proofs in a purely mathematical sense. The problem is the underlying ideal assumptions about

15 Refer to Appendix D to get a more detailed overview about **NP**and **NP**-hard problems.

16 The notion of a random function is briefly introduced in Section 2.1.2 and more thoroughly addressed in Chapter 8.

the randomness properties of the cryptographic hash functions, this is not something that is otherwise used in mathematically rigorous proofs.

In this book, we don't consider provable security (with or without the random oracle model) as a security notion of its own. Instead, we see it as a special case of conditional security, namely one where the intractability assumption represents the condition.

1.2.2.2 Practical Perspective

So far, we have argued about the security of a cryptosystem from a purely theoretical viewpoint. In practice, however, any (theoretically secure) cryptosystem must be implemented, and there are many things that can go wrong here (e.g., [16]). For example, the cryptographic key in use may be kept in memory and extracted from there, for example, using a cold boot attack[17] [17], or the user of a cryptosystem may be subject to all kinds of phishing and social engineering attacks.

Historically, the first such attacks tried to exploit the compromising emanations that occur in all information-processing physical systems. These are unintentional intelligence-bearing signals that, if intercepted and properly analyzed, may disclose the information transmitted, received, handled, or otherwise processed by a piece of equipment. In the late 1960s and early 1970s, the U.S. National Security Agency (NSA) coined the term *TEMPEST* to refer to this field of study (i.e., to secure electronic communications equipment from potential eavesdroppers), and vice versa, the ability to intercept and interpret those signals from other sources.[18] Hence, the term TEMPEST is a codename (not an acronym[19]) that is used broadly to refer to the entire field of emission security or emanations security (EMSEC). There

17 This attack exploits the fact that many dynamic random access memory (DRAM) chips don't lose their contents when a system is switched off immediately, but rather lose their contents gradually over a period of seconds, even at standard operating temperatures and even if the chips are removed from the motherboard. If kept at low temperatures, the data on these chips persist for minutes or even hours. In fact, the researchers showed that residual data can be recovered using simple techniques that require only temporary physical access to a machine, and that several popular disk encryption software packages, such as Microsoft's BitLocker, Apple's FileVault, and TrueCrypt (the predecessor of VeraCrypt) were susceptible to cold boot attacks. The feasibility and simplicity of such attacks has seriously challenged the security of many disk encryption software solutions.

18 https://www.nsa.gov/news-features/declassified-documents/cryptologic-spectrum/assets/files/tempest.pdf.

19 The U.S. government has stated that the term TEMPEST is not an acronym and does not have any particular meaning (it is therefore not included in this book's list of abbreviations and acronyms). However, in spite of this disclaimer, multiple acronyms have been suggested, such as "Transmitted Electro-Magnetic Pulse / Energy Standards & Testing," "Telecommunications ElectroMagnetic Protection, Equipment, Standards & Techniques," "Transient ElectroMagnetic Pulse Emanation STandard," "Telecommunications Electronics Material Protected from Emanating Spurious Transmissions," and—more jokingly—"Tiny ElectroMagnetic Particles Emitting Secret Things."

are several U.S. and NATO standards that basically define three levels of TEMPEST requirements: NATO SDIP-27 Levels A, B, and C; but this is far beyond the scope of this book.

In addition to cold boot attacks and exploiting compromising emanations, people have been very innovative in finding possibilities to mount attacks against presumably tamper-resistant hardware devices that employ invasive measuring techniques (e.g., [18, 19]). Most importantly, there are attacks that exploit side channel information an implementation may leak when a computation is performed. Side channel information is neither input nor output, but refers to some other information that may be related to the computation, such as timing information or power consumption. Attacks that try to exploit such information are commonly referred to as *side channel attacks*. Let us start with two mind experiments to better illustrate the notion of a side channel attack.[20]

- Assume somebody has written a secret note on a pad and has torn off the respective paper sheet. Is there a possibility to reconstruct the note? An obvious possibility is to go for a surveillance camera and examine the respective recordings. A less obvious possibility is to exploit the fact that pressing the pen on the paper sheet may have caused the underlying paper sheet to experience some pressure, and this, in turn, may have caused the underlying paper sheet to show the same groove-like depressions (representing the actual writing). Equipped with the appropriate tools, an expert may be able to reconstruct the note. Pressing the pen on a paper sheet may have caused a side channel to exist, even if the original paper sheet is destroyed.
- Consider a house with two rooms. In one room are three light switches and in the other room are three lightbulbs, but the wiring of the light switches and bulbs is unknown. In this setting, somebody's task is to find out the wiring, but he or she can enter each room only once. From a mathematical viewpoint, one can argue (and possibly even prove) that this task is impossible to solve. But from a physical viewpoint (and taking into account side channel information), the task can be solved: One can enter the room with the light switches, permanently turn on one bulb, and turn on another bulb for some time (e.g., a few seconds). One then enters the room with the lightbulbs. The bulb that is lit is easily identified and refers to the switch that has been permanently switched on. But the other two bulbs are not lit, and hence one cannot easily assign them to the respective switches. But one can measure the temperature of the lightbulbs. The one that is warmer more likely refers to the switch that has been switched on for some time. This information can be used to distinguish the two cases and to solve the task accordingly. Obviously, the

20 The second mind experiment was proposed by Artur Ekert.

trick is to measure the temperature of the lightbulbs and to use this information as a side channel.

In analogy to these mind experiments, there are many side channel attacks that have been proposed to defeat the security of cryptosystems, some of which have turned out to be very powerful. The first side channel attack that opened people's eyes and the field in the 1990s was a timing attack against a vulnerable implementation of the RSA cryptosystem [20]. The attack exploited the correlation between a cryptographic key and the running time of the algorithm that employed the key. Since then, many implementations of cryptosystems have been shown to be vulnerable against timing attacks and some variants, such as cache timing attacks or branch prediction analysis. In 2003, it was shown that remotely mounting timing attacks over computer networks is feasible [21], and since 2018 we know that almost all modern processors that support speculative and out-of-order command execution are susceptible to sophisticated timing attacks.[21] Other side channel attacks exploit the power consumption of an implementation of an algorithm that is being executed (usually called power consumption or power analysis attacks [22]), faults that are induced (usually named differential fault analysis [23, 24]), protocol failures [25], the sounds that are generated during a computation [26, 27], and many more.

Side channel attacks exploit side channel information. Hence, a reasonable strategy to mitigate a specific side channel attack is to avoid the respective side channel in the first place. If, for example, one wants to protect an implementation against timing attacks, then timing information must not leak. At first sight, one may be tempted to add a random delay to every computation, but this simple mechanism does not work in the field (because the effect of random delays can be compensated by having an adversary repeat the measurement many times). But there may be other mechanisms that work. If, for example, one ensures that all operations take an equal amount of time (i.e., the timing behavior is largely independent from the input), then one can mitigate such attacks. But constant-time programming has turned out to be difficult, certainly more difficult than it looks at first sight. Also, it is sometimes possible to blind the input and to prevent the adversary from knowing the true value. Both mechanisms have the disadvantage of slowing down the computations. There are fewer possibilities to protect an implementation against power consumption attacks. For example, dummy registers and gates can be added on which useless operations are performed to balance power consumption into a constant value. Whenever an operation is performed, a complementary operation is also performed on a dummy element to assure that the total power consumption remains balanced according to some higher value. Protection against differential fault analysis is less

21 The first such attacks have been named *Meltdown* and *Spectre*. They are documented at https://www.spectreattack.com.

general and even more involved. In [23], for example, the authors suggest a solution that requires a cryptographic computation to be performed twice and to output the result only if they are the same. The main problem with this approach is that it roughly doubles the execution time. Also, the probability that the fault will not occur twice is not sufficiently small (and this makes the attack harder to implement, but not impossible). The bottom line is that the development of adequate and sustainable protection mechanisms to mitigate differential fault analysis attacks remains a timely research topic. The same is true for failure analysis and acoustic cryptanalysis, and it may even be true for other side channel attacks that will be found and published in the future. Once a side channel is known, one can usually do something to avoid the respective attacks. But keep in mind that many side channels may exist that are still unknown today.

The existence and difficulty of mitigating side channel attacks have inspired theoreticians to come up with a model for defining and delivering cryptographic security against an adversary who has access to information leaked from the physical execution of a cryptographic algorithm [28]. The original term used to refer to this type of cryptography is *physically observable cryptography*. More recently, however, researchers have coined the term *leakage-resilient cryptography* to refer to essentially the same idea. Even after many years of research, it is still questionable whether physically observable or leakage-resilient cryptography can be achieved in the first place (e.g., [29]). It is certainly a legitimate and reasonable design goal, but it may not be a very realistic one. Maybe we will know one day.

1.2.2.3 Design Principles

In the past, we have seen many examples in which people have tried to improve the security of a cryptographic system by keeping secret its design and internal working principles. This approach is sometimes referred to as "security through obscurity." Many of these systems do not work and can be broken trivially.[22] This insight has a long tradition in cryptography, and there is a well-known cryptographic principle—*Kerckhoffs's principle*[23]—that basically states that a cryptographic system should be designed so as to remain secure, even if the adversary knows all the details of the system, except for the values explicitly declared to be secret, such as secret keys [30]. We follow this principle in this book, and we only address cryptosystems for which we can assume that the adversary knows the details. This assumption is in line with our requirement that the adversaries should be assumed to be as powerful as possible (to obtain the strong security definitions detailed in Section 1.2.2).

22 Note that "security through obscurity" may work outside the realm of cryptography.
23 The principle is named after Auguste Kerckhoffs, who lived from 1835 to 1903.

In spite of Kerckhoffs's principle, the design of a secure cryptographic system remains a difficult and challenging task. One has to make assumptions, and it is not clear whether these assumptions really hold in reality. For example, one usually assumes a certain set of countermeasures to protect against specific attacks. If the adversary attacks the system in another way, then there is hardly anything that can be done about it. Similarly, one has to assume the system to operate in a "typical" environment. If the adversary can manipulate the environment, then he or she may be able to change the operational behavior of the system, and hence to open up new vulnerabilities. The bottom line is that cryptographic systems that are based on make-believe, ad hoc approaches, and heuristics are often broken in the field in some new and ingenious ways. Instead, the design of a secure cryptographic system should be based on firm foundations. Ideally, it consists of the following two steps:

1. In a *definitional step*, the problem the cryptographic system is intended to solve is identified, precisely defined, and formally specified.
2. In a *constructive step*, a cryptographic system that satisfies the definition distilled in step one, possibly while relying on intractability assumptions, is designed.

Again, it is important to note that most parts of modern cryptography rely on intractability assumptions and that relying on such assumptions seems to be unavoidable. But there is still a huge difference between relying on an explicitly stated intractability assumption or just assuming (or rather hoping) that an ad hoc construction satisfies some unspecified and vaguely specified goals. This basically distinguishes cryptography as a science from cryptography as an art.

1.3 HISTORICAL BACKGROUND INFORMATION

Cryptography has a long and thrilling history that is addressed in many books (e.g., [31–33]). In fact, probably since the very beginning of the spoken and—even more importantly—written word, people have tried to transform "data to render its meaning unintelligible (i.e., to hide its semantic content), prevent its undetected alteration, or prevent its unauthorized use" [1]. According to this definition, these people have always employed cryptography and cryptographic techniques. The mathematics behind these early systems may not have been very advanced, but they still employed cryptography and cryptographic techniques. For example, Gaius Julius Caesar[24] used an encryption system in which every letter in the Latin alphabet was substituted with the letter that is found three positions afterward in the lexical order (i.e., "A"

24 Gaius Julius Caesar was a Roman emperor who lived from 102 BC to 44 BC.

is substituted with "D," "B" is substituted with "E," and so on). Today, this simple additive cipher is known as *Caesar cipher*. Later on, people employed encryption systems that used more advanced and involved mathematical transformations. Many books on cryptography contain numerous examples of historically relevant encryption systems—they are not repeated in this book (unless a few remarks in Section 9.2); the encryption systems in use today are simply too different.

Until World War II, cryptography was considered to be an art (rather than a science) and was primarily used in military and diplomacy. The following two developments and scientific achievements turned cryptography from an art into a science:

- During World War II, Claude E. Shannon[25] developed a mathematical theory of communication [34] and a related communication theory of secrecy systems [35] when he was working at AT&T Laboratories.[26] After their publication, the two theories started a new branch of research that is commonly referred to as *information theory* (again, refer to Appendix C for a brief introduction to information theory). It is used to prove the unconditional security of cryptographic systems.
- As mentioned earlier, Diffie and Hellman developed and proposed the idea of public key cryptography at Stanford University in the 1970s.[27] Their vision was to employ trapdoor functions to encrypt and digitally sign electronic data and documents. As introduced in Section 2.1.3 and further addressed in Chapter 5, a trapdoor function is a function that is easy to compute but hard to invert—unless one knows and has access to some trapdoor information. This information represents the private key held by a particular entity.

Diffie and Hellman's work culminated in a key agreement protocol (i.e., the Diffie-Hellman key exchange protocol described in Section 12.3) that allows two parties that share no secret to exchange a few messages over a public channel and to establish a shared (secret) key. This key can, for example, then be used to encrypt and decrypt data. After Diffie and Hellman published their discovery, a number of

25 Claude E. Shannon was a mathematician who lived from 1916 to 2001.

26 Similar studies were done by Norbert Wiener, who lived from 1894 to 1964.

27 Similar ideas were pursued by Ralph C. Merkle at the University of California at Berkeley [36]. More than a decade ago, the British government revealed that public key cryptography, including the Diffie-Hellman key agreement protocol and the RSA public key cryptosystem, was invented at the Government Communications Headquarters (GCHQ) in Cheltenham in the early 1970s by James H. Ellis, Clifford Cocks, and Malcolm J. Williamson under the name *nonsecret encryption* (NSE). You may refer to the note "The Story of Non-Secret Encryption" written by Ellis in 1997 to get the story (use a search engine to find the note in an Internet archive). Being part of the world of secret services and intelligence agencies, Ellis, Cocks, and Williamson were not allowed to openly talk about their discovery.

public key cryptosystems were developed and proposed. Some of these systems are still in use, such as RSA [37] and Elgamal [38]. Other systems, such as the ones based on the knapsack problem,[28] have been broken and are no longer in use. Some practically important public key cryptosystems are overviewed and discussed in Part III of this book.

Since the early 1990s, we have seen a wide deployment and massive commercialization of cryptography. Today, many companies develop, market, and sell cryptographic techniques, mechanisms, services, and products (implemented in hardware or software) on a global scale. There are cryptography-related conferences and trade shows[29] one can attend to learn more about products that implement cryptographic techniques, mechanisms, and services. One must be cautious here, because the term *crypto* is sometimes also used in a more narrow sense that is limited to blockchain-based distributed ledger technologies (DLTs). This is especially true since the rise of Bitcoin, idxrEthereum, and a few other cryptocurrencies.

The major goal of this book is to provide a basic understanding of what is actually going on in the field. If you want to learn more about the practical use of cryptography to secure Internet and Web applications, you may also refer to [39–42] or any other book about Internet and Web security in general, and Internet security protocols in particular. These practical applications of cryptography are not addressed (or repeated) in this book.

In Section D.5, we briefly introduce the notion of a quantum computer. As of this writing, the issue of whether it is ever possible to build and operate a sufficiently large and stable quantum computer is still open and controversially discussed in the community. But if such a computer can be built, then we know that many cryptosystems in use today can be broken efficiently. This applies to almost all public key cryptosystems (because these systems are typically based on the integer factorization problem or discrete logarithm problem that can both be solved on a quantum computer in polynomial time), and it partly applies to secret key cryptosystems (because it is known how to reduce the steps required to perform an exhaustive key search for an n-bit cipher from 2^n to $2^{n/2}$).

Against this background, people have started to look for cryptographic primitives that remain secure even if sufficiently large and stable quantum computers can be built and operated. The resulting area of research is known as *post-quantum*

28 The *knapsack problem* is a well-known problem in computational complexity theory and applied mathematics. Given a set of items, each with a cost and a value, determine the number of each item to include in a collection so that the total cost is less than some given cost and the total value is as large as possible. The name derives from the scenario of choosing treasures to stuff into your knapsack when you can only carry so much weight.

29 The most important trade show is the RSA Conference that is held annually. Refer to https://www.rsaconference.com for more information.

cryptography (PQC).[30] The goal is to design, implement, and deploy quantum-safe cryptosystems. If y refers to the time it takes to come up with such cryptosystems, x the time information needs to be secure, and z the time it takes to build a practically relevant, i.e., sufficiently large and stable, quantum computer, then *Mosca's Theorem*[31] states that one should worry if $x + y > z$. This is not mathematically precise, but it still gives some hints about when to worry about quantum computers and PQC.

In the past couple of years, PQC has attracted a lot of interest and funding, and many researchers have come up with proposals for PQC. In the case of secret key cryptography, resistance against quantum computers can be provided by doubling the key length. This is simple and straightforward. In the case of public key cryptography, however, things are more involved and new design paradigms are needed here. This is where code-based, hash-based, lattice-based, isogeny-based, and multivariate-based cryptosystems may come into play. As further addressed in Section 18.3, there is a NIST competition going on to evaluate and standardize post-quantum public key cryptographic algorithms to be used in the field. This is in fact a hot topic in contemporary cryptographic research.

1.4 OUTLINE OF THE BOOK

The rest of this book is organized as follows:

- In Chapter 2, "Cryptographic Systems Overview," we introduce, briefly overview, and put into perspective the three classes of cryptographic systems (i.e., unkeyed cryptosystems, secret key cryptosystems, and public key cryptosystems) with their major representatives.
- In Chapter 3, "Random Generators," we begin Part I on unkeyed cryptosystems by elaborating on how to generate random values or bits.
- In Chapter 4, "Random Functions," we introduce the notion of a random function that plays a pivotal role in contemporary cryptography. Many cryptosystems used in the field can, at least in principle, be seen as a random function.
- In Chapter 5, "One-Way Functions," we focus on functions that are one-way. Such functions are heavily used in public key cryptography.
- In Chapter 6, "Cryptographic Hash Functions," we conclude Part I by outlining cryptographic hash functions and their use in contemporary cryptography.

30 A somehow better term would be *quantum-resistant cryptography*, but PQC has prevailed.
31 The theorem was originally proposed by Michele Mosca (https://faculty.iqc.uwaterloo.ca/mmosca/).

- In Chapter 7, "Pseudorandom Generators," we begin Part II on secret key cryptosystems by elaborating on how to use a pseudorandom generator seeded with a secret key to generate pseudorandom values (instead of truly random ones).
- In Chapter 8, "Pseudorandom Functions," we do something similar for random functions: We explain how to construct pseudorandom functions that use a secret key as input and generate an output that looks as if it were generated by a random function.
- In Chapter 9, "Symmetric Encryption," we overview and discuss the symmetric encryption systems that are most frequently used in the field.
- In Chapter 10, "Message Authentication," we address message authentication and explain how secret key cryptography can be used to generate and verify message authentication codes.
- In Chapter 11, "Authenticated Encryption," we conclude Part II by combining symmetric encryption and message authentication in authenticated encryption. In many regards, AE represents the state of the art of cryptography as it stands today.
- In Chapter 12, "Key Establishment," we begin Part III on public key cryptosystems by elaborating on the practically relevant problem of how to establish a secret key that is shared between two or more entities.
- In Chapter 13, "Asymmetric Encryption," we overview and discuss the asymmetric encryption systems that are most frequently used in the field.
- In Chapter 14, "Digital Signatures," we elaborate on digital signatures and respective digital signature systems (DSSs).
- In Chapter 15, "Zero-Knowledge Proofs of Knowledge," we explore the notion of an interactive proof system that may have the zero-knowledge property, and discuss its application in zero-knowledge proofs of knowledge for entity authentication.
- In Chapter 16, "Key Management," we begin Part IV by discussing some aspects related to key management that represents the Achilles' heel of applied cryptography.
- In Chapter 17, "Summary," we conclude with some remarks about the current state of the art in cryptography.

- In Chapter 18, "Outlook," we predict possible and likely future developments and trends in the field, including PQC.

This book includes several appendixes. Appendixes A to D summarize the mathematical foundations and principles, including discrete mathematics (Appendix A), probability theory (Appendix B), information theory (Appendix C), and complexity theory (Appendix D). Finally, a list of mathematical symbols and a list of abbreviations and acronyms are provided at the end of the book.

Note that cryptography is a field of study that is far too broad to be addressed in a single book, and that one has to refer to additional and more in-depth material, such as the literature referenced at the end of each chapter, to learn more about a particular topic. Another possibility to learn more is to experiment and play around with cryptographic systems. As mentioned in the Preface, there are several software packages that can be used for this purpose, among which CrypTool is a particularly good choice.

The aims of this book are threefold: (1) to provide an overview, (2) to give an introduction into each of the previously mentioned topics and areas of research and development, and (3) to put everything into perspective. Instead of trying to be comprehensive and complete, this book tries to ensure that you still see the forest for the trees. In the next chapter, we delve more deeply into the various representatives of the three classes of cryptosystems introduced above.

References

[1] Shirey, R., *Internet Security Glossary, Version 2*, RFC 4949 (FYI 36), August 2007.

[2] Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996.

[3] Stamp, M., and R.M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*. John Wiley & Sons, New York, 2007.

[4] Swenson, C., *Modern Cryptanalysis: Techniques for Advanced Code Breaking*. Wiley Publishing, Indianapolis, IN, 2008.

[5] Joux, A., *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, Boca Raton, FL, 2009.

[6] Cole, E., *Hiding in Plain Sight: Steganography and the Art of Covert Communication*. Wiley Publishing, Indianapolis, IN, 2003.

[7] Wayner, P., *Disappearing Cryptography: Information Hiding — Steganography and Watermarking*, 3rd edition. Morgan Kaufmann Publishers, Burlington, MA, 2009.

[8] Arnold, M., M. Schmucker, and S.D. Wolthusen, *Techniques and Applications of Digital Watermarking and Content Protection*. Artech House Publishers, Norwood, MA, 2003.

[9] Katzenbeisser, S., and F. Petitcolas, *Information Hiding*. Artech House Publishers, Norwood, MA, 2015.

[10] Kelsey, J., B. Schneier, and D. Wagner, "Protocol Interactions and the Chosen Protocol Attack," *Proceedings of the 5th International Workshop on Security Protocols*, Springer-Verlag, 1997, pp. 91–104.

[11] Oppliger, R., "Disillusioning Alice and Bob," *IEEE Security & Privacy*, Vol. 15, No. 5, September/October 2017, pp. 82–84.

[12] Diffie, W., and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, IT-22(6), 1976, pp. 644–654.

[13] Maurer, U.M., "Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms," *Proceedings of CRYPTO '94*, Springer-Verlag, LNCS 839, 1994, pp. 271–281.

[14] Rabin, M.O., "Digitalized Signatures and Public-Key Functions as Intractable as Factorization," MIT Laboratory for Computer Science, MIT/LCS/TR-212, 1979.

[15] Bellare, M., and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 1993, pp. 62-73.

[16] Anderson, R., "Why Cryptosystems Fail," *Communications of the ACM*, Vol. 37, No. 11, November 1994, pp. 32–40.

[17] Halderman, J.A., et al., "Lest We Remember: Cold Boot Attacks on Encryption Keys," *Communications of the ACM*, Vol. 52, No. 5, May 2009, pp. 91–98.

[18] Anderson, R., and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, November 1996, pp. 1–11.

[19] Anderson, R., and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," *Proceedings of the 5th International Workshop on Security Protocols*, Springer-Verlag, LNCS 1361, 1997, pp. 125–136.

[20] Kocher, P., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Proceedings of CRYPTO '96*, Springer-Verlag, LNCS 1109, 1996, pp. 104–113.

[21] Brumley, D., and D. Boneh, "Remote timing attacks are practical," *Proceedings of the 12th Usenix Security Symposium*, USENIX Association, 2003.

[22] Kocher, P., J. Jaffe, and B. Jun, "Differential Power Analysis," *Proceedings of CRYPTO '99*, Springer-Verlag, LNCS 1666, 1999, pp. 388–397.

[23] Boneh, D., R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Proceedings of EUROCRYPT '97*, Springer-Verlag, LNCS 1233, 1997, pp. 37–51.

[24] Biham, E., and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Proceedings of CRYPTO '97*, Springer-Verlag, LNCS 1294, 1997, pp. 513–525.

[25] Bleichenbacher, D., "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1," *Proceedings of CRYPTO '98*, Springer-Verlag, LNCS 1462, 1998, pp. 1–12.

[26] Asonov, D., and R. Agrawal, "Keyboard Acoustic Emanations," *Proceedings of IEEE Symposium on Security and Privacy*, 2004, pp. 3–11.

[27] Zhuang, L., Zhou, F., and J.D. Tygar, "Keyboard Acoustic Emanations Revisited," *Proceedings of ACM Conference on Computer and Communications Security*, November 2005, pp. 373–382.

[28] Micali, S., and L. Reyzin, "Physically Observable Cryptography," *Proceedings of Theory of Cryptography Conference (TCC 2004)*, Springer-Verlag, LNCS 2951, 2004, pp. 278–296.

[29] Renauld, M., et al., "A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices," *Proceedings of EUROCRYPT 2011*, Springer-Verlag, LNCS 6632, 2011, pp. 109–128.

[30] Kerckhoffs, A., "La Cryptographie Militaire," *Journal des Sciences Militaires*, Vol. IX, January 1883, pp. 5–38, February 1883, pp. 161-191.

[31] Kahn, D., *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, New York, 1996.

[32] Bauer, F.L., *Decrypted Secrets: Methods and Maxims of Cryptology*, 2nd edition. Springer-Verlag, New York, 2000.

[33] Levy, S., *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Viking Penguin, New York, 2001.

[34] Shannon, C.E., "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, No. 3/4, July/October 1948, pp. 379–423/623–656.

[35] Shannon, C.E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, Vol. 28, No. 4, October 1949, pp. 656–715.

[36] Merkle, R.C., "Secure Communication over Insecure Channels," *Communications of the ACM*, Vol. 21, No. 4, April 1978, pp. 294–299.

[37] Rivest, R.L., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120–126.

[38] Elgamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm," *IEEE Transactions on Information Theory*, Vol. 31, No. 4, 1985, pp. 469–472.

[39] Oppliger, R., *Internet and Intranet Security*, 2nd edition. Artech House Publishers, Norwood, MA, 2002.

[40] Oppliger, R., *Security Technologies for the World Wide Web*, 2nd edition. Artech House Publishers, Norwood, MA, 2003.

[42] Oppliger, R., *End-to-End Encrypted Messaging*. Artech House Publishers, Norwood, MA, 2020.

[41] Oppliger, R., *SSL and TLS: Theory and Practice*, 3rd edition. Artech House Publishers, Norwood, MA, 2023.

Chapter 2

Cryptographic Systems Overview

As mentioned in Section 1.2.1, there are three classes of cryptographic systems: unkeyed cryptosystems, secret key cryptosystems, and public key cryptosystems. In this chapter, we introduce, overview, and provide some preliminary definitions for the most important representatives of these classes in Sections 2.1–2.3, and we conclude with some final remarks in Section 2.4. All cryptosystems mentioned in this chapter will be revisited and more thoroughly addressed in later chapters of the book.

2.1 UNKEYED CRYPTOSYSTEMS

According to Definition 1.5, unkeyed cryptosystems use no secret parameters. The most important representatives of this class are random generators, random functions, one-way functions, and cryptographic hash functions. We address them in this particular order.

2.1.1 Random Generators

Randomness is the most important ingredient for cryptography, and most cryptographic systems in use today depend on some form of randomness. This is certainly true for key generation and probabilistic encryption, but it is also true for many other cryptographic algorithms and systems. In Section 9.3, we will see that a perfectly secret encryption system (i.e., the one-time pad), requires a random bit for the encryption of every single plaintext message bit. This means that the one-time pad (in order to provide perfect secrecy) requires huge quantities of random bits. But also in many other cases do we need ways to generate sequences of random values

or bits. This is where the notions of a *random generator* or a *random bit generator* as captured in Definition 2.1 come into play.

Definition 2.1 (Random generator) *A* random generator *is a device or entity that outputs a sequence of statistically independent and unbiased values. If the output values are bits, then the random generator is also called a* random bit generator.

A random bit generator is depicted in Figure 2.1 as a gray box. It is important to note that such a generator has no input, and that it only generates an output. Also, because the output is a sequence of statistically independent and unbiased bits, all bits must occur with the same probability; that is, $\Pr[0] = \Pr[1] = 1/2$, or—more generally—all 2^k different k-tuples of bits must occur with the same probability $1/2^k$ for all integers $k \geq 1$. Luckily, there are statistical tests that can be used to verify these properties. Passing these tests is a necessary but usually not sufficient prerequisite for the output of a random generator to be suitable for cryptographic purposes and applications.

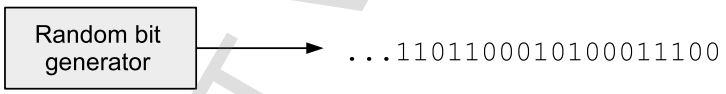


Figure 2.1 A random bit generator.

Ideally, a random (bit) generator is a physical device, but it may also be a logical entity that behaves like a physical device. In either case, it cannot be implemented in a deterministic way. Instead, it must be inherently nondeterministic, meaning that an implementation must use some physical events or phenomena (for which the outcomes are impossible to predict). Alternatively speaking, every (true) random generator requires a naturally occurring source of randomness. The engineering task of finding such a source and exploiting it in a device that may serve as a generator of binary sequences that are free of biases and correlations is a very challenging one. As mentioned above, the output of such a generator must fulfill statistical tests, but it must also withstand various types of sophisticated attacks.

Random generators and their design principles and security properties are further addressed in Chapter 3.

2.1.2 Random Functions

As mentioned above, a random generator is to output some random-looking values. In contrast, a *random function* (that is sometimes also called a *random oracle*) is not characterized by its output, but rather by the way it is chosen. This idea is captured in Definition 2.2.

Definition 2.2 (Random function) *A* random function *is a function $f : X \to Y$ that is chosen randomly from $\mathrm{Funcs}[X, Y]$, i.e., the set of all functions that map elements of the domain X to elements of the codomain Y.*

For input value $x \in X$, a random function can output any value $y = f(x) \in f(X) \subseteq Y$. The only requirement is that the same input x always maps to the same output y. Except for that, everything is possible and does not really matter (for the function to be random).

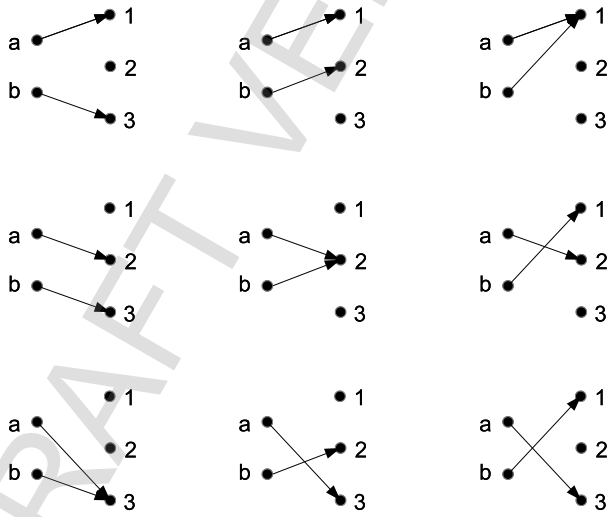


Figure 2.2 Elements of $\mathrm{Funcs}[\{a, b\}, \{1, 2, 3\}]$.

Note that there are $|Y|^{|X|}$ functions in $\mathrm{Funcs}[X, Y]$, and this number is incredibly large—even for moderately sized X and Y. If, for example, $X = \{a, b\}$ and $Y = \{1, 2, 3\}$, then $\mathrm{Funcs}[X, Y]$ comprises $3^2 = 9$ functions, and this number is sufficiently small that its elements can be illustrated in Figure 2.2. If we moderately

increase the sets and let X be the set of all 2-bit strings and Y the set of all 3-bit strings, then $|X| = 2^2$ and $|Y| = 2^3$, and hence $\mathrm{Funcs}[X, Y]$ already comprises $(2^3)^{2^2} = (2^3)^4 = 2^{12} = 4,096$ functions. What this basically means is that there are $4,096$ possible functions to map 2-bit strings to 3-bit strings. This can no longer be illustrated on a paper sheet. Let us consider an even more realistic setting, in which X and Y are both 128 bits long. Here, $\mathrm{Funcs}[X, Y]$ comprises

$$(2^{128})^{2^{128}} = 2^{128 \cdot 2^{128}} = 2^{2^7 \cdot 2^{128}} = 2^{2^{135}}$$

functions, and this number is so large that it would require 2^{135} bits, if one wanted to number the functions and use an index to refer to a particular function from $\mathrm{Funcs}[X, Y]$. This is clearly impractical.

Also note that random functions are not meant to be implemented in the field. Instead, they represent conceptual constructs that are mainly used in security proofs. Referring to the security game of Figure 1.2 in Section 1.2.2.1, the random function yields the ideal system and it is shown that no adversary can tell the real system (for which the security needs to be proven) apart from it. If this can in fact be shown, then the real system behaves like a random function, and this, in turn, means that the adversary must try out all possibilities. Since there are so many possibilities, this task can be assumed to be computationally infeasible, and hence one can conclude that the real system is secure for all practical purposes.

If $X = Y$ and $\mathrm{Funcs}[X, X]$ is restricted to the set of all possible permutations of X (i.e., $\mathrm{Perms}[X]$), then we can say that a *random permutation* is a randomly chosen permutation from $\mathrm{Perms}[X]$. Everything said above about random functions is also true for random permutations. We revisit the notions of random functions and random permutations, as well as the way they are used in cryptography in Chapter 4.

2.1.3 One-Way Functions

Informally speaking, a function $f : X \to Y$ is one way if it is easy to compute but hard to invert. Referring to complexity theory, *easy* means that the computation can be done efficiently (i.e., in polynomial time), whereas *hard* means that it is not known how to do the computation efficiently, that is, no efficient algorithm is known to exist.[1] More formally, the notion of a *one-way function* is captured in Definition 2.3 and illustrated in Figure 2.3.

Definition 2.3 (One-way function) *A function $f : X \to Y$ is* one way *if $f(x)$ can be computed efficiently for all $x \in X$, but $f^{-1}(f(x))$ cannot be computed efficiently, meaning that $f^{-1}(y)$ cannot be computed efficiently for $y \in_R Y$.*

1 Note that it is not impossible that such an algorithm exists; it is just not known.

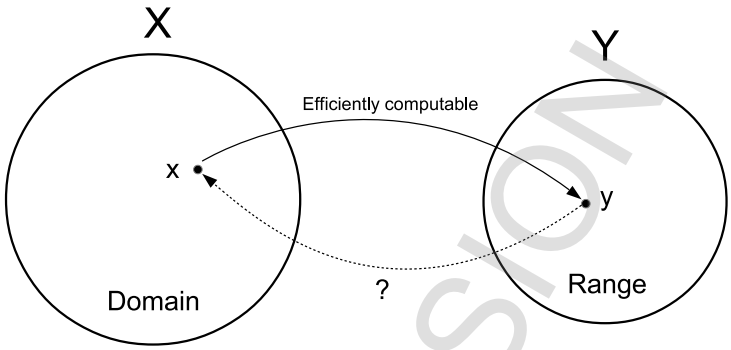


Figure 2.3 A one-way function.

In this definition, X represents the domain of f, Y represents the range, and the expression $y \in_R Y$ reads as "an element y that is randomly chosen from Y." Consequently, it must be possible to efficiently compute $f(x)$ for all $x \in X$, whereas it must not—or only with a negligible probability—be possible to compute $f^{-1}(y)$ for a y that is randomly chosen from Y. To be more precise, one must state that it may be possible to compute $f^{-1}(y)$, but that the entity that wants to do the computation does not know how to do it. In either case, Definition 2.3 is not precise in a mathematically strong sense, because we have not yet defined what an efficient computation is. In essence, a computation is said to be efficient if the (expected) running time of the algorithm that does the computation is bounded by a polynomial in the length of the input. Otherwise (i.e., if the expected running time is not bounded by a polynomial in the length of the input), the algorithm requires super-polynomial time and is said to be inefficient. For example, an algorithm that requires exponential time is clearly superpolynomial. This notion of efficiency (and the distinction between polynomial and superpolynomial running time algorithms) is yet coarse, but still the best we have to work with.

There are many real-world examples of one-way functions. If, for example, we have a telephone book, then the function that assigns a telephone number to each name is easy to compute (because the names are sorted alphabetically) but hard to invert (because the telephone numbers are not sorted numerically). Also, many physical processes are inherently one way. If, for example, we smash a bottle into pieces, then it is generally infeasible to put the pieces together and reassemble the

bottle. Similarly, if we drop a bottle from a bridge, then it falls down, whereas the reverse process never occurs by itself. Last but not least, time is inherently one way, and it is (currently) not known how to travel back in time. As a consequence of this fact, we continuously age and have no possibility to make ourselves young again.

In contrast to the real world, there are only a few mathematical functions conjectured to be one way. The most important examples are centered around *modular exponentiation*: Either $f(x) = g^x \pmod m$, $f(x) = x^e \pmod m$, or $f(x) = x^2 \pmod m$ for a properly chosen modulus m. While the argument x is in the exponent in the first function, x represents the base of the exponentiation function in the other two functions. Inverting the first function requires computing discrete logarithms, whereas inverting the second (third) function requires computing e-th (square) roots. The three functions are used in many public key cryptosystems: The first function is, for example, used in the Diffie-Hellman key exchange protocol (Section 12.3), the second function is used in the RSA public key cryptosystem (Section 13.3.1), and the third function is used in the Rabin encryption system (Section 13.3.2). We will discuss all of these systems later in the book. It is, however, important to note that none of these functions has been shown to be one way in a mathematically strong sense, and that it is theoretically not even known whether one-way functions exist at all. This means that the one-way property of these functions is just an assumption that may turn out to be wrong (or illusory) some day in the future—we don't think so, but it may still be the case.

In the general case, a one-way function cannot be inverted efficiently. But there may still be some one-way functions that can be inverted efficiently, if and—as it is hoped—only if some extra information is known. This brings in the notion of a *trapdoor (one-way) function* as captured in Definition 2.4.

Definition 2.4 (Trapdoor function) *A one-way function $f : X \to Y$ is a* trapdoor function *(or a* trapdoor one-way function*, respectively), if there is some extra information (i.e., the* trapdoor*) with which f can be inverted efficiently, i.e., $f^{-1}(f(x))$ can be computed efficiently for all $x \in X$ or $f^{-1}(y)$ can be computed efficiently for $y \in_R Y$.*

Among the functions mentioned above, $f(x) = x^e \pmod m$ and $f(x) = x^2 \pmod m$ have a trapdoor, namely the prime factorization of m. Somebody who knows the prime factors of m can also efficiently invert these functions. In contrast, the function $f(x) = g^x \pmod m$ is not known to have a trapdoor if m is a prime number.

The mechanical analog of a trapdoor (one-way) function is a padlock. It can be closed by everybody (if it is in unlocked state), but it can be opened only by somebody who holds the proper key. In this analogy, a padlock without a keyhole represents a one-way function. In the real world, this is not a particularly useful

artifact, but in the digital world, as we will see, there are many applications that make use of it.

If X and Y are the same, then a one-way function $f : X \to X$ that is a permutation (i.e., $f \in Perms[X]$), is called a *one-way permutation*. Similar to Definition 2.4, a one-way permutation $f : X \to X$ is a *trapdoor permutation* (or a *trapdoor one-way permutation*, respectively), if it has a trapdoor. Consequently, one-way permutations and trapdoor permutations are special cases of one-way functions and trapdoor functions, namely ones in which the domain and the range are the same and the functions themselves are permutations.

One-way functions including trapdoor functions, one-way permutations, and trapdoor permutations are further addressed in Chapter 5. In this chapter, we will also explain why one has to consider families of such functions to be mathematically correct. In Part III of the book, we then elaborate on the use one-way and trapdoor functions in public key cryptography.

2.1.4 Cryptographic Hash Functions

Hash functions are widely used and have many applications in computer science. Informally speaking, a hash function is an efficiently computable function that takes an arbitrarily large input and generates an output of a usually much smaller size. This idea is captured in Definition 2.5 and illustrated in Figure 2.4.

Definition 2.5 (Hash function) *A function $h : X \to Y$ is a* hash function, *if $h(x)$ can be computed efficiently for all $x \in X$ and $|X| \gg |Y|$.*

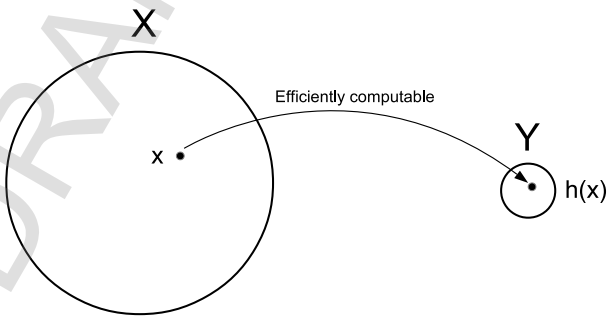


Figure 2.4 A hash function.

The elements of X and Y are typically strings of characters from a given alphabet. If Σ_{in} is the input alphabet and Σ_{out} is the output alphabet, then a hash function h can be written as $h : \Sigma_{in}^{*} \rightarrow \Sigma_{out}^{n}$, or $h : \Sigma_{in}^{n_{max}} \rightarrow \Sigma_{out}^{n}$ if the input size is restricted to n_{max} for some technical reasons.[2] In either case, the output is n characters long. In many practical settings, Σ_{in} and Σ_{out} are identical and refer to the binary alphabet $\Sigma = \{0, 1\}$. In such a setting, the hash function h takes as input an arbitrarily long bitstring and generates as output a bitstring of fixed size n.

In cryptography, we are talking about strings that are a few hundred bits long. Also, we are talking about hash functions that have specific (security) properties, such as one-wayness (or preimage resistance, respectively), second-preimage resistance, and/or collision resistance. These properties are introduced and fully explained in Chapter 6. In the meantime, it suffices to know that a hash function used in cryptography (i.e., a cryptographic hash function), must fulfill two basic requirements:

- On the one hand, it must be hard to invert the function; that is, the function is one-way or preimage resistant.[3]
- On the other hand, it must be hard to find a collision, meaning that it is either hard to find a second preimage for a given hash value; that is, the function is second-preimage resistant, or it is hard to find two preimages that hash to the same value (i.e., the function is collision resistant).

According to Definition 2.6, the first requirement can be combined with any of the two possibilities from the second requirement.

Definition 2.6 (Cryptographic hash function) *A hash function h is* cryptographic, *if it is either one-way and second-preimage resistant or one-way and collision resistant.*

Cryptographic hash functions have many applications in cryptography. Most importantly, a cryptographic hash function h can be used to hash arbitrarily sized messages to bitstrings of fixed size. This is illustrated in Figure 2.5, where the ASCII-encoded message "This is a file that includes some important but long statements. Consequently, we may need a short representation of this file." is hashed to `0x492165102a1a9e3179a2139d429e32ce4f7fd837` (in hexadecimal notation). This value represents the *fingerprint* or *digest* of the message and—in some sense—stands for it. The second-preimage resistance property implies that it is difficult—or even computationally infeasible—to find another message that hashes to the same value. It also implies that a minor modification of the message leads to

2 One such reason may be that the input length must be encoded in a fixed-length field in the padding.
3 The two terms are used synonymously here.

a completely different hash value that looks random. If, for example, a second point were added to the message given above, then the resulting hash value would be `0x2049a3fe86abcb824d9f9bc957f00cfa7c1cae16` (that is completely independent from `0x492165102a1a9e3179a2139d429e32ce4f7fd837`). If the collision resistance property is required, then it is even computationally infeasible to find two arbitrary messages that hash to the same value.[4]

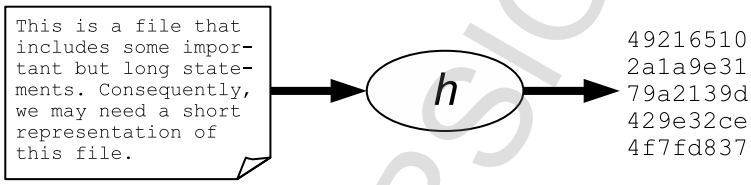


Figure 2.5 A cryptographic hash function.

Examples of cryptographic hash functions that are used in the field are MD5, SHA-1 (depicted in Figure 2.5), the representatives from the SHA-2 family, and SHA-3 or KECCAK. These functions generate hash values of different sizes.[5]

Cryptographic hash functions, their design principles, and their security properties are further addressed in Chapter 6.

Random generators, random functions, one-way functions, and cryptographic hash functions are unkeyed cryptosystems that are omnipresent in cryptography, and that are used as building blocks in more sophisticated cryptographic systems and applications. The next class of cryptosystems we look at are secret key cryptosystems. Referring to Definition 1.6, these cryptosystems use secret parameters that are shared among all participating entities.

2.2 SECRET KEY CRYPTOSYSTEMS

The most important representatives of secret key cryptosystems are pseudorandom generators, pseudorandom functions, systems for symmetric encryption and message authentication, as well as authenticated encryption. For all of these systems we

4 Because the task of finding two arbitrary messages that hash to the same value is simpler than finding a message that hashes to a given value, collision resistance is a stronger property than second-preimage resistance.

5 The output of MD5 is 128 bits long. The output of SHA-1 is 160 bits long. Many other hash functions generate an output of variable size.

briefly explain what they are all about and what it means by saying that they are secure. Note, however, that the notion of security will be explained in more detail in the respective chapters in Part II of the book.

2.2.1 Pseudorandom Generators

In Section 2.1.1, we introduced the notion of a random generator that can be used to generate random values. If a large number of such values is needed, then it may be more appropriate to use a *pseudorandom generator* (PRG) instead of—or rather in combination with—a true random generator. More specifically, one can use a random generator to randomly generate a short value (i.e., a seed), and a PRG to stretch this short value into a much longer sequence of values that appear to be random. The notions of a PRG and a *pseudorandom bit generator* (PRBG) are captured in Definition 2.7.

Definition 2.7 (PRG and PRBG) *A* PRG *is an efficiently computable function that takes as input a relatively short value of length n, called the* seed*, and generates as output a value of length $l(n)$ with $l(n) \gg n$ that appears to be random (and is therefore called* pseudorandom*). If the input and output values are bit sequences, then the PRG is a PRBG.*

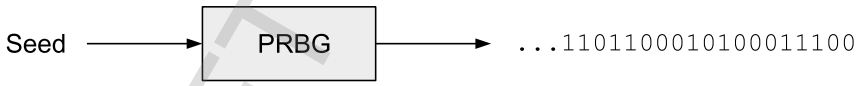


Figure 2.6 A PRBG.

A PRBG is illustrated in Figure 2.6.[6] Formally speaking, a PRBG G is a mapping from $\mathcal{K} = \{0,1\}^n$ to $\{0,1\}^{l(n)}$, where $l(n)$ represents a stretch function (i.e., a function that stretches an n-bit input value into a longer $l(n)$-bit output value with $n < l(n) \leq \infty$):

$$G : \mathcal{K} \longrightarrow \{0,1\}^{l(n)}$$

Note that Definition 2.7 is not precise in a mathematically strong sense, because we have not yet defined what we mean by saying that a bit sequence "appears

6 Note the subtle difference between Figures 2.1 and 2.6. Both generators output bit sequences. But while the random bit generator has no input, the PRBG has a seed that represents the input.

to be random." Unlike a true random generator, a PRG operates deterministically, and this, in turn, means that a PRG always outputs the same values if seeded with the same input value. A PRG thus represents a *finite state machine* (FSM), and hence the sequence of the generated values needs to be cyclic (with a potentially very large cycle). This is why we cannot require that the output of a PRG is truly random, but only that it appears to be so (for a computationally bounded adversary).

This is the starting point for a security definition: We say that a PRG is secure, if its output is indistinguishable from the output of a true random generator. Again referring to the security game of Section 1.2.2.1, we consider an adversary who can have a generator output arbitrarily many values, and his or her task is to decide whether these values are generated by a true random generator or a PRG (i.e., whether they are randomly or pseudorandomly generated). If he or she can do so (with a probability that is better than guessing), then the PRG does not appear to behave like a random generator and is therefore not assumed to be secure. Needless to say that the adversary can employ all statistical tests mentioned in Section 2.1.1 and further explored in Section 3.3 to make his or her decision.

We will explore PRGs and (cryptographically) secure PRGs in more detail in Chapter 7. As suggested by the title of [1], pseudorandomness and PRGs are key ingredients and have many applications in cryptography, such as key generation and additive stream ciphers.

2.2.2 Pseudorandom Functions

We have just seen how to use a PRG to "simulate" a true random generator (and this is why we call the respective generator *pseudorandom* instead of *random*). Following a similar line of argumentation, we may try to "simulate" a random function as introduced in Section 2.1.2 with a *pseudorandom function* (PRF). Remember that a random function $f : X \to Y$ is randomly chosen from $\mathrm{Funcs}[X, Y]$, and that the size of this set; that is $|\mathrm{Funcs}[X, Y]| = |Y|^{|X|}$, is so incredibly large that we cannot number its elements and use an index to refer to a particular function. Instead, we can better use a subset of $\mathrm{Funcs}[X, Y]$ that is sufficiently small so that we can number its elements and use a moderately sized index to refer to a particular function from the subset. If we use a secret key as an index into the subset, then we can have something like a random function without its disadvantages. This is the plan, and a respective definition for a PRF is given in Definition 2.8.[7]

7 The notion of a function family (or family of functions, respectively) is formally introduced in Section A.1.1. Here, it is sufficient to have an intuitive understanding for the term.

Definition 2.8 (PRF) *A* PRF *is a family $F : \mathcal{K} \times X \to Y$ of (efficiently computable) functions, where each $k \in \mathcal{K}$ determines a function $f_k : X \to Y$ that is indistinguishable from a random function; that is, a function randomly chosen from* $\mathrm{Funcs}[X, Y]$.

Note that there are "only" $|\mathcal{K}|$ elements in F, whereas there are $|Y|^{|X|}$ elements in $\mathrm{Funcs}[X, Y]$. This means that we can use a relatively small key to determine a particular function $f_k \in F$, and this function still behaves like a random function, meaning that it is computationally indistinguishable from a truly random function. In our security game this means that, when interacting[8] with either a random function or a PRF, an adversary cannot tell the two cases apart. In other words, he or she cannot or can only tell with a negligible probability whether he or she is interacting with a random function or "only" a pseudorandom one. Such a PRF (that is indistinguishable from a random function) then behaves like a random function and is considered to be "secure," meaning that it can be used for cryptographic purposes and applications.

A *pseudorandom permutation* (PRP) is defined similarly: A PRP is a family $P : \mathcal{K} \times X \to X$ of (efficiently computable) permutations, where each $k \in \mathcal{K}$ determines a permutation $p_k : X \to X$ that is indistinguishable from a random permutation; that is, a permutation randomly chosen from $\mathrm{Perms}[X]$.

PRFs and PRPs are important in modern cryptography mainly because many cryptographic constructions that are relevant in practice can be seen this way: A cryptographic hash function is a PRF (with no key); a key derivation function (KDF) is a PRF with a seed acting as key; a block cipher is a PRP; a PRG can be built from a PRF and vice versa, and so on. PRFs and PRPs and their security properties will be further addressed in Chapter 8.

2.2.3 Symmetric Encryption

When people talk about cryptography, they often refer to confidentiality protection using a *symmetric encryption system* that, in turn, can be used to encrypt and decrypt data. *Encryption* refers to the process that maps a plaintext message to a ciphertext, whereas *decryption* refers to the reverse process (i.e., the process that maps a ciphertext back to the plaintext message). Formally speaking, a symmetric encryption system (or *cipher*) can be defined as suggested in Definition 2.9.

Definition 2.9 (Symmetric encryption system) *Let $\mathcal{M}$ be a plaintext message space,*[9] *$\mathcal{C}$ a ciphertext space, and $\mathcal{K}$ a key space. A* symmetric encryption system *or* cipher *refers to a pair (E, D) of families of efficiently computable functions:*

8 Interacting means that the adversary can have arbitrarily many input values of his or her choice be mapped to respective output values.

9 In some other literature, the plaintext message space is denoted by $\mathcal{P}$.

- $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$ *denotes a family* $\{E_k : k \in \mathcal{K}\}$ *of* encryption functions $E_k : \mathcal{M} \rightarrow \mathcal{C}$*;*
- $D : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M}$ *denotes a family* $\{D_k : k \in \mathcal{K}\}$ *of respective* decryption functions $D_k : \mathcal{C} \rightarrow \mathcal{M}$.

For every message $m \in \mathcal{M}$ *and key* $k \in \mathcal{K}$*, the functions* D_k *and* E_k *must be inverse to each other; that is,* $D_k(E_k(m)) = m$.[10]

In a typical setting, $\mathcal{M} = \mathcal{C} = \{0,1\}^*$ refers to the set of all arbitrarily long binary strings, whereas $\mathcal{K} = \{0,1\}^l$ refers to the set of all l bits long keys. Hence, l stands for the key length of the symmetric encryption system (e.g., $l = 128$).

While the decryption functions need to be deterministic, the encryption functions can be deterministic or probabilistic. In the second case, they usually take some random data as additional input (this is not formalized in Definition 2.9). As will become clear later in the book, probabilistic encryption functions tend to be more secure than deterministic ones.

In the description of a (symmetric or asymmetric) encryption system, we often use an algorithmic notation: Generate then stands for a key generation algorithm (that can be omitted in the symmetric case because the key is just randomly selected from $\mathcal{K}$), Encrypt for an algorithm that implements the encryption function, and Decrypt for an algorithm that implements the decryption function. Using this notation, the working principle of a symmetric encryption system is illustrated in Figure 2.7. First, the Generate algorithm generates a key k by randomly selecting an element from $\mathcal{K}$. This key is distributed to either side of the communication channel (this is why the encryption system is called "symmetric" in the first place), namely to the sender (or the sending device, respectively) on the left side and the recipient (or the receiving device, respectively) on the right side. The sender can encrypt a plaintext message $m \in \mathcal{M}$ with its implementation of the encryption function or Encrypt algorithm and k. The resulting ciphertext $c = E_k(m) \in \mathcal{C}$ is sent to the recipient over the communication channel that can be insecure (drawn as a dotted line in Figure 2.7). On the right side, the recipient can decrypt c with its implementation of the decryption function or Decrypt algorithm and the same key k. If the decryption is successful, then the recipient is able to retrieve and continue to use the original message m.

The characteristic feature of a symmetric encryption system is in fact that k is the same on either side of the communication channel, meaning that k is a secret shared by the sender and the recipient. Another characteristic feature is that the system can operate on individual bits and bytes (typically representing a *stream*

10 In some symmetric encryption systems, it does not matter whether one encrypts first and then decrypts or one decrypts first and then encrypts: that is, $D_k(E_k(m)) = E_k(D_k(m)) = m$.

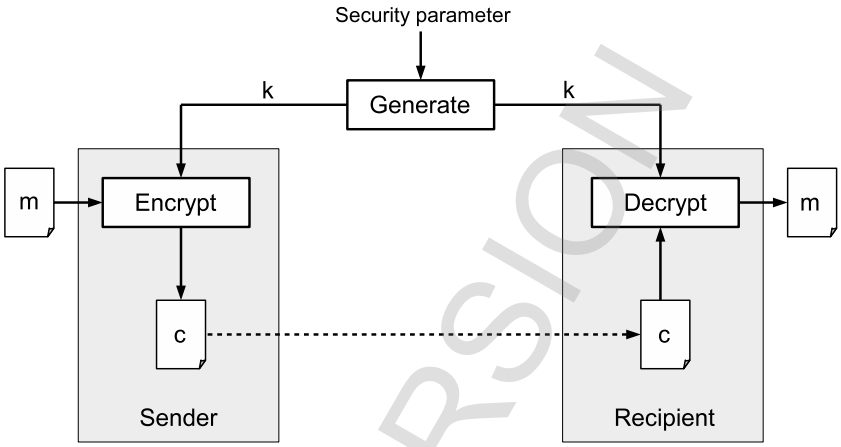


Figure 2.7 The working principle of a symmetric encryption system.

cipher) or on larger blocks (typically representing a *block cipher*). While there are modes of operation that turn a block cipher into a stream cipher, the opposite is not known to be true, meaning that there is no mode of operation that effectively turns a stream cipher into a block cipher.

To make meaningful statements about the security of a symmetric encryption system or cipher, one must define the adversary and the task he or she needs to solve to be successful (Definition 1.8). With regard to the adversary, one must specify his or her computing power and the types of attacks he or she is able to mount, such as ciphertext-only attacks, chosen-plaintext attacks, or even chosen-ciphertext attacks. With regard to the task, one must specify whether he or she must decrypt a ciphertext, determine a key, determine a few bits from either the plaintext or the key, or do something else. Consequently, there are several notions of security one may come up with. If, for example, an adversary has infinite computing power but is still not able to decrypt a ciphertext within a finite amount of time, then the respective cipher is unconditionally or information-theoretically secure. We already mentioned that the one-time pad yields an example of such an information-theoretically secure cipher (that provides perfect secrecy). If the adversary is theoretically able to decrypt a ciphertext within a finite amount of time, but the computing power required to do so is beyond his or her capabilities, then the respective cipher is “only” conditionally or computationally secure. This means that the system can be broken in theory

(e.g., by an exhaustive key search), but the respective attacks are assumed to be computationally infeasible to mount by the adversaries one has in mind.

In Chapter 9, we will introduce, formalize, discuss, and put into perspective several notions of security, including semantic security. If a cipher is semantically secure, then it is computationally infeasible to retrieve any meaningful information about a plaintext message from a given ciphertext, even if the adversary can mount a chosen-plaintext attack, and hence has access to an encryption oracle. All symmetric encryption systems in use today are at least semantically secure. In this chapter, we will also outline ciphers that are either historically relevant, such as the Data Encryption Standard (DES) and RC4, or practically important, such as the Advanced Encryption Standard (AES) and Salsa20/ChaCha20. Note that there are many other ciphers proposed in the literature. The majority of them have been broken, but some still remain secure.

2.2.4 Message Authentication

While encryption systems are to protect the confidentiality of data, there are applications that require rather the authenticity and integrity of data to be protected—either in addition or instead of the confidentiality. Consider, for example, a financial transaction. It is nice to have the confidentiality of this transaction be protected, but it is somehow more important to protect its authenticity and integrity. The typical way to achieve this is to have the sender add an *authentication tag* to the message and to have the recipient verify the tag before he or she accepts the message as being genuine. This is conceptually similar to an error correction code. But in addition to protect a message against transmission errors, an authentication tag also protects a message against tampering and deliberate fraud. This means that the tag itself needs to be protected against an adversary who may try to modify the message and/or the tag.

From a bird's eye perspective, there are two possibilities to construct an authentication tag: Either through the use of public key cryptography and a *digital signature* (as explained later in this book) or through the use of secret key cryptography and a *message authentication code* (MAC[11]). The second possibility is captured in Definition 2.10.

Definition 2.10 (MAC) *A* MAC *is an authentication tag that can be computed and verified with a secret parameter (e.g., a secret key).*

In the case of a message sent from one sender to one recipient, the secret parameter must be shared between the two entities. If, however, the message is sent

11 In some literature, the term *message integrity code* (MIC) is used synonymously and interchangeably. However, this term is not used in this book.

to multiple recipients, then the secret parameter must be shared among the sender and all receiving entities. In this case, the distribution and management of the secret parameter yields a major challenge (and is probably one of the Achilles' heels of the entire system).

Similar to a symmetric encryption system, one can introduce and formally define a system to compute and verify MACs. In this book, we sometimes use the term *message authentication system* to refer to such a system (contrary to many other terms used in this book, this term is not widely used in the literature). It is formalized in Definition 2.11.

Definition 2.11 (Message authentication system) *Let $\mathcal{M}$ be a message space, $\mathcal{T}$ a tag space, and $\mathcal{K}$ a key space. A* message authentication system *then refers to a pair (A, V) of families of efficiently computable functions:*

- $A : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ *denotes a family* $\{A_k : k \in \mathcal{K}\}$ *of* authentication functions $A_k : \mathcal{M} \to \mathcal{T}$*;*
- $V : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{valid, invalid\}$ *denotes a family* $\{V_k : k \in \mathcal{K}\}$ *of* verification functions $V_k : \mathcal{M} \times \mathcal{T} \to \{valid, invalid\}$.

For every message $m \in \mathcal{M}$ and key $k \in \mathcal{K}$, $V_k(m, t)$ must yield $valid$ if and only if t is a valid authentication tag for m and k; that is, $t = A_k(m)$, and hence $V_k(m, A_k(m))$ must yield $valid$.

Typically, $\mathcal{M} = \{0,1\}^*$, $\mathcal{T} = \{0,1\}^{l_{tag}}$ for some fixed tag length l_{tag}, and $\mathcal{K} = \{0,1\}^{l_{key}}$ for some fixed key length l_{key}. It is often the case that $l_{tag} = l_{key} = 128$, meaning that the tags and keys are 128 bits long each.

The working principle of a message authentication system is depicted in Figure 2.8. Again, the Generate algorithm randomly selects a key k from $\mathcal{K}$ that is sent to the sender (on the left side) and the recipient (on the right side). The sender uses the authentication function or Authenticate algorithm to compute an authentication tag t from m and k. Both m and t are sent to the recipient. The recipient, in turn, uses the verification function or Verify algorithm to check whether t is a valid tag with respect to m and k. The resulting Boolean value yields the output of the Verify algorithm; it can either be $valid$ or $invalid$.

To argue about the security of a message authentication system, we must define the adversary and the task he or she must solve to be successful (i.e., to break the security of the system). Similar to symmetric encryption systems, we may consider adversaries with infinite computing power to come up with systems that are unconditionally or information-theoretically secure, or—more realistically—adversaries with finite computing power to come up with systems that are "only" conditionally or computationally secure.

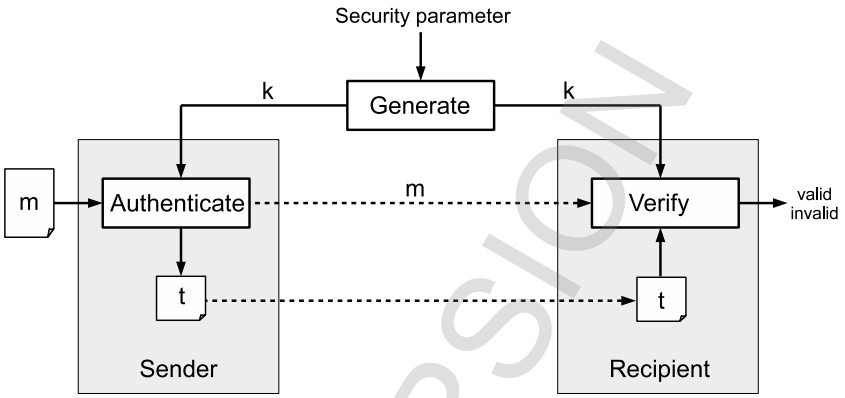


Figure 2.8 The working principle of a message authentication system.

As we will see in Chapter 10, there are message authentication systems that are unconditionally or information-theoretically secure, but require a unique key to authenticate a single message, and there are systems that are "only" conditionally or computationally secure, but can use a single and relatively short key to authenticate multiple messages. Needless to say, this is more appropriate and useful, and hence most systems used in the field are conditionally or computationally secure. Furthermore, the notion of (computational) security we are heading for is unforgeability, meaning that it must be computationally infeasible to generate a valid tag for a new message. This requirement will be clarified and more precisely defined in Chapter 10, when we elaborate on message authentication, MACs, and respective message authentication systems.

2.2.5 Authenticated Encryption

For a long time, people used symmetric encryption systems to encrypt messages and message authentication systems to generate MACs that were then appended to the messages. But it was not clear how (i.e., in what order), the two cryptographic primitives had to be applied and combined to achieve the best level of security. In general, there are three approaches or generic composition methods. Let $m \in \mathcal{M}$ be a message, k_e an encryption key, and k_a an authentication key from respective key spaces. The generic composition methods can then be described as follows:

- In *Encrypt-then-MAC* (EtM) the message m is first encrypted, and the resulting ciphertext is then authenticated, before the ciphertext and the MAC are sent together to the recipient. Mathematically speaking, the data that is sent to the recipient is $E_{k_e}(m) \parallel A_{k_a}(E_{k_e}(m))$. EtM is used, for example, in IP security (IPsec).
- In *Encrypt-and-MAC* (E&M) the message m is encrypted and authenticated independently, meaning that—in contrast to EtM—a MAC is generated for the plaintext and sent together with the ciphertext to the recipient. In this case, the data that is sent to the recipient is $E_{k_e}(m) \parallel A_{k_a}(m)$. E&M is used, for example, in Secure Shell (SSH).
- In *MAC-then-Encrypt* (MtE) a MAC is first generated for the plaintext message, and the message with the appended MAC is then encrypted. The resulting ciphertext (that comprises both the message and the MAC) is then sent to the recipient. In this case, the data that is sent to the recipient is $E_{k_e}(m \parallel A_{k_a}(m))$. MtA was used, for example, in former versions of the SSL/TLS protocols [2].

Since 2001 it is known that encrypting a message and subsequently applying a MAC to the ciphertext (i.e., the EtM method), provides the best level of security [3], and most of today's security protocols follow this approach, i.e., they apply message encryption prior to authentication. More specifically, most people combine message encryption and authentication in what is called *authenticated encryption* (AE) or *authenticated encryption with associated data* (AEAD). AEAD basically refers to AE where all data are authenticated but not all data are encrypted. The exemplary use case is an IP packet, where the payload is encrypted and authenticated but the header is only authenticated (because it must be accessible to the intermediate routers in the clear). AE is further addressed in Chapter 11. Most importantly, there are several modes of operation for block ciphers that provide AE (or AEAD, respectively).

The next class of cryptosystems we look at are public key cryptosystems. According to Definition 1.7, these are cryptosystems that use secret parameters that are not shared among all participating entities, meaning that they are held in private.

2.3 PUBLIC KEY CRYPTOSYSTEMS

Instead of sharing all secret parameters, the entities that participate in a public key cryptosystem hold two distinct sets of parameters: One that is private (collectively referred to as the *private* or *secret key* and abbreviated as sk), and one that is

published (collectively referred to as the *public key* and abbreviated as pk).[12] A necessary but usually not sufficient prerequisite for a public key cryptosystem to be secure is that both keys—the private key and the public key—are yet mathematically related, but it is still computationally infeasible to compute one from the other. Another prerequisite is that the public keys are available in a form that provides authenticity and integrity. If somebody is able to introduce faked public keys, then he or she is usually able to mount very powerful attacks. This is why we usually require public keys to be published in some certified form, and hence the notions of (public key) certificates and public key infrastructures (PKI) come into play. This topic will be further addressed in Section 16.4.

The fact that public key cryptosystems use secret parameters that are not shared among all participating entities suggests that the respective algorithms are executed by different entities, and hence that such cryptosystems are best defined as sets of algorithms (that are then executed by these different entities). We adopt this viewpoint in this book and define public key cryptosystems as sets of algorithms. In the case of an asymmetric encryption system, for example, there is a key generation algorithm Generate, an encryption algorithm Encrypt, and a decryption algorithm Decrypt. The Generate and Encrypt algorithms are usually executed by the sender of a message, whereas the Decrypt algorithm is executed by the recipient(s). As discussed later, other public key cryptosystems may employ other sets of algorithms.

In the following sections, we briefly overview the most important public key cryptosystems used in the field, such as key establishment, asymmetric encryption, and digital signatures. The overview is superficial here, and the technical details are provided in Part II of the book (this also applies to zero-knowledge proofs of knowledge that are not addressed here).

Because public key cryptography is computationally less efficient than secret key cryptography, it usually makes a lot of sense to combine both types of cryptography in *hybrid cryptosystems*. In such a system, public key cryptography is mainly used for authentication and key establishment, whereas secret key cryptography is used for everything else (most notably bulk data encryption). Hybrid cryptosystems are frequently used and very widely deployed in the field. In fact, almost every non-trivial application of cryptography employs some form of hybrid cryptography.

2.3.1 Key Establishment

If two or more entities want to employ and make use of secret key cryptography, then they must share a secret parameter that represents a cryptographic key. Consequently, in a large system many secret keys must be generated, stored, managed,

12 It depends on the cryptosystem whether it matters which set of parameters is used to represent the private key and which set of parameters is used to represent the public key.

used, and destroyed (at the end of their life cycle) in a secure way. If, for example, n entities want to securely communicate with each other, then there are

$$\left(\begin{array}{c} n \\ 2 \end{array} \right) = \frac{n(n-1)}{1 \cdot 2} = \frac{n^2 - n}{2}$$

such keys. This number grows in the order of n^2, and hence the establishment of secret keys is a major practical problem—sometimes called the n^2-problem—and probably the Achilles' heel of the large-scale deployment of secret key cryptography. For example, if $n = 1,000$ entities want to securely communicate with each other, then there are

$$\left(\begin{array}{c} 1,000 \\ 2 \end{array} \right) = \frac{1,000^2 - 1,000}{2} = 499,500$$

keys. Even for moderately large n, the generation, storage, management, usage, and destruction of all such keys is prohibitively expensive and the antecedent distribution of them is next to impossible. Things even get worse in dynamic systems, where entities may join and leave at will. In such a system, the distribution of keys is impossible, because it is not even known in advance who may want to join. This means that one has to establish keys when needed, and there are basically two approaches to achieve this:

- The use of a *key distribution center* (KDC) that provides the entities with the keys needed to securely communicate with each other;
- The use of a *key establishment protocol* that allows the entities to establish the keys themselves.

A prominent example of a KDC is the Kerberos authentication and key distribution system [4]. KDCs in general and Kerberos in particular have many disadvantages. The most important disadvantage is that each entity must unconditionally trust the KDC and share a master key with it. There are situations in which this level of trust is neither justified nor can it be accepted by the participating entities. Consequently, the use of a key establishment protocol that employs public key cryptography yields a viable alternative that is advantageous in most situations and application settings.

In a simple key establishment protocol, an entity randomly generates a key and uses a secure channel to transmit it to the peer entity (or peer entities, respectively). The secure channel can be implemented with any asymmetric encryption system: The entity that randomly generates the key encrypts the key with the public key of the peer entity. This protocol is simple and straightforward, but it has the problem

that the security depends on the quality and security of the key generation process—which yields a PRG. Consequently, it is advantageous to have a mechanism in place that allows two (or even more) entities to establish and agree on a commonly shared key. This is where the notion of a key agreement or key exchange protocol comes into play (as opposed to a key distribution protocol).

Even today, the most important key exchange protocol was proposed by Diffie and Hellman in their landmark paper [5] that opened the field of public key cryptography. It solves a problem that looks impossible to solve: How can two entities that have no prior relationship and do not share any secret use a public channel to agree on a shared secret? Imagine a room in which people can only shout messages at each other. How can two persons in this room agree on a secret? As we will see in Chapter 12, the *Diffie-Hellman key exchange protocol* solves this problem in a simple and ingenious way. In this chapter, we will also discuss a few variants and their security, and say a few words about the use of quantum cryptography as an alternative way of exchanging keying material.

2.3.2 Asymmetric Encryption Systems

Similar to a symmetric encryption system, an asymmetric encryption system can be used to encrypt and decrypt plaintext messages. The major difference between a symmetric and an asymmetric encryption system is that the former employs secret key cryptography and respective techniques, whereas the latter employs public key cryptography and respective techniques.

As already emphasized in Section 2.1.3, an asymmetric encryption system can be built from a trapdoor function—or, more specifically—from a family of trapdoor functions. Each public key pair comprises a public key pk that yields a one-way function and a private key sk that yields a trapdoor (needed to efficiently compute the inverse of the one-way function). To send a secret message to the recipient, the sender uses the recipient's public key and applies the respective one-way function to the plaintext message. The resulting ciphertext is then sent to the recipient. The recipient, in turn, is the only entity that supposedly holds the trapdoor (information) needed to invert the one-way function and to decrypt the ciphertext accordingly. More formally, an asymmetric encryption system can be defined as in Definition 2.12.

Definition 2.12 (Asymmetric encryption system) *An* asymmetric encryption system *consists of the following three efficient algorithms:*

- $\mathsf{Generate}(1^k)$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^k *(in unary notation*[13]*), and generates as output a public key pair* (pk, sk) *that is in line with the security parameter.*
- $\mathsf{Encrypt}(pk, m)$ *is a deterministic or probabilistic encryption algorithm that takes as input a public key* pk *and a plaintext message* m*, and generates as output a ciphertext* $c = \mathsf{Encrypt}(pk, m)$*.*
- $\mathsf{Decrypt}(sk, c)$ *is a deterministic decryption algorithm that takes as input a private key* sk *and a ciphertext* c*, and generates as output a plaintext message* $m = \mathsf{Decrypt}(sk, c)$*.*

For every plaintext message m *and public key pair* (pk, sk)*, the* Encrypt *and* Decrypt *algorithms must be inverse to each other; that is,* $\mathsf{Decrypt}(sk, \mathsf{Encrypt}(pk, m)) = m$*.*

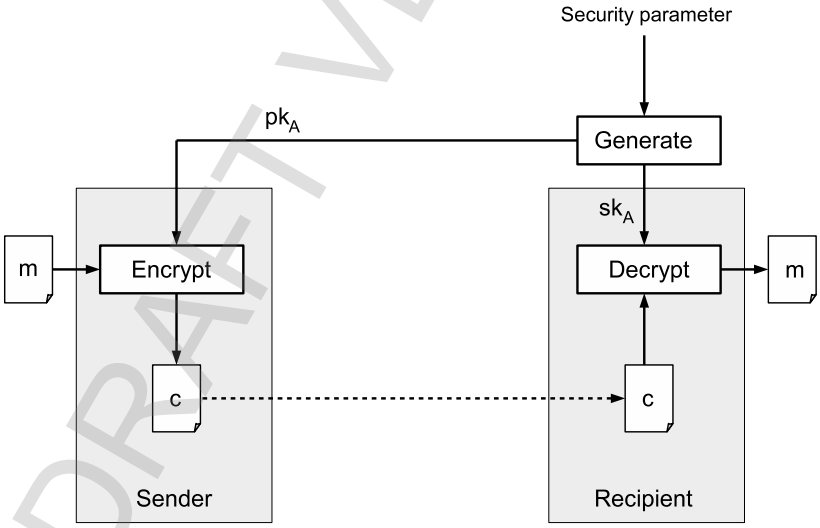


Figure 2.9 The working principle of an asymmetric encryption system.

13 This means that a 1 is repeated k times.

The working principle of an asymmetric encryption system is illustrated in Figure 2.9. At the top of the figure, the Generate algorithm is to generate a public key pair for entity A that is the recipient of a message. In preparation for the encryption, A's public key pk_A is provided to the sender on the left side. The sender then subjects the message m to the one-way function represented by pk_A, and sends the respective ciphertext $c = \mathsf{Encrypt}_{pk_A}(m)$ to A. On the right side, A knows its secret key sk_A that represents a trapdoor to the one-way function and can be used to decrypt c and retrieve the plaintext message $m = \mathsf{Decrypt}_{sk_A}(c)$ accordingly. Hence, the output of the Decrypt algorithm is the original message m.

There are many asymmetric encryption systems that have been proposed in the literature, such as Elgamal, RSA, and Rabin. These systems are based on the three exemplary one-way functions mentioned in Section 2.1.3 (in this order). Because it is computationally infeasible to invert these functions, the systems provide a reasonable level of security—even in their basic forms (that are sometimes called textbook versions).

In Chapter 13, we will elaborate on these asymmetric encryption systems, but we will also address notions of security that cannot be achieved with them. The strongest notion of security is again defined in the game-theoretical setting: An adversary can select two equally long plaintext messages and has one of them encrypted. If he or she cannot tell whether the respective ciphertext is the encryption of the first or the second plaintext message with a probability that is better than guessing, then the asymmetric encryption system leaks no information and is therefore assumed to be (semantically) secure. This may hold even if the adversary has access to a decryption oracle, meaning that he or she can have any ciphertext of his or her choice be decrypted—except, of course, the ciphertext the adversary is challenged with. We will more thoroughly explain this setting and present variants of the basic asymmetric encryption systems that remain secure even in this setting.

2.3.3 Digital Signatures

Digital signatures can be used to protect the authenticity and integrity of messages, or—more generally—data objects. According to [6], a *digital signature* refers to "a value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity." Similarly, the term digital signature is defined as "data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, e.g. by the recipient" in ISO/IEC 7498-2 [7]. Following the second definition, there are two classes of digital signatures:

- If data representing the digital signature is appended to a data unit (or message), then one refers to a *digital signature with appendix*.
- If a data unit is cryptographically transformed in a way that it represents both the data unit (or message) that is signed and the digital signature, then one refers to a *digital signature giving message recovery*. In this case, the data unit is recovered if and only if the signature is successfully verified.

In either case, the entity that digitally signs a data unit or message is called the *signer* or *signatory*, whereas the entity that verifies the digital signature is called the *verifier*. In a typical setting, both the signer and the verifier are computing devices operating on a user's behalf. The formal definitions of the two respective digital signature systems (DSS) are given in Definitions 2.13 and 2.14.

Definition 2.13 (DSS with appendix) *A* DSS with appendix *consists of the following three efficiently computable algorithms:*

- $\mathsf{Generate}(1^k)$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^k*, and generates as output a public key pair* (pk, sk) *that is in line with the security parameter.*
- $\mathsf{Sign}(sk, m)$ *is a deterministic or probabilistic signature generation algorithm that takes as input a signing key* sk *and a message* m*, and generates as output a digital signature* s *for* m*.*
- $\mathsf{Verify}(pk, m, s)$ *is a deterministic signature verification algorithm that takes as input a verification key* pk*, a message* m*, and a purported digital signature* s *for* m*, and generates as output a binary decision whether the signature is valid.*

$\mathsf{Verify}(pk, m, s)$ *must yield* $valid$ *if and only if* s *is a valid digital signature for* m *and* pk*. This means that for every message* m *and every public key pair* (pk, sk)*,* $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m))$ *must yield* $valid$*.*

Definition 2.14 (DSS giving message recovery) *A* DSS giving message recovery *consists of the following three efficiently computable algorithms:*

- $\mathsf{Generate}(1^k)$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^k*, and generates as output a public key pair* (pk, sk) *that is in line with the security parameter.*
- $\mathsf{Sign}(sk, m)$ *is a deterministic or probabilistic signature generation algorithm that takes as input a signing key* sk *and a message* m*, and generates as output a digital signature* s *giving message recovery.*

- Recover(pk, s) *is a deterministic message recovery algorithm that takes as input a verification key* pk *and a digital signature* s*, and generates as output either the message that is digitally signed or a notification indicating that the digital signature is invalid.*

Recover(pk, s) *must yield* m *if and only if* s *is a valid digital signature for* m *and* pk*. This means that for every message* m *and every public key pair* (pk, sk)*,* Recover$(pk,$ Sign$(sk, m))$ *must yield* m*.*

Note that the Generate and Sign algorithms are identical in this algorithmic notation, and that the only difference refers to the Verify and Recover algorithms. While the Verify algorithm takes the message m as input, this value is not needed by the Recover algorithm. Instead, the message m is automatically recovered if the signature turns out to be valid.

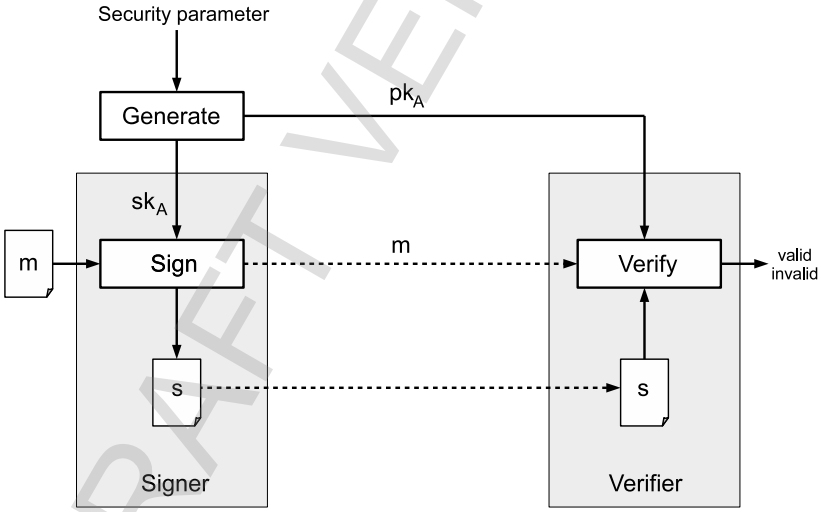


Figure 2.10 The working principle of a DSS with appendix.

The working principle of a DSS with appendix is illustrated in Figure 2.10. This time, the Generate algorithm is applied on the left side (i.e., the signer' side). The signer uses the secret key sk_A (representing the trapdoor) to sign message m; that is, s = Sign(sk_A, m). This message m and the respective signature s are then

sent to the verifier. This verifier, in turn, uses the Verify algorithm to verify s. More specifically, it takes $m, s,$ and pk_A as input values and ouputs either $valid$ or $invalid$ (obviously depending on the validity of the signature).

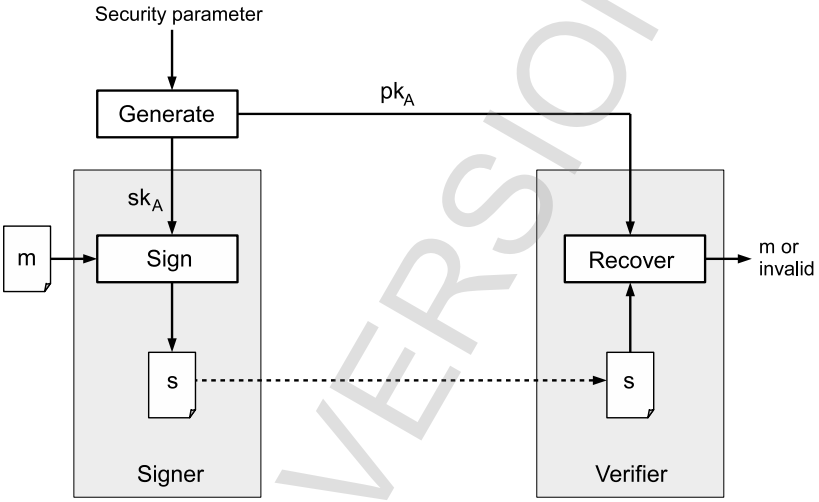


Figure 2.11 The working principle of a DSS giving message recovery.

If the DSS is giving message recovery, then the situation is slightly different. As illustrated in Figure 2.11, the beginning is the same. But instead of sending m and s to the recipient, the signatory only sends s. The signature encodes the message. So when the recipient subjects s to the Recover algorithm, the output is either m (if the signature is valid) or $invalid$.

With the proliferation of the Internet in general and Internet-based electronic commerce in particular, digital signatures and the legislation thereof have become important and timely topics. In fact, many DSS with specific and unique properties have been developed, proposed, and published in the literature. Again, the most important DSSs are overviewed, discussed, and put into perspective in Chapter 14. These are RSA, Rabin, Elgamal, and some variations thereof, such as the Digital Signature Algorithm (DSA) and its elliptic curve version (ECDSA).

Similar to asymmetric encryption systems, the security discussion for digital signatures is nontrivial and subtle, and there are several notions of security discussed in the literature. The general theme is that it must be computationally infeasible for

an adversary to generate a new valid-looking signature, even if he or she has access to an oracle that provides him or her with arbitrarily many signatures for messages of his or her choice. In technical parlance, this means that the DSS must resist existential forgery, even if the adversary can mount an adaptive chosen-message attack. Again, we will revisit this topic and explain the various notions of security for DSSs, as well as some constructions to achieve these levels of security in Chapter 14.

2.4 FINAL REMARKS

In this chapter, we briefly introduced and provided some preliminary definitions for the most important representatives of the three main classes of cryptosystems distinguished in this book, namely unkeyed cryptosystems, secret key cryptosystems, and public key cryptosystems. We want to note (again) that this classification scheme is somewhat arbitrary, and that other classification schemes may be possible as well.

The cryptosystems that are preliminarily defined in this chapter are revisited, more precisely defined (in a mathematically strong sense), discussed, and put into perspective in the remaining chapters of the book. For all of these systems, we also dive more deeply into the question of what it means for such a system to be secure. This leads us to various notions of security that can be found in the literature. Some of these notions are equivalent, whereas others are fundamentally different. In this case, it is interesting to know the exact relationship of the notions, and what notion is actually the strongest one. We then also provide cryptosystems that conform to this (strongest) notion of security.

Following this line of argumentation, it is a major theme in cryptography to better understand and formally define notions of security, and to prove that particular cryptosystems are secure in this sense. It is another theme to start with cryptographic building blocks that are known to be secure, and to ask how these building blocks can be composed or combined in more advanced cryptographic protocols or systems so that their security properties still apply. Alternatively speaking, how can secure cryptographic building blocks be composed or combined in a modular fashion so that the result remains secure? This is an interesting and practically relevant question addressed in cryptographic research areas and frameworks like *universal composability* [8] or *constructive cryptography* [9]. These topics are beyond the scope of this book and not further addressed here. Instead, we "only" provide an overview of the cryptographic building blocks that are available and that are assumed to be secure when considered in isolation. Just keep in mind that a cryptosystem that is secure in isolation does not need to remain secure when combined or composed with others. So, research areas and frameworks like universal composability and

constructive cryptography are crucial for the overall security of a cryptographic application.

References

[1] Luby, M., *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, Princeton, NJ, 1996.

[2] Oppliger, R., *SSL and TLS: Theory and Practice*, 2nd edition. Artech House Publishers, Norwood, MA, 2016.

[3] Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)," *Proceedings of CRYPTO 2001*, Springer-Verlag, LNCS 2139, 2001, pp. 310–331.

[4] Oppliger, R., *Authentication Systems for Secure Networks*. Artech House Publishers, Norwood, MA, 1996.

[5] Diffie, W., and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. 22, No. 6, 1976, pp. 644–654.

[6] Shirey, R., *Internet Security Glossary, Version 2*, Informational RFC 4949 (FYI 36), August 2007.

[7] ISO/IEC 7498-2, *Information Processing Systems—Open Systems Interconnection Reference Model—Part 2: Security Architecture*, 1989.

[8] Canetti, R., "Universally Composable Security: A New Paradigm for Cryptographic Protocols," Cryptology ePrint Archive, *Report 2000/067*, 2000, https://eprint.iacr.org/2000/067.

[9] Maurer, U.M., "Constructive Cryptography – A New Paradigm for Security Definitions and Proofs," *Proceedings of the 2011 International Conference on Theory of Security and Applications (TOSCA 2011)*, Springer-Verlag, LNCS 6993, 2012, pp. 33–56.