Chapter 3

Some Cryptographic Schemes, Protocols, and Security Definitions

3.1 Introduction

3.1.1 Purpose of this Chapter

The purpose of this chapter is to set the stage for the course. We present some basic classical cryptographic schemes and investigate how they can be composed to obtain a cryptographic protocol allowing two parties to communicate securely over an insecure channel or network. We point out the difficulties and subtleties arising if one wants to define security mathematically precisely. A main purpose of the course is to develop the understanding of security definitions and proofs. In summary, the goals of this chapter are:

- to present some important basic cryptographic schemes,
- to start discussing how their security could be defined,
- to present some ad-hoc ways of using some of these schemes in a cryptographic protocol,
- to discover various flaws of such naïvely designed protocols,
- to present a well-designed cryptographic protocol for secure communication,
- to start discussing how the security of this protocol can be defined and proved, and
- to introduce several types of computational problems arising in cryptography.

It is our goal to challenge (and hit the limits of) an informal treatment of cryptography, thus providing the motivation for a rigorous mathematical treatment.

3.1.2 The Story of this Chapter

In this chapter we explore how two parties, called Alice and Bob, can establish a secure connection between them, similar to what the widely-used and well-known TLS protocol tries to achieve. We discuss this problem in a somewhat ad-hoc manner, in line with how security protocols are often discussed informally in the literature. We proceed step by step, introducing a new scheme whenever we need it to solve a problem that still remains to be solved.

3.2 Symmetric Encryption

Let us begin the story. If we want to transmit information securely, we certainly need to protect the confidentiality, i.e., the message must remain secret from an eavesdropper. This is achieved by using encryption, assuming (for now) that Alice and Bob share a secret key. The confidentiality property is also called secrecy or privacy.[1]

3.2.1 Concept and Definition

The model of a *symmetric cryptosystem*, also called a *symmetric encryption scheme*, is shown in Figure 3.1. The message m, often called *plaintext*, is encrypted by a sender Alice, using a *secret key* k, resulting in the *ciphertext* c. The encryption transformation can be probabilistic, i.e., involve fresh random bits for each message to be encrypted.

The ciphertext is transmitted to the receiver Bob over a non-confidential channel to which an adversary has access, either only read access, or read/write access, depending on the assumed model. The secret key is shared between sender and receiver by some secure means, e.g. by sending it before-hand over a secure channel, for instance by use of a trusted courier. The receiver can decrypt the ciphertext, using the secret key, resulting in the plaintext.

Formally, a symmetric cryptosystem can be defined as follows:

Definition 3.1. A *symmetric cryptosystem* (or *symmetric encryption scheme*) for message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$, consists of a probability distribution P_K over $\mathcal{K}$,[2] as well as

[1] We avoid the term privacy because it is generally used specifically to refer to the protection of personal data.

[2] The key distribution is almost always the uniform distribution.

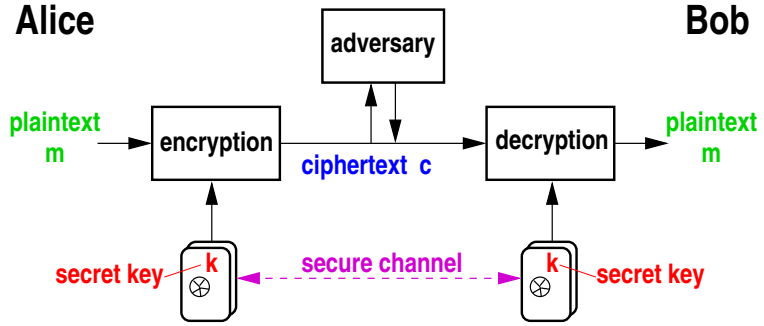

Figure 3.1: Model of a symmetric cryptosystem. A symmetric cryptosystem can be probabilistic, i.e., the encryption transformation can involve random bits generated afresh for each message.

- a possibly probabilistic *encryption* function E: $\mathcal{M} \times \mathcal{K} \to \mathcal{C}$,[3] and
- a *decryption* function d: $\mathcal{C} \times \mathcal{K} \to \mathcal{M} \cup \{\perp\}$,[4]

such that $d(E(m,k),k) = m$ with probability 1 for all m and k (correctness).

We will always assume that messages, ciphertexts, and keys are bit-strings, i.e., $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{K}$ are subsets of $\{0,1\}^*$.

A deterministic symmetric cryptosystem is also called a *cipher*.

3.2.2 Discussion of the Definition

We point out that by such a definition of a cryptographic scheme, its intended use in an application is generally not explicit, even though it might be somehow implicit. There can be different ways of using the scheme, some of which may be secure while others are insecure.

For example, one typically may want to encrypt many messages with the same key, but it also makes sense to use any given key only for encrypting a single message, and then to change the key. In the former, multi-message case, a deterministic symmetric cryptosystem is problematic since an adversary can at least recognize when the same message is repeated.[5] Another technique for

[3] A probabilistic function is a conditional probability distribution $P_{C|MK}$ of the ciphertext, given the message and the key. Such a conditional distribution can equivalently be understood as a function mapping pairs (m,k) to probability distributions over $\mathcal{C}$.

[4] The symbol $\perp$ means that no decrypted message is generated, for example because the ciphertext is invalid in the sense that it does, for the given key, not correspond to any message.

[5] This problem does not arise if the application is known to never repeat a message, for example because the messages contain an increasing counter or the time.

achieving multi-message security is to keep a state in the encryption function between messages, which would require a generalization of Definition 3.1.

Moreover, one probably understands that the randomness should be chosen afresh for every message to be encrypted, but, again, this is not made explicit in the definition of the scheme.

As explained in Section 1.3, the above definition does not include the security condition; it only captures a correctness condition, namely that decryption with the correct key must always work. The identity function (i.e., no encryption at all) satisfies the definition but can hardly be considered secure.

Moreover, the definition, as stated, does not include a *practicality* condition, namely that the encryption and decryption functions must in some sense be efficiently computable (see Section 1.3.3).

3.2.3 Towards Defining Security of Symmetric Encryption

A necessary requirement for a symmetric cryptosystem to be secure is that the key space be large enough so that an exhaustive key search is infeasible. In practice, a commonly used key length is 128 bits (i.e., $|\mathcal{K}| = 2^{128}$).

An intuitive security definition for an encryption scheme could be that it is infeasible to compute the plaintext when given the ciphertext, without knowledge of the secret key. However, such a security definition is unsatisfactory for two reasons.

- First, it should be infeasible to compute *any* information about the plaintext, even to guess a single bit with success probability significantly greater than what would be possible without seeing the ciphertext.
- Second, this should be true even if the adversary can mount various attacks on the cryptosystem, for instance adaptively choose a plaintext and receive the corresponding ciphertext (under the unknown secret key), as well as choose a ciphertext and receive the corresponding plaintext.

Hence we need a more adequate security definition. A natural approach appears to be to require that no *partial* information should leak. But how can one capture the notion of partial information? The goal of the following definition is to capture this.[6] We first discuss a so-called *game-based* security definition and then briefly mention the security definition in constructive cryptography. Game-based security means that a certain game, defined between a challenger and an adversary, cannot be won by any feasible adversary substantially better than what is trivially possible.

[6] In the literature, the term *semantic security* is used to refer to definitions of the following form: For any binary function of the message, and assuming one is restricted to efficient (say, polynomial-time) computation, then one cannot guess the binary value of that function when given the ciphertext substantially better than when the ciphertext is not given. It has been shown that this security notion is in an asymptotic sense equivalent to the IND-CPA security notion discussed here.

In our terminology (see Chapter 2), we simply define the computational problem corresponding to the "attack-game", which can be a game, a distinction problem, or a bit-guessing problem. In such a description, the term "adversary" does not appear. But we may neverthelss sometimes use expressions like "the adversary chooses" which are closer to what one would see in the general literature. We also point out that in the literature, the term "game" is used for all three types of computational problems, whereas we only use it for the first type.

We now define the so-called "IND-CPA game" (actually a bit-guessing problem consisting of guessing the bit B) which leads to the so-called IND-CPA security notion. Here "CPA" stands for *chosen-plaintext attack* and "IND" stands for indistinguishability.

Definition 3.2. The *t-message IND-CPA bit-guessing problem* $[\![S_t^{\mathsf{sym-ind-cpa}}; B]\!]$ *for symmetric encryption* is defined as follows: B is a uniform random bit, and the system $S_t^{\mathsf{sym-ind-cpa}}$ is defined as follows:

1. A random secret key k is chosen according to the key distribution P_K.
2. Up to t times, $S_t^{\mathsf{sym-ind-cpa}}$ takes as input a message m and returns $E(m, k)$ (for fresh randomness).
3. $S_t^{\mathsf{sym-ind-cpa}}$ takes as input a pair (m_0, m_1) of messages of the same length[7] and returns the encryption of m_B, i.e., $c = E(m_B, k)$ (for fresh randomness).
4. If less than t inputs were given in Step 2, Step 2 is continued up to a maximum of t such inputs.

In the literature, the above definition is typically stated as follows:

> A symmetric encryption scheme is CPA-*secure* (or IND-CPA-secure) if every polynomial-time adversary (algorithm) has negligible[8] advantage in the above game, where t is not bounded (i.e., only bounded by the adversary's running time).

We make a few remarks about Definition 3.2:

- The existence of Steps 2 and 4 is also referred to as the availability (to the adversary) of an encryption oracle.
- A *fixed-challenge* version of the security definition is obtained if one takes the maximum over all message pairs (m_0, m_1) rather than allowing (the adversary or distinguisher) to choose the pair after seeing some ciphertexts.

[7] Why is it necessary to require that the messages are of the same length?
[8] for a suitable negligibility notion

- The definition can be strengthened by also allowing queries to a decryption oracle (where, of course, the challenge ciphertext is forbidden). This notion is referred to as IND-CCA (chosen-ciphertext attack) and is not discussed further at this point.
- As discussed in Section 2.1, one could describe the computational problem in Definition 3.2 as a distinction problem rather than a bit-guessing problem.
- One can think of $S_t^{\mathsf{sym-ind-cpa}}$ as being decomposed into three systems: a system outputting a secret key, a system performing encryptions, and a system performing the described game.

Exercise 3.1. Show that the above fixed-challenge version of the security definition is strictly weaker than the general CPA security definition.

It is *a priori* not clear whether CPA security suffices for a particular application, or whether the stronger CCA security is required, or whether one needs an even stronger definition. (For example, one could allow parts of the secret key to leak to the adversary.) More precisely, it is not clear which security notion is sufficient for which application, in particular for establishing a secure channel between Alice and Bob.

As explained in Section 1.4.2, this is an intrinsic dilemma of game-based security definitions. It may not be clear which definition suffices for the application within a certain protocol. This is a problem for two reasons.

1. We want a security proof for our overall protocol and therefore need to understand which security notion for encryption is required.
2. If we work with an unnecessarily strong security notion, then a scheme achieving that notion may not exist at all, or if one exists, it may be significantly less efficient than one satisfying a weaker notion.

These problems will be addressed by introducing constructive cryptography.

Actually, here is yet another security definition for symmetric encryption. The adversary can choose up to t messages m_i, with $1 \leq i \leq t$, and either always gets the encryption of a fresh random message of the same length as the chosen message, or he gets the real encryption (with the unknown secret key) of the message. (There are also two versions, depending on whether the choice of messages is adaptive or not.) Investigating this security definition and its relation to the CPA game (Definition 3.2) is perhaps the subject of an exercise.

3.2.4 The One-Time Pad

In this section we discuss the one-time pad (OTP, see Figure 3.2), the best-known provably secure cryptosystem, as a motivating example. We present the traditional security statement for OTP encryption and explain three intrinsic problems with this security statement for the one-time pad:

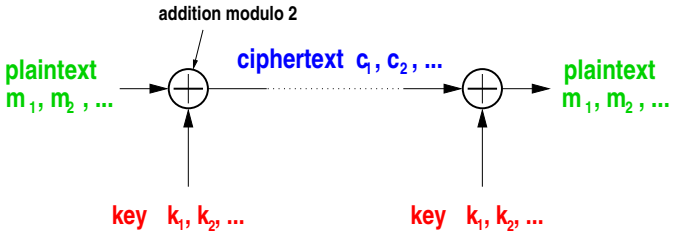

Figure 3.2: The one-time pad (OTP) encryption scheme. The plaintext (consisting of bits $m_1, m_2, \ldots$) the key (consisting of bits $k_1, k_2, \ldots$) and the ciphertext (consisting of bits $c_1, c_2, \ldots$) are bit-strings of equal length, say n bits. The key is a uniformly random bit-string that is used only once. Encryption [decryption] consists of XOR-ing (i.e., adding bit-wise modulo 2) the key to the plaintext [ciphertext].

1. In Section 3.2.6, we show that, despite the proof, the OTP can nevertheless be insecure, for two different reasons.

2. It is not necessarily clear how one can use one-time pad encryption securely within a larger protocol, i.e., how it can securely be composed with other cryptographic schemes (e.g. authentication).

3. In Section 3.2.7, we argue that the security proof cannot be carried over to a more practical setting where the (long) secret key is replaced by a pseudo-random key generated from a short secret key.

The constructive approach to defining security will solve all these problems.

3.2.5 The Traditional Security Proof for the One-Time Pad

The OTP encryption scheme can be proved to be information-theoretically secure, i.e., unbreakable even for a *computationally unbounded* adversary. How can one formulate and prove this claim?

The usual way to argue about the security of OTP encryption is to show that the plaintext (also called the message) and the ciphertext are statistically independent. This means that the ciphertext gives no information about the plaintext, independent of the available computing power. In other words, if one has to guess the plaintext when given the ciphertext one can just as well ignore the ciphertext.

Proposition 3.1. In one-time pad encryption, for a given message length n, the plaintext and the ciphertext are statistically independent (for all plaintext distributions),[9] i.e.,

$$\mathsf{P}_{CM}(c, m) \;=\; \mathsf{P}_C(c) \cdot \mathsf{P}_M(m)$$

for all m and c.

Exercise 3.2. Prove the following more general statement of which the proposition is a special case. Consider a group G with group operation $\star$ and consider an arbitrary random variable X over G as well as an *independent* random variable U with uniform distribution over G. Then the random variables $U \star X$ and X are statistically independent. The proof also shows that $U \star X$ is uniform, no matter what the distribution of X is.

We note that statistical independence can also be captured by the condition $\mathsf{P}_{M|C}(m, c) = \mathsf{P}_M(m)$ for all m and c, which states that the distribution of the message does not change if one learns the ciphertext. We also note that the fact that $\mathsf{P}_{C|M}(\cdot, m)$ is independent of m can be seen as an information-theoretic version of the CPA security definition: The ciphertext has the same distribution for all messages.

In the following two subsections we discuss two reasons why this security statement is not sufficient.

3.2.6 Two "Security Problems" of the One-Time Pad

Despite the above security proof, the OTP appears to be very insecure, at least for two independent reasons.

- First, an adversary (Eve) with access to the communication channel can modify the message in a controlled manner, even though she gets no information about the transmitted message. The adversary can add (bit-wise modulo 2, i.e., XOR) an arbitrary offset δ to the transmitted message, simply by adding δ to the ciphertext. If OTP encryption were used in a banking application, an attacker could for instance change the bank account number of a money transfer.

- Second, suppose that the OTP is used to encrypt one of the two messages "Yes" or "No". Since the two messages have different lengths, the ciphertext leaks complete information about the message, at least in a naïve implementation of the OTP. This appears to be in contradiction to the above claim that the ciphertext is independent of the plaintext message. This makes clear that when using OTP encryption, either one should pad all messages to have the same length, or one should accept the fact that the length of the message leaks.

[9] This property is called *perfect secrecy*, a term introduced by Shannon.

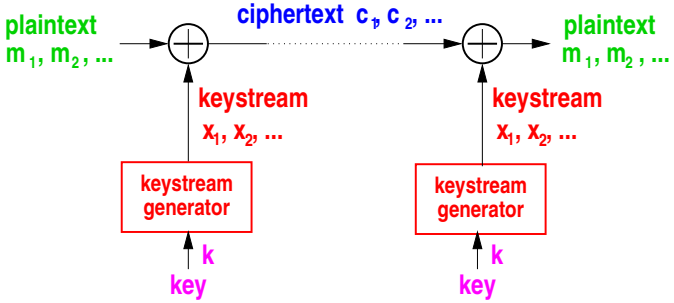

Figure 3.3: Binary additive stream cipher. It is obtained from the OTP encryption by replacing the random key $k_1, k_2, \ldots$ by the output $x_1, x_2, \ldots$ of a pseudo-random bit generator (also called key-stream generator) with a short key k. This system can be used for encrypting arbitrarily long messages, and also many consecutive messages if sender and receiver use consecutive portions of the key-stream.

3.2.7 Computational Security: Additive Stream Ciphers

To illustrate the third short-coming of the security statement (Proposition 3.1), we consider a variation of OTP encryption used in practice, where the uniformly random key is replaced by a key-stream sequence that is generated deterministically from a short secret key by a so-called key-stream generator. This is called an *additive stream cipher* (see Figure 3.3).

How should one define the security of such a stream cipher? Our goal is that the two security definitions for OTP encryption and stream cipher encryption are similar. Because the entire key-stream is generated from a short secret key, the system cannot be information-theoretically secure. One could (theoretically) break the system by trying all possible keys until the correct one is identified as the only key that results in meaningful plaintext after decryption.[10] Hence we must find a way to formulate what it means for this system to be *computationally secure*.

3.2.8 Addressing the Three Problems

All three problems mentioned in Section 3.2.4 can be solved by *constructive cryptography* introduced later (see Section 3.3 for a treatment of the OTP example).

The first criticism, namely that there are attacks against the OTP not covered by the security proof, could be eliminated by stating that we assume the communication channel to be *authenticated* (but not necessarily confidential). Then

[10] This requires a sufficient amount of ciphertext and assumes that the plaintext is redundant, i.e., that one can distinguish between meaningful and meaningless messages.

the security proof could be interpreted as stating that OTP encryption "constructs" a secure channel from an authenticated channel (and a shared secret key). The fact that the message length leaks can be solved by simply stating this fact as part of the specification of the secure channel. What "construct" means is a question we will address later.

The second criticism, namely that the statement of Proposition 3.1 does not (necessarily) guarantee composability with other cryptographic schemes, is resolved in constructive cryptography by the fact that resources constructed by a protocol (i.e., a secure channel) can be used as the assumed resources in further construction steps.

The third criticism, namely that the security definition does not extend to computational security,[11] can be solved by caturing this aspect as an adequate relaxation of the constructed resource.

3.2.9 Pseudo-randomness: The Concept

In practice one often approximates the generation of random bits by a deterministic algorithm, starting from a random *seed* or *key* value. This is called a *pseudo-random bit generator (PRBG)*. There are two reasons for using a PRBG rather than a true random bit generator:

- Implementing a random bit generator on a given hardware platform may be infeasible or impossible.
- The generated random bit sequence must be reproducible at another location. This is a typical problem in cryptography where secret keys should be short. But the problem also occurs in other contexts, for example in scientific simulations where one wants a result to be reproducible.

It is delicate to define pseudo-randomness properly. A bit generator that outputs the constant sequence $000000\cdots0000$ would probably not be considered a PRBG, nor one that outputs $010101\cdots0101$. How do we capture the requirement that the output sequence must "look random"? Any fixed list of statistical or other properties (e.g. large period) is arbitrary and falls short of capturing all aspects of "looking random". The actual definition used in cryptography is based on a very important general paradigm for security definitions, namely *indistinguishability*. A PRBG is secure if it cannot be distinguished from a truly random bit generator by any *feasible* distinguisher algorithm (see Figure 3.4), except with very small (negligible) advantage.

The notions of feasible and negligible are usually defined asymptotically as (some form of) polynomial-time computable and as decreasing faster than the inverse of any polynomial, respectively. We will address this later.

[11] Recall that the term "computational" is often used to state that security only holds against computationally bounded adversaries.

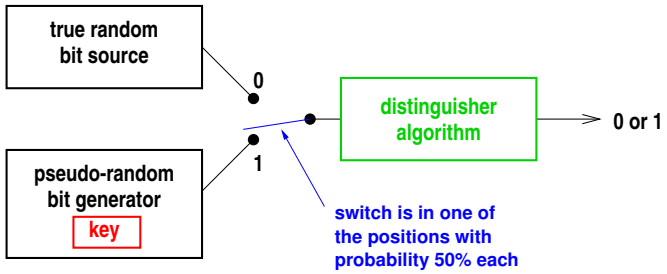

Figure 3.4: Distinguishing the output of a pseudo-random bit generator from a truly random bit generator. This is a special case of a distinction problem (see Section 2.1.2), where the figure appears mirrored, with the distinguisher on the left side.

An important concept for generating pseudo-random bits is that of a deterministic function g which expands a given random input to a longer pseudo-random output. Let U_m denote a random variable with uniform distribution over the bit-strings of length m. Recall the concept of a distinction problem from Section 2.1.

Definition 3.3. The *PRG distinction problem* for a function $g : \{0,1\}^k \to \{0,1\}^m$ for $m > k$ is

$$\langle g(U_k) \,|\, U_m \rangle,$$

i.e., the problem of distinguishing $g(U_k)$ (for a uniform input U_k) from a uniform n-bit string U_n.

In the literature, a pseudo-random generator is defined as follows:

Definition 3.4. (Informal) A function $g : \{0,1\}^k \to \{0,1\}^m$ for $m > k$ is called a *pseudo-random generator (PRG)* if the advantage of any feasible distinguishing algorithm for $\langle g(U_k) \,|\, U_m \rangle$ is negligible.[12]

One generally also needs an efficient algorithm for computing the function, but in line with the explanations of Section 1.3.3, we have not included this condition in the (informal) definition of a PRG. Actually, what one finds in the literature is the condition that the function g is required to be *efficiently computable*[13] or, more often, that g is not considered to be a function but a specific algorithm (for cumputing a function).

Instead of working with Definition 3.4, which is not yet formal, we will work with the *distinction problem* of distinguishing $g(U_k)$ and U_m. We refer to Section 2.1.2 for the definition of a distinction problem.

[12] To make this definition formal, one would need to consider an asymptotic setting, where g would be a family of functions, k would be a security parameter, and m would be a function of k.

[13] As pointed out in Section 1.3.3, such a condition is actually not sufficient.

An important natural question, which we later answer positively, is the following: If a PRG is applied iteratively, by using part of the output as the new input, does one obtain a PRG with a greater expansion factor? For example, if we start from a uniform random n-bit key and use a PRG $g : \{0,1\}^n \to \{0,1\}^{2n}$ to generate $2n$ pseudo-random bits, then we can reuse half of these bits as a new input to g, resulting in $3n$ pseudo-random bits. This process can be repeated, resulting in an algorithm that starts from a seed value and produces a pseudo-random sequence of arbitrary length, i.e., to obtain a PRBG as previously discussed.

3.2.10 Block Ciphers, AES, and Pseudo-random Functions

An important type of cipher used in practice are so-called *block ciphers*, a stateless encryption function where plaintext and ciphertext are n-bit strings (usually $n = 64$ or $n = 128$).

It is important to point out that a block cipher should not be used as a symmetric cryptosystem, but a block cipher serves as a building block to define an encryption function. Such a particular way of using a block cipher is called a *mode*. Three such modes, which are not discussed in the course notes, are cipher block chaining (CBC), counter mode, and cipher feedback mode.

Definition 3.5. A *block cipher* with block length n and key length m is a function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ such that for every key k, $f(\cdot\,, k)$ is a bijection (i.e., a permutation) $\{0,1\}^n \to \{0,1\}^n$.

The practicality requirement is that one knows efficient algorithms for computing $f(x,k)$ and the inverse $f^{-1}(\cdot\,, k)$.

The most prominent example of a block cipher is the so-called *Data Encryption Standard (DES)* with $n = 64$ and $m = 56$, developed in the 70's and standardized by the U.S. government. The currently most widely used block cipher is the so-called *Advanced Encryption Standard (AES)*, standardized by the U.S. government in 2000. It allows for different parameter sizes; the most often used parameters are $n = 128$ and $m = 256$.

When used in the natural mode called electronic codebook mode, a message is encrypted by cutting it into n-bit blocks and encrypting each block separately. This encryption mode has a security problem (see exercises).

A block cipher can also be used in other modes of operation which we do not discuss at this point.

A very strong and natural security requirement can be captured via the problem, parameterized by t, of distinguishing the following two systems (see Section 2.1.2):

- a system that, for a uniformly chosen key k, answers t queries $f(x,k)$ for (adaptively chosen) values of x.

- a system that, for a function $g : \{0,1\}^n \to \{0,1\}^n$ chosen uniformly from the set of functions $\{0,1\}^n \to \{0,1\}^n$, answers t queries $g(x)$ for (adaptively chosen) values of x.

This distiction problem will be discussed later in more detail.

A function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ for which no feasible distinguisher has a non-negligible advantage in this distinction problem is called a *pseudo-random function (PRF)*.

3.3 Constructive Cryptography: A First Example

3.3.1 The Basic Paradigm

In the constructive cryptography framework, a cryptographic mechanism (e.g. the one-time pad) is interpreted as a method for *constructing* a certain *resource* (called the *constructed resource*) from certain *assumed resources*. The construction is achieved by each (honest) party attaching a system, executing its protocol, to its interface of the assumed resource. Such an attached protocol system is called a *converter*.

A resource is a system with interfaces to the involved parties, typically including some honest parties and a (hypothetical) adversary. The constructed resource captures everything that is guaranteed, including what the honest parties can achieve and what an adversary can (and can not) do.

We point out that security properties appear in constructive cryptography very differently when compared to other treatments of the security of cryptographic schemes, such as encryption. In the constructive approach, the *use* of a scheme is made explicit (which is important) in the converters, and the security of the scheme (for that way of using it) is captured by the fact that a certain construction is achieved. This statement may or may not be equivalent to a certain game-based security definition of a scheme. In the best case, the two types of statements are equivalent, which provides justification for the game-based security definition.

3.3.2 The "Real World" and the "Ideal World"

One particular way to think about constructions is as follows. One can think of two worlds, a "real world" where the assumed resources are available and the protocol is executed, and an "ideal world" where the constructed resource is available. One argues that an adversary could, in the ideal world, achieve anything that she could achieve in the real world. This argument involves, as a thought experiment, a *simulator system* which transforms the (constructed) resource in the ideal world into the system in the real world.

However, we point out that constructive cryptography is a more general approach and the view described below is only a specific instantiation. It can not capture all construction statements one wants to make. There exist some other frameworks, most prominently the so-called *Universal Composability (UC) framework* by Ran Canetti, which make use in one form or another of the described "real world/ ideal world" paradigm.

3.3.3 The One-time Pad Example

We need to define the real system and the ideal system for the one-time pad example. We refer to Figure 3.5.

The constructed resource is a secure channel, denoted as SEC (later we also use the symbol $\bullet\!\!\longrightarrow\!\!\bullet$).[14] A secure channel (for sending a single message from A to B) is a resource system with three interfaces, for a sender A, a receiver B, and an adversary E. A can input a message, B receives the message (upon request), and E can learn nothing except for the length of the message.

The assumed resources consist of an authenticated channel, denoted as AUT or also as $\bullet\!\!\longrightarrow$, and (in parallel) a shared secret key, denoted as KEY or also as $\bullet\!\!=\!\!\bullet$, which generates a fresh uniform string of a certain length and (upon request) outputs it to Alice and Bob, while outputting nothing to Eve. The channel AUT takes a message (from a certain message space) at the A-interface and (upon request) outputs the message at the B-interface and also the E-interface.[15] The real-world system consists of the parallel composition of AUT and KEY (the assumed resource), denoted [AUT, KEY] (or, alternatively $[\bullet\!\!\longrightarrow, \bullet\!\!=\!\!\bullet]$), with the converters (protocol systems) otp-enc and otp-dec attached. Here we assume that otp-enc takes as input a message of *variable length*, takes the initial segment of the same length from the key, XORs it to the message, and sends the result over the channel AUT.

In algebraic notation, the real-world system is hence

$$\text{otp-dec}^{\text{B}}\ \text{otp-enc}^{\text{A}}\ [\text{KEY}, \text{AUT}],$$

where the superscript (e.g. A) specifies at which interface the protocol system is connected.

How can we argue that OTP encryption constructs a *secure* channel? After all, in the real-world system, when a message is input, the adversary receives something, namely the ciphertext, while she receives nothing in the ideal world, i.e., in the constructed resource (except for the message length $|m|$, the leakage of which is a feature of the constructed resource). How can we resolve this apparent problem?

[14] This notation is explained in the *Information Security* lecture notes. In a nutshell, the $\bullet$ means that the corresponding end-point of the channel is exclusive for the attached entity.

[15] In contrast to a secure channel, an authenticated channel is one in which E also learns the message sent by A but, like for a secure channel, E cannot influence the output for B.

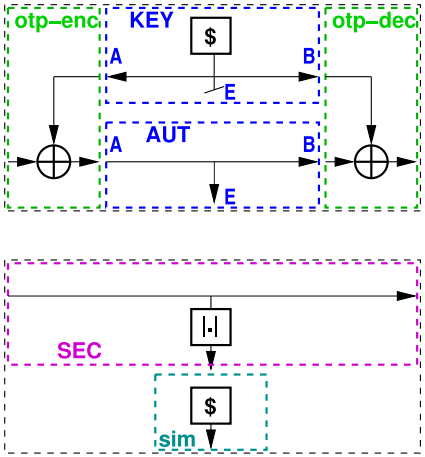

Figure 3.5: The one-time pad (OTP) explained as an example in the constructive cryptography framework. The OTP constructs a secure channel SEC (the constructed resource) from an authenticated channel AUT and a secret key KEY (the assumed resources). There exists a simulator sim which, when attached to interface E of the constructed resource, makes it equivalent to the real-world system with the converters (otp-enc and otp-dec) attached to the assumed resources. Equation (3.1) is an alternative (algebraic) way of describing the systems and stating their equivalence.

3.3.4 The Simulator

Since the ciphertext is statistically independent of the message, we can argue that the adversary is not better off in the ideal world (i.e., when the secure channel is available), compared to the real world where OTP encryption is used. If the secure channel were used, she could *simulate* the ciphertext (she would receive in the real-world system) by herself. More precisely, if a simulator generating a random ciphertext (of appropriate length[16]) is attached to the adversary interface of the constructed resource (see Figure 3.5), then the resulting (ideal-world) system is equivalent to the real-world system. Here equivalent means that the entire input-output behavior is exactly the same. In other words, both systems, the systems in the real and the ideal world, would behave in exactly the same manner in any possible environment or application they might be placed in. This means that the real-world system is just as good as the ideal-world

[16]The length must be given as an input to the simulator, hence it must be "leaked" by the secure channel.

system if one assumes that generating a uniform random string is easy for the adversary E.

Written as an equation of systems, the requirement can be stated as

$$\text{otp-dec}^{\text{B}}\ \text{otp-enc}^{\text{A}}\ [\text{KEY}, \text{AUT}] \quad \equiv \quad \text{sim}^{\text{E}}\ \text{SEC}. \tag{3.1}$$

3.3.5 Computational Security

Let us now switch to the stream-cipher setting. The security proof for OTP encryption (Section 3.2.5) did not carry over to the stream-cipher setting. In contrast, the above simulation-based security proof does.

The assumed and constructed resources *are the same*, no matter whether information-theoretic or computational security is considered. In the example we consider here, also the same simulator can be used for both settings (OTP and stream-cipher). However, the real-world system with stream-cipher encryption and the ideal-world system with simulator are not identical anymore. But all we want to argue is that they behave essentially identically in any realistic application context, where realistic means that only efficient computations occur. Written as an equation of systems, the requirement can be stated as

$$\text{dec}^{\text{B}}\ \text{enc}^{\text{A}}\ [\text{KEY}, \text{AUT}] \quad \approx \quad \text{sim}^{\text{E}}\ \text{SEC},$$

where the notion of systems being *approximately* equivalent is defined via the problem of distinguishing the two systems $\text{dec}^{\text{B}}\ \text{enc}^{\text{A}}\ [\text{KEY}, \text{AUT}]$ and $\text{sim}^{\text{E}}\ \text{SEC}$. In asymptotic terms, and informally stated, the construction of SEC from [KEY, AUT] is computationally secure if the distinguishing advantage

$$\Delta^{D}(\text{dec}^{\text{B}}\text{enc}^{\text{A}}[\text{KEY}, \text{AUT}], \text{sim}^{\text{E}}\text{SEC})$$

is negligible for all efficient distinguishers D. However, we will later see ways to capture computational security in a more direct and elegant way, without need for introducing a computational model (which is anyway arbitrary) and asymptotics.

3.3.6 Composability

An important goal will be that constructions are *composable*. This means, for example, that if one proves an encryption scheme secure, then one should be able to use it in any application that requires a secure communication channel when only an authenticated channel and a secret key is available.

Similarly, when only an insecure channel and a key is available but a certain authentication mechanism constructs an authenticated channel from an insecure channel and a secret key, then the combination of the authentication mechanism and secure encryption should construct a secure channel from an insecure channel and two secret keys.

3.4 Message Authentication Codes

In the previous section we saw that the best encryption scheme can possibly be insecure if the transmitted ciphertexts are not authenticated. In this section we propose a solution: *message authentication*, based on a secret key.

Two relevant questions that will arise are whether the same key used for encryption can also be used for message authentication, and whether the combination of encryption and message authentication is secure if each of the schemes is individually secure (according to some notion).

3.4.1 The Concept

If an adversary can control a communication channel, then she can always delete messages. This cannot be prevented by cryptographic means. But what cryptography can achieve is that a receiver can detect (and ignore) messages *not* originating from a specific sender. This is the goal of message authentication codes (MAC) in a context where sender and receiver share a secret key.

Definition 3.6. A *message authentication code (MAC)* for message space $\mathcal{M}$, key space $\mathcal{K}$, and authenticator space[17] $\mathcal{A}$, is a function $f : \mathcal{M} \times \mathcal{K} \to \mathcal{A}$ together with a probability distribution P_K over $\mathcal{K}$,[18]

For practicality, one also requires an efficient algorithm for computing the function f.

A MAC is used as follows:

1. The sender computes the MAC for the message and transmits the message together with the (appended) MAC-value (also called authenticator or tag).
2. The receiver computes the MAC on the received message and compares it to the received MAC-value. If it is correct, then the message is accepted, otherwise it is rejected.

3.4.2 Security Definition

The game-based security definition for a MAC is obtained by considering the following MAC-forgery game between an adversary and a challenger.

Definition 3.7. *t-message MAC-forgery game G_t^{mac}* for a MAC f:

1. A secret key $k \in \mathcal{K}$ is chosen according to P_K.

[17] also called tag space

[18] As for encryption, the key distribution is almost always the uniform distribution.

2. G_t^{mac} takes up to t messages as input and returns their MAC-values: For every message m, $f(m, k)$ is returned.
3. G_t^{mac} takes an input (m', a) for $\in \mathcal{M}$ and $a \in \mathcal{A}$. The game is won if $a = f(m', k)$ and m' was not given as an input in step 2.

A MAC-function is called *secure (under a chosen-message attack)* if no feasible (adversary) algorithm wins this game with non-negligible probability. One could refine the definition of the above game by also allowing a certain number v of verification queries, where the adversary provides a pair (m, z) and the challenger replies with a binary value indicating whether $f(m, k) = z$. Since such a query can also be done by asking for the MAC-value $f(m, k)$, this is not a highly relevant generalization.

3.4.3 The CBC-MAC

The so-called CBC-MAC[19] is one of the most widely-used message authentication schemes. The CBC-MAC construction is shown in Figure 3.6 and explained in the figure caption, for the specific choice of instantiating the internal block cipher with AES.

A relevant question is whether and why the CBC-MAC is indeed a secure MAC scheme. We will investigate this question later. In a nutshell, we will prove that if AES is replaced by a uniform random function, then the CBC-MAC system is indistinguishable (except with very small advantage) from a uniform random function $\{0,1\}^* \to \{0,1\}^n$, where n is the block length of the block cipher.

3.4.4 Description in Terms of a System Algebra

A block cipher, say AES, can be understood as a system with two interfaces (left and right). A system implementing AES first takes at the right interface as input a 128-bit key and then continuously answers AES encryption queries (at the left interface) by taking as input a plaintext block and returning the encryption (under the key) of the block. Let us denote this system as AES.

The system AES can be understood as being connected at the right interface to a single-interface system denoted $\mathbf{U}_{128}$, a source outputting a uniform 128-bit string. The combined system can be written as $\mathsf{AES}\ \mathbf{U}_{128}$ and is again a single-interface system (with the interface on the left side).

Moreover, we can also understand the CBC-MAC construction as a system with two interfaces, denoted CBC, which takes at the left interface as inputs messages (of arbitrary length) and returns the corresponding MAC. At the right interface it is connected to a system that answers block encryption queries, for

[19] CBC stands for cipher block chaining.

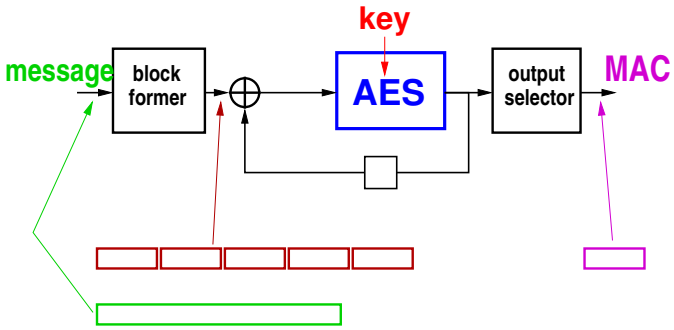

Figure 3.6: CBC-MAC, a message authentication code (MAC) based on a block cipher, for example AES. The message is processed in the block former to obtain a sequence of blocks, for example by appending the encoding of the length and by padding with 0's to obtain a multiple of the block length. Then the sequence of blocks is processed using the PRF (instantiated as AES in the figure). The output selector takes the last output block as the MAC.

example the system AES $\mathbf{U}_{128}$. The overall system is hence the single-interface system

$$\mathsf{CBC}\ \mathsf{AES}\ \mathbf{U}_{128}.$$

3.4.5 Combining Encryption and Authentication

To achieve secure communication over the Internet, one must apply both encryption and a MAC scheme. An important question is how they can be combined in a secure manner. Should one first apply the MAC to the message and then encrypt the result, or should one first encrypt the message and then apply the MAC function to the resulting ciphertext? This is a simple example of the kind of question that comes up when one designs cryptographic protocols.

We briefly describe the construction achieved by applying a MAC-function in the described manner and then discuss how this step can be combined with encryption.

The constructive viewpoint of applying a MAC is illustrated in Figure 3.7. The MAC function is used in the two converters mac and chk, where mac generates the MAC for a given message and appends it to the message, and chk verifies the MAC (shown as a function C which delivers the message only if the MAC value is correct and otherwise delivers some dummy symbol). This constructs an authenticated channel (not shown in the figure) from a insecure channel INS and a secret key KEY. Actually, the authenticated channel is weaker than the channel $\bullet\!\longrightarrow$ discussed earlier since the adversary can delete the mes-

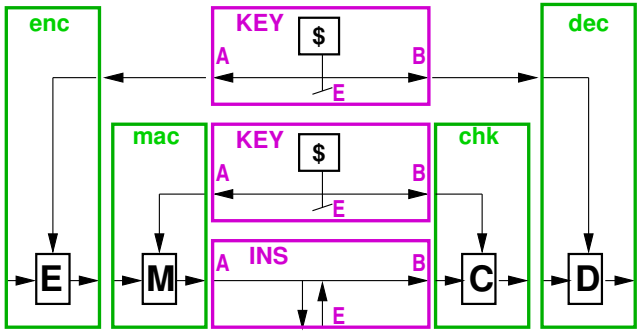

Figure 3.7: The Encrypt-then-MAC paradigm.

sage, i.e., interrupt the channel. The symbol we will be using for such a channel is $\bullet\!\!\rightarrow$ We remark that here INS is not yet formally defined and that the claimed construction statement would of course require a proof, which we defer to later.

Now the already discussed construction provided by encryption can be applied, involving a second (independent) key, and resulting in the combined protocol which is often referred to as Encrypt-then-MAC since the message entering the system first gets encrypted and then "MACed".

Later we will discuss (and prove) why such construction steps can be composed, i.e., why and in which sense the Encrypt-then-MAC paradigm is secure.

3.4.6 Encryption and Authentication with a Single Key

Let us return to a question we asked earlier, namely whether the same key could be used for encryption and message authentication. One could be tempted to argue that an encryption scheme protects the key (as otherwise it would be insecure), and hence re-using the key is safe.

However, the answer to the above question is "no". One can show (as an exercise) that the secret key can in general not be used for both encryption and for authentication. Such a combined scheme can be insecure even if both involved schemes are individually secure.

A natural idea to avoid the need for sharing two secret keys is to use a PRG to expand a given secret key into a longer bit-string, which can then be split into two keys that can be considered to be independent (in a computational sense). Therefore a single key suffices to perform several tasks that require independent keys. We discuss later how the security of such key expansion can be captured and proved.

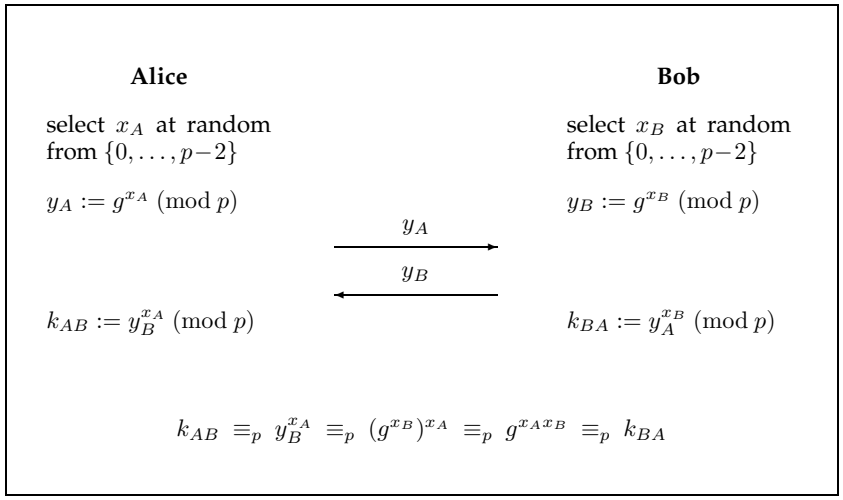

Figure 3.8: The Diffie-Hellman key agreement protocol. The prime p and the generator g are publicly known parameters.

3.5 Key Agreement

So far we have ignored the problem of how Alice and Bob obtain a shared secret key in the first place. They could meet physically and exchange a key, or they could send a trusted courier to distribute a secret key. Both these possibilities are highly impractical when Alice and Bob are users or servers on the Internet who wish to establish a secure connection. The Diffie-Hellman protocol provides a remarkable solution.

In this section we do not formally introduce the concept of a key agreement protocol nor discuss a formal security definition.

3.5.1 The Diffie-Hellman Key-Agreement Protocol

In this section we describe the famous key-agreement protocol proposed by Diffie and Hellman in 1976,[20] a corner-stone of modern cryptography. It allows two parties, say Alice and Bob, to generate a shared secret key by communicating only over an authenticated (but not confidential) channel to which an adversary has read access. We refer the reader to the Appendix for a repetition

[20] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.

of some basic number theory and algebra.[21]

The Diffie-Hellman protocol (see Figure 3.8), as originally proposed[22], makes use of exponentiation modulo a large prime p (of, for instance, 2048 bits):

$$x \mapsto g^x \pmod p.$$

While $y = g^x \pmod p$ can be computed efficiently (see Appendix A.3.5), even if p, g, and x are numbers of several hundred or thousands of digits, computing x when given p, g, and $y = g^x$ is generally believed to be computationally highly infeasible. This problem is known as (a version of) the *discrete logarithm problem*, which will be discussed later. The prime p and the basis g (e.g. $g = 2$) are public parameters, possibly generated once and for all for all users of the system.

The Diffie-Hellman protocol is shown in Figure 3.8. Alice selects an exponent x_A at random, computes $y_A = g^{x_A}$ modulo p, and sends y_A over an authenticated but otherwise insecure channel to Bob. Bob proceeds analogously, selects an exponent x_B at random, computes $y_B = g^{x_B}$ modulo p, and sends y_B to Alice. Then Alice computes the value

$$k_{AB} := y_B^{x_A} = (g^{x_B})^{x_A} = g^{x_B x_A}$$

modulo p, and Bob computes, analogously,

$$k_{BA} := y_A^{x_B} = (g^{x_A})^{x_B} = g^{x_A x_B}$$

modulo p. The simple but crucial observation is that

$$k_{AB} = k_{BA},$$

due to the commutativity of multiplication (in the exponent). In other words, Alice and Bob arrive at the same shared secret value.

3.5.2 The Mechanical Analog

The Diffie-Hellman protocol can be nicely explained by a mechanical analog, as shown in Figure 3.9. The exponentiation operation (e.g. the operation $x_A \mapsto g^{x_A}$) can be thought of as locking a padlock, an operation that is easy to perform but impossible (in a computational sense) to invert. Note that the padlock in this analog has no key; once locked, it can not be opened anymore. However, a party can remember the open state of the lock (i.e., x_A). Alice and Bob can exchange their locked padlocks (i.e., y_A and y_B), keeping a copy in the open state. Then they can both generate the same configuration, namely the two padlocks interlocked. For the adversary, this is impossible without breaking open one of the two padlocks.

[21] This material is considered a repetition of material already discussed in the courses *Discrete Mathematics* and *Information Security* (in the Computer Science curriculum at ETH Zurich). It is assumed that the reader is familiar with these schemes and their mathematical foundations.

[22] Since then, other versions, for instance based on elliptic curves, have been proposed.

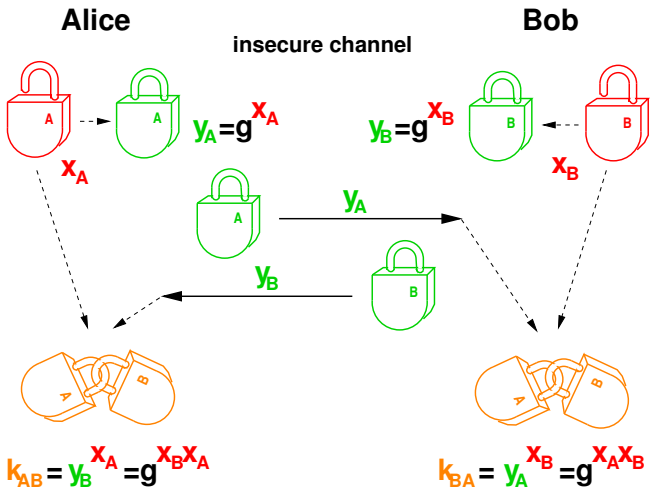

Figure 3.9: Mechanical analog of the Diffie-Hellman protocol.

3.5.3 Generalization to Arbitrary Cyclic Groups

The described Diffie-Hellman protocol can be generalized from computation modulo p (i.e., the group $\mathbb{Z}_p^*$) to any cyclic group $G = \langle g \rangle$, generated by a generator g, in which the discrete logarithm problem is computationally hard. The only modifications are that multiplication modulo p is replaced by the group operation of G (and x_A and x_B must hence be chosen from $\{0, \ldots, |G| - 1\}$). In practice, one often uses elliptic curves (as the group G) for which the discrete logarithm problem is believed to be even substantially harder than for the group $\mathbb{Z}_p^*$, for comparable group sizes.

3.5.4 From a Random Group Element to a Uniform Key

The Diffie-Hellman protocol as described does not yet provide a shared secret key to Alice and Bob because the binary representation of a uniformly random group elements is generally not a uniformly random string, actually far from it. For almost all groups, the natural representation is not dense, i.e., only a (small) subset of bitstrings of a certain length actually represent group elements. For the case of $\mathbb{Z}_p^*$, the representation is dense, but one would actually work only in a subgroup of $\mathbb{Z}_p^*$ of size at most $(p-1)/2$, and these elements would not be densely represented.

If one wants to obtain a k-bit secret key, Alice and Bob need to apply a function $\gamma : G \to \{0,1\}^k$ mapping group elements to bitstrings. In ad-hoc implementations of the Diffie-Hellman protocol, a fixed function is used, for example a cryptographic hash function. In an implementation designed to be provably secure, the function γ must be a so-called *extractor*, a function which, informally, maps any input with sufficient entropy to an essentially uniform random string. However, an extractor function must be randomized, i.e., it takes as a second input a short uniformly random string, often called *seed*. This seed must be random, but it need not be secret. Hence it can for example be chosen by Alice and sent to Bob (authentically) together with the group element.

Extractors will be discussed later.

3.5.5 Towards Defining Security

A possible security definition could state that it must be computationally infeasible (for an adversary) to compute the shared value $k_{AB} = g^{x_A x_B}$ when given $y_A = g^{x_A}$ and $y_B = g^{x_B}$. This is known as the so-called CDH-problem (see below).

Is this security definition, namely that computing the key is infeasible, sufficient? This raises the more general question of how one can show that a security definition is sufficient or that it is not sufficient? These questions are of crucial importance in cryptography, not just for understanding the Diffie-Hellman protocol. That a security definition is not sufficient can for example be shown by demonstrating an application in which the scheme, even though secure according to the definition, fails nevertheless.

For the Diffie-Hellman protocol, the security definition stating that the key is hard to compute (i.e., the CDH assumption, see below) is not sufficient. Namely, it does not exclude the possibility that part of the secret key can be computed. Is this bad? Yes! One can imagine an application where the generated key is used as a key for one-time pad encryption. Since part of the key is known, part of the message leaks to the adversary.

A stronger security definition could require that the adversary learns essentially nothing about the secret key, in a computational sense. But how could we define "nothing" in a computational sense? One approach is to say that there exists no efficiently computable predicate of the key which can efficiently be guessed better when the protocol execution is observed compared to when the protocol execution cannot be observed. However, such a definition, even if made precise, would still not be sufficient because an application might leak partial information about the key, and in such a case learning anything more about the key should be impossible. For example, even if the adversary learns all but one bit of the key, guessing the last bit should still be hard.

Another natural security definition trying to capture that no information leaks (in a computational sense) is the following, similar to the IND-CPA security definition for symmetric encryption. Given the transcript of the Diffie-

Hellman protocol, and assuming that the adversary is also given a value which is either the correct secret value K_{AB} shared by Alice and Bob, or an independent uniformly random group element, she cannot distinguish between the two cases. This is formalized as the DDH-problem in the next section.

This is still not necessarily a satisfactory definition since it is not clear what it really means in an application. A better definition appears to be the following. Executing the protocol should be equivalent (in a computational sense) to an idealized setting where Alice and Bob have access to a random bit-string (the key) to which Eve has no access. This can be formalized in constructive cryptography (see later). It will be possible to show that such a secret key is constructed under the assumption that the DDH-problem is hard (see below).

3.5.6 Underlying Computational Problems

We state three computational problems for a cyclic group G of order $|G| = q$ generated by a generator g (i.e., $G = \langle g \rangle$). As mentioned above, a concrete example is the multiplicative group modulo a large prime p, i.e., $G = \mathbb{Z}_p^*$, with $|G| = q = p - 1$. It is assumed that the group G, the generator g, and the order q are known. The first two problems are games (according to Section 2.1) while the last problem is a distinction problem.

Definition 3.8. The *discrete logarithm (DL) problem* for $G = \langle g \rangle$ is the problem of computing, for a given (uniformly chosen) group element $A \in G$, the exponent $a \in \mathbb{Z}_q$ such that $A = g^a$.

Definition 3.9. The *computational Diffie-Hellman (CDH) problem* for G is the problem of computing, for given (uniformly chosen) group elements $A, B \in G$, the group element C such that $C = g^{ab}$, where $A = g^a$ and $B = g^b$.

Definition 3.10. The *decisional Diffie-Hellman (DDH) problem* for G is the problem of distinguishing, for given elements $A, B, C \in G$, the case where A, B, C are independent and uniformly random in G from the case where A and B are uniformly random and $C = g^{ab}$, where $A = g^a$ and $B = g^b$. (A triple (A, B, C) of the latter type is called a Diffie-Hellman triple.)

For each of these problems one can phrase the assumption that the problem is computationally hard.[23] These assumptions are called the *DL-assumption*, the *CDH-assumption*, and the *DDH-assumption*, respectively.

It is easy to see that the CDH-assumption implies the DL-assumption, and the DDH-assumption implies the CDH-assumption. This is proved by trivial reductions. For example, if one could solve the CDH problem, then one could solve an instance of the DDH problem by simply solving the corresponding CDH problem and checking whether or not the third component of the DDH instance matches.

[23] Note that to define hardness formally (see possibly later), one would need to define an asymptotic family of groups and a specific binary representation of the group elements.

The CDH assumption is necessary but not sufficient for the Diffie-Hellman protocol to be secure in a meaningful sense. We will see later that the DDH-assumption is necessary and sufficient. However, in the so-called *random oracle model (ROM)*, discussed later, the CDH assumption is sufficient.

Since the DL problem appears more likely to be hard than the CDH problem, and, similarly, since the CDH problem appears more likely to be hard than the DDH problem, it is desirable to prove reductions between these problems. One can actually prove that for essentially all groups the CDH-assumption is equivalent to the DL-assumption, where for solving one instance of the DL problem one needs several calls to an oracle solving the CDH problem.[24] In contrast, no reduction from the CDH problem to the DDH problem is known.

3.6 Public-key Encryption and the RSA System

In this section we take a small detour from our path towards designing a secure communication protocol. This allows us to introduce another basic cryptographic concept, *public-key encryption (PKE)*, as well as the so-called RSA scheme (more precisely, the RSA trapdoor one-way permutation), which will in Section 3.7 also serve as the basis of a *digital signature scheme*.

3.6.1 Concept and Definition

The concept of public-key encryption is shown in Figure 3.10. A user (here Bob) generates a key pair consisting of a *secret key* (also called *private key* or *decryption key*) s_B and the corresponding *public key* (also called *encryption key*) p_B, which he makes public, for example by putting it on his web-page, authenticated by a certificate.[25] There is an encryption operation, making use of the public key (but not the secret key) by which any entity (e.g. Alice) can encrypt a message for Bob. Bob (and only he) can decrypt the ciphertext using the secret key.

Definition 3.11. A *public-key encryption (PKE) scheme*[26] for message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, public-key space $\mathcal{P}$, and secret-key space $\mathcal{S}$, consists of:[27]

- a *key-pair distribution*, a probability distribution over $\mathcal{P} \times \mathcal{S}$,
- a randomized[28] *encryption function* E: $\mathcal{M} \times \mathcal{P} \to \mathcal{C}$,

[24] Ueli Maurer, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, Springer-Verlag, 1994.

[25] A certificate is a statement signed by a trusted party, often called a certification authority (CA), which confirms the binding of the public key to a particular entity.

[26] also called a *public-key cryptosystem (PKC)*

[27] Note that we are using a capital E for the encryption function because it is a (function-valued) random variable.

[28] A randomized function is a conditional probability distribution. Here we do not make the randomness needed for key generation and encryption explicit.

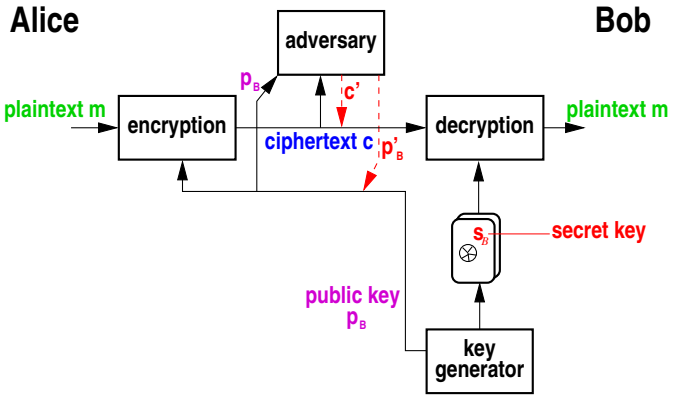

Figure 3.10: Scenario of using public-key encryption. An adversary can try to replace the public key p_B by a fake public key p'_B (which is why p_B must be authenticated) or try to replace the ciphertext c by a fake ciphertext c' (which is why c must be authenticated) .

- a *decryption function* d: $\mathcal{C} \times \mathcal{S} \to \mathcal{M} \cup \{\perp\}$,[29]

such that for any message $m \in \mathcal{M}$ and any key pair (p, s) with positive probability, $\mathsf{Pr}\big(d(E(m,p),s) = m\big) = 1$ (correctness).

The practicality condition is that efficient algorithms for sampling from the key-pair distribution and for encryption and decryption are given.

3.6.2 Security Definitions for PKE

Informally, security of a PKE-scheme means that it must be computationally infeasible to compute "useful" information about the plaintext for a given ciphertext without knowledge of the secret key.

As for the case of symmetric encryption, "leaking no useful information" can be captured by demanding that an adversary cannot distinguish the ciphertexts of any two messages chosen by her.

One must also specify what type of attack an adversary can perform. In a PKE-scheme, anybody can encrypt messages, hence we do not need to consider an encryption oracle in the definition. But to achieve a high level of security, one can allow the adversary to have access to a decryption oracle. This is summarized in the following bit-guessing problem (see Section 2.1.3), in the literature usually called a game between the adversary and a challenger.

[29]The symbol $\perp$ means that no decrypted message is generated, for example because the ciphertext is invalid in the sense that it does not correspond to any message.

Definition 3.12. The t-message CCA bit-guessing problem for public-key encryption, $[\![S_t^{\mathsf{pke-ind-cca}}; B]\!]$, is defined as follows: B is a uniform random bit, and the system $S_t^{\mathsf{pke-ind-cca}}$ is defined as follows:

1. A random secret key/public key pair (s, p) is sampled according to the key-pair distribution, and output (upon request).
2. Up to t times, $S_t^{\mathsf{pke-ind-cca}}$ takes as input a ciphertext c and returns $d(c, s)$
3. $S_t^{\mathsf{pke-ind-cca}}$ takes as input a pair (m_0, m_1) of messages of the same length and returns the encryption of m_B, i.e., $c = E(m_B, p)$ (for fresh and independent randomness).
4. If less than t inputs were given in Step 2, Step 2 is continued up to a maximum of t such inputs.

Definition 3.13. If in the above definition no decryption queries are allowed ($t = 0$, i.e., steps 2 and 4 are omitted), then it is called the *CPA bit-guessing problem for public-key encryption*, denoted $[\![S^{\mathsf{pke-ind-cpa}}; B]\!]$.

Informally, a PKE-scheme is called *CCA-secure* (respectively *CPA secure*) if every feasible (adversary) algorithm has only negligible advantage in the respective game.

This IND-CCA security definition seems intuitively very strong, but we do not (yet) know (nor address the question) whether and why this definition is adequate or too strong or too weak in a certain application context. Not surprisingly, the answer actually depends on the type of application.

3.6.3 Using Public-key Encryption

In order for a public-key cryptosystem to be applicable, the public key must be authenticated. This is achieved by transmitting it over an authenticated channel. Such a channel can, for instance, be obtained by the use of so-called *certificates*, which will not be discussed in depth in this course.

Public-key encryption is especially useful in a context with many entities, not just two. For example, B has published his public key authentically, and every entity should be able to send a secret message to B. This is the typical setting when public-key cryptography is used. It corresponds, for example, to a setting where many entities can connect to a server whose public key is publicly known.

Public-key encryption is used primarily for key management,[30] i.e., the transmitted (encrypted) message is a randomly chosen key, often called a session key (or a pre-master secret), that serves as a secret key for symmetric encryption and authentication. One advantage of this approach is the efficiency:

[30]Public-key cryptosystems are also used for various cryptographic protocols (e.g. e-voting protocols) where often a special property (e.g. a homomorphism) of the cryptosystem is exploited.

symmetric encryption can be implemented two to four orders of magnitude faster than public-key encryption.

Conversely, one can also use a key agreement scheme such as the Diffie-Hellman protocol (combined with symmetric encryption) as a public-key encryption scheme, as described below.

3.6.4 Using the Diffie-Hellman Protocol as a PKE-Scheme

The Diffie-Hellman protocol can be used as a PKE-scheme, as follows. Let the group G and the generator g be fixed and the order $|G| = q$ be known, and let a secure symmetric encryption scheme (e, d) with key space $\mathcal{K}$ (e.g. $\mathcal{K} = \{0,1\}^k$ for a suitable k) be given. Let γ be a function $G \to \mathcal{K}$ such that a random element of G is mapped by γ to an essentially random element of $\mathcal{K}$.

- **Key generation** (by Bob): Choose x_B uniformly at random from $\mathbb{Z}_q$. The secret key is x_B, the public key is $y_B = g^{x_B}$.
- **Encryption:** Choose x uniformly at random from $\mathbb{Z}_q$. The ciphertext for a message m is the pair $\left(g^x, e(m, \gamma(y_B^x), r)\right)$, where r is a uniformly random value from the randomness space of e.
- **Decryption:** left as an exercise.

For the special case where the OTP is used for symmetric encryption (more specifically, the message m is assumed to be represented as a group element and the encryption is simply multiplication in the group with the key, i.e., the ciphertext is $m \cdot k_{AB}$), this scheme is known as the so-called *ElGamal public-key cryptosystem*.

Exercise 3.3. Show that the described PKE-scheme is IND-CPA secure (under the DDH-assumption), but that it is not IND-CCA secure.

3.6.5 Trapdoor One-way Permutations

We now discuss another approach to devising a public-key encryption scheme, based on the following notion.

Definition 3.14. A *trapdoor one-way permutation (TOWP)* for domain $\mathcal{X}$, codomain $\mathcal{Y}$, parameter set $\mathcal{P}$, and trapdoor set $\mathcal{T}$ consists of

- a *parameter-trapdoor distribution*, a probability distribution over $\mathcal{P} \times \mathcal{T}$,
- a function $f\colon \mathcal{X} \times \mathcal{P} \to \mathcal{Y}$, and
- a function $g : \mathcal{Y} \times \mathcal{T} \to \mathcal{X} \cup \{\perp\}$,

such that for all $x \in \mathcal{X}$ and for all pairs $(p, t) \in \mathcal{P} \times \mathcal{T}$ with positive probability, $g(f(x, p), t) = x$ (correctness).

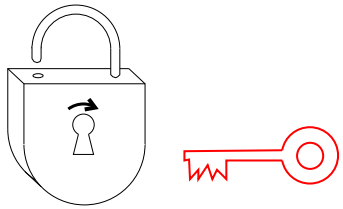

Figure 3.11: Mechanical analog of a trapdoor one-way permutation.

The practicality condition is that efficient algorithms for sampling from the parameter-trapdoor distribution and for computing f and g are given.

The security requirement is that the following TOWP inversion game (a game in the sense of Section 2.1.1) must be computationally hard, i.e., no feasible algorithm has non-negligible success probability.

Definition 3.15. The *TOWP inversion game* is defined as follows:

1. The system generates a pair (p, t) according to the parameter-trapdoor distribution, chooses a uniformly random $x \in \mathcal{X}$, and (upon request) outputs $f(x, p)$ and p.
2. The system takes an input x' and the game is won if $x' = x$.

Note that the definition of the game does not exclude that *partial* information about x can be computed from $f(x, p)$ without knowledge of the trapdoor.

The mechanical analog of a TOWP is shown in Figure 3.11.

3.6.6 The RSA Trapdoor One-way Permutation

This section briefly summarizes the RSA scheme[31] already discussed (in the Computer Science curriculum) in the courses *Discrete Mathematics* and *Information Security*, where it was called a public-key encryption scheme. The connection of the RSA TOWP and the related (naïve) public-key encryption scheme will be discussed later.

To understand the RSA TOWP, we need the following simple theorem.

Theorem 3.1. *Let G be some finite group (with neutral element 1), and let $e \in \mathbb{Z}$ be a given exponent relatively prime to $|G|$ (i.e. $\gcd(e, |G|) = 1$). The e-th root of $y \in G$, namely $x \in G$ satisfying $x^e = y$, can be computed according to*

$$x = y^d,$$

[31] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, Vol. 21, No. 2, pp. 120–126, 1978.

where d is the multiplicative inverse of e modulo $|G|$, i.e.,

$$d \equiv_{|G|} e^{-1}.$$

Proof. We have

$$(x^e)^d = x^{ed} = x^{k\cdot|G|+1} = \underbrace{\left(x^{|G|}\right)}_{=1}{}^k \cdot x = x,$$

where $k = (ed-1)/|G|$ is an integer. The under-braced term is equal to 1 because of Lagrange's theorem which states that the order of a subgroup of a finite group G divides the group order $|G|$. In particular, the order of every element divides $|G|$. □

When $|G|$ is known, then d satisfying $d \equiv_{|G|} e^{-1}$ can be computed using the extended Euclidean algorithm. No general method is known for computing e-th roots in a group G without knowing its order. This can be exploited to define a TOWP.

Motivated by the Diffie-Hellman protocol also based on modular exponentiation, Rivest, Shamir and Adleman suggested as a possible class of groups the groups $\mathbb{Z}_n^*$, where $n = pq$ is the product of two sufficiently large secret primes p and q. The order of $\mathbb{Z}_n^*$,

$$|\mathbb{Z}_n^*| = \varphi(n) = (p-1)(q-1),$$

can be computed only if the prime factors p and q of n are known.

Exercise 3.4. Show that from n and $\varphi(n)$ one can efficiently compute p and q.

We have seen that

$$\mathbb{Z}_n^* \to \mathbb{Z}_n^* : \ x \ \mapsto \ x^e \pmod n,$$

is a permutation and that

$$\mathbb{Z}_n^* \to \mathbb{Z}_n^* : \ x \ \mapsto \ x^d \pmod n,$$

is the inverse permutation, where d can be computed according to

$$d \ = \ e^{-1} \pmod{\varphi(n)}. \tag{3.2}$$

Exercise 3.5. Show that the RSA function is actually a permutation $\mathbb{Z}_n \to \mathbb{Z}_n$, making use of the Chinese remainder theorem. There is hence no need to restrict inputs to $\mathbb{Z}_n^*$.

We can hence define the RSA TOWP as follows.

Definition 3.16. RSA trapdoor one-way permutation: The parameter-trapdoor distribution is defined by the following process: A pair of primes, p and q, is chosen according to some distribution, let $n = pq$, then e with $\gcd\left(e, \varphi(n)\right) = 1$ is chosen according to some distribution. The parameter is the pair $p = (n, e)$ and the trapdoor is $t = (n, d)$ where d is defined by (3.2). Moreover, $\mathcal{X}$ is a set contained in $\mathbb{Z}_n$ for all possible n. Then f is defined as $f(x, p) = x^e$ in $\mathbb{Z}_n$, and g is defined as $g(y, t) = y^d$ in $\mathbb{Z}_n$.

Note that f and g are efficiently computable. Typically, p and q are chosen such that n has a certain bit-length (e.g. 2048 bits), and quite often p and q are restricted to be of (roughly) equal bit-size. Often, e is chosen to be a small constant, for example $e = 3$, and p and q are chosen to satisfy $\gcd(p-1, e) = 1$ and $\gcd(q-1, e) = 1$.

Let us briefly discuss the security of the RSA TOWP. It is not known whether computing e-th roots modulo n when $\gcd(e, \varphi(n)) = 1$ is easier than factoring n, but it is widely believed that the two problems are computationally about equally hard.[32] Factoring large integers is believed to be computationally infeasible. If no significant breakthrough in factoring is made and if processor speeds keep improving at the same rate as they are now, then a 2048-bit modulus appears to be secure for the next 30 years. However, factoring is known to be efficient on a quantum computer (Shor's polynomial-time algorithm), and RSA and other factoring-based cryptographic schemes will be insecure once sufficiently large quantum computers can be built.

3.6.7 Public-key encryption Using a TOWP

A naïve way to use a TOWP as a PKE-scheme is to let the domain of the TOWP be the message space and to encrypt simply by applying the TOWP, where the parameter of the TOWP is the public key of the PKE-scheme and the parameter/trapdoor pair is the secret key of the PKE-scheme. In terms of mechanical analogs, this corresponds to using the (receiver's) padlock to lock a box, which can then be sent over an insecure (or authenticated) channel. The receiver can decrypt, i.e., remove the lock. If the RSA TOWP is used, the resulting naïve (or plain) version of the public-key cryptosystem is shown in Figure 3.12.

However, it is important to note that naïve RSA encryption, as described above, is deterministic and therefore insecure, for example in applications with a small message space because each message in the message space could be tested. Even if the message space is large, the scheme is still not secure. Therefore a reasonable public-key cryptosystem based on RSA is probabilistic, by introducing appropriate (per-message) randomness. We might later address the problem of designing a suitable way of using a TOWP as a (secure) PKE-scheme.

[32] One can prove that breaking RSA by a generic algorithm (only performing operations in the ring $\mathbb{Z}_n$) is equivalent to factoring: D. Aggarwal and U. Maurer, Breaking RSA generically is equivalent to factoring, Advances in Cryptology - EUROCRYPT 2009, LNCS, vol. 5479, pp. 36–53, 2009.

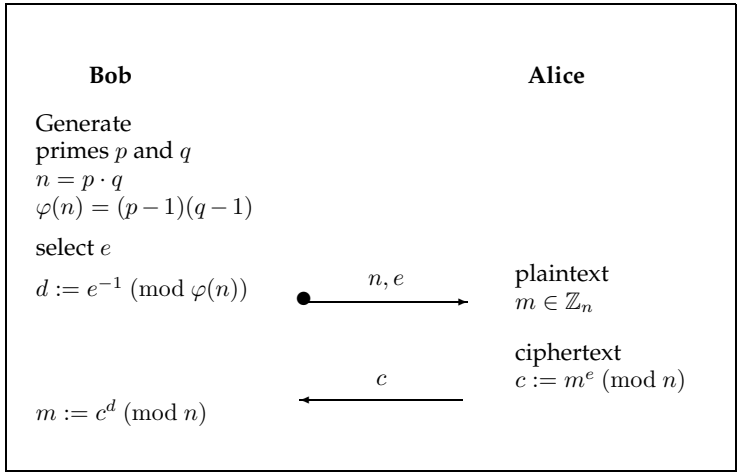

Figure 3.12: The naïve or plain RSA public-key cryptosystem. Bob's public key is the pair (n, e) and his secret key is d. Alice can encrypt a message m, represented as a number in $\mathbb{Z}_n$, by raising it to the e-th power modulo n. Bob decrypts a ciphertext c by raising it to the d-th power. The symbol • indicates that this communication is authenticated.

3.7 Digital Signatures

We have seen that the Diffie-Hellman protocol is not secure unless the values y_A and y_B are exchanged via authenticated channels. We can not use a MAC scheme to guarantee the authenticity because Alice and Bob do not share a key (yet); generating such a key is the goal of executing the protocol. But one can use digital signatures to authenticate the messages, assuming that Alice and Bob know each other's signature public key. This can be achieved, for example, by certificates.

3.7.1 Concept and Definition

The concept of a digital signature scheme is shown in Figure 3.13. A user (on the left) generates a key pair consisting of a secret signing key (or private key) and the corresponding public signature verification key, which he makes public, for instance by putting it on his web-page, authenticated by a certificate. Knowledge of the secret key enables one to sign messages, while the public key allows one to verify signatures.

Informally, a digital signature scheme consists of three efficient algorithms:

- The *key generator* is a probabilistic algorithm which generates a key pair, consisting of a *signing key* (or secret key or private key) and a *signature verification key* (or public key), according to a certain probability distribution.
- The *signing algorithm* takes as inputs a signing key and a message and computes the signature for the message. Signing can be probabilistic.
- The *signature verification algorithm* takes as inputs a signature verification key, a message, and a signature, and outputs a bit (which can be interpreted as "accept" or "reject").

More formally, we can define a digital signature scheme as follows:

Definition 3.17. A *digital signature scheme (DSS)* for message space $\mathcal{M}$, signature set $\mathcal{S}$, verification-key set $\mathcal{V}$, and signing-key set $\mathcal{Z}$, consists of:

- a *key-pair distribution*, a probability distribution over $\mathcal{V} \times \mathcal{Z}$,
- a *signing function*[33] σ: $\mathcal{M} \times \mathcal{Z} \to \mathcal{S}$,
- a *verification function* τ: $\mathcal{M} \times \mathcal{S} \times \mathcal{V} \to \{0, 1\}$,

such that signatures are correctly verified with probability 1 (correctness), i.e., $\tau(m, \sigma(m, z), v) = 1$ for all (v, z) with positive probability and for all $m \in \mathcal{M}$.

The practicality condition is that efficient algorithms for sampling from the key-pair distribution and for signing and verification are given.

3.7.2 Security Definition

Security of a DSS is defined via the following game (a game in the sense of Section 2.1.1), where, as usual, security means that no feasible (adversary) algorithm has a non-negligible success probability. We describe the game in the usual terminology of the cryptographic literature, and it is immediate to phrase it as a system in the sense of Section 2.1.1.

Definition 3.18. *t-message signature forgery game G_t^{sig}*:

1. A random secret key/public key pair (z, v) according to the key-pair distribution, and output (upon request).
2. G_t^{sig} takes up to t messages as input and returns the signature: For every message m, $\sigma(m, z)$ is returned.
3. G_t^{sig} takes an input (m', s) for $m' \in \mathcal{M}$ and $s \in \mathcal{S}$. The game is won if $\tau(m', s, v) = 1$ and m' was not given as input in Step 2.[34]

[33] This function could also be randomized.

[34] Why is the winning condition in the above game not defined as $s = \sigma(m', z)$ and m' was not given as input in Step 2?

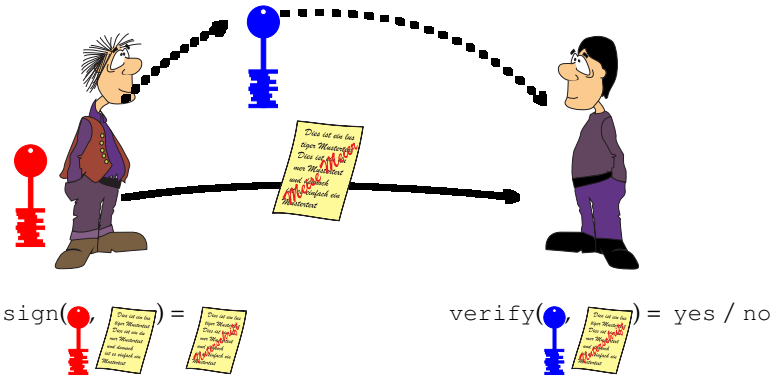

Figure 3.13: Illustration of a digital signature scheme.

3.7.3 An RSA-based Signature Scheme

The RSA TOWP (or, more generally, any TOWP) can also be used for generating digital signatures. One exploits the fact that anybody knowing the public key can compute e-th powers, whereas only the owner of the secret key can compute e-th roots.

The naïve approach would hence be to define the signature z for a message m (represented as an element of $\mathbb{Z}_n$) as the e-th root of m, i.e., $s = m^{1/e} = m^d$. However, such a scheme would be insecure. One attack exploits the multiplicative property of RSA (namely, we have $(mm')^d = m^d m'^d$). One can hence forge the signature for the message mm' (the product is modulo n) if one has obtained the signatures for m and m'.

To turn the RSA-TOWP into a secure signature scheme, one must use a carefully designed encoding of the message into an element of $\mathbb{Z}_n$. This will not be discussed here.

Another problem is that one wants to sign long messages, not just messages that can be represented as a number modulo n. A natural approach to encoding messages is therefore to apply a so-called cryptographic hash function h (see Section 3.7.4 below), to the message before applying the d-th power operation: Alice's signature s for a message m is

$$s := (h(m))^d \pmod{n}.$$

A signature z can be verified by checking the equation

$$s^e \equiv_n h(m).$$

This signature scheme (see Figure 3.14) is sometimes referred to as *full domain hash (FDH)*.

A necessary security requirement is that it must be infeasible to find two messages m and m' with the same hash value, i.e., such that $h(m) = h(m')$.

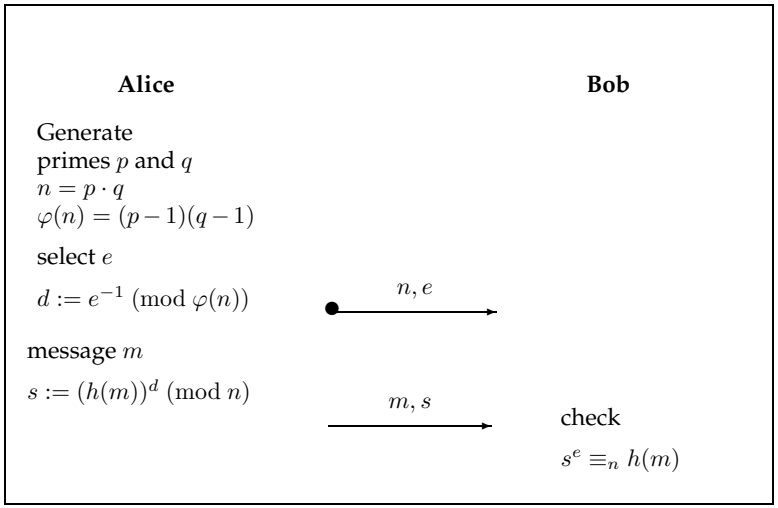

Figure 3.14: The full-domain hash (FDH) digital signature scheme based on the RSA TOWP.

(Such a hash function h is so-called *collision-resistant*; see below). Otherwise an attacker could generate two message hashing to the same value, have one of them signed, and thereby also have a signature for the other message.

However, collision-resistance is not sufficient for the FDH signature scheme to be secure. If the hash function were replaced by a purely theoretical model of a truly random function (a so-called *random oracle*) then the security of the FDH signature scheme can be proved. One talks about a security proof in the so-called *random oracle model (ROM)*. Unfortunately, one can show that there cannot exist a hash function that "behaves like" a random oracle. This means that a proof in the ROM cannot be carried over into a security proof when a concrete hash function is used, no matter how cleverly it is designed.

3.7.4 Collision-Resistant Hash Functions

Definition 3.19. A *hash function* is a function $h : D \to R$ where $|D| \gg |R|$, typically $D = \{0,1\}^*$ and $R = \{0,1\}^k$ for some suitable k, e.g. $k = 160$.

Hash functions (of course, efficiently computable ones) have many applications in computer science, including databases. The purpose of a hash function is to assign a (in a certain sense) unique short tag to an arbitrarily long bit-string. In non-cryptographic applications it should be very unlikely that two different bit-strings occurring in the application map to the same tag. In cryptography it

is important that the uniqueness is guaranteed even in presence of a malicious adversary trying to provoke such a collision.

Definition 3.20. *Collision-finding game* G_t^{coll} for a hash function h:

1. For up to t inputs x, G_t^{coll} returns $h(x)$.
2. The game is won if in Step 1, $h(x) = h(x')$ for two distinct inputs $x \neq x'$.

Definition 3.21. (Informal.) A parameterized hash function family $\{h_c : c \in \mathcal{C}\}$ (e.g. $\mathcal{C} = \{0,1\}^s$ for some s) is *collision-resistant* if no feasible algorithm wins the above collision-finding game for a uniformly random $c \in \mathcal{C}$ (given to the algorithm) with non-negligible success probability.

We make a few remarks about these definitions:

- Note that a typical value z in the range of h has many preimages $x \in D$ with $h(x) = z$. What the definition asks for is only that it be difficult to *find* two such values, not that they do not exist.
- Definition 3.21 makes no sense for a fixed hash function h (even though one sometimes finds it stated in the literature for a concrete hash function). The reason is that a fixed *instance* of a computational problem cannot be hard, only a computational problem with a large instance set can be hard. More concretely, for a fixed h, there exists (of course) an algorithm that outputs a collision for h, namely a collision fixed in its code.[35] For this reason one often sees the following definition.

For a hash function with k-bit output one can find collisions in time $O(2^{k/2})$, due to the so-called birthday paradox. Some well-known hash functions are SHA-1 and MD5. There have been some recent impressive successes in the cryptanalysis of these hash functions, leading to severe potential problems in some applications. These attacks reduce the number of operations needed for finding a collision considerably below the $O(2^{k/2})$ required for a brute-force attack. In 2015, the U.S. National Institute of Standards and Technology (NIST) has standardized a hash function under the name SHA-3 after a serious scientific competition for hash function proposals. SHA-3 is generally considered to be secure. Note that all these hash functions are not defined as *families* and hence are not collision-resistant in the sense of Definition 3.21.

3.7.5 Hash-then-Sign

There are other ways (not discussed here) for obtaining a signature scheme from the RSA-TOWP or from other principles. We do not discuss how to design such a scheme that is provably secure if the RSA-TOWP is secure, and we just mention that such methods appearing in standards are not provably secure.

[35] This does not mean that anybody knows such an algorithm.

Such a construction usually allows only to sign messages of a fixed length (for example 256 bits). Assume hence that we are given a secure signature scheme for fixed-length messages. In practice one wants to sign messages of arbitrary length. Again, a natural approach to obtaining a signature scheme for messages of arbitrary length is to simply hash the message and then to apply the basic signature scheme to the hash value. This is known as the *hash-then-sign* principle.

One can prove (as an exercise) that if the basic signature scheme is secure and the hash function is collision-resistant, then the *hash-then-sign* principle yields a secure signature scheme. More precisely, one can show that any algorithm for forging signatures can either be used to find a collision for the hash function or to find a forgery for the basic signature scheme.

Note that the full-domain hash (FDH) signature scheme for a TOWP and the hash-then-sign principle discussed here are different concepts, despite the fact that in both the message is first hashed. The TOWP is not by itself a signature scheme, which is why FDH is generally not provably secure without making additional assumptions.

3.8 Wrapping up this Chapter

In this chapter we have seen a number of fundamental cryptographic concepts as well as some concrete schemes. We have discussed some security definitions for these schemes. For the one-time pad, and also informally for MACs, we have illustrated the constructive viewpoint we will be using later in this course.

The story of the chapter results in a concrete protocol for establishing a secure channel between two entities Alice and Bob, assuming Alice and Bob have an authenticated copy of their communication partner's signature verification key, i.e., we assume the initial availability of single-use authenticated channels from Alice to Bob and from Bob to Alice. These channels are in practice constructed by the use of certificates.

In summary, and informally described, the protocol works as follows. We only consider the case where a single message is sent securely, but by appropriate changes one obtains a protocol for sending an arbitrary number of messages.

1. Alice and Bob execute the Diffie-Hellman protocol, where the two Diffie-Hellman messages are signed using the RSA-based FDH signature scheme, for a specific hash function h. The signatures on the received messages are verified, and if the verification fails for a party, that party aborts the protocol.
2. Alice and Bob each use the resulting key to generate two keys, using a PRG g.
3. Alice expands the first of these keys, using the PRBG based on iterated application of g, to obtain a key as long as the message to be transmitted.

4. Alice uses the one-time pad with the generated key to encrypt the message.
5. Alice uses a MAC scheme, with the second key from step 2, to compute a MAC for the ciphertext.
6. Alice sends the ciphertext and the MAC to Bob over an insecure channel.
7. Bob verifies the received MAC on the received ciphertext, using the second key. If the check fails, he aborts.
8. Bob expands the first of the two keys, using the PRBG.
9. Bob decrypts the ciphertext and outputs the message.

This protocol can be proved to be secure under certain assumptions. Secure here means that it constructs a secure communication channel according to the constructive cryptography paradigm. The assumptions are:[36]

- The DDH-problem is hard for the group used in the Diffie-Hellman protocol.
- The RSA function is a TOWP (according to Definition 3.14).
- The function g is a PRG (according to Definition 3.4).
- The MAC scheme is secure (according to Definition 3.6).

Turned around, this means that any distinguisher that could distinguish the "ideal setting" from the "real setting" could be used (by a reduction) to break one of the above assumptions.

However, this only holds in an idealized setting where the hash function h is replaced by a random oracle. Such a proof is called a proof in the random oracle model.

So what remains to be done? The following questions, most of which will be addressed in this course, are now well-motivated. Some of the questions are concrete, and some are of a general conceptual type.

- What kind of mathematical objects are computational problems, methods for solving them, and reductions? And how do we measure how well a method solves a problem?
- Can the assumption that a hash function behaves like a random oracle be made precise, and can there be a hash function with this very strong property? We will actually see that the answer to the latter question is negative, which means that the random oracle model is in a sense not sound.
- How can we prove that the FDH signature scheme is secure if the underlying function (e.g. RSA) is a TOWP (and h is a random oracle)?

[36] Of course, one also needs to assume that the certification authority works correctly, that the computers on which the protocol is running are not compromised, etc.

- Can one show that RSA is a TOWP, based on the assumption that factoring is hard?
- How can we argue that the DDH problem is indeed hard. Which groups must be excluded because the DDH problem is easy?
- Why is the continued PRG-expansion a secure PRBG?
- How can one construct a PRG from weaker assumptions, for example that a certain function is hard to invert (i.e., is a one-way function)?
- Why is the CBC-MAC scheme secure if the underlying block cipher is secure, and are there better MAC schemes?
- How can we develop a framework in which the individual security statements for the various schemes used can naturally be composed? In other words, how can we provably exclude a failure of our intuition or hidden effects resulting from an unexpected interference between the security definitions of the various schemes used?
- Which other tasks, besides secure communication, can be solved by cryptography?

In order to address these and many other questions, we need to proceed with mathematical precision and rigor.