instantaneously communicate with all of his troops, but unfortunately the enemy could listen in on all of his broadcasts. The need for secure and efficient ciphers became paramount and led to the invention of machine ciphers, such as Germany's Enigma machine. This was a device containing a number of rotors, each of which had many wires running through its center. Before a letter was encrypted, the rotors would spin in a predetermined way, thereby altering the paths of the wires and the resultant output. This created an immensely complicated polyalphabetic cipher in which the number of cipher alphabets was enormous. Further, the rotors could be removed and replaced in a vast number of different starting configurations, so breaking the system involved knowing both the circuits through the rotors and figuring out that day's initial rotor configuration.

Despite these difficulties, during World War II the British managed to decipher a large number of messages encrypted on Enigma machines. They were aided in this endeavor by Polish cryptographers who, just before hostilities commenced, shared with Britain and France the methods that they had developed for attacking Enigma. But determining daily rotor configurations and analyzing rotor replacements was still an immensely difficult task, especially after Germany introduced an improved Enigama machine having an extra rotor. The existence of Britain's ULTRA project to decrypt Enigma remained secret until 1974, but there are now several popular accounts. Military intelligence derived from ULTRA was of vital importance in the Allied war effort.

Another WWII cryptanalytic success was obtained by United States cryptographers against a Japanese cipher machine that they code-named Purple. This machine used switches, rather than rotors, but again the effect was to create an incredibly complicated polyalphabetic cipher. A team of cryptographers, led by William Friedman, managed to reconstruct the design of the Purple machine purely by analyzing intercepted encrypted messages. They then built their own machine and proceeded to decrypt many important diplomatic messages.

In this section we have barely touched the surface of the history of cryptography from antiquity through the middle of the 20th century. Good starting points for further reading include Simon Singh's light introduction [128] and David Kahn's massive and comprehensive, but fascinating and quite readable, book *The Codebreakers* [58].

1.7 Symmetric and asymmetric ciphers

We have now seen several different examples of ciphers, all of which have a number of features in common. Bob wants to send a secret message to Alice. He uses a secret key k to scramble his plaintext message m and turn it into a ciphertext c. Alice, upon receiving c, uses the secret key k to unscramble c and reconstitute m. If this procedure is to work properly, then both Alice and Bob must possess copies of the secret key k, and if the system is to provide security, then their adversary Eve must not know k, must not be able to guess k, and must not be able to recover m from c without knowing k.

In this section we formulate the notion of a cryptosystem in abstract mathematical terms. There are many reasons why this is desirable. In particular, it allows us to highlight similarities and differences between different systems, while also providing a framework within which we can rigorously analyze the security of a cryptosystem against various types of attacks.

1.7.1 Symmetric ciphers

Returning to Bob and Alice, we observe that they must share knowledge of the secret key k. Using that secret key, they can both encrypt and decrypt messages, so Bob and Alice have equal (or symmetric) knowledge and abilities. For this reason, ciphers of this sort are known as *symmetric ciphers.* Mathematically, a symmetric cipher uses a key k chosen from a space (i.e., a set) of possible keys $\mathcal{K}$ to encrypt a plaintext message m chosen from a space of possible messages $\mathcal{M}$, and the result of the encryption process is a ciphertext c belonging to a space of possible ciphertexts $\mathcal{C}$.

Thus encryption may be viewed as a function

$$e : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$$

whose domain $\mathcal{K} \times \mathcal{M}$ is the set of pairs (k, m) consisting of a key k and a plaintext m and whose range is the space of ciphertexts $\mathcal{C}$. Similarly, decryption is a function

$$d : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

Of course, we want the decryption function to "undo" the results of the encryption function. Mathematically, this is expressed by the formula

$$d\big(k, e(k, m)\big) = m \qquad \text{for all } k \in \mathcal{K} \text{ and all } m \in \mathcal{M}.$$

It is sometimes convenient to write the dependence on k as a subscript. Then for each key k, we get a pair of functions

$$e_k : \mathcal{M} \longrightarrow \mathcal{C} \qquad \text{and} \qquad d_k : \mathcal{C} \longrightarrow \mathcal{M}$$

satisfying the decryption property

$$d_k\big(e_k(m)\big) = m \qquad \text{for all } m \in \mathcal{M}.$$

In other words, for every key k, the function d_k is the inverse function of the function e_k. In particular, this means that e_k must be one-to-one, since if $e_k(m) = e_k(m')$, then

$$m = d_k\big(e_k(m)\big) = d_k\big(e_k(m')\big) = m'.$$

It is safest for Alice and Bob to assume that Eve knows the encryption method that is being employed. In mathematical terms, this means that Eve knows the functions e and d. What Eve does not know is the particular key k that Alice and Bob are using. For example, if Alice and Bob use a simple substitution cipher, they should assume that Eve is aware of this fact. This illustrates a basic premise of modern cryptography called *Kerckhoff's principle*, which says that the security of a cryptosystem should depend only on the secrecy of the key, and not on the secrecy of the encryption algorithm itself.

If $(\mathcal{K}, \mathcal{M}, \mathcal{C}, e, d)$ is to be a successful cipher, it must have the following properties:

1. For any key $k \in \mathcal{K}$ and plaintext $m \in \mathcal{M}$, it must be easy to compute the ciphertext $e_k(m)$.

2. For any key $k \in \mathcal{K}$ and ciphertext $c \in \mathcal{C}$, it must be easy to compute the plaintext $d_k(c)$.

3. Given one or more ciphertexts $c_1, c_2, \ldots, c_n \in \mathcal{C}$ encrypted using the key $k \in \mathcal{K}$, it must be very difficult to compute any of the corresponding plaintexts $d_k(c_1), \ldots, d_k(c_n)$ without knowledge of k.

There is a fourth property that is desirable, although it is more difficult to achieve.

4. Given one or more pairs of plaintexts and their corresponding ciphertexts, $(m_1, c_1), (m_2, c_2), \ldots, (m_n, c_n)$, it must be difficult to decrypt any ciphertext c that is not in the given list without knowing k. This is known as security against a *chosen plaintext attack*.

Example 1.34. The simple substitution cipher does not have Property 4, since even a single plaintext/ciphertext pair (m, c) reveals most of the encryption table. Thus simple substitution ciphers are vulnerable to chosen plaintext attacks. See Exercise 1.41 for a further example.

In our list of four desirable properties for a cryptosystem, we have left open the question of what exactly is meant by the words "easy" and "hard." We defer a formal discussion of this profound question to Section 4.7 (see also Sections 2.1 and 2.6). For now, we informally take "easy" to mean computable in less than a second on a typical desktop computer and "hard" to mean that all of the computing power in the world would require several years (at least) to perform the computation.

1.7.2 Encoding schemes

It is convenient to view keys, plaintexts, and ciphertexts as numbers and to write those numbers in binary form. For example, we could take strings of

	32	00100000
(	40	00101000
)	41	00101001
,	44	00101100
.	46	00101110

A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
⋮	⋮	⋮
X	88	01011000
Y	89	01011001
Z	90	01011010

a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
⋮	⋮	⋮
x	120	01111000
y	121	01111001
z	122	01111010

Table 1.10: The ASCII encoding scheme

eight bits,[14] which give numbers from 0 to 255, and use them to represent the letters of the alphabet via

$$\texttt{a} = 00000000, \quad \texttt{b} = 00000001, \quad \texttt{c} = 00000010, \quad \ldots, \quad \texttt{z} = 00011001.$$

To distinguish lowercase from uppercase, we could let `A` = 00011011, `B` = 00011100, and so on. This encoding method allows up to 256 distinct symbols to be translated into binary form.

Your computer may use a method of this type, called the ASCII code,[15] to store data, although for historical reasons the alphabetic characters are not assigned the lowest binary values. Part of the ASCII code is listed in Table 1.10. For example, the phrase "`Bed bug.`" (including spacing and punctuation) is encoded in ASCII as

B	e	d		b	u	g	.
66	101	100	32	98	117	103	46
01000010	01100101	01100100	00100000	01100010	01110101	01100111	00101110

Thus where you see the phrase "`Bed bug.`", your computer sees the list of bits

$$0100001001100101011001000010000001100010011101010110011100101110.$$

Definition. An *encoding scheme* is a method of converting one sort of data into another sort of data, for example, converting text into numbers. The distinction between an encoding scheme and an encryption scheme is one of intent. An encoding scheme is assumed to be entirely public knowledge and used by everyone for the same purposes. An encryption scheme is designed to hide information from anyone who does not possess the secret key. Thus an encoding scheme, like an encryption scheme, consists of an encoding function and its inverse decoding function, but for an encoding scheme, both functions are public knowledge and should be fast and easy to compute.

[14]A *bit* is a 0 or a 1. The word "bit" is an abbreviation for *binary digit.*

[15]ASCII is an acronym for American Standard Code for Information Interchange.

With the use of an encoding scheme, a plaintext or ciphertext may be viewed as a sequence of binary blocks, where each block consists of eight bits, i.e., of a sequence of eight ones and zeros. A block of eight bits is called a *byte*. For human comprehension, a byte is often written as a decimal number between 0 and 255, or as a two-digit hexadecimal (base 16) number between 00 and FF. Computers often operate on more than one byte at a time. For example, a 64-bit processor operates on eight bytes at a time.

1.7.3 Symmetric encryption of encoded blocks

In using an encoding scheme as described in Section 1.7.2, it is convenient to view the elements of the plaintext space $\mathcal{M}$ as consisting of bit strings of a fixed length B, i.e., strings of exactly B ones and zeros. We call B the *blocksize* of the cipher. A general plaintext message then consists of a list of message blocks chosen from $\mathcal{M}$, and the encryption function transforms the message blocks into a list of ciphertext blocks in $\mathcal{C}$, where each block is a sequence of B bits. If the plaintext ends with a block of fewer than B bits, we pad the end of the block with zeros. Keep in mind that this encoding process, which converts the original plaintext message into a sequence of blocks of bits in $\mathcal{M}$, is public knowledge.

Encryption and decryption are done one block at a time, so it suffices to study the process for a single plaintext block, i.e., for a single $m \in \mathcal{M}$. This, of course, is why it is convenient to break a message up into blocks. A message can be of arbitrary length, so it's nice to be able to focus the cryptographic process on a single piece of fixed length. The plaintext block m is a string of B bits, which for concreteness we identify with the corresponding number in binary form. In other words, we identify $\mathcal{M}$ with the set of integers m satisfying $0 \le m < 2^B$ via

$$\overbrace{m_{B-1}m_{B-2}\cdots m_2m_1m_0}^{\text{list of } B \text{ bits of } m} \longleftrightarrow \overbrace{m_{B-1}\cdot 2^{B-1} + \cdots + m_2\cdot 2^2 + m_1\cdot 2 + m_0}^{\text{integer between 0 and } 2^B-1}.$$

Here $m_0, m_1, \ldots, m_{B-1}$ are each 0 or 1.

Similarly, we identify the key space $\mathcal{K}$ and the ciphertext space $\mathcal{C}$ with sets of integers corresponding to bit strings of a certain blocksize. For notational convenience, we denote the blocksizes for keys, plaintexts, and ciphertexts by B_k, B_m, and B_c. They need not be the same. Thus we have identified $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$ with sets of positive integers

$$\begin{aligned}\mathcal{K} &= \{k \in \mathbb{Z} \;: 0 \le k < 2^{B_k}\},\\ \mathcal{M} &= \{m \in \mathbb{Z} : 0 \le m < 2^{B_m}\},\\ \mathcal{C} &= \{c \in \mathbb{Z} \;: 0 \le c < 2^{B_c}\}.\end{aligned}$$

An important question immediately arises: how large should Alice and Bob make the set $\mathcal{K}$, or equivalently, how large should they choose the key blocksize B_k? If B_k is too small, then Eve can check every number from 0 to $2^{B_k} - 1$

until she finds Alice and Bob's key. More precisely, since Eve is assumed to know the decryption algorithm d (Kerckhoff's principle), she takes each $k \in \mathcal{K}$ and uses it to compute $d_k(c)$. Assuming that Eve is able to distinguish between valid and invalid plaintexts, eventually she will recover the message.

This attack is known as an *exhaustive search attack* (also sometimes referred to as a *brute-force attack*), since Eve exhaustively searches through the key space. With current technology, an exhaustive search is considered to be infeasible if the space has at least 2^{80} elements. Thus Bob and Alice should definitely choose $B_k \geq 80$.

For many cryptosystems, especially the public key cryptosystems that form the core of this book, there are refinements on the exhaustive search attack that effectively replace the size of the space with its square root. These methods are based on the principle that it is easier to find matching objects (collisions) in a set than it is to find a particular object in the set. We describe some of these *meet-in-the-middle* or *collision attacks* in Sections 2.7, 4.4, 4.5, 6.2, and 6.10. If meet-in-the-middle attacks are available, then Alice and Bob should choose $B_k \geq 160$.

1.7.4 Examples of symmetric ciphers

Before descending further into a morass of theory and notation, we pause to give a mathematical description of some elementary symmetric ciphers.

Let p be a large prime,[16] say $2^{159} < p < 2^{160}$. Alice and Bob take their key space $\mathcal{K}$, plaintext space $\mathcal{M}$, and ciphertext space $\mathcal{C}$ to be the same set,

$$\mathcal{K} = \mathcal{M} = \mathcal{C} = \{1, 2, 3, \ldots, p-1\}.$$

In fancier terminology, $\mathcal{K} = \mathcal{M} = \mathcal{C} = \mathbb{F}_p^*$ are all taken to be equal to the group of units in the finite field $\mathbb{F}_p$.

Alice and Bob randomly select a key $k \in \mathcal{K}$, i.e., they select an integer k satisfying $1 \leq k < p$, and they decide to use the encryption function e_k defined by

$$e_k(m) \equiv k \cdot m \pmod{p}. \tag{1.9}$$

Here we mean that $e_k(m)$ is set equal to the unique positive integer between 1 and p that is congruent to $k \cdot m$ modulo p. The corresponding decryption function d_k is

$$d_k(c) \equiv k' \cdot c \pmod{p},$$

where k' is the inverse of k modulo p. It is important to note that although p is very large, the extended Euclidean algorithm (Remark 1.15) allows us to calculate k' in fewer than $2 \log_2 p + 2$ steps. Thus finding k' from k counts as "easy" in the world of cryptography.

[16]There are in fact many primes in the interval $2^{159} < p < 2^{160}$. The prime number theorem implies that almost 1% of the numbers in this interval are prime. Of course, there is also the question of identifying a number as prime or composite. There are efficient tests that do this, even for very large numbers. See Section 3.4.

It is clear that Eve has a hard time guessing k, since there are approximately 2^{160} possibilities from which to choose. Is it also difficult for Eve to recover k if she knows the ciphertext c? The answer is yes, it is still difficult. Notice that the encryption function

$$e_k : \mathcal{M} \longrightarrow \mathcal{C}$$

is surjective (onto) for any choice of key k. This means that for every $c \in \mathcal{C}$ and any $k \in \mathcal{K}$ there exists an $m \in \mathcal{M}$ such that $e_k(m) = c$. Further, any given ciphertext may represent any plaintext, provided that the plaintext is encrypted by an appropriate key. Mathematically, this may be rephrased by saying that given any ciphertext $c \in \mathcal{C}$ and any plaintext $m \in \mathcal{M}$, there exists a key k such that $e_k(m) = c$. Specifically this is true for the key

$$k \equiv m^{-1} \cdot c \pmod{p}. \tag{1.10}$$

This shows that Alice and Bob's cipher has Properties 1, 2, and 3 as listed on page 38, since anyone who knows the key k can easily encrypt and decrypt, but it is hard to decrypt if you do not know the value of k. However, this cipher does not have Property 4, since even a single plaintext/ciphertext pair (m, c) allows Eve to recover the private key k using the formula (1.10).

It is also interesting to observe that if Alice and Bob define their encryption function to be simply multiplication of integers $e_k(m) = k \cdot m$ with no reduction modulo p, then their cipher still has Properties 1 and 2, but Property 3 fails. If Eve tries to decrypt a single ciphertext $c = k \cdot m$, she still faces the (moderately) difficult task of factoring a large number. However, if she manages to acquire several ciphertexts $c_1, c_2, \ldots, c_n$, then there is a good chance that

$$\begin{aligned} \gcd(c_1, c_2, \ldots, c_n) &= \gcd(k \cdot m_1, k \cdot m_2, \ldots, k \cdot m_n) \\ &= k \cdot \gcd(m_1, m_2, \ldots, m_n) \end{aligned}$$

equals k itself or a small multiple of k. Note that it is an easy task to compute the greatest common divisor.

This observation provides our first indication of how reduction modulo p has a wonderful "mixing" effect that destroys properties such as divisibility. However, reduction is not by itself the ultimate solution. Consider the vulnerability of the cipher (1.9) to a chosen plaintext attack. As noted above, if Eve can get her hands on both a ciphertext c and its corresponding plaintext m, then she easily recovers the key by computing

$$k \equiv m^{-1} \cdot c \pmod{p}.$$

Thus even a single plaintext/ciphertext pair suffices to reveal the key, so the encryption function e_k given by (1.9) does not have Property 4 on page 38.

There are many variants of this "multiplication-modulo-p" cipher. For example, since addition is more efficient than multiplication, there is an "addition-modulo-p" cipher given by

$$e_k(m) \equiv m + k \pmod{p} \qquad \text{and} \qquad d_k(c) \equiv c - k \pmod{p},$$

which is nothing other than the shift or Caesar cipher that we studied in Section 1.1. Another variant, called an *affine cipher*, is a combination of the shift cipher and the multiplication cipher. The key for an affine cipher consists of two integers $k = (k_1, k_2)$ and encryption and decryption are defined by

$$\begin{aligned} e_k(m) &= k_1 \cdot m + k_2 \pmod{p}, \\ d_k(c) &= k_1' \cdot (c - k_2) \pmod{p}, \end{aligned} \tag{1.11}$$

where k_1' is the inverse of k_1 modulo p.

The affine cipher has a further generalization called the *Hill cipher*, in which the plaintext m, the ciphertext c, and the second part of the key k_2 are replaced by column vectors consisting of n numbers modulo p. The first part of the key k_1 is taken to be an n-by-n matrix with mod p integer entries. Encryption and decryption are again given by (1.11), but now multiplication $k_1 \cdot m$ is the product of a matrix and a vector, and k_1' is the inverse matrix of k_1 modulo p. Both the affine cipher and the Hill cipher are vulnerable to chosen plaintext attacks (see Exercises 1.41. and 1.42).

Example 1.35. As noted earlier, addition is generally faster than multiplication, but there is another basic computer operation that is even faster than addition. It is called *exclusive or* and is denoted by XOR or $\oplus$. At the lowest level, XOR takes two individual bits $\beta \in \{0,1\}$ and $\beta' \in \{0,1\}$ and yields

$$\beta \oplus \beta' = \begin{cases} 0 & \text{if } \beta \text{ and } \beta' \text{ are the same,} \\ 1 & \text{if } \beta \text{ and } \beta' \text{ are different.} \end{cases} \tag{1.12}$$

If you think of a bit as a number that is 0 or 1, then XOR is the same as addition modulo 2. More generally, the XOR of two bit strings is the result of performing XOR on each corresponding pair of bits. For example,

$$10110 \oplus 11010 = [1 \oplus 1]\,[0 \oplus 1]\,[1 \oplus 0]\,[1 \oplus 1]\,[0 \oplus 0] = 01100.$$

Using this new operation, Alice and Bob have at their disposal yet another basic cipher defined by

$$e_k(m) = k \oplus m \qquad \text{and} \qquad d_k(c) = k \oplus c.$$

Here $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$ are the sets of all binary strings of length B, or equivalently, the set of all numbers between 0 and $2^B - 1$.

This cipher has the advantage of being highly efficient and completely symmetric in the sense that e_k and d_k are the same function. If k is chosen randomly and is used only once, then this cipher is known as *Vernam's one-time pad*. In Section 4.56 we show that the one-time pad is provably secure. Unfortunately, it requires a key that is as long as the plaintext, which makes it too cumbersome for most practical applications. And if k is used to encrypt more than one plaintext, then Eve may be able to exploit the fact that

$$c \oplus c' = (k \oplus m) \oplus (k \oplus m') = m \oplus m'$$

to extract information about m or m'. It's not obvious how Eve would proceed to find k, m, or m', but simply the fact that the key k can be removed so easily, revealing the potentially less random quantity $m \oplus m'$, should make a cryptographer nervous. Further, this method is vulnerable in some situations to a chosen plaintext attack; see Exercise 1.46.

1.7.5 Random bit sequences and symmetric ciphers

We have arrived, at long last, at the fundamental question regarding the creation of secure and efficient symmetric ciphers. Is it possible to use a single relatively short key k (say consisting of 160 random bits) to securely and efficiently send arbitrarily long messages? Here is one possible construction. Suppose that we could construct a function

$$R : \mathcal{K} \times \mathbb{Z} \longrightarrow \{0, 1\}$$

with the following properties:

1. For all $k \in \mathcal{K}$ and all $j \in \mathbb{Z}$, it is easy to compute $R(k, j)$.
2. Given an arbitrarily long sequence of integers $j_1, j_2, \ldots, j_n$ and given all of the values $R(k, j_1), R(k, j_2), \ldots, R(k, j_n)$, it is hard to determine k.
3. Given any list of integers $j_1, j_2, \ldots, j_n$ and given all of the values

   $$R(k, j_1), R(k, j_2), \ldots, R(k, j_n),$$

   it is hard to guess the value of $R(k, j)$ with better than a 50% chance of success for any value of j not already in the list.

If we could find a function R with these three properties, then we could use it to turn an initial key k into a sequence of bits

$$R(k, 1), R(k, 2), R(k, 3), R(k, 4), \ldots, \qquad (1.13)$$

and then we could use this sequence of bits as the key for a one-time pad as described in Example 1.35.

The fundamental problem with this approach is that the sequence of bits (1.13) is not truly random, since it is generated by the function R. Instead, we say that the sequence of bits (1.13) is a *pseudorandom sequence* and we call R a *pseudorandom number generator*.

Do pseudorandom number generators exist? If so, they would provide examples of the one-way functions defined by Diffie and Hellman in their groundbreaking paper [36], but despite more than a quarter century of work, no one has yet proven the existence of even a single such function. We return to this fascinating subject in Sections 2.1 and 8.2. For now, we content ourselves with a few brief remarks.

Although no one has yet conclusively proven that pseudorandom number generators exist, many candidates have been suggested, and some of these proposals have withstood the test of time. There are two basic approaches to constructing candidates for R, and these two methods provide a good illustration of the fundamental conflict in cryptography between security and efficiency.

The first approach is to repeatedly apply an ad hoc collection of mixing operations that are well suited to efficient computation and that appear to be very hard to untangle. This method is, disconcertingly, the basis for most practical symmetric ciphers, including the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES), which are the two systems most widely used today. See Section 8.10 for a brief description of these modern symmetric ciphers.

The second approach is to construct R using a function whose efficient inversion is a well-known mathematical problem that is believed to be difficult. This approach provides a far more satisfactory theoretical underpinning for a symmetric cipher, but unfortunately, all known constructions of this sort are far less efficient than the ad hoc constructions, and hence are less attractive for real-world applications.

1.7.6 Asymmetric ciphers make a first appearance

If Alice and Bob want to exchange messages using a symmetric cipher, they must first mutually agree on a secret key k. This is fine if they have the opportunity to meet in secret or if they are able to communicate once over a secure channel. But what if they do not have this opportunity and if every communication between them is monitored by their adversary Eve? Is it possible for Alice and Bob to exchange a secret key under these conditions?

Most people's first reaction is that it is not possible, since Eve sees every piece of information that Alice and Bob exchange. It was the brilliant insight of Diffie and Hellman[17] that under certain hypotheses, it is possible. The search for efficient (and provable) solutions to this problem, which is called *public key* (or *asymmetric*) *cryptography*, forms one of the most interesting parts of mathematical cryptography and is the principal focus of this book.

We start by describing a nonmathematical way to visualize public key cryptography. Alice buys a safe with a narrow slot in the top and puts her safe in a public location. Everyone in the world is allowed to examine the safe and see that it is securely made. Bob writes his message to Alice on a piece of paper and slips it through the slot in the top of the safe. Now only a person with the key to the safe, which presumably means only Alice, can retrieve and read Bob's message. In this scenario, Alice's public key is the safe, the

[17]The history is actually somewhat more complicated than this; see our brief discussion in Section 2.1 and the references listed there for further reading.

encryption algorithm is the process of putting the message in the slot, and the decryption algorithm is the process of opening the safe with the key. Note that this setup is not far-fetched; it is used in the real world. For example, the night deposit slot at a bank has this form, although in practice the "slot" must be well protected to prevent someone from inserting a long thin pair of tongs and extracting other people's deposits!

A useful feature of our "safe-with-a-slot" cryptosystem, which it shares with actual public key cryptosystems, is that Alice needs to put only one safe in a public location, and then everyone in the world can use it repeatedly to send encrypted messages to Alice. There is no need for Alice to provide a separate safe for each of her correspondents. And there is also no need for Alice to open the safe and remove Bob's message before someone else such as Carl or Dave uses it to send Alice a message.

We are now ready to give a mathematical formulation of an asymmetric cipher. As usual, there are spaces of keys $\mathcal{K}$, plaintexts $\mathcal{M}$, and ciphertexts $\mathcal{C}$. However, an element k of the key space is really a pair of keys,

$$k = (k_{\mathsf{priv}}, k_{\mathsf{pub}}),$$

called the *private key* and the *public key*, respectively. For each public key k_{pub} there is a corresponding encryption function

$$e_{k_{\mathsf{pub}}} : \mathcal{M} \longrightarrow \mathcal{C},$$

and for each private key k_{priv} there is a corresponding decryption function

$$d_{k_{\mathsf{priv}}} : \mathcal{C} \longrightarrow \mathcal{M}.$$

These have the property that if the pair $(k_{\mathsf{priv}}, k_{\mathsf{pub}})$ is in the key space $\mathcal{K}$, then

$$d_{k_{\mathsf{priv}}}\bigl(e_{k_{\mathsf{pub}}}(m)\bigr) = m \qquad \text{for all } m \in \mathcal{M}.$$

If an asymmetric cipher is to be secure, it must be difficult for Eve to compute the decryption function $d_{k_{\mathsf{priv}}}(c)$, even if she knows the public key k_{pub}. Notice that under this assumption, Alice can send k_{pub} to Bob using an insecure communication channel, and Bob can send back the ciphertext $e_{k_{\mathsf{pub}}}(m)$, without worrying that Eve will be able to decrypt the message. To easily decrypt, it is necessary to know the private key k_{priv}, and presumably Alice is the only person with that information. The private key is sometimes called Alice's *trapdoor information*, because it provides a trapdoor (i.e., a shortcut) for computing the inverse function of $e_{k_{\mathsf{pub}}}$. The fact that the encryption and decryption keys k_{pub} and k_{priv} are different makes the cipher asymmetric, whence its moniker.

It is quite intriguing that Diffie and Hellman created this concept without finding a candidate for an actual pair of functions, although they did propose a similar method by which Alice and Bob can securely exchange a random piece of data whose value is not known initially to either one. We describe Diffie

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
S	C	J	A	X	U	F	B	Q	K	T	P	R	W	E	Z	H	V	L	I	G	Y	D	N	M	O

Table 1.11: Simple substitution encryption table for exercise 1.3

and Hellman's key exchange method in Section 2.3 and then go on to discuss a number of asymmetric ciphers, including ElGamal (Section 2.4), RSA (Section 3.2), ECC (Section 5.4), and NTRU (Section 6.10), whose security relies on the presumed difficulty of a variety of different mathematical problems.

Exercises

Section 1.1. Simple substitution ciphers

1.1. Build a cipher wheel as illustrated in Figure 1.1, but with an inner wheel that rotates, and use it to complete the following tasks. (For your convenience, there is a cipher wheel that you can print and cut out at `www.math.brown.edu/~jhs/MathCrypto/CipherWheel.pdf`.)

(a) Encrypt the following plaintext using a rotation of 11 clockwise.

"A page of history is worth a volume of logic."

(b) Decrypt the following message, which was encrypted with a rotation of 7 clockwise.

`AOLYLHYLUVZLJYLAZILAALYAOHUAOLZLJYLALZAOHALCLYFIVKFNBLZZLZ`

(c) Decrypt the following message, which was encrypted by rotating 1 clockwise for the first letter, then 2 clockwise for the second letter, etc.

`XJHRFTNZHMZGAHIUETXZJNBWNUTRHEPOMDNBJMAUGORFAOIZOCC`

1.2. Decrypt each of the following Caesar encryptions by trying the various possible shifts until you obtain readable text.

(a) `LWKLQNWKDWLVKDOOQHYHUVHHDELOOERDUGORYHOBDVDWUHH`

(b) `UXENRBWXCUXENFQRLQJUCNABFQNWRCJUCNAJCRXWORWMB`

(c) `BGUTBMBGZTFHNLXMKTIPBMAVAXXLXTEPTRLEXTOXKHHFYHKMAXFHNLX`

1.3. For this exercise, use the simple substitution table given in Table 1.11.

(a) Encrypt the plaintext message

`The gold is hidden in the garden.`

(b) Make a decryption table, that is, make a table in which the ciphertext alphabet is in order from `A` to `Z` and the plaintext alphabet is mixed up.

(c) Use your decryption table from (b) to decrypt the following message.

`IBXLX JVXIZ SLLDE VAQLL DEVAU QLB`

1.4. Each of the following messages has been encrypted using a simple substitution cipher. Decrypt them. For your convenience, we have given you a frequency table