Chapter 1

Introduction

1.1 Cryptography and Modern Cryptography

The Concise Oxford Dictionary (2006) defines cryptography as *the art of writing or solving codes*. This definition may be historically accurate, but it does not capture the essence of modern cryptography. First, it focuses solely on the problem of secret communication. This is evidenced by the fact that the definition specifies "codes", elsewhere defined as "a system of pre-arranged signals, especially used to ensure secrecy in transmitting messages". Second, the definition refers to cryptography as an art form. Indeed, until the 20th century (and arguably until late in that century), cryptography was an art. Constructing good codes, or breaking existing ones, relied on creativity and personal skill. There was very little theory that could be relied upon and there was not even a well-defined notion of what constitutes a good code.

In the late 20th century, this picture of cryptography radically changed. A rich theory emerged, enabling the rigorous study of cryptography as a *science*. Furthermore, the field of cryptography now encompasses much more than secret communication. For example, it deals with the problems of message authentication, digital signatures, protocols for exchanging secret keys, authentication protocols, electronic auctions and elections, digital cash and more. In fact, modern cryptography can be said to be concerned with problems that may arise in *any* distributed computation that may come under internal or external attack. Without attempting to provide a perfect definition of modern cryptography, we would say that it is *the scientific study of techniques for securing digital information, transactions, and distributed computations.*

Another very important difference between classical cryptography (say, before the 1980s) and modern cryptography relates to who uses it. Historically, the major consumers of cryptography were military and intelligence organizations. Today, however, cryptography is everywhere! Security mechanisms that rely on cryptography are an integral part of almost any computer system. Users (often unknowingly) rely on cryptography every time they access a secured website. Cryptographic methods are used to enforce access control in multi-user operating systems, and to prevent thieves from extracting trade secrets from stolen laptops. Software protection methods employ encryption, authentication, and other tools to prevent copying. The list goes on and on.

In short, cryptography has gone from an art form that dealt with secret communication for the military to a science that helps to secure systems for ordinary people all across the globe. This also means that cryptography is becoming a more and more central topic within computer science.

The focus of this book is *modern* cryptography. Yet we will begin our study by examining the state of cryptography before the changes mentioned above. Besides allowing us to ease into the material, it will also provide an understanding of where cryptography has come from so that we can later appreciate how much it has changed. The study of "classical cryptography" — replete with ad-hoc constructions of codes, and relatively simple ways to break them — serves as good motivation for the more rigorous approach that we will be taking in the rest of the book.[1]

1.2 The Setting of Private-Key Encryption

As noted above, cryptography was historically concerned with secret communication. Specifically, cryptography was concerned with the construction of *ciphers* (now called *encryption schemes*) for providing secret communication between two parties sharing some information in advance. The setting in which the communicating parties share some secret information in advance is now known as the *private-key* (or the *symmetric-key*) setting. Before describing some historical ciphers, we discuss the private-key setting and encryption in more general terms.

In the private-key setting, two parties share some secret information called a *key*, and use this key when they wish to communicate secretly with each other. A party sending a message uses the key to *encrypt* (or "scramble") the message before it is sent, and the receiver uses the same key to *decrypt* (or "unscramble") and recover the message upon receipt. The message itself is called the *plaintext*, and the "scrambled" information that is actually transmitted from the sender to the receiver is called the *ciphertext*; see Figure 1.1. The shared key serves to distinguish the communicating parties from any other parties who may be eavesdropping on their communication (assumed to take place over a public channel).

In this setting, the same key is used to convert the plaintext into a ciphertext and back. This explains why this setting is also known as the *symmetric-key* setting, where the symmetry lies in the fact that both parties hold the same key which is used for both encryption and decryption. This is in contrast to

[1] This is our primary intent in presenting this material and, as such, this chapter should not be taken as a representative historical account. The reader interested in the history of cryptography should consult the references at the end of this chapter.

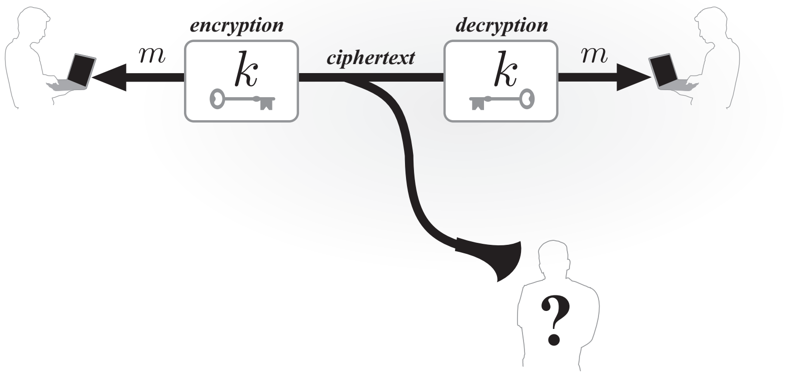

FIGURE 1.1: The basic setting of private-key encryption.

the setting of *asymmetric* encryption (introduced in Chapter 9), where the sender and receiver do not share any secrets and different keys are used for encryption and decryption. The private-key setting is the classic one, as we will see later in this chapter.

An implicit assumption in any system using private-key encryption is that the communicating parties have some way of initially sharing a key in a secret manner. (Note that if one party simply sends the key to the other over the public channel, an eavesdropper obtains the key too!) In military settings, this is not a severe problem because communicating parties are able to physically meet in a secure location in order to agree upon a key. In many modern settings, however, parties cannot arrange any such physical meeting. As we will see in Chapter 9, this is a source of great concern and actually limits the applicability of cryptographic systems that rely solely on private-key methods. Despite this, there are still many settings where private-key methods suffice and are in wide use; one example is disk encryption, where the *same* user (at different points in time) uses a fixed secret key to both write to and read from the disk. As we will explore further in Chapter 10, private-key encryption is also widely used in conjunction with asymmetric methods.

The syntax of encryption. A private-key encryption scheme is comprised of three algorithms: the first is a procedure for generating keys, the second a procedure for encrypting, and the third a procedure for decrypting. These have the following functionality:

1. The *key-generation algorithm* Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution that is determined by the scheme.

2. The *encryption algorithm* Enc takes as input a key k and a plaintext message m and outputs a ciphertext c. We denote by $\mathsf{Enc}_k(m)$ the encryption of the plaintext m using the key k.

3. The *decryption algorithm* Dec takes as input a key k and a ciphertext c and outputs a plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}_k(c)$.

The set of all possible keys output by the key-generation algorithm is called the *key space* and is denoted by $\mathcal{K}$. Almost always, Gen simply chooses a key uniformly at random from the key space (in fact, one can assume without loss of generality that this is the case). The set of all "legal" messages (i.e., those supported by the encryption algorithm) is denoted $\mathcal{M}$ and is called the *plaintext* (or *message*) *space*. Since any ciphertext is obtained by encrypting some plaintext under some key, the sets $\mathcal{K}$ and $\mathcal{M}$ together define a set of all possible ciphertexts denoted by $\mathcal{C}$. An encryption scheme is fully defined by specifying the three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and the plaintext space $\mathcal{M}$.

The basic correctness requirement of any encryption scheme is that for every key k output by Gen and every plaintext message $m \in \mathcal{M}$, it holds that

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m.$$

In words, decrypting a ciphertext (using the appropriate key) yields the original message that was encrypted.

Recapping our earlier discussion, an encryption scheme would be used by two parties who wish to communicate as follows. First, Gen is run to obtain a key k that the parties share. When one party wants to send a plaintext m to the other, he computes $c := \mathsf{Enc}_k(m)$ and sends the resulting ciphertext c over the public channel to the other party.[2] Upon receiving c, the other party computes $m := \mathsf{Dec}_k(c)$ to recover the original plaintext.

Keys and Kerckhoffs' principle. As is clear from the above formulation, if an eavesdropping adversary knows the algorithm Dec as well as the key k shared by the two communicating parties, then that adversary will be able to decrypt all communication between these parties. It is for this reason that the communicating parties must share the key k secretly, and keep k completely secret from everyone else. But maybe they should keep the decryption algorithm Dec a secret, too? For that matter, perhaps all the algorithms constituting the encryption scheme (i.e., Gen and Enc as well) should be kept secret? (Note that the plaintext space $\mathcal{M}$ is typically assumed to be known, e.g., it may consist of English-language sentences.)

In the late 19th century, Auguste Kerckhoffs gave his opinion on this matter in a paper he published outlining important design principles for military

[2] Throughout the book, we use ":=" to denote the assignment operation. A list of common notation can be found in the back of the book.

ciphers. One of the most important of these principles (now known simply as Kerckhoffs' principle) is the following:

> *The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.*

In other words, the encryption scheme itself should *not* be kept secret, and so only the key should constitute the secret information shared by the communicating parties.

Kerckhoffs' intention was that an encryption scheme should be designed so as to be secure *even if* an adversary knows the details of all the component algorithms of the scheme, as long as the adversary doesn't know the key being used. Stated differently, Kerckhoffs' principle demands that *security rely solely on the secrecy of the key.* But why?

There are three primary arguments in favor of Kerckhoffs' principle. The first is that it is much easier for the parties to maintain secrecy of a short key than to maintain secrecy of an algorithm. It is easier to share a short (say, 100-bit) string and store this string securely than it is to share and securely store a program that is thousands of times larger. Furthermore, details of an algorithm can be leaked (perhaps by an insider) or learned through reverse engineering; this is unlikely when the secret information takes the form of a randomly-generated string.

A second argument in favor of Kerckhoffs' principle is that in case the key *is* exposed, it will be much easier for the honest parties to change the key than to replace the algorithm being used. Actually, it is good security practice to refresh a key frequently even when it has not been exposed, and it would be much more cumbersome to replace the software being used instead.

Finally, in case many pairs of people (say, within a company) need to encrypt their communication, it will be significantly easier for all parties to use the same algorithm/program, but different keys, than for everyone to use a different program (which would furthermore depend on the party with whom they are communicating).

Today, Kerckhoffs' principle is understood as not only advocating that security should not rely on secrecy of the algorithms being used, but also demanding that these algorithms be made *public.* This stands in stark contrast to the notion of "security by obscurity" which is the idea that improved security can be achieved by keeping a cryptographic algorithm hidden. Some of the advantages of "open cryptographic design", where algorithm specifications are made public, include the following:

1. Published designs undergo public scrutiny and are therefore likely to be stronger. Many years of experience have demonstrated that it is very difficult to construct good cryptographic schemes. Therefore, our confidence in the security of a scheme is much higher if it has been extensively studied (by experts other than the designers of the scheme themselves) and no weaknesses have been found.

2. It is better for security flaws, if they exist, to be revealed by "ethical hackers" (leading, hopefully, to the system being fixed) rather than having these flaws be known only to malicious parties.

3. If the security of the system relies on the secrecy of the algorithm, then reverse engineering of the code (or leakage by industrial espionage) poses a serious threat to security. This is in contrast to the secret key which is not part of the code, and so is not vulnerable to reverse engineering.

4. Public design enables the establishment of standards.

As simple and obvious as it may sound, the principle of open cryptographic design (i.e., Kerckhoffs' principle) is ignored over and over again with disastrous results. It is very dangerous to use a proprietary algorithm (i.e., a non-standardized algorithm that was designed in secret by some company), and only publicly tried and tested algorithms should be used. Fortunately, there are enough good algorithms that are standardized and not patented, so that there is no reason whatsoever today to use something else.

Attack scenarios. We wrap up our general discussion of encryption with a brief discussion of some basic types of attacks against encryption schemes. In order of severity, these are:

- Ciphertext-only attack: This is the most basic type of attack and refers to the scenario where the adversary just observes a ciphertext (or multiple ciphertexts) and attempts to determine the underlying plaintext (or plaintexts).

- Known-plaintext attack: Here, the adversary learns one or more pairs of plaintexts/ciphertexts encrypted under the same key. The aim of the adversary is then to determine the plaintext that was encrypted in some *other* ciphertext (for which it does not know the corresponding plaintext).

- Chosen-plaintext attack: In this attack, the adversary has the ability to obtain the encryption of plaintexts of its choice. It then attempts to determine the plaintext that was encrypted in some other ciphertext.

- Chosen-ciphertext attack: The final type of attack is one where the adversary is even given the capability to obtain the decryption of ciphertexts of its choice. The adversary's aim, once again, is to determine the plaintext that was encrypted in some other ciphertext (whose decryption the adversary is unable to obtain directly).

The first two types of attacks are *passive* in that the adversary just receives some ciphertexts (and possibly some corresponding plaintexts as well) and then launches its attack. In contrast, the last two types of attacks are *active* in that the adversary can adaptively ask for encryptions and/or decryptions of its choice.

The first two attacks described above are clearly realistic. A ciphertext-only attack is the easiest to carry out in practice; the only thing the adversary needs is to eavesdrop on the public communication line over which encrypted messages are sent. In a known-plaintext attack it is assumed that the adversary somehow also obtains the plaintext messages corresponding to the ciphertexts that it viewed. This is often realistic because not all encrypted messages are confidential, at least not indefinitely. As a trivial example, two parties may always encrypt a "hello" message whenever they begin communicating. As a more complex example, encryption may be used to keep quarterly earnings results secret until their release date. In this case, anyone eavesdropping and obtaining the ciphertext will later obtain the corresponding plaintext. Any reasonable encryption scheme must therefore remain secure against an adversary that can launch a known-plaintext attack.

The two latter active attacks may seem somewhat strange and require justification. (When do parties encrypt and decrypt whatever an adversary wishes?) We defer a more detailed discussion of these attacks to the place in the text where security against these attacks is formally defined: Section 3.5 for chosen-plaintext attacks and Section 3.7 for chosen-ciphertext attacks.

Different applications of encryption may require the encryption scheme to be resilient to different types of attacks. It is not always the case that an encryption scheme secure against the "strongest" type of attack should be used, since it may be less efficient than an encryption scheme secure against "weaker" attacks. Therefore, the latter may be preferred if it suffices for the application at hand.

1.3 Historical Ciphers and Their Cryptanalysis

In our study of "classical cryptography" we will examine some historical ciphers and show that they are completely insecure. As stated earlier, our main aims in presenting this material are (1) to highlight the weaknesses of an "ad-hoc" approach to cryptography, and thus motivate the modern, rigorous approach that will be discussed in the following section, and (2) to demonstrate that "simple approaches" to achieving secure encryption are unlikely to succeed, and show why this is the case. Along the way, we will present some central principles of cryptography which can be learned from the weaknesses of these historical schemes.

In this section (and this section only), plaintext characters are written in `lower case` and ciphertext characters are written in `UPPER CASE`. When describing attacks on schemes, we always apply Kerckhoffs' principle and assume that the scheme is known to the adversary (but the key being used is not).

Caesar's cipher. One of the oldest recorded ciphers, known as Caesar's cipher, is described in "De Vita Caesarum, Divus Iulius" ("The Lives of the Caesars, The Deified Julius"), written in approximately 110 C.E.:

> *There are also letters of his to Cicero, as well as to his intimates on private affairs, and in the latter, if he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.*

That is, Julius Caesar encrypted by rotating the letters of the alphabet by 3 places: `a` was replaced with `D`, `b` with `E`, and so on. Of course, at the end of the alphabet, the letters wrap around and so `x` was replaced with `A`, `y` with `B`, and `z` with `C`. For example, the short message `begin the attack now`, with spaces removed, would be encrypted as:

```
EHJLQWKHDWWDFNQRZ
```

making it unintelligible.

An immediate problem with this cipher is that the method is *fixed*. Thus, anyone learning how Caesar encrypted his messages would be able to decrypt effortlessly. This can be seen also if one tries to fit Caesar's cipher into the syntax of encryption described earlier: the key-generation algorithm Gen is trivial (that is, it does nothing) and there is no secret key to speak of.

Interestingly, a variant of this cipher called ROT-13 (where the shift is 13 places instead of 3) is widely used nowadays in various online forums. It is understood that this does not provide any cryptographic security, and ROT-13 is used merely to ensure that the text (say, a movie spoiler) is unintelligible unless the reader of a message consciously chooses to decrypt it.

The shift cipher and the sufficient key space principle. Caesar's cipher suffers from the fact that encryption is always done in the same way, and there is no secret key. The shift cipher is similar to Caesar's cipher, but a secret key is introduced.[3] Specifically, in the shift cipher the key k is a number between 0 and 25. Then, to encrypt, letters are rotated by k places as in Caesar's cipher. Mapping this to the syntax of encryption described earlier, this means that algorithm Gen outputs a random number k in the set $\{0, \ldots, 25\}$; algorithm Enc takes a key k and a plaintext written using English letters and shifts each letter of the plaintext forward k positions (wrapping around from `z` to `a`); and algorithm Dec takes a key k and a ciphertext written using English letters and shifts every letter of the ciphertext *backward* k positions (this time wrapping around from `a` to `z`). The plaintext message space $\mathcal{M}$ is defined to be

[3] In some books, "Caesar's cipher" and "shift cipher" are used interchangeably.

all finite strings of characters from the English alphabet (note that numbers, punctuation, or other characters are not allowed in this scheme).

A more mathematical description of this method can be obtained by viewing the alphabet as the numbers $0, \ldots, 25$ (rather than as English characters). First, some notation: if a is an integer and N is an integer greater than 1, we define $[a \bmod N]$ as the remainder of a upon division by N. Note that $[a \bmod N]$ is an integer between 0 and $N-1$, inclusive. We refer to the process mapping a to $[a \bmod N]$ as *reduction modulo* N; we will have much more to say about reduction modulo N beginning in Chapter 7.

Using this notation, encryption of a plaintext character m_i with the key k gives the ciphertext character $[(m_i+k) \bmod 26]$, and decryption of a ciphertext character c_i is defined by $[(c_i-k) \bmod 26]$. In this view, the message space $\mathcal{M}$ is defined to be any finite sequence of integers that lie in the range $\{0, \ldots, 25\}$.

Is the shift cipher secure? Before reading on, try to decrypt the following message that was encrypted using the shift cipher and a secret key k (whose value we will not reveal):

```
OVDTHUFWVZZPISLRLFZHYLAOLYL.
```

Is it possible to decrypt this message without knowing k? Actually, it is completely trivial! The reason is that there are only 26 possible keys. Thus, it is easy to try every key, and see which key decrypts the ciphertext into a plaintext that "makes sense". Such an attack on an encryption scheme is called a *brute-force attack* or *exhaustive search.* Clearly, any secure encryption scheme must not be vulnerable to such a brute-force attack; otherwise, it can be completely broken, irrespective of how sophisticated the encryption algorithm is. This brings us to a trivial, yet important, principle called the "sufficient key space principle":

> *Any secure encryption scheme must have a key space that is not vulnerable to exhaustive search.*[4]

In today's age, an exhaustive search may use very powerful computers, or many thousands of PC's that are distributed around the world. Thus, the number of possible keys must be very large (at least 2^{60} or 2^{70}).

We emphasize that the above principle gives a *necessary* condition for security, not a *sufficient* one. We will see next an encryption scheme that has a very large key space but which is still insecure.

Mono-alphabetic substitution. The shift cipher maps each plaintext character to a different ciphertext character, but the mapping in each case is given by the same shift (the value of which is determined by the key). The idea

[4] This is actually only true if the message space is larger than the key space (see Chapter 2 for an example where security is achieved using a small key space as long as the message space is even smaller). In practice, when very long messages are typically encrypted with the same key, the key space must not be vulnerable to exhaustive search.

behind mono-alphabetic substitution is to map each plaintext character to a different ciphertext character in an *arbitrary* manner, subject only to the fact that the mapping must be one-to-one in order to enable decryption. The key space thus consists of all *permutations* of the alphabet, meaning that the size of the key space is $26! = 26 \cdot 25 \cdot 24 \cdots 2 \cdot 1$ (or approximately 2^{88}) if we are working with the English alphabet. As an example, the key

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	E	U	A	D	N	B	K	V	M	R	O	C	Q	F	S	Y	H	W	G	L	Z	I	J	P	T

in which `a` maps to `X`, etc., would encrypt the message `tellhimaboutme` to `GDOOKVCXEFLGCD`. A brute force attack on the key space for this cipher takes much longer than a lifetime, even using the most powerful computer known today. However, this does not necessarily mean that the cipher is secure. In fact, as we will show now, it is easy to break this scheme even though it has a very large key space.

Assume that English-language text is being encrypted (i.e., the text is grammatically-correct English writing, not just text written using characters of the English alphabet). It is then possible to attack the mono-alphabetic substitution cipher by utilizing statistical patterns of the English language (of course, the same attack works for any language). The two properties of this cipher that are utilized in the attack are as follows:

1. In this cipher, the mapping of each letter is fixed, and so if `e` is mapped to `D`, then every appearance of `e` in the plaintext will result in the appearance of `D` in the ciphertext.

2. The probability distribution of individual letters in the English language (or any other) is known. That is, the average frequency counts of the different English letters are quite invariant over different texts. Of course, the longer the text, the closer the frequency counts will be to the average. However, even relatively short texts (consisting of only tens of words) have distributions that are "close enough" to the average.

The attack works by tabulating the probability distribution of the ciphertext and then comparing it to the known probability distribution of letters in English text (see Figure 1.2). The probability distribution being tabulated in the attack is simply the frequency count of each letter in the ciphertext (i.e., a table saying that `A` appeared 4 times, `B` appeared 11 times, and so on). Then, we make an initial guess of the mapping defined by the key based on the frequency counts. For example, since `e` is the most frequent letter in English, we will guess that the most frequent character in the ciphertext corresponds to the plaintext character `e`, and so on. Unless the ciphertext is quite long, some of the guesses are likely to be wrong. Even for quite short ciphertexts, however, the guesses will be good enough to enable relatively quick decryption (especially utilizing other knowledge of the English language, such as the fact

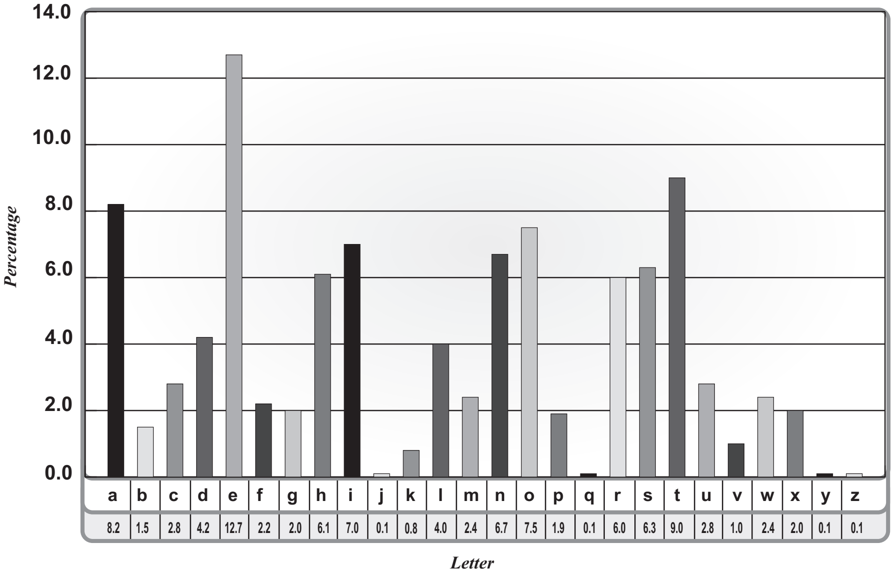

FIGURE 1.2: Average letter frequencies for English-language text.

that between `t` and `e`, the character `h` is likely to appear, and the fact that `u` generally follows `q`).

Actually, it should not be very surprising that the mono-alphabetic substitution cipher can be quickly broken, since puzzles based on this cipher appear in newspapers (and are solved by some people before their morning coffee)! We recommend that you try to decipher the following message — this should help convince you how easy the attack is to carry out (of course, you should use Figure 1.2 to help you):

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWE
OGWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFP
FWQVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILE
OGQILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX
```

We conclude that, although the mono-alphabetic cipher has a very large key space, it is still completely insecure.

An improved attack on the shift cipher. We can use character frequency tables to give an improved attack on the shift cipher. Specifically, our previous attack on the shift cipher required us to decrypt the ciphertext using each possible key, and then check to see which key results in a plaintext that "makes sense". A drawback of this approach is that it is difficult to automate, since it is difficult for a computer to check whether some plaintext "makes sense". (We do *not* claim this is impossible, as it can certainly be done using a dictionary of valid English words. We only claim that it is not trivial.) Moreover, there may be cases — we will see one below — where the plaintext characters are distributed according to English-language text but the plaintext itself is not valid English text, making the problem harder.

As before, associate the letters of the English alphabet with the numbers $0, \ldots, 25$. Let p_i, for $0 \leq i \leq 25$, denote the probability of the ith letter in normal English text. A simple calculation using known values of p_i gives

$$\sum_{i=0}^{25} p_i^2 \approx 0.065 \,. \tag{1.1}$$

Now, say we are given some ciphertext and let q_i denote the probability of the ith letter in this ciphertext (q_i is simply the number of occurrences of the ith letter divided by the length of the ciphertext). If the key is k, then we expect that q_{i+k} should be roughly equal to p_i for every i. (We use $i+k$ instead of the more cumbersome $[i + k \bmod 26]$.) Equivalently, if we compute

$$I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} p_i \cdot q_{i+j}$$

for each value of $j \in \{0, \ldots, 25\}$, then we expect to find that $I_k \approx 0.065$ where k is the key that is actually being used (whereas I_j for $j \neq k$ is expected to be different). This leads to a key-recovery attack that is easy to automate: compute I_j for all j, and then output the value k for which I_k is closest to 0.065.

The Vigenère (poly-alphabetic shift) cipher. As we have described, the statistical attack on the mono-alphabetic substitution cipher could be carried out because the mapping of each letter was fixed. Thus, such an attack can be thwarted by mapping different instances of the same plaintext character to different ciphertext characters. This has the effect of "smoothing out" the probability distribution of characters in the ciphertext. For example, consider the case that `e` is sometimes mapped to `G`, sometimes to `P`, and sometimes to `Y`. Then, the ciphertext letters `G`, `P`, and `Y` will most likely not stand out as more frequent, because other less-frequent characters will be also be mapped to them. Thus, counting the character frequencies will not offer much information about the mapping.

The Vigenère cipher works by applying multiple shift ciphers in sequence. That is, a short, secret word is chosen as the key, and then the plaintext is encrypted by "adding" each plaintext character to the next character of the key (as in the shift cipher), wrapping around in the key when necessary. For example, an encryption of the message `tellhimaboutme` using the key `cafe` would work as follows:

Plaintext:	`tellhimaboutme`
Key:	`cafecafecafeca`
Ciphertext:	`WFRQKJSFEPAYPF`

(The key need not be an actual English word.) This is exactly the same as encrypting the first, fifth, ninth, and so on characters with the shift cipher and key $k = 3$, the second, sixth, tenth, and so on characters with key $k = 1$, the third, seventh, and so on characters with $k = 6$ and the fourth, eighth, and so on characters with $k = 5$. Thus, it is a repeated shift cipher using different keys. Notice that in the above example `l` is mapped once to `R` and once to `Q`. Furthermore, the ciphertext character `F` is sometimes obtained from `e` and sometimes from `a`. Thus, the character frequencies in the ciphertext are "smoothed", as desired.

If the key is a sufficiently-long word (chosen at random), then cracking this cipher seems to be a daunting task. Indeed, it was considered by many to be an unbreakable cipher, and although it was invented in the 16th century a systematic attack on the scheme was only devised hundreds of years later.

Breaking the Vigenère cipher. A first observation in attacking the Vigenère cipher is that *if the length of the key is known*, then the task is relatively easy. Specifically, say the length of the key is t (this is sometimes called the period). Then the ciphertext can be divided into t parts where each part can be viewed as being encrypted using a single instance of the shift cipher. That is, let $k = k_1, \ldots, k_t$ be the key (each k_i is a letter of the alphabet) and let $c_1, c_2, \ldots$ be the ciphertext characters. Then, for every j ($1 \leq j \leq t$) the set of characters

$$c_j, c_{j+t}, c_{j+2t}, \ldots$$

were all encrypted by a shift cipher using key k_j. All that remains is therefore to determine, for each j, which of the 26 possible keys is the correct one. This is not as trivial as in the case of the shift cipher, because by guessing a single letter of the key it is no longer possible to determine if the decryption "makes sense". Furthermore, checking for all values of j simultaneously would require a brute force search through 26^t different possible keys (which is infeasible for t greater than, say, 15). Nevertheless, we can still use the statistical method described earlier. That is, for every set of ciphertext characters relating to a given key (that is, for each value of j), it is possible to tabulate the frequency of each ciphertext character and then check which of the 26 possible shifts yields the "right" probability distribution. Since this can be carried out separately for each key, the attack can be carried out very quickly; all that is required is to build t frequency tables (one for each of the subsets of the characters) and compare them to the real probability distribution.

An alternate, somewhat easier approach, is to use the improved method for attacking the shift cipher that we showed earlier. Recall that this improved attack does not rely on checking for a plaintext that "makes sense", but only relies on the underlying probability distribution of characters in the plaintext.

Either of the above approaches give successful attacks when the key length is known. It remains to show how to determine the length of the key.

Kasiski's method, published in the mid-19th century, gives one approach for solving this problem. The first step is to identify repeated patterns of length 2 or 3 in the ciphertext. These are likely to be due to certain bigrams or trigrams that appear very often in the English language. For example, consider the word "`the`" that appears very often in English text. Clearly, "`the`" will be mapped to different ciphertext characters, depending on its position in the text. However, if it appears twice in the same relative position, then it will be mapped to the same ciphertext characters. For example, if it appears in positions $t + j$ and $2t + i$ (where $i \neq j$) then it will be mapped to different characters each time. However, if it appears in positions $t + j$ and $2t + j$, then it will be mapped to the same ciphertext characters. In a long enough text, there is a good chance that "`the`" will be mapped repeatedly to the same ciphertext characters.

Consider the following concrete example with the key `beads` (spaces have been added for clarity):

Plaintext:	`the man and the woman retrieved the letter from the post office`
Key:	`bea dsb ead sbe adsbe adsbeadsb ean sdeads bead sbe adsb eadbea`
Ciphertext:	`VMF QTP FOH MJJ XSFCS SIMTNFZXF YIS EIYUIK HWPQ MJJ QSLV TGJKGF`

The word `the` is mapped sometimes to `VMF`, sometimes to `MJJ` and sometimes to `YIS`. However, it is mapped *twice* to `MJJ`, and in a long enough text it is likely that it would be mapped multiple times to each of the possibilities. The main observation of Kasiski is that the distance between such multiple appearances (except for some coincidental ones) is a multiple of the period length. (In the above example, the period length is 5 and the distance between the two appearances of `MJJ` is 40, which is 8 times the period length.) Therefore, the *greatest common divisor* of all the distances between the repeated sequences should yield the period length t or a multiple thereof.

An alternative approach called the *index of coincidence method*, is a bit more algorithmic and hence easier to automate. Recall that if the key-length is t, then the ciphertext characters

$$c_1, c_{1+t}, c_{1+2t}, \ldots$$

are encrypted using the same shift. This means that the frequencies of the characters in this sequence are expected to be identical to the character frequencies of standard English text *except in some shifted order.* In more detail: let q_i denote the frequency of the ith English letter in the sequence above (once again, this is simply the number of occurrences of the ith letter divided by the total number of letters in the sequence). If the shift used here is k_1 (this is just the first character of the key), then we expect q_{i+k_1} to be roughly equal to p_i for all i, where p_i is again the frequency of the ith letter in standard English text. But this means that the sequence $p_0, \ldots, p_{25}$ is just the sequence $q_0, \ldots, q_{25}$ shifted by k_1 places. As a consequence, we expect that (see Equation (1.1)):

$$\sum_{i=0}^{25} q_i^2 = \sum_{i=0}^{25} p_i^2 \approx 0.065\,.$$

This leads to a nice way to determine the key length t. For $\tau = 1, 2, \ldots$, look at the sequence of ciphertext characters $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$ and tabulate $q_0, \ldots, q_{25}$ for this sequence. Then compute

$$S_\tau \stackrel{\text{def}}{=} \sum_{i=0}^{25} q_i^2 .$$

When $\tau = t$ we expect to see $S_\tau \approx 0.065$ as discussed above. On the other hand, for $\tau \neq t$ we expect (roughly speaking) that all characters will occur with roughly equal probability in the sequence $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$, and so we expect $q_i \approx 1/26$ for all i. In this case we will obtain

$$S_\tau \approx \sum_{i=0}^{25} \frac{1}{26} \approx 0.038 ,$$

which is sufficiently different from 0.065 for this technique to work.

Ciphertext length and cryptanalytic attacks. The above attacks on the Vigenère cipher require a longer ciphertext than for previous schemes. For example, a large ciphertext is needed for determining the period if Kasiski's method is used. Furthermore, statistics are needed for t different parts of the ciphertext, and the frequency table of a message converges to the average as its length grows (and so the ciphertext needs to be approximately t times longer than in the case of the mono-alphabetic substitution cipher). Similarly, the attack that we showed for the mono-alphabetic substitution cipher requires a longer ciphertext than for the attacks on the shift cipher (which can work for messages consisting of just a single word). This phenomenon is not coincidental, and relates to the size of the key space for each encryption scheme.

Ciphertext-only vs. known-plaintext attacks. The attacks described above are all ciphertext-only attacks (recall that this is the easiest type of attack to carry out in practice). All the above ciphers are *trivially broken* if the adversary is able to carry out a known-plaintext attack; we leave a demonstration of this as an exercise.

Conclusions and discussion. We have presented only a few historical ciphers. Beyond their general historical interest, our aim in presenting them was to illustrate some important lessons regarding cryptographic design. Stated briefly, these lessons are:

1. *Sufficient key space principle:* Assuming sufficiently-long messages are being encrypted, a secure encryption scheme must have a key space that cannot be searched exhaustively in a reasonable amount of time. However, a large key space does not by itself imply security (e.g., the mono-alphabetic substitution cipher has a large key space but is trivial to break). Thus, a large key space is a necessary requirement, but not a sufficient one.

2. *Designing secure ciphers is a hard task:* The Vigenère cipher remained unbroken for a long time, partially due to its presumed complexity. Far more complex schemes have also been used, such as the German Enigma. Nevertheless, this complexity does not imply security and all historical ciphers can be completely broken. In general, it is very hard to design a secure encryption scheme, and such design should be left to experts.

The history of classical encryption schemes is fascinating, both with respect to the methods used as well as the influence of cryptography and cryptanalysis on world history (in World War II, for example). Here, we have only tried to give a taste of some of the more basic methods, with a focus on what modern cryptography can learn from these attempts.

1.4 The Basic Principles of Modern Cryptography

The previous section has given a taste of historical cryptography. It is fair to say that, historically, cryptography was more of an art than any sort of science: schemes were designed in an ad-hoc manner and then evaluated based on their perceived complexity or cleverness. Unfortunately, as we have seen, all such schemes (no matter how clever) were eventually broken.

Modern cryptography, now resting on firmer and more scientific foundations, gives hope of breaking out of the endless cycle of constructing schemes and watching them get broken. In this section we outline the main principles and paradigms that distinguish modern cryptography from classical cryptography. We identify three main principles:

1. *Principle 1* — the first step in solving any cryptographic problem is the formulation of a rigorous and precise definition of security.

2. *Principle 2* — when the security of a cryptographic construction relies on an unproven assumption, this assumption must be precisely stated. Furthermore, the assumption should be as minimal as possible.

3. *Principle 3* — cryptographic constructions should be accompanied by a rigorous proof of security with respect to a definition formulated according to principle 1, and relative to an assumption stated as in principle 2 (if an assumption is needed at all).

We now discuss each of these principles in greater depth.

1.4.1 Principle 1 – Formulation of Exact Definitions

One of the key intellectual contributions of modern cryptography has been the realization that formal definitions of security are *essential* prerequisites for the design, usage, or study of any cryptographic primitive or protocol. Let us explain each of these in turn:

1. *Importance for design:* Say we are interested in constructing a secure encryption scheme. If we do not have a firm understanding of what it is we want to achieve, how can we possibly know whether (or when) we have achieved it? Having an exact definition in mind enables us to better direct our design efforts, as well as to evaluate the quality of what we build, thereby improving the end construction. In particular, it is much better to define what is needed *first* and then begin the design phase, rather than to come up with a *post facto* definition of what has been achieved once the design is complete. The latter approach risks having the design phase end when the designers' patience is tried (rather than when the goal has been met), or may result in a construction that achieves *more* than is needed and is thus less efficient than a better solution.

2. *Importance for usage:* Say we want to use an encryption scheme within some larger system. How do we know which encryption scheme to use? If presented with a candidate encryption scheme, how can we tell whether it suffices for our application? Having a precise definition of the security achieved by a given scheme (coupled with a security proof relative to a formally-stated assumption as discussed in principles 2 and 3) allows us to answer these questions. Specifically, we can define the security that *we* desire in our system (see point 1, above), and then verify whether the definition satisfied by a given encryption scheme suffices for our purposes. Alternatively, we can specify the definition that we need the encryption scheme to satisfy, and look for an encryption scheme satisfying this definition. Note that it may not be wise to choose the "most secure" scheme, since a weaker notion of security may suffice for our application and we may then be able to use a more efficient scheme.

3. *Importance for study:* Given two encryption schemes, how can we compare them? Without any definition of security, the only point of comparison is efficiency, but efficiency alone is a poor criterion since a highly efficient scheme that is completely insecure is of no use. Precise specification of the level of security achieved by a scheme offers another point of comparison. If two schemes are equally efficient but the first one satisfies a stronger definition of security than the second, then the first is preferable.[5] There may also be a trade-off between security and efficiency (see the previous two points), but at least with precise definitions we can understand what this trade-off entails.

[5] Of course, things are rarely this simple.

Of course, precise definitions also enable rigorous proofs (as we will discuss when we come to principle 3), but the above reasons stand irrespective of this.

It is a mistake to think that formal definitions are not needed since "we have an intuitive idea of what security means". For starters, different people have different intuition regarding what is considered secure. Even one person might have multiple intuitive ideas of what security means, depending on the context. For example, in Chapter 3 we will study four different definitions of security for private-key encryption, each of which is useful in a different scenario. In any case, a formal definition is necessary for communicating your "intuitive idea" to someone else.

An example: secure encryption. It is also a mistake to think that formalizing definitions is trivial. For example, how would you formalize the desired notion of security for private-key encryption? (The reader may want to pause to think about this before reading on.) We have asked students many times how secure encryption should be defined, and have received the following answers (often in the following order):

1. *Answer 1 — an encryption scheme is secure if no adversary can find the secret key when given a ciphertext.* Such a definition of encryption completely misses the point. The aim of encryption is to protect the message being encrypted and the secret key is just the means of achieving this. To take this to an absurd level, consider an encryption scheme that ignores the secret key and just outputs the plaintext. Clearly, no adversary can find the secret key. However, it is also clear that no secrecy whatsoever is provided.[6]

2. *Answer 2 — an encryption scheme is secure if no adversary can find the plaintext that corresponds to the ciphertext.* This definition already looks better and can even be found in some texts on cryptography. However, after some more thought, it is also far from satisfactory. For example, an encryption scheme that reveals 90% of the plaintext would still be considered secure under this definition, as long as it is hard to find the remaining 10%. But this is clearly unacceptable in most common applications of encryption. For example, employment contracts are mostly standard text, and only the salary might need to be kept secret; if the salary is in the 90% of the plaintext that is revealed then nothing is gained by encrypting.

   If you find the above counterexample silly, refer again to footnote 6. The point once again is that if the definition as stated isn't what was meant, then a scheme could be proven secure without actually providing the necessary level of protection. (This is a good example of why *exact* definitions are important.)

[6] And lest you respond: "But that's not what I meant!", well, that's exactly the point: it is often not so trivial to formalize what one means.

3. *Answer 3 — an encryption scheme is secure if no adversary can determine any character of the plaintext that corresponds to the ciphertext.* This already looks like an excellent definition. However, other subtleties can arise. Going back to the example of the employment contract, it may be impossible to determine the actual salary or even any digit thereof. However, should the encryption scheme be considered secure if it leaks whether the encrypted salary is greater than or less than $100,000 per year? Clearly not. This leads us to the next suggestion.

4. *Answer 4 — an encryption scheme is secure if no adversary can derive any meaningful information about the plaintext from the ciphertext.* This is already close to the actual definition. However, it is lacking in one respect: it does not define what it means for information to be "meaningful". Different information may be meaningful in different applications. This leads to a very important principle regarding definitions of security for cryptographic primitives: *definitions of security should suffice for all potential applications.* This is essential because one can never know what applications may arise in the future. Furthermore, implementations typically become part of general cryptographic libraries which are then used in may different contexts and for many different applications. Security should ideally be guaranteed for all possible uses.

5. *The final answer — an encryption scheme is secure if no adversary can compute* any *function of the plaintext from the ciphertext.* This provides a very strong guarantee and, when formulated properly, is considered today to be the "right" definition of security for encryption. Even here, there are questions regarding the attack model that should be considered, and how this aspect of security should be defined.

Even though we have now hit upon the correct requirement for secure encryption, conceptually speaking, it remains to state this requirement mathematically and formally, and this is in itself a non-trivial task (one that we will address in detail in Chapters 2 and 3).

As noted in the "final answer", above, our formal definition must also specify the attack model: i.e., whether we assume a ciphertext-only attack or a chosen-plaintext attack. This illustrates a general principle used when formulating cryptographic definitions. Specifically, in order to fully define security of some cryptographic task, there are two distinct issues that must be explicitly addressed. The first is what is considered to be a break, and the second is what is assumed regarding the power of the adversary. The break is exactly what we have discussed above; i.e., an encryption scheme is considered broken if an adversary learns some function of the plaintext from a ciphertext. The *power of the adversary* relates to assumptions regarding the actions the adversary is assumed to be able to take, as well as the adversary's computational power. The former refers to considerations such as whether the adversary is assumed only to be able to eavesdrop on encrypted messages (i.e., a ciphertext-only attack), or whether we assume that the adversary can also actively request encryptions of any plaintext that it likes (i.e., carry out a chosen-plaintext attack). A second issue that must be considered is the computational power of the adversary. For all of this book, except Chapter 2, we will want to ensure security against any *efficient* adversary, by which we mean any adversary running in polynomial time. (A full discussion of this point appears in Section 3.1.2. For now, it suffices to say that an "efficient" strategy is one that can be carried out in a lifetime. Thus "feasible" is arguably a more accurate term.) When translating this into concrete terms, we might require security against any adversary utilizing decades of computing time on a supercomputer.

In summary, any definition of security will take the following general form:

> *A cryptographic scheme for a given task is secure if no adversary of a specified power can achieve a specified break.*

We stress that the definition never assumes anything about the adversary's *strategy*. This is an important distinction: we are willing to assume something about the adversary's capabilities (e.g., that it is able to mount a chosen-plaintext attack but not a chosen-ciphertext attack), but we are *not* willing to assume anything about *how it uses* its abilities. We call this the "arbitrary adversary principle": security must be guaranteed for *any* adversary within the class of adversaries having the specified power. This principle is important because it is impossible to foresee what strategies might be used in an adversarial attack (and history has proven that attempts to do so are doomed to failure).

Mathematics and the real world. A definition of security essentially provides a mathematical formulation of a real-world problem. If the mathematical definition does not appropriately model the real world, then the definition may be useless. For example, if the adversarial power under consideration is too weak (and, in practice, adversaries have more power), or the break is such that it allows real attacks that were not foreseen (like one of the early answers regarding encryption), then "real security" is not obtained, even if a "mathematically-secure" construction is used. In short, a definition of security must accurately model the real world in order for it to deliver on its mathematical promise of security.

It is quite common, in fact, for a widely-accepted definition to be ill-suited for some new application. As one notable example, there are encryption schemes that were proven secure (relative to some definition like the ones we have discussed above) and then implemented on smart-cards. Due to physical properties of the smart-cards, it was possible for an adversary to monitor the *power usage* of the smart-card (e.g., how this power usage fluctuated over time) as the encryption scheme was being run, and it turned out that this information could be used to determine the key. There was nothing wrong with the security definition or the proof that the scheme satisfied this definition; the problem was simply that there was a mismatch between the definition and the real-world implementation of the scheme on a smart-card.

This should not be taken to mean that definitions (or proofs, for that matter) are useless! The definition — and the scheme that satisfies it — may still be appropriate for other settings, such as when encryption is performed on an end-host whose power usage cannot be monitored by an adversary. Furthermore, one way to achieve secure encryption on a smart-card would be to further *refine* the definition so that it takes power analysis into account. Or, perhaps hardware countermeasures for power analysis can be developed, with the effect of making the original definition (and hence the original scheme) appropriate for smart-cards. The point is that with a definition you at least know where you stand, even if the definition turns out not to accurately model the particular setting in which a scheme is used. In contrast, with no definition it is not even clear what went wrong.

This possibility of a disconnect between a mathematical model and the reality it is supposed to be modeling is not unique to cryptography but is something that occurs throughout science. To take an example from the field of computer science, consider the meaning of a *mathematical proof* that there exist well-defined problems that computers cannot solve.[7] The immediate question that arises is *what does it mean for "a computer to solve a problem"*? Specifically, a mathematical proof can be provided only when there is some mathematical definition of what a computer is (or to be more exact, what the process of computation is). The problem is that computation is a real-world process, and there are many different ways of computing. In order for us to be really convinced that the "unsolvable problem" is really unsolvable, we must be convinced that our mathematical definition of computation captures the *real-world process* of computation. How do we know when it does?

This inherent difficulty was noted by Alan Turing who studied questions of what can and cannot be solved by a computer. We quote from his original paper [140] (the text in square brackets replaces original text in order to make it more reader friendly):

> *No attempt has yet been made to show [that the problems we have defined to be solvable by a computer] include [exactly those problems] which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is "What are the possible processes which can be carried out in [computation]?"*
>
> *The arguments which I shall use are of three kinds.*
>
> (a) *A direct appeal to intuition.*

[7] Those who have taken a course in computability theory will be familiar with the fact that such problems do indeed exist (e.g., the Halting Problem).

(b) *A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).*

(c) *Giving examples of large classes of [problems that can be solved using a given definition of computation].*

In some sense, Turing faced the exact same problem as cryptographers. He developed a mathematical model of computation but needed to somehow be convinced that the model was a good one. Likewise, cryptographers define notions of security and need to be convinced that their definitions imply meaningful security guarantees in the real world. As with Turing, they may employ the following tools to become convinced:

1. *Appeals to intuition:* the first tool when contemplating a new definition of security is to see whether it implies security properties that we intuitively expect to hold. This is a minimum requirement, since (as we have seen in our discussion of encryption) our initial intuition usually results in a notion of security that is too weak.

2. *Proofs of equivalence:* it is often the case that a new definition of security is justified by showing that it is equivalent to (or stronger than) a definition that is older, more familiar, or more intuitively-appealing.

3. *Examples:* a useful way of being convinced that a definition of security suffices is to show that the different real-world attacks we are familiar with are ruled out by the definition.

In addition to all of the above, and perhaps most importantly, we rely on the *test of time* and the fact that with time, the scrutiny and investigation of both researchers and practitioners testifies to the soundness of a definition.

1.4.2 Principle 2 – Reliance on Precise Assumptions

Most modern cryptographic constructions cannot be proven secure unconditionally. Indeed, proofs of this sort would require resolving questions in the theory of computational complexity that seem far from being answered today. The result of this unfortunate state of affairs is that security typically relies upon some assumption. The second principle of modern cryptography states that assumptions must be precisely stated. This is for three main reasons:

1. *Validation of the assumption:* By their very nature, assumptions are statements that are not proven but are rather conjectured to be true. In order to strengthen our belief in some assumption, it is necessary for the assumption to be studied. The more the assumption is examined and tested without being successfully refuted, the more confident we are that the assumption is true. Furthermore, study of an assumption can provide positive evidence of its validity by showing that it is implied by some other assumption that is also widely believed.

If the assumption being relied upon is not precisely stated and presented, it cannot be studied and (potentially) refuted. Thus, a pre-condition to raising our confidence in an assumption is having a precise statement of what exactly is assumed.

2. *Comparison of schemes:* Often in cryptography, we may be presented with two schemes that can both be proven to satisfy some definition but each with respect to a different assumption. Assuming both schemes are equally efficient, which scheme should be preferred? If the assumption on which one scheme is based is *weaker* than the assumption on which the second scheme is based (i.e., the second assumption implies the first), then the first scheme is to be preferred since it may turn out that the second assumption is false while the first assumption is true. If the assumptions used by the two schemes are incomparable, then the general rule is to prefer the scheme that is based on the better-studied assumption, or the assumption that is simpler (for the reasons highlighted in the previous paragraphs).

3. *Facilitation of proofs of security:* As we have stated, and will discuss in more depth in principle 3, modern cryptographic constructions are presented together with proofs of security. If the security of the scheme cannot be proven unconditionally and must rely on some assumption, then a mathematical proof that "the construction is secure if the assumption is true" can only be provided if there is a precise statement of what the assumption is.

One observation is that it is always possible to just assume that a construction *itself* is secure. If security is well defined, this is also a precise assumption (and the proof of security for the construction is trivial)! Of course, this is not accepted practice in cryptography for a number of reasons. First of all, as noted above, an assumption that has been tested over the years is preferable to a new assumption that is introduced just to prove a given construction secure. Second, there is a general preference for assumptions that are simpler to state, since such assumptions are easier to study and to refute. So, for example, an assumption of the type that some mathematical problem is hard to solve is simpler to study and work with than an assumption that an encryption schemes satisfies a complex (and possibly unnatural) security definition. When a simple assumption is studied at length and still no refutation is found, we have greater confidence in its being correct. Another advantage of relying on "lower-level" assumptions (rather than just assuming a construction is secure) is that these low-level assumptions can typically be shared amongst a number of constructions. If a specific instantiation of the assumption turns out to be false, it can simply be replaced (within any higher-level construction based on that assumption) by a *different* instantiation of that assumption.

The above methodology is used throughout this book. For example, Chapters 3 and 4 show how to achieve secure communication (in a number of ways), assuming that a primitive called a "pseudorandom function" exists. In these chapters nothing is said at all about how such a primitive can be constructed. In Chapter 5, we then discuss how pseudorandom functions are constructed in practice, and in Chapter 6 we show that pseudorandom functions can be constructed from even lower-level primitives.

1.4.3 Principle 3 – Rigorous Proofs of Security

The first two principles discussed above lead naturally to the current one. Modern cryptography stresses the importance of rigorous proofs of security for proposed schemes. The fact that exact definitions and precise assumptions are used means that such a proof of security is possible. However, why is a proof necessary? The main reason is that the security of a construction or protocol cannot be checked in the same way that software is typically checked. For example, the fact that encryption and decryption "work" and that the ciphertext looks garbled, does not mean that a sophisticated adversary is unable to break the scheme. Without a *proof* that no adversary of the specified power can break the scheme, we are left only with our intuition that this is the case. Experience has shown that intuition in cryptography and computer security is disastrous. There are countless examples of unproven schemes that were broken, sometimes immediately and sometimes years after being presented or deployed.

Another reason why proofs of security are so important is related to the potential damage that can result if an insecure system is used. Although software bugs can sometimes be very costly, the potential damage that may result from someone breaking the encryption scheme or authentication mechanism of a bank is huge. Finally, we note that although many bugs exist in software, things basically work due to the fact that typical users do not try to make their software fail. In contrast, attackers use amazingly complex and intricate means (utilizing specific properties of the construction) to attack security mechanisms with the clear aim of breaking them. Thus, although proofs of correctness are always desirable in computer science, they are absolutely essential in the realm of cryptography and computer security. We stress that the above observations are not just hypothetical, but are conclusions that have been reached after years of empirical evidence and experience.

The reductionist approach. We conclude by noting that most proofs in modern cryptography use what may be called the reductionist approach. Given a theorem of the form

> *"Given that Assumption X is true, Construction Y is secure according to the given definition",*

a proof typically shows how to *reduce* the problem given by Assumption X to the problem of breaking Construction Y. More to the point, the proof will typically show (via a constructive argument) how any adversary breaking

Construction Y can be used as a sub-routine to violate Assumption X. We will have more to say about this in Section 3.1.3.

Summary – Rigorous vs. Ad-Hoc Approaches to Security

The combination of the above three principles constitutes a rigorous approach to cryptography that is distinct from the ad-hoc approach of classical cryptography. The ad-hoc approach may fail on any one of the above three principles, but often ignores them all. Unfortunately, ad hoc solutions are still designed and deployed by those who wish to obtain a "quick and dirty" solution to a problem (or by those who are just simply unaware). We hope that this book will contribute to an awareness of the importance of the rigorous approach, and its success in developing new, mathematically-secure schemes.

References and Additional Reading

In this chapter, we have studied just a few of the known historical ciphers. There are many others of both historical and mathematical interest, and we refer the reader to textbooks by Stinson [138] or Trappe and Washington [139] for further details. The role of these schemes in history (and specifically in the history of war) is a fascinating subject that is covered in the book by Kahn [81].

We discussed the differences between the historical, non-rigorous approach to cryptography (as exemplified by historical ciphers) and a rigorous approach based on precise definitions and proofs. Shannon [127] was the first to take the latter approach. Modern cryptography, which relies on (computational) assumptions in addition to definitions and proofs, was begun in the seminal paper by Goldwasser and Micali [69]. We will study this in Chapter 3.

Exercises

1.1 Decrypt the ciphertext provided at the end of the section on mono-alphabetic substitution.

1.2 Provide a formal definition of the Gen, Enc, and Dec algorithms for both the mono-alphabetic substitution and Vigenère ciphers.

1.3 Consider an improved version of the Vigenère cipher, where instead of using multiple shift ciphers, multiple mono-alphabetic substitution ciphers are used. That is, the key consists of t random permutations of the alphabet, and the plaintext characters in positions $i, t+i, 2t+i$, and so on are encrypted using the ith permutation. Show how to break this version of the cipher.

1.4 In an attempt to prevent Kasiski's attack on the Vigenère cipher, the following modification has been proposed. Given the period t of the cipher, the plaintext is broken up into blocks of size t. Recall that within each block, the Vigenère cipher works by encrypting the ith character with the ith key (using a shift cipher). Letting the key be $k_1, \ldots, k_t$, this means the ith character in each block is encrypted by adding k_i to it, modulo 26. The proposed modification is to encrypt the ith character in the jth block by adding $k_i + j$ modulo 26.

(a) Show that decryption can be carried out.

(b) Describe the effect of the above modification on Kasiski's attack.

(c) Devise an alternate way to determine the period for this scheme.

1.5 Show that the shift, substitution, and Vigenère ciphers are all trivial to break using a known-plaintext attack. How much known plaintext is needed to completely recover the key for each of the ciphers?

1.6 Show that the shift, substitution, and Vigenère ciphers are all trivial to break using a chosen-plaintext attack. How much plaintext must be encrypted in order for the adversary to completely recover the key? Compare to the previous question.