Chapter 1
Introduction to Cryptography and Data Security

This section will introduce the most important terms of modern cryptology and will teach an important lesson about proprietary vs. openly known algorithms. We will also introduce modular arithmetic which is also of major importance in public-key cryptography.

In this chapter you will learn:

- The general rules of cryptography
- Key lengths for short-, medium- and long-term security
- The difference between different types of attacks against ciphers
- A few historical ciphers, and on the way we will learn about modular arithmetic, which is of major importance for modern cryptography as well
- Why one should only use well-established encryption algorithms

1.1 Overview of Cryptology (and This Book)

If we hear the word *cryptography* our first associations might be e-mail encryption, secure website access, smart cards for banking applications or code breaking during World War II, such as the famous attack against the German Enigma encryption machine (Fig. 1.1).

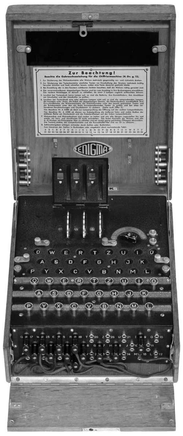

Fig. 1.1 The German Enigma encryption machine (reproduced with permission from the Deutsches Museum, Munich)

Cryptography seems closely linked to modern electronic communication. However, cryptography is a rather old business, with early examples dating back to about 2000 B.C., when non-standard "secret" hieroglyphics were used in ancient Egypt. Since Egyptian days cryptography has been used in one form or the other in many, if not most, cultures that developed written language. For instance, there are documented cases of secret writing in ancient Greece, namely the *scytale* of Sparta (Fig. 1.2), or the famous Caesar cipher in ancient Rome, about which we will learn later in this chapter. This book, however, strongly focuses on modern cryptographic methods and also teaches many data security issues and their relationship with cryptography.

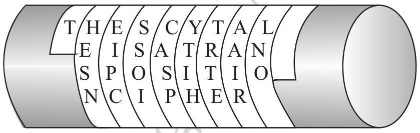

Fig. 1.2 Scytale of Sparta

Let's now have a look at the field of *cryptography* (Fig. 1.3). The first thing

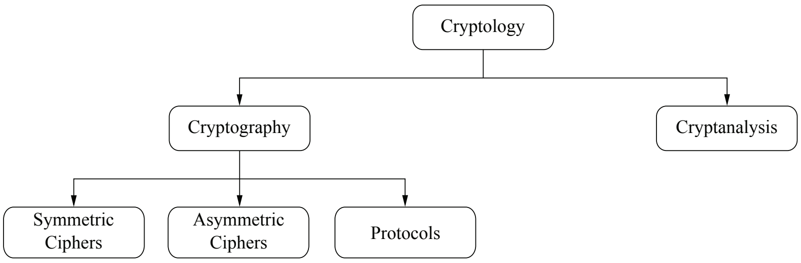

Fig. 1.3 Overview of the field of cryptology

that we notice is that the most general term is *cryptology* and not *cryptography*. Cryptology splits into two main branches:

Cryptography is the science of secret writing with the goal of hiding the meaning of a message.
Cryptanalysis is the science and sometimes art of *breaking* cryptosystems. You might think that code breaking is for the intelligence community or perhaps organized crime, and should not be included in a serious classification of a scientific discipline. However, most cryptanalysis is done by respectable researchers in academia nowadays. Cryptanalysis is of central importance for modern cryptosystems: without people who try to break our crypto methods, we will never know whether they are really secure or not. See Sect. 1.3 for more discussion about this issue.

Because cryptanalysis is the only way to assure that a cryptosystem is secure, it is an integral part of cryptology. Nevertheless, the focus of this book is on crypto**graphy**: We introduce most important practical crypto algorithms in detail. These are all crypto algorithms that have withstood cryptanalysis for a long time, in most cases for several decades. In the case of crypt**analysis** we will mainly restrict ourselves to providing state-of-the-art results with respect to breaking the crypto algorithms that are introduced, e.g., the factoring record for breaking the RSA scheme.

Let's now go back to Fig. 1.3. Cryptography itself splits into three main branches:

Symmetric Algorithms are what many people assume cryptography is about: two parties have an encryption and decryption method for which they share a secret key. All cryptography from ancient times until 1976 was exclusively based on symmetric methods. Symmetric ciphers are still in widespread use, especially for data encryption and integrity check of messages.
Asymmetric (or Public-Key) Algorithms In 1976 an entirely different type of cipher was introduced by Whitfield Diffie, Martin Hellman and Ralph Merkle. In public-key cryptography, a user possesses a secret key as in symmetric cryptography but also a public key. Asymmetric algorithms can be used for applications such as digital signatures and key establishment, and also for classical data encryption.
Cryptographic Protocols Roughly speaking, crypto protocols deal with the application of cryptographic algorithms. Symmetric and asymmetric algorithms

can be viewed as building blocks with which applications such as secure Internet communication can be realized. The Transport Layer Security (TLS) scheme, which is used in every Web browser, is an example of a cryptographic protocol.

Strictly speaking, hash functions, which will be introduced in Chap. 11, form a third class of algorithms but at the same time they share some properties with symmetric ciphers.

In the majority of cryptographic applications in practical systems, symmetric and asymmetric algorithms (and often also hash functions) are all used together. This is sometimes referred to as *hybrid schemes*. The reason for using both families of algorithms is that each has specific strengths and weaknesses.

The main focus of this book is on symmetric and asymmetric algorithms, as well as hash functions. However, we will also introduce basic security protocols. In particular, we will introduce several key establishment protocols and what can be achieved with crypto protocols: confidentiality of data, integrity of data, authentication of data, user identification, etc.

1.2 Symmetric Cryptography

This section deals with the concepts of symmetric ciphers and it introduces the historic substitution cipher. Using the substitution cipher as an example, we will learn the difference between brute-force and analytical attacks.

1.2.1 Basics

Symmetric cryptographic schemes are also referred to as *symmetric-key*, *secret-key*, and *single-key* schemes or algorithms. Symmetric cryptography is best introduced with an easy to understand problem: There are two users, Alice and Bob, who want to communicate over an insecure *channel* (Fig. 1.4). The term channel might sound a bit abstract but it is just a general term for the communication link: This can be the Internet, a stretch of air in the case of mobile phones or wireless LAN communication, or any other communication media you can think of. The actual problem starts with the bad guy, Oscar[1], who has access to the channel, for instance, by hacking into an Internet router or by listening to the radio signals of a Wi-Fi communication. This type of unauthorized listening is called *eavesdropping*. Obviously, there are many situations in which Alice and Bob would prefer to communicate without Oscar listening. For instance, if Alice and Bob represent two offices of a car manufacturer, and they are transmitting documents containing the business strategy for the introduction of new car models in the next few years, these documents should

[1] The name Oscar was chosen to remind us of the word opponent.

not get into the hands of their competitors, or of foreign intelligence agencies for that matter.

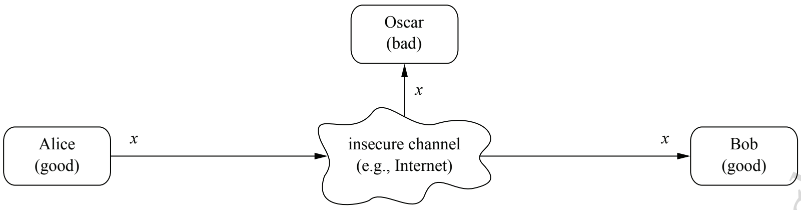

Fig. 1.4 Communication over an insecure channel

In this situation, symmetric cryptography offers a powerful solution: Alice encrypts her message x using a symmetric algorithm, yielding the ciphertext y. Bob receives the ciphertext and decrypts the message. Decryption is, thus, the inverse process of encryption (Fig. 1.5). What is the advantage? If we have a strong encryption algorithm, the ciphertext will look like random bits to Oscar and will contain no information whatsoever that is useful to him.

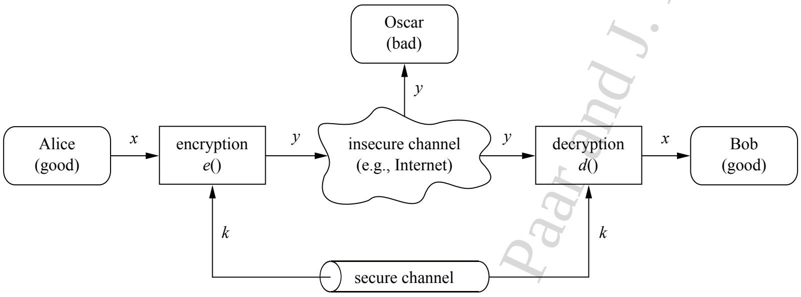

Fig. 1.5 Symmetric-key cryptosystem

The variables x, y and k in Fig. 1.5 are important in cryptography and have special names:

- x is called *plaintext* or *cleartext*,
- y is called *ciphertext*,
- k is called the *key*,
- the set of all possible keys is called the *key space*.

The system needs a secure channel for distribution of the key between Alice and Bob. The secure channel shown in Fig. 1.5 can, for instance, be a human who is transporting the key in a wallet between Alice and Bob. This is, of course, a somewhat cumbersome method. An example where this method works nicely is the pre-shared keys used in Wi-Fi Protected Access (WPA) encryption in wireless

LANs. Later in this book we will learn methods for establishing keys over insecure channels. In any case, the key has only to be transmitted once between Alice and Bob and can then be used for securing many subsequent communications.

One important and also counterintuitive fact in this situation is that both the encryption and the decryption algorithms are publicly known. It seems that keeping the *encryption algorithm* secret should make the whole system harder to break. However, secret algorithms also mean untested algorithms: The only way to find out whether an encryption method is strong, i.e., cannot be broken by a determined attacker, is to make it public and have it analyzed by other cryptographers. Please see Sect. 1.3 for more discussion on this topic. The only thing that should be kept secret in a sound cryptosystem is the key.

Remarks:

1. Of course, if Oscar gets hold of the key, he can easily decrypt the message since the algorithm is publicly known. Hence it is crucial to note that the problem of transmitting a message securely is reduced to the problems of transmitting a key secretly and of storing the key in a secure fashion.
2. In this scenario we only consider the problem of confidentiality, that is, of hiding the contents of the message from an eavesdropper. We will see later in this book that there are many other things we can do with cryptography, such as preventing Oscar from making unnoticed changes to the message (message integrity) or assuring that a message really comes from Alice (sender authentication).

1.2.2 Simple Symmetric Encryption: The Substitution Cipher

We will now learn one of the simplest methods for encrypting text, the *substitution (= replacement) cipher*. Historically this type of cipher has been used many times, and it is a good illustration of basic cryptography. We will use the substitution cipher for learning some important facts about key lengths and about different ways of attacking ciphers.

The goal of the substitution cipher is the encryption of text (as opposed to bits in modern digital systems). The idea is very simple: We substitute each letter of the alphabet with another one.

Example 1.1.

$$\begin{aligned} \texttt{A} &\rightarrow \texttt{k} \\ \texttt{B} &\rightarrow \texttt{d} \\ \texttt{C} &\rightarrow \texttt{w} \\ &\cdots \end{aligned}$$

For instance, the pop group ABBA would be encrypted as kddk.

◇

We assume that we choose the substitution table completely randomly, so that an attacker is not able to guess it. Note that the substitution table is the key of this cryptosystem. As always in symmetric cryptography, the key has to be distributed between Alice and Bob in a secure fashion.

Example 1.2. Let’s look at another ciphertext:

```
iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb
             hcc hwwhbsqvqbre hwq vhlq
```

◇

This does not seem to make too much sense and looks like decent cryptography. *However, the substitution cipher is not secure at all!* Let’s look at ways of breaking the cipher.

First Attack: Brute-Force or Exhaustive Key Search

Brute-force attacks are based on a simple concept: Oscar, the attacker, has the ciphertext from eavesdropping on the channel and happens to have a short piece of plaintext, e.g., the header of a file that was encrypted. Oscar now simply decrypts the first piece of ciphertext with *all possible* keys. Again, the key for this cipher is the substitution table. If the resulting plaintext matches the short piece of plaintext, he knows that he has found the correct key.

Definition 1.2.1 Basic Exhaustive Key Search or Brute-force Attack
Let (x,y) denote the pair of plaintext and ciphertext, and let $K = \{k_1,\ldots,k_\kappa\}$ be the key space of all possible keys k_i. A brute-force attack now checks for every $k_i \in K$ if

$$d_{k_i}(y) \overset{?}{=} x.$$

If the equality holds, a possible correct key is found; if not, proceed with the next key.

In practice, a brute-force attack can be more complicated because incorrect keys can give false positive results. We will address this issue in Sect. 5.2.

It is important to note that a brute-force attack against symmetric ciphers is always possible *in principle*. Whether it is feasible in practice depends on the key space, i.e., on the number of possible keys that exist for a given cipher. If testing all the keys on many modern computers takes too much time, i.e., several decades, the cipher is *computationally secure* against a brute-force attack.

Let's determine the key space of the substitution cipher: When choosing the replacement for the first letter A, we randomly choose one letter from the 26 letters of the alphabet (in the example above we chose k). The replacement for the next alphabet letter B was randomly chosen from the remaining 25 letters, etc. Thus there exist the following number of different substitution tables:

$$\text{key space of the substitution cipher} = 26 \cdot 25 \cdots 3 \cdot 2 \cdot 1 = 26! \approx 2^{88}$$

Even with hundreds of thousands of high-end PCs such a search would take several decades! Thus, we are tempted to conclude that the substitution cipher is secure. But this is incorrect because there is another, more powerful attack.

Second Attack: Letter Frequency Analysis

First we note that the brute-force attack from above treats the cipher as a black box, i.e., we do not analyze the internal structure of the cipher. The substitution cipher can easily be broken by such an analytical attack.

The major weakness of the cipher is that each plaintext symbol always maps to the same ciphertext symbol. That means that the statistical properties of the plaintext are preserved in the ciphertext. If we go back to the second example we observe that the letter q occurs most frequently in the text. From this we know that q must be the substitution for one of the frequent letters in the English language.

For practical attacks, the following properties of language can be exploited:

1. Determine the frequency of every ciphertext letter. The frequency distribution, often even of relatively short pieces of encrypted text, will be close to that of the given language in general. In particular, the most frequent letters can often easily be spotted in ciphertexts. For instance, in English E is the most frequent letter (about 13%), T is the second most frequent letter (about 9%), A is the third most frequent letter (about 8%), and so on. Table 1.1 lists the letter frequency distribution of English.
2. The method above can be generalized by looking at pairs or triples, or quadruples, and so on of ciphertext symbols. For instance, in English (and some other European languages), the letter Q is almost always followed by a U. This behavior can be exploited to detect the substitution of the letter Q and the letter U.
3. If we assume that word separators (blanks) have been found (which is only sometimes the case), one can often detect frequent short words such as THE, AND, etc. Once we have identified one of these words, we immediately know three letters (or whatever the length of the word is) for the entire text.

In practice, the three techniques listed above are often combined to break substitution ciphers.

Example 1.3. If we analyze the encrypted text from Example 1.2, we obtain:

```
WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON
              ALL ARRANGEMENTS ARE MADE
```

Table 1.1 Relative letter frequencies of the English language

Letter	Frequency	Letter	Frequency
A	0.0817	N	0.0675
B	0.0150	O	0.0751
C	0.0278	P	0.0193
D	0.0425	Q	0.0010
E	0.1270	R	0.0599
F	0.0223	S	0.0633
G	0.0202	T	0.0906
H	0.0609	U	0.0276
I	0.0697	V	0.0098
J	0.0015	W	0.0236
K	0.0077	X	0.0015
L	0.0403	Y	0.0197
M	0.0241	Z	0.0007

◇

Lesson learned Good ciphers should hide the statistical properties of the encrypted plaintext. The ciphertext symbols should appear to be random. Also, a large key space alone is not sufficient for a strong encryption function.

1.3 Cryptanalysis

This section deals with recommended key lengths of symmetric ciphers and different ways of attacking crypto algorithms. It is stressed that a cipher should be secure even if the attacker knows the details of the algorithm.

1.3.1 General Thoughts on Breaking Cryptosystems

If we ask someone with some technical background what breaking ciphers is about, he/she will most likely say that code breaking has to do with heavy mathematics, smart people and large computers. We have images in mind of the British code breakers during World War II, attacking the German Enigma cipher with extremely smart mathematicians (the famous computer scientist Alan Turing headed the efforts) and room-sized electro-mechanical computers. However, in practice there are also other methods of code breaking. Let’s look at different ways of breaking cryptosystems *in the real world* (Fig. 1.6).

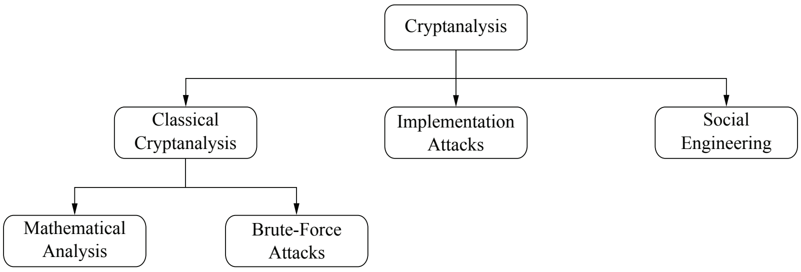

Fig. 1.6 Overview of cryptanalysis

Classical Cryptanalysis

Classical cryptanalysis is understood as the science of recovering the plaintext x from the ciphertext y, or, alternatively, recovering the key k from the ciphertext y. We recall from the earlier discussion that cryptanalysis can be divided into analytical attacks, which exploit the internal structure of the encryption method, and brute-force attacks, which treat the encryption algorithm as a black box and test all possible keys.

Implementation Attacks

Side-channel analysis can be used to obtain a secret key, for instance, by measuring the electrical power consumption of a processor which operates on the secret key. The power trace can then be used to recover the key by applying signal processing techniques. In addition to power consumption, electromagnetic radiation or the run-time behavior of algorithms can give information about the secret key and are, thus, useful side channels.[2] Note also that implementation attacks are mostly relevant against cryptosystems to which an attacker has physical access, such as smart cards. In most Internet-based attacks against remote systems, implementation attacks are usually not a concern.

Social Engineering Attacks

Bribing, blackmailing, tricking or classical espionage can be used to obtain a secret key by involving humans. For instance, forcing someone to reveal his/her secret key, e.g., by holding a gun to his/her head can be quite successful. Another, less violent, attack is to call people whom we want to attack on the phone, and say: “This is

[2] Before you switch on the digital oscilloscope in your lab in order to reload your Geldkarte (the Geldkarte is the electronic wallet function integrated in most German bank cards) to the maximum amount of €200: Modern smart cards have built-in countermeasures against side channel attacks and are very hard to break.

the IT department of your company. For important software updates we need your password". It is always surprising how many people are naïve enough to actually give out their passwords in such situations.

This list of attacks against cryptographic system is certainly not exhaustive. For instance, buffer overflow attacks or malware can also reveal secret keys in software systems. You might think that many of these attacks, especially social engineering and implementation attacks, are "unfair," but there is little fairness in real-world cryptography. If people want to break your IT system, they are already breaking the rules and are, thus, unfair. The major point to learn here is:

An attacker always looks for the weakest link in your cryptosystem. That means we have to choose strong algorithms *and* we have to make sure that social engineering and implementation attacks are not practical.

Even though both implementation attacks and social engineering attacks can be quite powerful in practice, this book mainly assumes attacks based on mathematical cryptanalysis.

Solid cryptosystems should adhere to *Kerckhoffs' Principle*, postulated by Auguste Kerckhoffs in 1883:

Definition 1.3.1 Kerckhoffs' Principle
A cryptosystem should be secure even if the attacker (Oscar) knows all details about the system, with the exception of the secret key. In particular, the system should be secure when the attacker knows the encryption and decryption algorithms.

Important Remark: Kerckhoffs' Principle is counterintuitive! It is extremely tempting to design a system which appears to be more secure because we keep the details hidden. This is called *security by obscurity*. However, experience and military history has shown time and again that such systems are almost always weak, and they are very often broken easily as soon as the secret design has been reverse-engineered or leaked out through other means. An example is the Content Scrambling System (CSS) for DVD content protection, which was broken easily once it was reverse-engineered. This is why a cryptographic scheme must remain secure even if its description becomes available to an attacker.

1.3.2 How Many Key Bits Are Enough?

During the 1990s there was much public discussion about the key length of ciphers. Before we provide some guidelines, there are two crucial aspects to remember:

1. The discussion of key lengths for symmetric crypto algorithms is only relevant if a brute-force attack is the best known attack. As we saw in Sect. 1.2.2 during the security analysis of the substitution cipher, if there is an analytical attack that

works, a large key space does not help at all. Of course, if there is the possibility of social engineering or implementation attacks, a long key also does not help.

2. The key lengths for symmetric and asymmetric algorithms are dramatically different. For instance, an 80-bit symmetric key provides roughly the same security as a 1024-bit RSA (RSA is a popular asymmetric algorithm) key.

Both facts are often misunderstood, especially in the semitechnical literature.

Table 1.2 gives a rough indication of the security of symmetric ciphers *with respect to brute-force attacks*. As described in Sect. 1.2.2, a large key space is a necessary but not sufficient condition for a secure symmetric cipher. The cipher must also be strong against analytical attacks.

Table 1.2 Estimated time for successful brute-force attacks on symmetric algorithms with different key lengths

Key length	Security estimation
56–64 bits	short term: a few hours or days
112–128 bits	long term: several decades in the absence of quantum computers
256 bits	long term: several decades, even with quantum computers that run the currently known quantum computing algorithms

Foretelling the Future Of course, predicting the future tends to be tricky: We can't really foresee new technical or theoretical developments with certainty. As you can imagine, it is very hard to know what kinds of computers will be available in the year 2030. For medium-term predictions, *Moore's Law* is often assumed. Roughly speaking, Moore's Law states that computing power doubles every 18 months while the costs stay constant. This has the following implications in cryptography: If today we need one month and computers worth \$1,000,000 to break a cipher X, then:

- The cost for breaking the cipher will be \$500,000 in 18 months (since we only have to buy half as many computers),
- \$250,000 in 3 years,
- \$125,000 in 4.5 years, and so on.

It is important to stress that Moore's Law is an exponential function. In 15 years, i.e., after 10 iterations of computer power doubling, we can do $2^{10} = 1024$ as many computations for the same money we would need to spend today. Stated differently, we only need to spend about 1/1000th of today's money to do the same computation. In the example above that means that we can break cipher X in 15 years within one month at a cost of about $\$1,000,000/1024 \approx \$1000$. Alternatively, with \$1,000,000, an attack can be accomplished within 45 minutes in 15 years from now. Moore's Law behaves similarly to a bank account with a 50% interest rate: The compound interest grows very, very quickly. Unfortunately, there are few trustworthy banks which offer such an interest rate.

1.4 Modular Arithmetic and More Historical Ciphers

In this section we use two historical ciphers to introduce modular arithmetic with integers. Even though the historical ciphers are no longer relevant, modular arithmetic is extremely important in modern cryptography, especially for asymmetric algorithms. Ancient ciphers date back to Egypt, where substitution ciphers were used. A very popular special case of the substitution cipher is the *Caesar cipher*, which is said to have been used by Julius Caesar to communicate with his army. The Caesar cipher simply shifts the letters in the alphabet by a constant number of steps. When the end of the alphabet is reached, the letters repeat in a cyclic way, similar to numbers in modular arithmetic.

To make computations with letters more practicable, we can assign each letter of the alphabet a number. By doing so, an encryption with the Caesar cipher simply becomes a (modular) addition with a fixed value. Instead of just adding constants, a multiplication with a constant can be applied as well. This leads us to the *affine cipher*.

Both the Caesar cipher and the affine cipher will now be discussed in more detail.

1.4.1 Modular Arithmetic

Almost all crypto algorithms, both symmetric ciphers and asymmetric ciphers, are based on arithmetic within a finite number of elements. Most number sets we are used to, such as the set of natural numbers or the set of real numbers, are infinite. In the following we introduce modular arithmetic, which is a simple way of performing arithmetic in a finite set of integers.

Let's look at an example of a finite set of integers from everyday life:

Example 1.4. Consider the hours on a clock. If you keep adding one hour, you obtain:

$$1h, 2h, 3h, \ldots, 11h, 12h, 1h, 2h, 3h, \ldots, 11h, 12h, 1h, 2h, 3h, \ldots$$

Even though we keep adding one hour, we never leave the set.
◇

Let's look at a general way of dealing with arithmetic in such finite sets.

Example 1.5. We consider the set of the nine numbers:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$$

We can do regular arithmetic as long as the results are smaller than 9. For instance:

$$2 \times 3 = 6$$
$$4 + 4 = 8$$

But what about $8+4$? Now we try the following rule: Perform regular integer arithmetic and divide the result by 9. We then consider **only the remainder** rather than the original result. Since $8+4=12$, and $12/9$ has a remainder of 3, we write:

$$8+4 \equiv 3 \bmod 9$$

◇

We now introduce an exact definition of the modulo operation:

Definition 1.4.1 Modulo Operation
Let $a,r,m \in \mathbb{Z}$ (where $\mathbb{Z}$ is a set of all integers) and $m > 0$. We write

$$a \equiv r \bmod m$$

if m divides $a-r$.
m is called the modulus *and r is called the* remainder.

There are a few implications from this definition which go beyond the casual rule "divide by the modulus and consider the remainder." We discuss these implications below.

Computation of the Remainder

It is always possible to write $a \in \mathbb{Z}$, such that

$$a = q \cdot m + r \quad \text{for} \quad 0 \leq r < m \tag{1.1}$$

Since $a-r = q \cdot m$ (m divides $a-r$) we can now write: $a \equiv r \bmod m$. Note that $r \in \{0,1,2,\ldots,m-1\}$.

Example 1.6. Let $a=42$ and $m=9$. Then

$$42 = 4 \cdot 9 + 6$$

and therefore $42 \equiv 6 \bmod 9$.

◇

The Remainder Is Not Unique

It is somewhat surprising that for every given modulus m and number a, there are (infinitely) many valid remainders. Let's look at another example:

Example 1.7. We want to reduce 12 modulo 9. Here are several results which are correct according to the definition:

- $12 \equiv 3 \bmod 9,\;\; 3$ is a valid remainder since $9|(12-3)$
- $12 \equiv 21 \bmod 9,\;\; 21$ is a valid remainder since $9|(21-3)$
- $12 \equiv -6 \bmod 9,\;\; -6$ is a valid remainder since $9|(-6-3)$

where the "$x|y$" means "x divides y". There is a system behind this behavior. The set of numbers

$$\{\ldots,-24,-15,-6,3,12,15,24,\ldots\}$$

form what is called an *equivalence class*. There are eight other equivalence classes for the modulus 9:

$$\begin{gathered}\{\ldots,-27,-18,-9,\;0,\;9,18,27,\ldots\}\\ \{\ldots,-26,-17,-8,\;1,\;10,19,28,\ldots\}\\ \vdots\\ \{\ldots,-19,-10,-1,\;8,\;17,26,35,\ldots\}\end{gathered}$$

$\diamond$

All Members of a Given Equivalence Class Behave Equivalently

For a given modulus m, it does not matter which element from a class we choose for a given computation. This property of equivalent classes has major practical implications. If we have involved computations with a fixed modulus — which is usually the case in cryptography — we are free to choose the class element that results in the easiest computation. Let's look first at an example:

Example 1.8. The core operation in many practical public-key schemes is an exponentiation of the form $x^e \bmod m$, where x,e,m are very large integers, say, 2048 bits each. Using a toy-size example, we can demonstrate two ways of doing modular exponentiation. We want to compute $3^8 \bmod 7$. The first method is the straightforward approach, and for the second one we switch between equivalent classes.

- $3^8 = 6561 \equiv 2 \bmod 7$, since $6561 = 937 \cdot 7 + 2$
  Note that we obtain the fairly large intermediate result 6561 even though we know that our final result cannot be larger than 6.
- Here is a much smarter method: First we perform two partial exponentiations:

  $$3^8 = 3^4 \cdot 3^4 = 81 \cdot 81$$

  We can now replace the intermediate results 81 by another member of the same equivalence class. The smallest positive member modulo 7 in the class is 4 (since $81 = 11 \cdot 7 + 4$). Hence:

  $$3^8 = 81 \cdot 81 \equiv 4 \cdot 4 = 16 \bmod 7$$

From here we obtain the final result easily as $16 \equiv 2 \bmod 7$.

Note that we could perform the second method without a pocket calculator since the numbers never become larger than 81. For the first method, on the other hand, dividing 6561 by 7 is mentally already a bit challenging. As a general rule we should remember that it is almost always of computational advantage to apply the modulo reduction as soon as we can in order to keep the numbers small.

◇

Of course, the final result of any modulo computation is always the same, no matter how often we switch back and forth between equivalent classes.

Which Remainder Do We Choose?

By agreement, we usually choose r in Eq. (1.1) such that:

$$0 \leq r \leq m-1.$$

However, mathematically it does not matter which member of an equivalent class we use.

1.4.2 Integer Rings

After studying the properties of modulo reduction we are now ready to define in more general terms a structure that is based on modulo arithmetic. Let's look at the mathematical construction that we obtain if we consider the set of integers from zero to $m-1$ together with the operations addition and multiplication:

Definition 1.4.2 Ring
The integer ring $\mathbb{Z}_m$ *consists of:*

1. *The set* $\mathbb{Z}_m = \{0,1,2,\ldots,m-1\}$
2. *Two operations "$+$" and "$\times$" for all* $a,b \in \mathbb{Z}_m$ *such that:*
   1. $a+b \equiv c \bmod m$, $(c \in \mathbb{Z}_m)$
   2. $a \times b \equiv d \bmod m$, $(d \in \mathbb{Z}_m)$

Let's first look at an example for a small integer ring.

Example 1.9. Let $m = 9$, i.e., we are dealing with the ring $\mathbb{Z}_9 = \{0,1,2,3,4,5,6,7,8\}$. Let's look at a few simple arithmetic operations:

$$6+8 = 14 \equiv 5 \bmod 9$$
$$6 \times 8 = 48 \equiv 3 \bmod 9$$

◇

More about rings and finite fields which are related to rings is discussed in Sect. 4.2. At this point, the following properties of rings are important:

- We can add and multiply any two numbers and the result is always in the ring. A ring is said to be *closed*.
- Addition and multiplication are *associative*, e.g., $a+(b+c)=(a+b)+c$, and $a\cdot(b\cdot c)=(a\cdot b)\cdot c$ for all $a,b,c\in\mathbb{Z}_m$.
- There is the *neutral element 0 with respect to addition*, i.e., for every element $a\in\mathbb{Z}_m$ it holds that $a+0\equiv a \bmod m$.
- For any element a in the ring, there is always the negative element $-a$ such that $a+(-a)\equiv 0 \bmod m$, i.e., the *additive inverse* always exists.
- There is the *neutral element 1 with respect to multiplication*, i.e., for every element $a\in\mathbb{Z}_m$ it holds that $a\times 1\equiv a \bmod m$.
- The *multiplicative inverse* exists only for some, but not for all, elements. Let $a\in\mathbb{Z}$, the inverse a^{-1} is defined such that

  $$a\cdot a^{-1}\equiv 1 \bmod\ m$$

  If an inverse exists for a, we can divide by this element since $b/a\equiv b\cdot a^{-1} \bmod m$.
- It takes some effort to *find* the inverse (usually employing the Euclidean algorithm, which is taught in Sect. 6.3). However, there is an easy way of telling whether an inverse for a given element *a exists* or not:
  An element $a\in\mathbb{Z}$ has a multiplicative inverse a^{-1} if and only if $\gcd(a,m)=1$, where gcd is the *greatest common divisor* , i.e., the largest integer that divides both numbers a and m. The fact that two numbers have a gcd of 1 is of great importance in number theory, and there is a special name for it: if $\gcd(a,m)=1$, then a and m are said to be *relatively prime* or coprime.

  Example 1.10. Let's see whether the multiplicative inverse of 15 exists in $\mathbb{Z}_{26}$. Because

  $$\gcd(15,26)=1$$

  the inverse must exist. On the other hand, since

  $$\gcd(14,26)=2\neq 1$$

  the multiplicative inverse of 14 does not exist in $\mathbb{Z}_{26}$.
  ◇

Another ring property is that $a\times(b+c)=(a\times b)+(a\times c)$ for all $a,b,c\in\mathbb{Z}_m$, i.e., the *distributive law* holds. In summary, roughly speaking, we can say that the ring $\mathbb{Z}_m$ is the set of integers $\{0,1,2,\ldots,m-1\}$ in which we can add, subtract, multiply, and sometimes divide.

As mentioned earlier, the ring $\mathbb{Z}_m$, and thus integer arithmetic with the modulo operation, is of central importance to modern public-key cryptography. In practice, the integers involved have a length of 150–4096 bits so that efficient modular computations are a crucial aspect.

1.4.3 Shift Cipher (or Caesar Cipher)

We now introduce another historical cipher, the *shift cipher*. It is actually a special case of the substitution cipher and has a very elegant mathematical description.

The shift cipher itself is extremely simple: We simply shift every plaintext letter by a fixed number of positions in the alphabet. For instance, if we shift by 3 positions, `A` would be substituted by `d`, `B` by `e`, etc. The only problem arises towards the end of the alphabet: what should we do with `X, Y, Z`? As you might have guessed, they should "wrap around". That means `X` should become `a`, `Y` should become `b`, and `Z` is replaced by `c`. Allegedly, Julius Caesar used this cipher with a three-position shift.

The shift cipher also has an elegant description using modular arithmetic. For the mathematical statement of the cipher, the letters of the alphabet are encoded as numbers, as depicted in Table 1.3.

Table 1.3 Encoding of letters for the shift cipher

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Both the plaintext letters and the ciphertext letters are now elements of the ring $\mathbb{Z}_{26}$. Also, the key, i.e., the number of shift positions, is also in $\mathbb{Z}_{26}$ since more than 26 shifts would not make sense (27 shifts would be the same as 1 shift, etc.). The encryption and decryption of the shift cipher follows now as:

Definition 1.4.3 Shift Cipher
Let $x, y, k \in \mathbb{Z}_{26}$.
Encryption: $e_k(x) \equiv x + k \bmod 26$.
Decryption: $d_k(y) \equiv y - k \bmod 26$.

Example 1.11. Let the key be $k = 17$, and the plaintext is:

$$\texttt{ATTACK} = x_1, x_2, \ldots, x_6 = 0, 19, 19, 0, 2, 10.$$

The ciphertext is then computed as

$$y_1, y_2, \ldots, y_6 = 17, 10, 10, 17, 19, 1 = \texttt{rkkrtb}$$

◇

As you can guess from the discussion of the substitution cipher earlier in this book, the shift cipher is not secure at all. There are two ways of attacking it:

1. Since there are only 26 different keys (shift positions), one can easily launch a brute-force attack by trying to decrypt a given ciphertext with all possible 26 keys. If the resulting plaintext is readable text, you have found the key.
2. As for the substitution cipher, one can also use letter frequency analysis.

1.4.4 Affine Cipher

Now, we try to improve the shift cipher by generalizing the encryption function. Recall that the actual encryption of the shift cipher was the addition of the key $y_i = x_i + k \bmod 26$. The *affine cipher* encrypts by multiplying the plaintext by one part of the key followed by addition of another part of the key.

Definition 1.4.4 Affine Cipher
Let $x, y, a, b \in \mathbb{Z}_{26}$
Encryption: $e_k(x) = y \equiv a \cdot x + b \bmod 26.$
Decryption: $d_k(y) = x \equiv a^{-1} \cdot (y - b) \bmod 26.$
with the key: $k = (a, b)$*, which has the restriction:* $\gcd(a, 26) = 1.$

The decryption is easily derived from the encryption function:

$$\begin{aligned} a \cdot x + b &\equiv y \bmod 26 \\ a \cdot x &\equiv (y - b) \bmod 26 \\ x &\equiv a^{-1} \cdot (y - b) \bmod 26 \end{aligned}$$

The restriction $\gcd(a, 26) = 1$ stems from the fact that the key parameter a needs to be inverted for decryption. We recall from Sect. 1.4.2 that an element a and the modulus must be relatively prime for the inverse of a to exist. Thus, a must be in the set:

$$a \in \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\} \tag{1.2}$$

But how do we find a^{-1}? For now, we can simply compute it by trial and error: For a given a we simply try all possible values a^{-1} until we obtain:

$$a \cdot a^{-1} \equiv 1 \bmod 26$$

For instance, if $a = 3$, then $a^{-1} = 9$ since $3 \cdot 9 = 27 \equiv 1 \bmod 26$. Note that a^{-1} also always fulfills the condition $\gcd(a^{-1}, 26) = 1$ since the inverse of a^{-1} always exists. In fact, the inverse of a^{-1} is a itself. Hence, for the trial-and-error determination of a^{-1} one only has to check the values given in Eq. (1.2).

Example 1.12. Let the key be $k = (a,b) = (9,13)$, and the plaintext be

$$\texttt{ATTACK} = x_1, x_2, \ldots, x_6 = 0, 19, 19, 0, 2, 10.$$

The inverse a^{-1} of a exists and is given by $a^{-1} = 3$. The ciphertext is computed as

$$y_1, y_2, \ldots, y_6 = 13, 2, 2, 13, 5, 25 = \texttt{nccnfz}$$

◇

Is the affine cipher secure? No! The key space is only a bit larger than in the case of the shift cipher:

$$\begin{aligned} \text{key space} &= (\text{\#values for } a) \times (\text{\#values for } b) \\ &= 12 \times 26 = 312 \end{aligned}$$

A key space with 312 elements can, of course, still be searched exhaustively, i.e., brute-force attacked, in a fraction of a second with current desktop PCs. In addition, the affine cipher has the same weakness as the shift and substitution cipher: The mapping between plaintext letters and ciphertext letters is fixed. Hence, it can easily be broken with letter frequency analysis.

The remainder of this book deals with strong cryptographic algorithms which are of practical relevance.

1.5 Discussion and Further Reading

This book addresses practical aspects of cryptography and data security and is intended to be used as an introduction; it is suited for classroom use, distance learning and self-study. At the end of each chapter, we provide a discussion section in which we briefly describe topics for readers interested in further study of the material.

About This Chapter: Historical Ciphers and Modular Arithmetic This chapter introduced a few historical ciphers. However, there are many, many more, ranging from ciphers in ancient times to WW II encryption methods. To readers who wish to learn more about historical ciphers and the role they played over the centuries, the books by Bauer [13], Kahn [97] and Singh [157] are highly recommended. Besides making fascinating bedtime reading, these books help one to understand the role that military and diplomatic intelligence played in shaping world history. They also help to show modern cryptography in a larger context.

The mathematics introduced in this chapter, modular arithmetic, belongs to the field of number theory. This is a fascinating subject area which is, unfortunately, historically viewed as a "branch of mathematics without applications". Thus, it is rarely taught outside mathematics curricula. There is a wealth of books on number theory. Among the classic introductory books are references [129, 148]. A particularly accessible book written for non-mathematications is [156].

Research Community and General References Even though cryptography has matured considerably over the last 30 years, it is still a relatively young field compared to other disciplines, and every year brings many new developments and discoveries. Many research results are published at events organized by the International Association for Cryptologic Research (IACR). The proceedings of the three IACR conferences CRYPTO, EUROCRYPT, and ASIACRYPT as well as the IACR workshops Cryptographic Hardware and Embedded Systems (CHES), Fast Software Encryption (FSE), Public Key Cryptography (PKC) and Theory of Cryptograpy (TCC), are excellent sources for tracking the recent developments in the field of cryptology at large. Two important conferences which deal with the larger issue of security (of which cryptography is one aspect) are the IEEE Symposium on Security and Privacy and the USENIX Security forum. All of the events listed take place annually.

There are several good books on cryptography. As reference sources, the *Handbook of Applied Cryptography* [120] and the more recent *Encyclopedia of Cryptography and Security* [168] are highly recommended; both make excellent additions to this textbook.

Provable Security Due to our focus on practical cryptography, this book omits most aspects related to the theoretical foundations of crypto algorithms and protocols. Especially in modern cryptographic research, there is a strong desire to provide statements about cryptographic schemes which are provable in a strict mathematical sense. For this, the goals of both a security system and the adversary are described in a formal model. Often, proofs are achieved by reducing the security of a system to certain assumptions, e.g., that factorization of integers is hard or that a hash function is collision free.

The field of provable security is quite large. We list now some important subareas. A recent survey on the specific area of provable public-key encryption is given in [55]. Provable security is closely related to *cryptographic foundations*, which studies the general assumptions and approaches needed. For instance, the interrelationship between certain presumably hard problems (e.g., integer factorization and discrete logarithm) are studied. The standard references are [81, 83]. *Zero-knowledge proofs* are concerned with proving a certain knowledge towards another party without revealing the secret. They were originally motivated by proving an entity's identity without revealing a password or key. However, they are typically not used that way any more. An early reference is [139], and a more recent tutorial is given in [82]. *Multiparty computation* can be used to compute answers such as the outcome of an election or determining the highest bid in an auction based on encrypted data. The interesting part is that when the protocol is completed the participants know only their own input and the answer but nothing about the encrypted data of the other participants. Good reference sources are [112] and [83, Chap. 7].

A few times this book also touches upon provable security, for instance the relationship between Diffie–Hellman key exchange and the Diffie–Hellman problem (cf. Sect. 8.4), the block cipher based hash functions in Sect. 11.3.2 or the security of the HMAC message authentication scheme in Sect. 12.2.

As a word of caution, it should be mentioned that even though very practical results have been derived from research in the provable security of crypto schemes, many findings are only of limited practical value. Also, the whole field is not without controversy [84, 102].

Secure System Design Cryptography is often an important tool for building a secure system, but on the other hand secure system design encompasses many other aspects. Security systems are intended to protect something valuable, e.g., information, monetary values, personal property, etc. The main objective of secure system design is to make breaking the system more costly than the value of the protected assets, where the "cost" should be measured in monetary value but also in more abstract terms such as effort or reputation. Generally speaking, adding security to a system often narrows its usability.

In order to approach the problem systematically, several general frameworks exist. They typically require that assets and corresponding security needs have to be defined, and that the attack potential and possible attack paths must be evaluated. Finally, adequate countermeasures have to be specified in order to realize an appropriate level of security for a particular application or environment.

There are standards which can be used for evaluation and help to define a secure system. Among the more prominent ones are ISO/IEC [94] (15408, 15443-1, 15446, 19790, 19791, 19792, 21827), the Common Criteria for Information Technology Security Evaluation [46], the German IT-Grundschutzhandbuch [37], FIPS PUBS [77] and many more.

1.6 Lessons Learned

- Never ever develop your own crypto algorithm unless you have a team of experienced cryptanalysts checking your design.
- Do not use unproven crypto algorithms (i.e., symmetric ciphers, asymmetric ciphers, hash functions) or unproven protocols.
- Attackers always look for the weakest point of a cryptosystem. For instance, a large key space by itself is no guarantee for a cipher being secure; the cipher might still be vulnerable against analytical attacks.
- Key lengths for symmetric algorithms in order to thwart exhaustive key-search attacks are:
    - 64 bits: insecure except for data with extremely short-term value.
    - 112–128 bits: long-term security of several decades, including attacks by intelligence agencies unless they possess quantum computers. Based on our current knowledge, attacks are only feasible with quantum computers (which do not exist and perhaps never will).
    - 256 bit: as above, but possibly against attacks by quantum computers.

- Modular arithmetic is a tool for expressing historical encryption schemes, such as the affine cipher, in a mathematically elegant way.