Introduction

Cryptography is the science of keeping secrets secret. Assume a sender referred to here and in what follows as *Alice* (as is commonly used) wants to send a message m to a receiver referred to as *Bob*. She uses an insecure communication channel. For example, the channel could be a computer network or a telephone line. There is a problem if the message contains confidential information. The message could be intercepted and read by an eavesdropper. Or, even worse, the adversary, as usual referred to here as *Eve*, might be able to modify the message during transmission in such a way that the legitimate recipient Bob does not detect the manipulation.

One objective of cryptography is to provide methods for preventing such attacks. Other objectives are discussed in Section .

Encryption and Secrecy

The fundamental and classical task of cryptography is to provide *confidentiality* by *encryption methods*. The message to be transmitted – it can be some text, numerical data, an executable program or any other kind of information – is called the *plaintext*. Alice *encrypts* the plaintext m and obtains the *ciphertext* c. The ciphertext c is transmitted to Bob. Bob turns the ciphertext back into the plaintext by *decryption*. To *decrypt*, Bob needs some secret information, a secret *decryption key*.[1] Adversary Eve still may intercept the ciphertext. However, the encryption should guarantee secrecy and prevent her from deriving any information about the plaintext from the observed ciphertext.

Encryption is very old. For example, *Caesar's shift cipher*[2] was introduced more than 2000 years ago. Every encryption method provides an encryption algorithm E and a decryption algorithm D. In classical encryption schemes, both algorithms depend on the same secret key k. This key k is used for both encryption and decryption. These encryption methods are therefore called

[1] Sometimes the terms *encipher* and *decipher* are used instead of encrypt and decrypt.

[2] Each plaintext character is replaced by the character 3 to the right modulo 26, i.e., a is replaced by d, b by e,..., x by a, y by b and z by c.

symmetric. For example, in Caesar's cipher the secret key is the offset 3 of the shift. We have

$$D(k, E(k, m)) = m \text{ for each plaintext } m.$$

Symmetric encryption and the important examples DES (data encryption standard) and AES (advanced encryption standard) are discussed in Chapter 1.

In 1976, W. Diffie and M.E. Hellman published their famous paper, *New Directions in Cryptography* ([DifHel76]). There they introduced the revolutionary concept of *public-key cryptography*. They provided a solution to the long standing problem of key exchange and pointed the way to digital signatures. The *public-key encryption* methods (comprehensively studied in Chapter 2) are *asymmetric*. Each recipient of messages has his personal key $k = (pk, sk)$, consisting of two parts: pk is the encryption key and is made public, sk is the decryption key and is kept secret. If Alice wants to send a message m to Bob, she encrypts m by use of Bob's publicly known encryption key pk. Bob decrypts the ciphertext by use of his decryption key sk, which is known only to him. We have

$$D(sk, E(pk, m)) = m.$$

Mathematically speaking, public-key encryption is a so-called *one-way function* with a *trapdoor*. Everyone can easily encrypt a plaintext using the public key pk, but the other direction is difficult. It is practically impossible to deduce the plaintext from the ciphertext, without knowing the secret key sk (which is called the trapdoor information).

Public-key encryption methods require more complex computations and are less efficient than classical symmetric methods. Thus symmetric methods are used for the encryption of large amounts of data. Before applying symmetric encryption, Alice and Bob have to agree on a key. To keep this key secret, they need a secure communication channel. It is common practice to use public-key encryption for this purpose.

The Objectives of Cryptography

Providing confidentiality is not the only objective of cryptography. Cryptography is also used to provide solutions for other problems:

1. *Data integrity*. The receiver of a message should be able to check whether the message was modified during transmission, either accidentally or deliberately. No one should be able to substitute a false message for the original message, or for parts of it.
2. *Authentication*. The receiver of a message should be able to verify its origin. No one should be able to send a message to Bob and pretend to

be Alice (*data origin authentication*). When initiating a communication, Alice and Bob should be able to identify each other (*entity authentication*).
3. *Non-repudiation*. The sender should not be able to later deny that she sent a message.

If messages are written on paper, the medium – paper – provides a certain security against manipulation. Handwritten personal signatures are intended to guarantee authentication and non-repudiation. If electronic media are used, the medium itself provides no security at all, since it is easy to replace some bytes in a message during its transmission over a computer network, and it is particularly easy if the network is publicly accessible, like the Internet.

So, while encryption has a long history,[3] the need for techniques providing data integrity and authentication resulted from the rapidly increasing significance of electronic communication.

There are symmetric as well as public-key methods to ensure the integrity of messages. Classical symmetric methods require a secret key k that is shared by sender and receiver. The message m is augmented by a *message authentication code* (MAC). The code is generated by an algorithm and depends on the secret key. The augmented message $(m, MAC(k, m))$ is protected against modifications. The receiver may test the integrity of an incoming message $(m, \overline{m})$ by checking whether

$$MAC(k, m) = \overline{m}.$$

Message authentication codes may be implemented by keyed hash functions (see Chapter 2).

Digital signatures require public-key methods (see Chapter 2 for examples and details). As with classical handwritten signatures, they are intended to provide authentication and non-repudiation. Note that non-repudiation is an indispensable feature if digital signatures are used to sign contracts. Digital signatures depend on the secret key of the signer – they can be generated only by him. On the other hand, anyone can check whether a signature is valid, by applying a publicly known verification algorithm *Verify*, which depends on the public key of the signer. If Alice wants to sign the message m, she applies the algorithm *Sign* with her secret key sk and gets the signature $Sign(sk, m)$. Bob receives a signature s for message m, and may then check the signature by testing whether

$$Verify(pk, s, m) = ok,$$

with Alice's public key pk.

It is common not to sign the message itself, but to apply a *cryptographic hash function* (see Section 2.4) first and then sign the hash value. In schemes

[3] For the long history of cryptography, see [Kahn67].

like the famous RSA (named after its inventors: Rivest, Shamir and Adleman), the decryption algorithm is used to generate signatures and the encryption algorithm is used to verify them. This approach to digital signatures is therefore often referred to as the "hash-then-decrypt" paradigm (see Section 2.4.5 for details). More sophisticated signature schemes, like the probabilistic signature scheme (PSS), require more steps. Modifying the hash value by pseudorandom sequences turns signing into a probabilistic procedure (see Section 2.4.5).

Digital signatures depend on the message. Distinct messages yield different signatures. Thus, like classical message authentication codes, digital signatures can also be used to guarantee the integrity of messages.

Attacks

The primary goal of cryptography is to keep the plaintext secret from eavesdroppers trying to get some information about the plaintext. As discussed before, adversaries may also be active and try to modify the message. Then, cryptography is expected to guarantee the integrity of the messages. Adversaries are assumed to have complete access to the communication channel.

Cryptanalysis is the science of studying attacks against cryptographic schemes. Successful attacks may, for example, recover the plaintext (or parts of the plaintext) from the ciphertext, substitute parts of the original message, or forge digital signatures. Cryptography and cryptanalysis are often subsumed by the more general term *cryptology*.

A fundamental assumption in cryptanalysis was first stated by A. Kerkhoff in the nineteenth century. It is usually referred to as *Kerkhoff's Principle*. It states that the adversary knows all the details of the cryptosystem, including algorithms and their implementations. According to this principle, the security of a cryptosystem must be entirely based on the secret keys.

Attacks on the secrecy of an encryption scheme try to recover plaintexts from ciphertexts, or even more drastically, to recover the secret key. The following survey is restricted to passive attacks. The adversary, as usual we call her Eve, does not try to modify the messages. She monitors the communication channel and the end points of the channel. So she may not only intercept the ciphertext, but (at least from time to time) she may be able to observe the encryption and decryption of messages. She has no information about the key. For example, Eve might be the operator of a bank computer. She sees incoming ciphertexts and sometimes also the corresponding plaintexts. Or she observes the outgoing plaintexts and the generated ciphertexts. Perhaps she manages to let encrypt plaintexts or decrypt ciphertexts of her own choice.

The possible attacks depend on the actual resources of the adversary Eve. They are usually classified as follows:

1. *Ciphertext-only attack*. Eve has the ability to obtain ciphertexts. This is likely to be the case in any encryption situation. Even if Eve cannot perform the more sophisticated attacks described below, one must assume that she can get access to encrypted messages. An encryption method that cannot resist a ciphertext-only attack is completely insecure.
2. *Known-plaintext attack*. Eve has the ability to obtain plaintext-ciphertext pairs. Using the information from these pairs, she attempts to decrypt a ciphertext for which she does not have the plaintext. At first glance, it might appear that such information would not ordinarily be available to an attacker. However, it very often is available. Messages may be sent in standard formats which Eve knows.
3. *Chosen-plaintext attack*. Eve has the ability to obtain ciphertexts for plaintexts of her choosing. Then she attempts to decrypt a ciphertext for which she does not have the plaintext. While again this may seem unlikely, there are many cases in which Eve can do just this. For example, she sends some interesting information to her intended victim which she is confident he will encrypt and send out. This type of attack assumes that Eve must first obtain whatever plaintext-ciphertext pairs she wants and then do her analysis, without any further interaction. This means that she only needs access to the encrypting device once.
4. *Adaptively-chosen-plaintext attack*. This is the same as the previous attack, except now Eve may do some analysis on the plaintext-ciphertext pairs, and subsequently get more pairs. She may switch between gathering pairs and performing the analysis as often as she likes. This means that she has either lengthy access to the encrypting device or can somehow make repeated use of it.
5. *Chosen- and adaptively-chosen-ciphertext attack*. These two attacks are similar to the above plaintext attacks. Eve can choose ciphertexts and gets the corresponding plaintexts. She has access to the decryption device.

Cryptographic Protocols

Encryption and decryption algorithms, cryptographic hash functions or *pseudorandom generators* (see Section 1.1, Chapter 7) are the basic building blocks (also called cryptographic primitives) for solving problems involving secrecy, authentication or data integrity.

In many cases a single building block is not sufficient to solve the given problem: different primitives must be combined. A series of steps must be executed to accomplish a given task. Such a well-defined series of steps is called a *cryptographic protocol*. As is also common, we add another condition: we require that two or more parties are involved. We only use the term protocol if at least two people are required to complete the task.

As a counter example, take a look at digital signature schemes. A typical scheme for generating a digital signature first applies a cryptographic hash function h to the message m and then, in a second step, computes the signature by applying a public-key decryption algorithm to the hash value $h(m)$. Both steps are done by one person. Thus, we do not call it a protocol.

Typical examples of protocols are protocols for user identification. There are many situations where the identity of a user Alice has to be verified. Alice wants to log in to a remote computer, for example, or to get access to an account for electronic banking. Passwords or PIN numbers are used for this purpose. This method is not always secure. For example, anyone who observes Alice's password or PIN when transmitted might be able to impersonate her. We sketch a simple *challenge-and-response* protocol which prevents this attack (however, it is not perfect; see Section 3.2.1).

The protocol is based on a public-key signature scheme, and we assume that Alice has a key $k = (pk, sk)$ for this scheme. Now, Alice can prove her identity to Bob in the following way.

1. Bob randomly chooses a "challenge" c and sends it to Alice.
2. Alice signs c with her secret key, $s := Sign(sk, c)$, and sends the "response" s to Bob.
3. Bob accepts Alice's proof of identity, if $Verify(pk, s, c) = ok$.

Only Alice can return a valid signature of the challenge c, because only she knows the secret key sk. Thus, Alice proves her identity, without showing her secret. No one can observe Alice's secret key, not even the verifier Bob.

Suppose that an eavesdropper Eve observed the exchanged messages. Later, she wants to impersonate Alice. Since Bob selects his challenge c at random (from a huge set), the probability that he uses the same challenge twice is very small. Therefore, Eve cannot gain any advantage by her observations.

The parties in a protocol can be friends or adversaries. Protocols can be attacked. The attacks may be directed against the underlying cryptographic algorithms or against the implementation of the algorithms and protocols. There may also be attacks against a protocol itself. There may be passive attacks performed by an eavesdropper, where the only purpose is to obtain information. An adversary may also try to gain an advantage by actively manipulating the protocol. She might pretend to be someone else, substitute messages or replay old messages.

Important protocols for key exchange, electronic elections, digital cash and interactive proofs of identity are discussed in Chapter 3.

Provable Security

It is desirable to design cryptosystems that are *provably secure*. Provably secure means that mathematical proofs show that the cryptosystem resists cer-

tain types of attacks. Pioneering work in this field was done by C.E. Shannon. In his information theory, he developed measures for the amount of information associated with a message and the notion of perfect secrecy. A *perfectly secret* cipher perfectly resists all ciphertext-only attacks. An adversary gets no information at all about the plaintext, even if his resources in computing power and time are unlimited. *Vernam's one-time pad* (see Section 1.1), which encrypts a message m by XORing it bitwise with a truly random bit string, is the most famous perfectly secret cipher. It even resists all the passive attacks mentioned. This can be mathematically proven by Shannon's theory. Classical information-theoretic security is discussed in Section 8.1; an introduction to Shannon's information theory may be found in Appendix B. Unfortunately, Vernam's one-time pad and all perfectly secret ciphers are usually impractical. It is not practical in most situations to generate and handle truly random bit sequences of sufficient length as required for perfect secrecy.

More recent approaches to provable security therefore abandon the ideal of perfect secrecy and the (unrealistic) assumption of unbounded computing power. The computational complexity of algorithms is taken into account. Only attacks that might be *feasible* in practice are considered. Feasible means that the attack can be performed by an *efficient algorithm*. Of course, here the question about the right notion of efficiency arises. Certainly, algorithms with non-polynomial running time are inefficient. Vice versa algorithms with polynomial running time are often considered as the efficient ones. In this book, we also adopt this notion of efficiency.

The way a cryptographic scheme is attacked might be influenced by random events. Adversary Eve might toss a coin to decide which case she tries next. Therefore, *probabilistic algorithms* are used to model attackers. Breaking an encryption system, for example by a ciphertext-only attack, means that a probabilistic algorithm with polynomial running time manages to derive information about the plaintext from the ciphertext, with some non-negligible probability. Probabilistic algorithms can toss coins, and their control flow may be at least partially directed by these random events. By using random sources, they can be implemented in practice. They must not be confused with non-deterministic algorithms. The notion of probabilistic (polynomial) algorithms and the underlying probabilistic model are discussed in Chapter 4.

The security of a public-key cryptosystem is based on the hardness of some computational problem (there is no efficient algorithm for solving the problem). For example, the secret keys of an RSA scheme could be easily figured out if computing the prime factors of a large integer were possible.[4]

[4] What "large" means depends on the available computing power. Today, a 1024-bit integer is considered as large.

However, it is believed that factoring large integers is infeasible.[5] There are no mathematical proofs for the hardness of the computational problems used in public-key systems. Therefore, security proofs for public-key methods are always conditional: they depend on the validity of the underlying assumption.

The assumption usually states that a certain function f is one way; i.e., f can be computed efficiently, but it is infeasible to compute x from $f(x)$. The assumptions, as well as the notion of a one-way function, can be made very precise by the use of probabilistic polynomial algorithms. The probability of successfully inverting the function by a probabilistic polynomial algorithm is negligibly small, and negligibly small means that it is asymptotically less than any given polynomial bound (see Chapter 5, Definition 5.12). Important examples, like the factoring, discrete logarithm and quadratic residuosity assumptions, are included in this book (see Chapter 5).

There are analogies to the classical notions of security. Shannon's perfect secrecy has a computational analogy: *ciphertext indistinguishability* (or *semantic security*). An encryption is perfectly secret if and only if an adversary cannot distinguish between two plaintexts, even if her computing resources are unlimited: if adversary Eve knows that a ciphertext c is the encryption of either m or m', she has no better chance than $1/2$ of choosing the right one. Ciphertext indistinguishability – also called *polynomial-time indistinguishability* – means that Eve's chance of successfully applying a probabilistic polynomial algorithm is at most negligibly greater than $1/2$ (Chapter 8, Definition 8.14).

As a typical result, it is proven in Section 8.4 that *public-key one-time pads* are ciphertext-indistinguishable. This means, for example, that the RSA public-key one-time pad is ciphertext-indistinguishable under the sole assumption that the RSA function is one way. A public-key one-time pad is similar to Vernam's one-time pad. The difference is that the message m is XORed with a pseudorandom bit sequence which is generated from a short truly random seed, by means of a one-way function.

Thus, one-way functions are not only the essential ingredients of public-key encryption and digital signatures. They also yield computationally perfect pseudorandom bit generators (Chapter 7). If f is a one-way function, it is not only impossible to compute x from $f(x)$, but certain bits (called hard-core bits) of x are equally difficult to deduce. This feature is called the bit security of a one-way function. For example, the least-significant bit is a hard-core bit for the RSA function $x \mapsto x^e \bmod n$. Starting with a truly random seed, repeatedly applying f and taking the hard-core bit in each step, you get a pseudorandom bit sequence. These bit sequences cannot be distinguished from truly random bit sequences by an efficient algorithm, or, equivalently (Yao's Theorem, Section 7.2), it is practically impossible to predict the next bit from the previous ones. So they are really computationally perfect.

[5] It is not known whether breaking RSA is easier than factoring the modulus. See Chapters 2 and 5 for a detailed discussion.

The bit security of important one-way functions is studied in detail in Chapter 6 including an in-depth analysis of the probabilities involved.

Randomness and the security of cryptographic schemes are closely related. There is no security without randomness. An encryption method provides secrecy only if the ciphertexts appear random to the adversary Eve. Vernam's one-time pad is perfectly secret, because, due to the truly random key string k, the encrypted message $m \oplus k$ [6] is a truly random bit sequence for Eve. The public-key one-time pad is ciphertext-indistinguishable, because if Eve applies an efficient probabilistic algorithm, she cannot distinguish the pseudo-random key string and, as a consequence, the ciphertext from a truly random sequence.

Public-key one-time pads are secure against passive eavesdroppers, who perform a ciphertext-only attack (see Section above for a classification of attacks). However, active adversaries, who perform adaptively-chosen-ciphertext attacks, can be a real danger in practice – as demonstrated by Bleichenbacher's 1-Million-Chosen-Ciphertext Attack (Section 2.3.3). Therefore, security against such attacks is also desirable. In Section 8.5, we study two examples of public-key encryption schemes which are secure against adaptively-chosen-ciphertext attacks, and their security proofs. One of the examples, Cramer-Shoup's public key encryption scheme, was the first practical scheme whose security proof is based solely on a standard number-theoretic assumption and a standard assumption of hash functions (collision-resistance).

The ideal cryptographic hash function is a *random function*. It yields hash values which cannot be distinguished from randomly selected and uniformly distributed values. Such a random function is also called a *random oracle*. Sometimes, the security of a cryptographic scheme can be proven in the *random oracle model*. In addition to the assumed hardness of a computational problem, such a proof relies on the assumption that the hash functions used in the scheme are truly random functions. Examples of such schemes include the public-key encryption schemes OAEP (Section 2.3.4) and SAEP (Section 8.5.1), the above mentioned signature scheme PSS and full-domain-hash RSA signatures (Section 2.4.5). We give the random-oracle proofs for SAEP and full-domain-hash signatures.

Truly random functions can not be implemented, nor even perfectly approximated in practice. Therefore, a proof in the random oracle model can never be a complete security proof. The hash functions used in practice are constructed to be good approximations to the ideal of random functions. However, there were surprising errors in the past (see Section 2.4).

We distinguished different types of attacks on an encryption scheme. In a similar way, the attacks on signature schemes can be classified and different levels of security can be defined. We introduce this classification in Chapter 9 and give examples of signature schemes whose security can be proven solely under standard assumptions (like the factoring or the strong RSA as-

[6] $\oplus$ denotes the bitwise XOR operator, see page 13.

sumption). No assumptions on the randomness of a hash function have to be made, in contrast, for example, to schemes like PSS. A typical security proof for the highest level of security is included. For the given signature scheme, we show that not a single signature can be forged, even if the attacker Eve is able to obtain valid signatures from the legitimate signer, for messages she has chosen adaptively.

The security proofs for public-key systems are always conditional and depend on (widely believed, but unproven) assumptions. On the other hand, Shannon's notion of perfect secrecy and, in particular, the perfect secrecy of Vernam's one-time pad are unconditional. Although perfect unconditional security is not reachable in most practical situations, there are promising attempts to design practical cryptosystems which provably come close to perfect information-theoretic security. The proofs are based on classical information-theoretic methods and do not depend on unproven assumptions. The security relies on the fact that communication channels are noisy or on the limited storage capacity of an adversary. Some results in this approach are reviewed in the chapter on provably secure encryption (Section 8.6).