Cryptography

Theory and Practice

Fourth Edition

Chapter 1

Introduction to Cryptography

In this chapter, we present a brief overview of the kinds of problems studied in cryptography and the techniques used to solve them. These problems and the cryptographic tools that are employed in their solution are discussed in more detail and rigor in the rest of this book. This introduction may serve to provide an informal, non-technical, non-mathematical summary of the topics to be addressed. As such, it can be considered to be optional reading.

1.1 Cryptosystems and Basic Cryptographic Tools

In this section, we discuss basic notions relating to encryption. This includes secret-key and public-key cryptography, block and stream ciphers, and hybrid cryptography.

1.1.1 Secret-key Cryptosystems

Cryptography has been used for thousands of years to help to provide confidential communications between mutually trusted parties. In its most basic form, two people, often denoted as ***Alice*** and ***Bob***, have agreed on a particular ***secret key***. At some later time, Alice may wish to send a secret message to Bob (or Bob might want to send a message to Alice). The key is used to transform the original message (which is usually termed the ***plaintext***) into a scrambled form that is unintelligible to anyone who does not possess the key. This process is called ***encryption*** and the scrambled message is called the ***ciphertext***. When Bob receives the ciphertext, he can use the key to transform the ciphertext back into the original plaintext; this is the ***decryption*** process. A ***cryptosystem*** constitutes a complete specification of the keys and how they are used to encrypt and decrypt information.

Various types of cryptosystems of increasing sophistication have been used for many purposes throughout history. Important applications have included sensitive communications between political leaders and/or royalty, military maneuvers, etc. However, with the development of the internet and applications such as electronic commerce, many new diverse applications have emerged. These include scenarios such as encryption of passwords, credit card numbers, email, documents, files, and digital media.

It should also be mentioned that cryptographic techniques are also widely used to protect stored data in addition to data that is transmitted from one party to another. For example, users may wish to encrypt data stored on laptops, on external hard disks, in the cloud, in databases, etc. Additionally, it might be useful to be able to perform computations on encrypted data (without first decrypting the data).

The development and deployment of a cryptosystem must address the issue of security. Traditionally, the threat that cryptography addressed was that of an eavesdropping adversary who might intercept the ciphertext and attempt to decrypt it. If the adversary happens to possess the key, then there is nothing that can be done. Thus the main security consideration involves an adversary who does not possess the key, who is still trying to decrypt the ciphertext. The techniques used by the adversary to attempt to "break" the cryptosystem are termed ***cryptanalysis***. The most obvious type of cryptanalysis is to try to guess the key. An attack wherein the adversary tries to decrypt the ciphertext with every possible key in turn is termed an ***exhaustive key search***. When the adversary tries the correct key, the plaintext will be found, but when any other key is used, the "decrypted" ciphertext will likely be random gibberish. So an obvious first step in designing a secure cryptosystem is to specify a very large number of possible keys, so many that the adversary will not be able to test them all in any reasonable amount of time.

The model of cryptography described above is usually called ***secret-key cryptography***. This indicates that there is one secret key, which is known to both Alice and Bob. That is, the key is a "secret" that is known to two parties. This key is employed both to encrypt plaintexts and to decrypt ciphertexts. The actual encryption and decryption functions are thus inverses of each other. Some basic secret-key cryptosystems are introduced and analyzed with respect to different security notions in Chapters 2 and 3.

The drawback of secret-key cryptography is that Alice and Bob must somehow be able to agree on the secret key ahead of time (before they want to send any messages to each other). This might be straightforward if Alice and Bob are in the same place when they choose their secret key. But what if Alice and Bob are far apart, say on different continents? One possible solution is for Alice and Bob to use a public-key cryptosystem.

1.1.2 Public-key Cryptosystems

The revolutionary idea of ***public-key cryptography*** was introduced in the 1970s by Diffie and Hellman. Their idea was that it might be possible to devise a cryptosystem in which there are two distinct keys. A ***public key*** would be used to encrypt the plaintext and a ***private key*** would enable the ciphertext to be decrypted. Note that a public key can be known to "everyone," whereas a private key is known to only one person (namely, the recipient of the encrypted message). So a public-key cryptosystem would enable anyone to encrypt a message to be transmitted to Bob, say, and only Bob could decrypt the message. The first and best-known example of a public-key cryptosystem is the *RSA Cyptosystem* that

was invented by Rivest, Shamir and Adleman. Various types of public-key cryptosystems are presented in Chapters 6, 7, and 9.

Public-key cryptography obviates the need for two parties to agree on a prior shared secret key. However, it is still necessary to devise a method to distribute public keys securely. But this is not necessarily a trivial goal to accomplish, the main issue being the correctness or authenticity of purported public keys. Certificates, which we will discuss a bit later, are one common method to deal with this problem.

1.1.3 Block and Stream Ciphers

Cryptosystems are usually categorized as ***block ciphers*** or ***stream ciphers***. In a block cipher, the plaintext is divided into fixed-sized chunks called ***blocks***. A block is specified to be a bitstring (i.e., a string of 0's and 1's) of some fixed length (e.g., 64 or 128 bits). A block cipher will encrypt (or decrypt) one block at a time. In contrast, a stream cipher first uses the key to construct a ***keystream***, which is a bitstring that has exactly the same length as the plaintext (the plaintext is a bitstring of arbitrary length). The encryption operation constructs the ciphertext as the exclusive-or of the plaintext and the keystream. Decryption is accomplished by computing the exclusive-or of the ciphertext and the keystream. Public-key cryptosystems are invariably block ciphers, while secret-key cryptosystems can be block ciphers or stream ciphers. Block ciphers are studied in detail in Chapter 4.

1.1.4 Hybrid Cryptography

One of the drawbacks of public-key cryptosystems is that they are much slower than secret-key cryptosystems. As a consequence, public-key cryptosystems are mainly used to encrypt small amounts of data, e.g., a credit card number. However, there is a nice way to combine secret- and public-key cryptography to achieve the benefits of both. This technique is called ***hybrid cryptography***. Suppose that Alice wants to encrypt a "long" message and send it to Bob. Assume that Alice and Bob do not have a prior shared secret key. Alice can choose a random secret key and encrypt the plaintext, using a (fast) secret-key cryptosystem. Alice then encrypts this secret key using Bob's public key. Alice sends the ciphertext and the encrypted key to Bob. Bob first uses his private decryption key to decrypt the secret key, and then he uses this secret key to decrypt the ciphertext.

Notice that the "slow" public-key cryptosystem is only used to encrypt a short secret key. The much faster secret-key cryptosystem is used to encrypt the longer plaintext. Thus, hybrid cryptography (almost) achieves the efficiency of secret-key cryptography, but it can be used in a situation where Alice and Bob do not have a previously determined secret key.

1.2 Message Integrity

This section discusses various tools that help to achieve integrity of data, including message authentication codes (MACs), signature schemes, and hash functions.

Cryptosystems provide ***secrecy*** (equivalently, ***confidentiality***) against an eavesdropping adversary, which is often called a ***passive adversary***. A passive adversary is assumed to be able to access whatever information is being sent from Alice to Bob; see Figure 1.1. However, there are many other threats that we might want to protect against, particularly when an ***active adversary*** is present. An active adversary is one who can alter information that is transmitted from Alice to Bob.

Figure 1.2 depicts some of the possible actions of an active adversary. An active adversary might

- alter the information that is sent from Alice to Bob,
- send information to Bob in such a way that Bob thinks the information originated from Alice, or
- divert information sent from Alice to Bob in such a way that a third party (Charlie) receives this information instead of Bob.

Possible objectives of an active adversary could include fooling Bob (say) into accepting "bogus" information, or misleading Bob as to who sent the information to him in the first place.

We should note that encryption, by itself, cannot protect against these kinds of active attacks. For example, a stream cipher is susceptible to a ***bit-flipping attack***. If some ciphertext bits are "flipped" (i.e., 0's are replaced by 1's and vice versa), then the effect is to flip the corresponding plaintext bits. Thus, an adversary can modify the plaintext in a predictable way, even though the adversary does not know what the plaintext bits are.

There are various types of "integrity" guarantees that we might seek to provide, in order to protect against the possible actions of an active adversary. Such an adversary might change the information that is being transmitted from Alice to Bob (and note that this information may or may not be encrypted). Alternatively, the adversary might try to "forge" a message and send it to Bob, hoping that he will think that it originated from Alice. Cryptographic tools that protect against these and related types of threats can be constructed in both the secret-key and public-key settings. In the secret-key setting, we will briefly discuss the notion of a ***message authentication code*** (or ***MAC***). In the public-key setting, the tool that serves a roughly similar purpose is a ***signature scheme***.

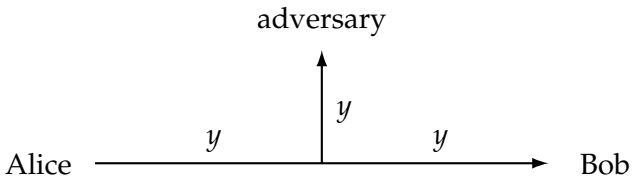

FIGURE 1.1: A passive adversary

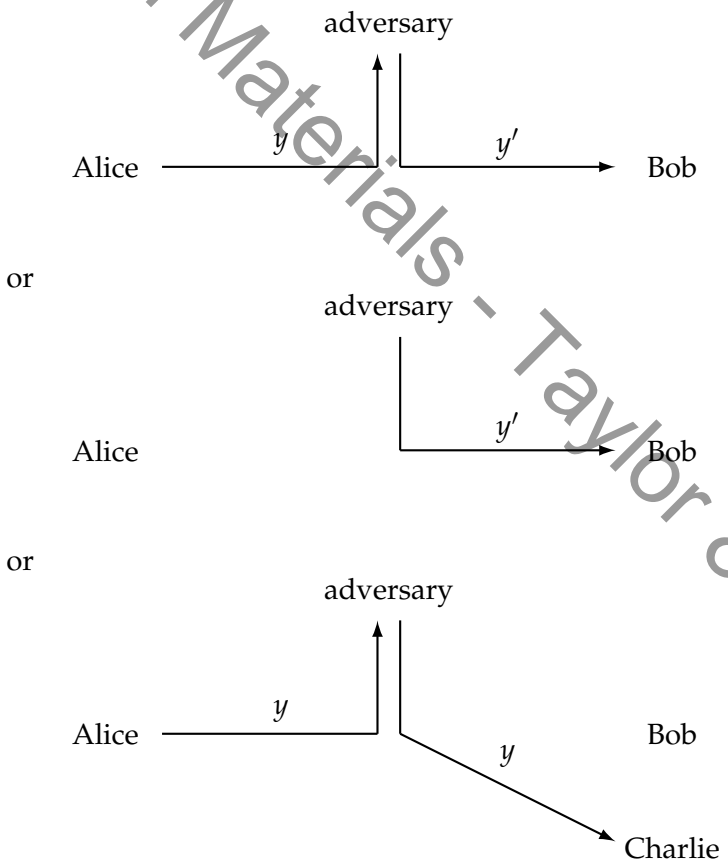

FIGURE 1.2: Active adversaries

1.2.1 Message Authentication Codes

A message authentication code requires Alice and Bob to share a secret key. When Alice wants to send a message to Bob, she uses the secret key to create a ***tag*** that she appends to the message (the tag depends on both the key and the message). When Bob receives the message and tag, he uses the key to re-compute the tag and checks to see if it is the same as the tag that he received. If so, Bob accepts the message as an authentic message from Alice; if not, then Bob rejects the message as being invalid. We note that the message may or may not be encrypted. MACs are discussed in Chapter 5.

If there is no need for confidentiality, then the message can be sent as plaintext. However, if confidentiality is desired, then the plaintext would be encrypted, and then the tag would be computed on the ciphertext. Bob would first verify the correctness of the tag. If the tag is correct, Bob would then decrypt the ciphertext. This process is often called ***encrypt-then-MAC*** (see Section 5.5.3 for a more detailed discussion of this topic).

For a MAC to be considered secure, it should be infeasible for the adversary to compute a correct tag for any message for which they have not already seen a valid tag. Suppose we assume that a secure MAC is being employed by Alice and Bob (and suppose that the adversary does not know the secret key that they are using). Then, if Bob receives a message and a valid tag, he can be confident that Alice created the tag on the given message (provided that Bob did not create it himself) and that neither the message nor the tag was altered by an adversary. A similar conclusion can be reached by Bob when he receives a message from Alice, along with a correct tag.

1.2.2 Signature Schemes

In the public-key setting, a signature scheme provides assurance similar to that provided by a MAC. In a signature scheme, the private key specifies a ***signing algorithm*** that Alice can use to sign messages. Similar to a MAC, the signing algorithm produces an output, which in this case is called a ***signature***, that depends on the message being signed as well as the key. The signature is then appended to the message. Notice that the signing algorithm is known only to Alice. On the other hand, there is a ***verification algorithm*** that is a public key (known to everyone). The verification algorithm takes as input a message and a signature, and outputs *true* or *false* to indicate whether the signature should be accepted as valid. One nice feature of a signature scheme is that anyone can verify Alice's signatures on messages, provided that they have an authentic copy of Alice's verification key. In contrast, in the MAC setting, only Bob can verify tags created by Alice (when Alice and Bob share a secret key). Signature schemes are studied in Chapter 8.

Security requirements for signature schemes are similar to MACs. It should be infeasible for an adversary to create a valid signature on any message not previously signed by Alice. Therefore, if Bob (or anyone else) receives a message and a valid tag (i.e., one that can be verified using Alice's public verification algorithm),

then the recipient can be confident that the signature was created by Alice and neither the message nor the signature was modified by an adversary.

One common application of signatures is to facilitate secure software updates. When a user purchases software from an online website, it typically includes a verification algorithm for a signature scheme. Later, when an updated version of the software is downloaded, it includes a signature (on the updated software). This signature can be verified using the verification algorithm that was downloaded when the original version of the software was purchased. This enables the user's computer to verify that the update comes from the same source as the original version of the software.

Signature schemes can be combined with public-key encryption schemes to provide confidentiality along with the integrity guarantees of a signature scheme. Assume that Alice wants to send a signed, encrypted (short) message to Bob. In this situation, the most commonly used technique is for Alice to first create a signature on the plaintext using her private signing algorithm, and then encrypt the plaintext and signature using Bob's public encryption key. When Bob receives the message, he first decrypts it, and then he checks the validity of the signature. This process is called ***sign-then-encrypt***; note that this is in some sense the reverse of the "encrypt-then-MAC" procedure that is used in the secret-key setting.

1.2.3 Nonrepudiation

There is one somewhat subtle difference between MACs and signature schemes. In a signature scheme, the verification algorithm is public. This means that the signature can be verified by anyone. So, if Bob receives a message from Alice containing her valid signature on the message, he can show the message and the signature to anyone else and be confident that the third party will also accept the signature as being valid. Consequently, Alice cannot sign a message and later try to claim that she did not sign the message, a property that is termed ***nonrepudiation***. This is useful in the setting of contracts, where we do not want someone to be able to renege on a signed contract by claiming (falsely) that their signature has been "forged," for example.

However, for a MAC, there is no third-party verifiability because the secret key is required to verify the correctness of the tag, and the key is known only to Alice and Bob. Even if the secret key is revealed to a third party (e.g., as a result of a court order), there is no way to determine if the tag was created by Alice or by Bob, because anything Bob can do, Alice can do as well, and vice versa. So a MAC does not provide nonrepudiation, and for this reason, a MAC is sometimes termed "deniable." It is interesting to note, however, that there are situations where deniability is desirable. This could be the case in real-time communications, where Alice and Bob want to be assured of the authenticity of their communications as they take place, but they do not want a permanent, verifiable record of this communication to exist. Such communication is analogous to an "off-the-record" conversation, e.g., between a journalist and an anonymous source. A MAC is useful in the con-

text of conversations of this type, especially if care is taken, after the conversation is over, to delete the secret keys that are used during the communication.

1.2.4 Certificates

We mentioned that verifying the authenticity of public keys, before they are used, is important. A certificate is a common tool to help achieve this objective. A ***certificate*** will contain information about a particular user or, more commonly, a website, including the website's public keys. These public keys will be signed by a trusted authority. It is assumed that everyone has possession of the trusted authority's public verification key, so anyone can verify the trusted authority's signature on a certificate. See Section 8.6 for more information about certificates.

This technique is used on the internet in *Transport Layer Security* (which is commonly called *TLS*). When a user connects to a secure website, say one belonging to a business engaged in electronic commerce, the website of the company will send a certificate to the user so the user can verify the authenticity of the website's public keys. These public keys will subsequently be used to set up a secure channel, between the user and the website, in which all information is encrypted. Note that the public key of the trusted authority, which is used to verify the public key of the website, is typically hard-coded into the web browser.

1.2.5 Hash Functions

Signature schemes tend to be much less efficient than MACs. So it is not advisable to use a signature scheme to sign "long" messages. (Actually, most signature schemes are designed to only sign messages of a short, fixed length.) In practice, messages are "hashed" before they are signed. A ***cryptographic hash function*** is used to compress a message of arbitrary length to a short, random-looking, fixed-length ***message digest***. Note that a hash function is a public function that is assumed to be known to everyone. Further, a hash function has no key. Hash functions are discussed in Chapter 5.

After Alice hashes the message, she signs the message digest, using her private signing algorithm. The original message, along with the signature on the message, is then transmitted to Bob, say. This process is called ***hash-then-sign***. To verify the signature, Bob will compute the message digest by hashing the message. Then he will use the public verification algorithm to check the validity of the signature on the message digest. When a signature is used along with public-key encryption, the process would actually be ***hash-then-sign-then-encrypt***. That is, the message is hashed, the message digest is then signed, and finally, the message and signature are encrypted.

A cryptographic hash function is very different from a hash function that is used to construct a hash table, for instance. In the context of hash tables, a hash function is generally required only to yield collisions[1] with a sufficiently small probability. On the other hand, if a cryptographic hash function is used, it should

[1] A ***collision*** for a function h occurs when $h(x) = h(y)$ for some $x \neq y$.

be computationally infeasible to find collisions, even though they must exist. Cryptographic hash functions are usually required to satisfy additional security properties, as discussed in Section 5.2.

Cryptographic hash functions also have other uses, such as for ***key derivation***. When used for key derivation, a hash function would be applied to a long random string in order to create a short random key.

Finally, it should be emphasized that hash functions cannot be used for encryption, for two fundamental reasons. First is the fact that hash functions do not have a key. The second is that hash functions cannot be inverted (they are not injective functions) so a message digest cannot be "decrypted" to yield a unique plaintext value.

1.3 Cryptographic Protocols

Cryptographic tools such as cryptosystems, signature schemes, hash functions, etc., can be used on their own to achieve specific security objectives. However, these tools are also used as components in more complicated protocols. (Of course, protocols can also be designed "from scratch," without making use of prior primitives.)

In general, a ***protocol*** (or ***interactive protocol***) refers to a specified sequence of messages exchanged between two (or possibly more) parties. A ***session*** of a protocol between Alice and Bob, say, will consist of one or more ***flows***, where each flow consists of a message sent from Alice to Bob or vice versa. At the end of the session, the parties involved may have established some common shared information, or confirmed possession of some previously shared information.

One important type protocol is an ***identification scheme***, in which one party "proves" their identity to another by demonstrating possession of a password, for example. More sophisticated identification protocols will instead consist of two (or more) flows, for example a challenge followed by a response, where the response is computed from the challenge using a certain secret or private key. Identification schemes are the topic of Chapter 10.

There are many kinds of protocols associated with various aspects of choosing keys or communicating keys from one party to another. In a ***key distribution scheme***, keys might be chosen by a trusted authority and communicated to one or more members of a certain network. Another approach, which does not require the participation of an active trusted authority, is called ***key agreement***. In a key agreement scheme, Alice and Bob (say) are able to end up with a common shared secret key, which should not become known to an adversary. These and related topics are discussed in Chapters 11 and 12.

A ***secret sharing scheme*** involves a trusted authority distributing "pieces" of information (called "shares") in such a way that certain subsets of shares can be suitably combined to reconstruct a certain predefined secret. One common type

of secret sharing scheme is a ***threshold scheme***. In a (k, n)-threshold scheme, there are n shares, and any k shares permit the reconstruction of the secret. On the other hand, $k-1$ or fewer shares provide no information about the value of the secret. Secret sharing schemes are studied in Chapter 11.

1.4 Security

A fundamental goal for a cryptosystem, signature scheme, etc., is for it to be "secure." But what does it mean to be secure and how can we gain confidence that something is indeed secure? Roughly speaking, we would want to say that an adversary cannot succeed in "breaking" a cryptosystem, for example, but we have to make this notion precise. Security in cryptography involves consideration of three different aspects: an ***attack model***, an ***adversarial goal***, and a ***security level***. We will discuss each of these in turn.

The attack model specifies the information that is available to the adversary. We will always assume that the adversary knows the scheme or protocol being used (this is called ***Kerckhoffs' Principle***). The adversary is also assumed to know the public key (if the system is a public-key system). On the other hand, the adversary is assumed not to know any secret or private keys being used. Possible additional information provided to the adversary should be specified in the attack model.

The adversarial goal specifies exactly what it means to "break" the cryptosystem. What is the adversary attempting to do and what information are they trying to determine? Thus, the adversarial goal defines a "successful attack."

The security level attempts to quantify the effort required to break the cryptosystem. Equivalently, what computational resources does the adversary have access to and how much time would it take to carry out an attack using those resources?

A statement of security for a cryptographic scheme will assert that a particular adversarial goal cannot be achieved in a specified attack model, given specified computational resources.

We now illustrate some of the above concepts in relation to a cryptosystem. There are four commonly considered attack models. In a ***known ciphertext attack***, the adversary has access to some amount of ciphertext that is all encrypted with the same unknown key. In a ***known plaintext attack***, the adversary gains access to some plaintext as well as the corresponding ciphertext (all of which is encrypted with the same key). In a ***chosen plaintext attack***, the adversary is allowed to choose plaintext, and then they are given the corresponding ciphertext. Finally, in a ***chosen ciphertext attack***, the adversary chooses some ciphertext and they are then given the corresponding plaintext.

Clearly a chosen plaintext or chosen ciphertext attack provides the adversary with more information than a known ciphertext attack. So they would be con-

sidered to be stronger attack models than a known ciphertext attack, since they potentially make the adversary's job easier.

The next aspect to study is the adversarial goal. In a ***complete break*** of a cryptosystem, the adversary determines the private (or secret) key. However, there are other, weaker goals that the adversary could potentially achieve, even if a complete break is not possible. For example, the adversary might be able to decrypt a previously unseen ciphertext with some specified non-zero probability, even though they have not been able to determine the key. Or, the adversary might be able to determine some partial information about the plaintext, given a previously unseen ciphertext, with some specified non-zero probability. "Partial information" could include the values of certain plaintext bits. Finally, as an example of a weak goal, the adversary might be able distinguish between encryptions of two given plaintexts.[2]

Other cryptographic primitives will have different attack models and adversarial goals. In a signature scheme, the attack model would specify what kind of (valid) signatures the adversary has access to. Perhaps the adversary just sees some previously signed messages, or maybe the adversary can request the signer to sign some specific messages of the adversary's choosing. The adversarial goal is typically to sign some "new" message (i.e., one for which the adversary does not already know a valid signature). Perhaps the adversary can find a valid signature for some specific message that the adversary chooses, or perhaps they can find a valid signature for any message. These would represent weak and strong adversarial goals, respectively.

Three levels of security are often studied, which are known as ***computational security***, ***provable security***, and ***unconditional security***.

Computational security means that a specific algorithm to break the system is computationally infeasible, i.e., it cannot be accomplished in a reasonable amount of time using currently available computational resources. Of course, a system that is computationally secure today may not be computationally secure indefinitely. For example, new algorithms might be discovered, computers may get faster, or fundamental new computing paradigms such as quantum computing might become practical. Quantum computing, if it becomes practical, could have an enormous impact on the security of many kinds of public-key cryptography; this is addressed in more detail in Section 9.1.

It is in fact very difficult to predict how long something that is considered secure today will remain secure. There are many examples where many cryptographic schemes have not survived as long as originally expected due to the reasons mentioned above. This has led to rather frequent occurrences of replacing standards with improved standards. For example, in the case of hash functions, there have been a succession of proposed and/or approved standards, denoted as *SHA-0*, *SHA-1*, *SHA-2* and *SHA-3*, as new attacks have been found and old standards have become insecure.

[2]Whether or not this kind of limited information can be exploited by the adversary in a malicious way is another question, of course.

An interesting example relating to broken predictions is provided by the public-key *RSA Cryptosystem*. In the August 1977 issue of *Scientific American*, the eminent mathematical expositor Martin Gardner wrote a column on the newly developed *RSA* public-key cryptosystem entitled "A new kind of cipher that would take millions of years to break." Included in the article was a challenge ciphertext, encrypted using a 512-bit key. However, the challenge was solved 17 years later, on April 26, 1994, by factoring the given public key (the plaintext was "the magic words are squeamish ossifrage"). The statement that the cipher would take millions of years to break probably referred to how long it would take to run the best factoring algorithm known in 1977 on the fastest computer available in 1977. However, between 1977 and 1994, there were several developments, including the following:

- computers became much faster,
- improved factoring algorithms were found, and
- the development of the internet facilitated large-scale distributed computations.

Of course, it is basically impossible to predict when new algorithms will be discovered. Also, the third item listed above can be regarded as a "paradigm shift" that was probably not on anyone's radar in 1977.

The next "level" of security we address is provable security (also known as ***reductionist security***), which refers to a situation where breaking the cryptosystem (i.e., achieving the adversarial goal) can be reduced in a complexity-theoretic sense to solving some underlying (assumed difficult) mathematical problem. This would show that breaking the cryptosystem is at least as difficult as solving the given hard problem. Provable security often involves reductions to the factoring problem or the discrete logarithm problem (these problems are studied in Sections 6.6 and 7.2, respectively).

Finally, unconditional security means that the cryptosystem cannot be broken (i.e., the adversarial goal is not achievable), even with unlimited computational resources, because there is not enough information available to the adversary (as specified in the attack model) for them to be able to do this. The most famous example of an unconditionally secure cryptosystem is the *One-time Pad*. In this cryptosystem, the key is a random bitstring having the same length as the plaintext. The ciphertext is formed as the exclusive-or of the plaintext and the key. For the *One-time Pad*, it can be proven mathematically that the adversary can obtain no partial information whatsoever about the plaintext (other than its length), given the ciphertext, provided the key is used to encrypt only one string of plaintext and the key has the same length as the plaintext. The *One-time Pad* is discussed in Chapter 3.

When we analyze a cryptographic scheme, our goal would be to show that the adversary cannot achieve a *weak* adversarial goal in a *strong* attack model, given *significant* computational resources.

The preceding discussion of security has dealt mostly with the situation of a cryptographic primitive such as a cryptosystem. However, cryptographic primitives are generally combined in complicated ways when protocols are defined and ultimately implemented. Even seemingly simple implementation decisions can lead to unexpected vulnerabilities. For example, when data is encrypted using a block cipher, it first needs to be split into fixed length chunks, e.g., 128-bit blocks. If the data does not exactly fill up an integral number of blocks, then some padding has to be introduced. It turns out that a standard padding technique, when used with the common CBC mode of operation, is susceptible to an attack known as a ***padding oracle attack***, which was discovered by Vaudenay in 2002 (see Section 4.7.1 for a description of this attack).

There are also various kinds of attacks against physical implementations of cryptography that are known as ***side channel attacks***. Examples of these include ***timing attacks***, ***fault analysis attacks***, ***power analysis attacks***, and ***cache attacks***. The idea is that information about a secret or private key might be leaked by observing or physically manipulating a device (such as a smart card) on which a particular cryptographic scheme is implemented. One example would be observing the time taken by the device to perform certain computations (a so-called "timing attack"). This leakage of information can take place even though the scheme is "secure."

1.5 Notes and References

There are many monographs and textbooks on the subject of cryptography. We will mention here a few general treatments that may be useful to readers.

For an accessible, non-mathematical treatment, we recommend

- *Everyday Cryptography: Fundamental Principles and Applications, Second Edition* by Keith Martin [127].

For a more mathematical point of view, the following recent texts are helpful:

- *An Introduction to Mathematical Cryptography* by J. Hoffstein, J. Pipher, and J. Silverman [96]
- *Introduction to Modern Cryptography, Second Edition* by J. Katz and Y. Lindell [104]
- *Understanding Cryptography: A Textbook for Students and Practitioners* by C. Paar and J. Pelzl [157]
- *Cryptography Made Simple* by Nigel Smart [185]
- *A Classical Introduction to Cryptography: Applications for Communications Security* by Serge Vaudenay [196].

For mathematical background, especially for public-key cryptography, we recommend

- *Mathematics of Public Key Cryptography* by Stephen Galbraith [84].

Finally, the following is a valuable reference, even though it is quite out of date:

- *Handbook of Applied Cryptography* by A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone [134].