CHAPTER 1

Introduction

In this chapter we briefly discuss the goals of cryptography (Section 1.1). In particular, we discuss the basic problems of secure encryption, digital signatures, and fault-tolerant protocols. These problems lead to the notions of pseudorandom generators and zero-knowledge proofs, which are discussed as well.

Our approach to cryptography is based on computational complexity. Hence, this introductory chapter also contains a section presenting the computational models used throughout the book (Section 1.3). Likewise, this chapter contains a section presenting some elementary background from probability theory that is used extensively in the book (Section 1.2).

Finally, we motivate the rigorous approach employed throughout this book and discuss some of its aspects (Section 1.4).

Teaching Tip. Parts of Section 1.4 may be more suitable for the last lecture (i.e., as part of the concluding remarks) than for the first one (i.e., as part of the introductory remarks). This refers specifically to Sections 1.4.2 and 1.4.3.

1.1. Cryptography: Main Topics

Historically, the term "cryptography" has been associated with the problem of designing and analyzing *encryption schemes* (i.e., schemes that provide secret communication over insecure communication media). However, since the 1970s, problems such as constructing unforgeable *digital signatures* and designing *fault-tolerant protocols* have also been considered as falling within the domain of cryptography. In fact, cryptography can be viewed as concerned with the design of any system that needs to withstand malicious attempts to abuse it. Furthermore, cryptography as redefined here makes essential use of some tools that need to be treated in a book on the subject. Notable examples include one-way functions, pseudorandom generators, and zero-knowledge proofs. In this section we briefly discuss these terms.

We start by mentioning that much of the content of this book relies on the assumption that one-way functions exist. The definition of one-way functions captures the sort of computational difficulty that is inherent to our entire approach to cryptography, an approach that attempts to capitalize on the computational limitations of any real-life adversary. Thus, if nothing is difficult, then this approach fails. However, if, as is widely believed, not only do hard problems exist but also instances of them can be efficiently generated, then these hard problems can be "put to work." Thus, "algorithmically bad news" (by which hard computational problems exist) implies good news for cryptography. Chapter 2 is devoted to the definition and manipulation of computational difficulty in the form of one-way functions.

1.1.1. Encryption Schemes

The problem of providing *secret communication over insecure media* is the most traditional and basic problem of cryptography. The setting consists of two parties communicating over a channel that possibly may be tapped by an adversary, called the *wire-tapper*. The parties wish to exchange information with each other, but keep the wire-tapper as ignorant as possible regarding the content of this information. Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called *encryption*, is applied by the sender (i.e., the party sending a message), while the other algorithm, called *decryption*, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message and sends the result, called the *ciphertext*, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it and retrieves the original message (called the *plaintext*).

In order for this scheme to provide secret communication, the communicating parties (at least the receiver) must know something that is not known to the wire-tapper. (Otherwise, the wire-tapper could decrypt the ciphertext exactly as done by the receiver.) This extra knowledge may take the form of the decryption algorithm itself or some parameters and/or auxiliary inputs used by the decryption algorithm. We call this extra knowledge the *decryption key*. Note that, without loss of generality, we can assume that the decryption algorithm is known to the wire-tapper and that the decryption algorithm needs two inputs: a ciphertext and a decryption key. We stress that the existence of a secret key, not known to the wire-tapper, is merely a necessary condition for secret communication.

Evaluating the "security" of an encryption scheme is a very tricky business. A preliminary task is to understand what "security" is (i.e., to properly define what is meant by this intuitive term). Two approaches to defining security are known. The first ("classic") approach is *information-theoretic*. It is concerned with the "information" about the plaintext that is "present" in the ciphertext. Loosely speaking, if the ciphertext contains information about the plaintext, then the encryption scheme is considered insecure. It has been shown that such a high (i.e., "perfect") level of security can be achieved only if the key in use is at least as long as the *total* length of the messages sent via the encryption scheme. The fact that the key has to be longer than the information exchanged using it is indeed a drastic limitation on the applicability of such encryption schemes. This is especially true when *huge* amounts of information need to be secretly communicated.

The second ("modern") approach, as followed in this book, is based on *computational complexity*. This approach is based on the fact that *it does not matter whether or not the ciphertext contains information about the plaintext*, but rather *whether or not this information can be efficiently extracted*. In other words, instead of asking whether or not it is *possible* for the wire-tapper to extract specific information, we ask whether or not it is *feasible* for the wire-tapper to extract this information. It turns out that the new (i.e., "computational-complexity") approach offers security even if the key is much shorter than the total length of the messages sent via the encryption scheme. For example, one can use "pseudorandom generators" (discussed later) that expand short keys into much longer "pseudo-keys," so that the latter are as secure as "real keys" of comparable length.

In addition, the computational-complexity approach allows the introduction of concepts and primitives that cannot exist under the information-theoretic approach. A typical example is the concept of *public-key encryption schemes*. Note that in the preceding discussion we concentrated on the decryption algorithm and its key. It can be shown that the encryption algorithm must get, in addition to the message, an auxiliary input that depends on the decryption key. This auxiliary input is called the *encryption key*. Traditional encryption schemes, and in particular all the encryption schemes used over the millennia preceding the 1980s, operate with an encryption key equal to the decryption key. Hence, the wire-tapper in these schemes must be ignorant of the encryption key, and consequently the *key-distribution problem* arises (i.e., how two parties wishing to communicate over an insecure channel can agree on a secret encryption/decryption key).[1] The computational-complexity approach allows the introduction of encryption schemes in which the encryption key can be known to the wire-tapper without compromising the security of the scheme. Clearly, the decryption key in such schemes is different from the encryption key, and furthermore it is infeasible to compute the decryption key from the encryption key. Such encryption schemes, called *public-key schemes*, have the advantage of trivially resolving the key-distribution problem, because the encryption key can be publicized.

In Chapter 5, which will appear in the second volume of this work and will be devoted to encryption schemes, we shall discuss private-key and public-key encryption schemes. Much attention is devoted to defining the security of encryption schemes. Finally, constructions of secure encryption schemes based on various intractability assumptions are presented. Some of the constructions presented are based on pseudorandom generators, which are discussed in Chapter 3. Other constructions use specific one-way functions such as the RSA function and/or the operation of squaring modulo a composite number.

1.1.2. Pseudorandom Generators

It turns out that pseudorandom generators play a central role in the construction of encryption schemes (and related schemes). In particular, pseudorandom generators

[1] The traditional solution is to exchange the key through an alternative channel that is secure, alas "more expensive to use," for example, by a convoy.

yield simple constructions of private-key encryption schemes, and this observation is often used in practice (usually implicitly).

Although the term "pseudorandom generators" is commonly used *in practice*, both in the context of cryptography and in the much wider context of probabilistic procedures, it is seldom associated with a precise meaning. We believe that using a term without clearly stating what it means is dangerous in general and particularly so in a tricky business such as cryptography. Hence, a precise treatment of pseudorandom generators is central to cryptography.

Loosely speaking, a pseudorandom generator is a deterministic algorithm that expands short random seeds into much longer bit sequences that *appear* to be "random" (although they are not). In other words, although the output of a pseudorandom generator is not really random, it is *infeasible* to tell the difference. It turns out that pseudorandomness and computational difficulty are linked in an even more fundamental manner, as pseudorandom generators can be constructed based on various intractability assumptions. Furthermore, the main result in this area asserts that pseudorandom generators exist if and only if one-way functions exist.

Chapter 3, devoted to pseudorandom generators, starts with a treatment of the concept of computational indistinguishability. Pseudorandom generators are defined next and are constructed using special types of one-way functions (defined in Chapter 2). Pseudorandom *functions* are defined and constructed as well. The latter offer a host of additional applications.

1.1.3. Digital Signatures

A notion that did not exist in the pre-computerized world is that of a "digital signature." The need to discuss digital signatures arose with the introduction of computer communication in the business environment in which parties need to commit themselves to proposals and/or declarations they make. Discussions of "unforgeable signatures" also took place in previous centuries, but the objects of discussion were handwritten signatures, not digital ones, and the discussion was not perceived as related to cryptography.

Relations between encryption and signature methods became possible with the "digitalization" of both and the introduction of the computational-complexity approach to security. Loosely speaking, a *scheme for unforgeable signatures* requires

- that each user be able *to efficiently generate his or her own signature* on documents of his or her choice,
- that each user be able *to efficiently verify* whether or not a given string is a signature of another (specific) user on a specific document, and
- that *no one be able to efficiently produce the signatures of other users* to documents that those users did not sign.

We stress that the formulation of unforgeable digital signatures also provides a clear statement of the essential ingredients of handwritten signatures. Indeed, the ingredients are each person's ability to sign for himself or herself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least difficult) to forge signatures in a manner that could pass the verification procedure. It is difficult to state to what extent handwritten signatures meet these requirements. In contrast, our discussion of digital signatures will supply precise statements concerning the extent to which digital signatures meet the foregoing requirements. Furthermore, schemes for unforgeable digital signatures can be constructed using the same computational assumptions as used in the construction of (private-key) encryption schemes.

In Chapter 6, which will appear in the second volume of this work and will be devoted to signature schemes, much attention will be focused on defining the security (i.e., unforgeability) of these schemes. Next, constructions of unforgeable signature schemes based on various intractability assumptions will be presented. In addition, we shall treat the related problem of message authentication.

Message Authentication

Message authentication is a task related to the setting considered for encryption schemes (i.e., communication over an insecure channel). This time, we consider the case of an active adversary who is monitoring the channel and may alter the messages sent on it. The parties communicating through this insecure channel wish to authenticate the messages they send so that the intended recipient can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a *scheme for message authentication* requires

- that each of the communicating parties be able *to efficiently generate an authentication tag* for any message of his or her choice,
- that each of the communicating parties be able *to efficiently verify* whether or not a given string is an authentication tag for a given message, and
- that *no external adversary* (i.e., a party other than the communicating parties) *be able to efficiently produce authentication tags* to messages not sent by the communicating parties.

In some sense, "message authentication" is similar to a digital signature. The difference between the two is that in the setting of message authentication it is not required that third parties (who may be dishonest) be able to verify the validity of authentication tags produced by the designated users, whereas in the setting of signature schemes it is required that such third parties be able to verify the validity of signatures produced by other users. Hence, digital signatures provide a solution to the message-authentication problem. On the other hand, a message-authentication scheme does not necessarily constitute a digital-signature scheme.

Signatures Widen the Scope of Cryptography

Considering the problem of digital signatures as belonging to cryptography widens the scope of this area from the specific secret-communication problem to a variety of problems concerned with limiting the "gain" that can be achieved by "dishonest" behavior of parties (who are either internal or external to the system). Specifically:

- In the secret-communication problem (solved by use of encryption schemes), one wishes to reduce, as much as possible, the information that a potential wire-tapper can extract from the communication between two designated users. In this case, the designated system consists of the two communicating parties, and the wire-tapper is considered as an external ("dishonest") party.
- In the message-authentication problem, one aims at prohibiting any (external) wire-tapper from modifying the communication between two (designated) users.
- In the signature problem, one aims at providing all users of a system a way of making self-binding statements and of ensuring that one user cannot make statements that would bind another user. In this case, the designated system consists of the set of all users, and a potential forger is considered as an internal yet dishonest user.

Hence, in the wide sense, *cryptography is concerned with any problem in which one wishes to limit the effects of dishonest users*. A general treatment of such problems is captured by the treatment of "fault-tolerant" (or cryptographic) protocols.

1.1.4. Fault-Tolerant Protocols and Zero-Knowledge Proofs

A discussion of signature schemes naturally leads to a discussion of cryptographic protocols, because it is a natural concern to ask under what circumstances one party should provide its signature to another party. In particular, problems like mutual simultaneous commitment (e.g., contract signing) arise naturally. Another type of problem, motivated by the use of computer communication in the business environment, consists of "secure implementation" of protocols (e.g., implementing secret and incorruptible voting).

Simultaneity Problems

A typical example of a simultaneity problem is that of simultaneous exchange of secrets, of which contract signing is a special case. The setting for a simultaneous exchange of secrets consists of two parties, each holding a "secret." The goal is to execute a protocol such that if both parties follow it correctly, then at termination each will hold its counterpart's secret, and in any case (even if one party cheats) the first party will hold the second party's secret if and only if the second party holds the first party's secret. Perfectly simultaneous exchange of secrets can be achieved only if we assume the existence of third parties that are trusted to some extent. In fact, simultaneous exchange of secrets can easily be achieved using the active participation of a trusted third party: Each party sends its secret to the trusted third party (using a secure channel). The third party, on receiving both secrets, sends the first party's secret to the second party and the second party's secret to the first party. There are two problems with this solution:

1. The solution requires the *active* participation of an "external" party in all cases (i.e., also in case both parties are honest). We note that other solutions requiring milder forms of participation of external parties do exist.
2. The solution requires the existence of a *totally trusted* third entity. In some applications, such an entity does not exist. Nevertheless, in the sequel we shall discuss the problem

of implementing a trusted third party by a set of users with an honest majority (even if the identity of the honest users is not known).

Secure Implementation of Functionalities and Trusted Parties

A different type of protocol problem is concerned with the secure implementation of functionalities. To be more specific, we discuss the problem of evaluating a function of local inputs each of which is held by a different user. An illustrative and motivating example is *voting*, in which the function is majority, and the local input held by user A is a single bit representing the vote of user A (e.g., "pro" or "con"). Loosely speaking, a protocol for securely evaluating a specific function must satisfy the following:

- *Privacy*: No party can "gain information" on the input of other parties, beyond what is deduced from the value of the function.
- *Robustness*: No party can "influence" the value of the function, beyond the influence exerted by selecting its own input.

It is sometimes required that these conditions hold with respect to "small" (e.g., minority) coalitions of parties (instead of single parties).

Clearly, if one of the users is known to be totally trustworthy, then there exists a simple solution to the problem of secure evaluation of any function. Each user simply sends its input to the trusted party (using a secure channel), who, upon receiving all inputs, computes the function, sends the outcome to all users, and erases all intermediate computations (including the inputs received) from its memory. Certainly, it is unrealistic to assume that a party can be trusted to such an extent (e.g., that it will voluntarily erase what it has "learned"). Nevertheless, the problem of implementing secure function evaluation reduces to the problem of implementing a trusted party. It turns out that a trusted party can be implemented by a set of users with an honest majority (even if the identity of the honest users is not known). This is indeed a major result in this field, and much of Chapter 7, which will appear in the second volume of this work, will be devoted to formulating and proving it (as well as variants of it).

Zero-Knowledge as a Paradigm

A major tool in the construction of cryptographic protocols is the concept of *zero-knowledge* proof systems and the fact that zero-knowledge proof systems exist for all languages in $\mathcal{NP}$ (provided that one-way functions exist). Loosely speaking, a zero-knowledge proof yields nothing but the validity of the assertion. Zero-knowledge proofs provide a tool for "forcing" parties to follow a given protocol properly.

To illustrate the role of zero-knowledge proofs, consider a setting in which a party, called Alice, upon receiving an encrypted message from Bob, is to send Carol the least significant bit of the message. Clearly, if Alice sends only the (least significant) bit (of the message), then there is no way for Carol to know Alice did not cheat. Alice could prove that she did not cheat by revealing to Carol the entire message as well as its decryption key, but that would yield information far beyond what had been required. A much better idea is to let Alice augment the bit she sends Carol with a zero-knowledge proof that this bit is indeed the least significant bit of the message. We stress that the foregoing statement is of the "$\mathcal{NP}$ type" (since the proof specified earlier can be efficiently verified), and therefore the existence of zero-knowledge proofs for $\mathcal{NP}$-statements implies that the foregoing statement can be proved without revealing anything beyond its validity.

The focus of Chapter 4, devoted to zero-knowledge proofs, is on the foregoing result (i.e., the construction of zero-knowledge proofs for any $\mathcal{NP}$-statement). In addition, we shall consider numerous variants and aspects of the notion of zero-knowledge proofs and their effects on the applicability of this notion.

1.2. Some Background from Probability Theory

Probability plays a central role in cryptography. In particular, probability is essential in order to allow a discussion of information or lack of information (i.e., secrecy). We assume that the reader is familiar with the basic notions of probability theory. In this section, we merely present the probabilistic notations that are used throughout this book and three useful probabilistic inequalities.

1.2.1. Notational Conventions

Throughout this entire book we shall refer to only *discrete* probability distributions. Typically, the probability space consists of the set of all strings of a certain length ℓ, taken with uniform probability distribution. That is, the sample space is the set of all ℓ-bit-long strings, and each such string is assigned probability measure $2^{-\ell}$. Traditionally, functions from the sample space to the reals are called *random variables*. Abusing standard terminology, we allow ourselves to use the term *random variable* also when referring to functions mapping the sample space into the set of binary strings. We often do not specify the probability space, but rather talk directly about random variables. For example, we may say that X is a random variable assigned values in the set of all strings, so that $\Pr[X = 00] = \frac{1}{4}$ and $\Pr[X = 111] = \frac{3}{4}$. (Such a random variable can be defined over the sample space $\{0, 1\}^2$, so that $X(11) = 00$ and $X(00) = X(01) = X(10) = 111$.) In most cases the probability space consists of all strings of a particular length. Typically, these strings represent random choices made by some randomized process (see next section), and the random variable is the output of the process.

How to Read Probabilistic Statements. All our probabilistic statements refer to functions of random variables that are defined beforehand. Typically, we shall write $\Pr[f(X) = 1]$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of a given symbol in a probabilistic statement refer to the same (unique) random variable*. Hence, if $B(\cdot, \cdot)$ is a Boolean expression depending on two variables and X is a random variable, then $\Pr[B(X, X)]$ denotes the probability that $B(x, x)$ holds when x is chosen with probability $\Pr[X = x]$.

Namely,

$$\mathsf{Pr}[B(X, X)] = \sum_x \mathsf{Pr}[X = x] \cdot \chi(B(x, x))$$

where χ is an indicator function, so that $\chi(B) = 1$ if event B holds, and equals zero otherwise. For example, for every random variable X, we have $\mathsf{Pr}[X = X] = 1$. We stress that if one wishes to discuss the probability that $B(x, y)$ holds when x and y are chosen independently with the same probability distribution, then one needs to define *two* independent random variables, both with the same probability distribution. Hence, if X and Y are two independent random variables, then $\mathsf{Pr}[B(X, Y)]$ denotes the probability that $B(x, y)$ holds when the pair (x, y) is chosen with probability $\mathsf{Pr}[X = x] \cdot \mathsf{Pr}[Y = y]$. Namely,

$$\mathsf{Pr}[B(X, Y)] = \sum_{x,y} \mathsf{Pr}[X = x] \cdot \mathsf{Pr}[Y = y] \cdot \chi(B(x, y))$$

For example, for every two independent random variables, X and Y, we have $\mathsf{Pr}[X = Y] = 1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Typical Random Variables. Throughout this entire book, U_n denotes a random variable uniformly distributed over the set of strings of length n. Namely, $\mathsf{Pr}[U_n = \alpha]$ equals 2^{-n} if $\alpha \in \{0, 1\}^n$, and equals zero otherwise. In addition, we shall occasionally use random variables (arbitrarily) distributed over $\{0, 1\}^n$ or $\{0, 1\}^{l(n)}$ for some function $l : \mathbb{N} \to \mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over $\{0, 1\}^n$, whereas in others it is distributed over $\{0, 1\}^{l(n)}$, for some function $l(\cdot)$, which is typically a polynomial. Another type of random variable, the output of a randomized algorithm on a fixed input, is discussed in Section 1.3.

1.2.2. Three Inequalities

The following probabilistic inequalities will be very useful in the course of this book. All inequalities refer to random variables that are assigned real values. The most basic inequality is the *Markov inequality*, which asserts that for random variables with bounded maximum or minimum values, some relation must exist between the deviation of a value from the expectation of the random variable and the probability that the random variable is assigned this value. Specifically, letting $\mathsf{E}(X) \stackrel{\text{def}}{=} \sum_v \mathsf{Pr}[X = v] \cdot v$ denote the expectation of the random variable X, we have the following:

Markov Inequality: *Let X be a non-negative random variable and v a real number. Then*

$$\mathsf{Pr}[X \geq v] \leq \frac{\mathsf{E}(X)}{v}$$

Equivalently, $\mathsf{Pr}[X \geq r \cdot \mathsf{E}(X)] \leq \frac{1}{r}$.

Proof:

$$\begin{aligned}\mathsf{E}(X) &= \sum_x \mathsf{Pr}[X = x] \cdot x \\ &\geq \sum_{x<v} \mathsf{Pr}[X = x] \cdot 0 + \sum_{x \geq v} \mathsf{Pr}[X = x] \cdot v \\ &= \mathsf{Pr}[X \geq v] \cdot v\end{aligned}$$

The claim follows. ■

The Markov inequality is typically used in cases in which one knows very little about the distribution of the random variable; it suffices to know its expectation and at least one bound on the range of its values. See Exercise 1.

Using Markov's inequality, one gets a "possibly stronger" bound for the deviation of a random variable from its expectation. This bound, called Chebyshev's inequality, is useful provided one has additional knowledge concerning the random variable (specifically, a good upper bound on its variance). For a random variable X of finite expectation, we denote by $\mathsf{Var}(X) \stackrel{\text{def}}{=} \mathsf{E}[(X - \mathsf{E}(X))^2]$ the variance of X and observe that $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$.

Chebyshev's Inequality: *Let X be a random variable, and $\delta > 0$. Then*

$$\mathsf{Pr}[|X - \mathsf{E}(X)| \geq \delta] \leq \frac{\mathsf{Var}(X)}{\delta^2}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{E}(X))^2$ and apply the Markov inequality. We get

$$\begin{aligned}\mathsf{Pr}[|X - \mathsf{E}(X)| \geq \delta] &= \mathsf{Pr}[(X - \mathsf{E}(X))^2 \geq \delta^2] \\ &\leq \frac{\mathsf{E}[(X - \mathsf{E}(X))^2]}{\delta^2}\end{aligned}$$

and the claim follows. ■

Chebyshev's inequality is particularly useful for analysis of the error probability of approximation via repeated sampling. It suffices to assume that the samples are picked in a pairwise-independent manner.

Corollary (Pairwise-Independent Sampling): *Let $X_1, X_2, \ldots, X_n$ be pairwise-independent random variables with the same expectation, denoted μ, and the same variance, denoted σ^2. Then, for every $\varepsilon > 0$,*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\sigma^2}{\varepsilon^2 n}$$

The X_i's are called *pairwise-independent* if for every $i \neq j$ and all a and b, it holds that $\mathsf{Pr}[X_i = a \wedge X_j = b]$ equals $\mathsf{Pr}[X_i = a] \cdot \mathsf{Pr}[X_j = b]$.