1

An Overview of the Subject

In this chapter we introduce some key definitions and general concepts of cryptography that will be used throughout the text. A few simple examples of cryptosystems are provided to help better illustrate the topics. The chapter also provides a road map of the various parts of the subject and where they will be developed in the book. Cryptography has a fascinating history, elements of which punctuate this chapter as well as latter portions of the text. We point out that the general tone of this chapter is informal, and some of the preliminary definitions given here are developed more rigorously in later chapters, after the needed mathematical concepts have been considered.

Basic Concepts

Cryptography is the science of protecting data and communications. One of its main components involves communicating messages or information between designated parties by changing the appearance of the messages (or data) in ways that aim to make it extremely difficult or impossible for other parties to eavesdrop on or interfere with the transmission. Other important aspects include *authentication*, which allows receiving parties the means to ascertain that the communication really does come from the designated sender, and *integrity*, which, among other things, ensures that the message received has not been altered. The subject of cryptography is as old as written languages. There have always been situations where it is important to convey a confidential message. A spy's life could depend on certain messages not being compromised; launch codes for nuclear and other weapons of mass destruction, if cracked, could cause the demise of a whole city, a country, or even the world. Keeping data and messages confidential has become an essential and almost daily issue for almost all of us in our high-tech society. When anyone sends out a personal e-mail, he or she certainly would like to know all who might be able to read it. A supervisor or even a curious coworker may have easy access. Cryptography is vital to electronic commerce, for otherwise it would not be possible to make credit card purchases over the Internet or to wire money from a bank to another location. As our point of departure into this exciting subject, we consider Figure 1.1, which shows the basic idea behind most cryptosystems.

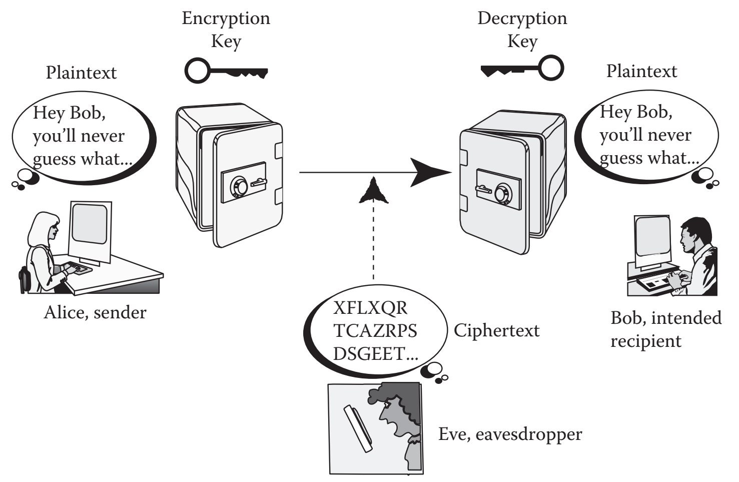


Figure 1.1 A basic reference illustration for a cryptosystem. Alice, the sender, wishes to send Bob, the intended recipient, a confidential message. On Alice's end, the message gets encrypted before it is sent to Bob, who, as the designated recipient, will be able to decrypt the message. Eve, the eavesdropper (a hacker), tries to intercept this message but will not have the key to decode it.

The characters **Alice** and **Bob** (the communicating parties) and **Eve** (the evil eavesdropper) of Figure 1.1 have become standard in cryptographic literature and are used throughout this book.* Another character—Mallory (the malicious manipulator)—will be later introduced as one who tries to fabricate messages or otherwise corrupt communications. As with any sort of security system, greater levels of security that are desired or required need more sophisticated systems to prevent breaches by intruders who must employ more sophisticated means to breach the system. The development in this book is given in essentially chronological order, with the math tools being introduced as they are needed. One general trend is that as cryptosystems have developed over several centuries, the mathematical foundations on which they rely have become increasingly sophisticated.

Any cryptographic system must anticipate **attacks** by **hackers** who might try to break the code of the transfer and thus compromise the data integrity. The advent of high-speed computers has had a tremendous impact on the standards for what are considered to be effective **ciphers** or **cryptosystems**, which are algorithms for rendering messages unintelligible except to the designated recipients. A cryptosystem has two parts: **encryption**, which is done at the sender's end of the message and means to put the actual **plaintext** (original message) into **ciphertext** (secret code), and **decryption**, which is done at the recipient's end and means to translate the ciphertext back into the original plaintext message. Encryption and decryption are done using a **key**, or perhaps two keys (one to "lock" the message in the encryption stage and the other to "unlock" it during decryption), along with algorithms that

* The names in our picture have become folk tradition in cryptography circles. Later, we will examine a related problem where a different sort of hacker tries to send phony messages to Bob, while attempting to make him think they came from Alice. Such an individual is called a "Mallory."

can perform the encryption and decryption. It can usually be assumed that hackers will be able to determine the cryptosystem that is used, but without complete knowledge about the corresponding keys, the system should remain secure. Thus, certain details of the keys must be kept secret.[*]

Depending on the usage, the consequences of an unauthorized break-in, and the skill level of anticipated hackers, a cryptosystem may be simple or may be very sophisticated. High-speed computers have made it possible to implement extremely sophisticated cryptosystems, but at the same time, hackers can use powerful computing tools to help them in being able to crack cryptosystems. Cryptography is a huge industry with many public and private companies working hard to keep the technology state-of-the-art and to keep one step ahead of hackers. The latest technologies in the field depend heavily on many mathematical tools ranging from abstract algebra and number theory to probability. The U.S. federal government is, of course, a big user and consumer of cryptography. Usage comes not only from the defense and intelligence industries (Pentagon, CIA, FBI, and so forth) but also from financial and technology industries. The branch in the U.S. government that is solely dedicated to cryptography is the *National Security Agency* (NSA). The NSA constantly and actively recruits people with mathematics and computer science degrees (from bachelor's degrees to PhDs).

Cryptosystems can be implemented on any alphabet. An **alphabet** is any finite set of symbols. Any ordered sequence of letters from a certain alphabet is called a **string** (from the alphabet).[†] For example, QXUZTKM is a string of length 7 (since it has seven alphabet characters) in the alphabet of the 26 uppercase English letters {A, B, C, …, Y, Z}. **Binary strings** (also called **bit strings**) are strings from the **binary alphabet** {0, 1}; for example, 011100 is a bit string of length 6 (since it consists of six characters). The individual digits in a binary string are called **bits**. The plaintext and ciphertext may be represented in different alphabets. Thus, although it is most convenient to input a plaintext message in a familiar alphabet (such as English letters and digits), the ciphertext produced by the computer would probably be formed in an alphabet that is efficient for computer architecture and manipulations (such as the binary alphabet).

All cryptosystems require algorithms and/or functions to accomplish the encryption and decryption processes. **Algorithms** are simply lists of instructions (or programs or procedures) designed to accomplish certain tasks. The concept of a function is also very general, involving rules or formulas that show how to get an associated output for each permissible input value. Functions can be described in many ways, using graphs, tables, formulas, or algorithms. For example, a table giving the daily high temperatures (rounded to the nearest degree Fahrenheit) at the Los Angeles International Airport for every day over the past five years is a function. The inputs are the days over the past five years, and the outputs are the corresponding high temperatures. To find the output of this function for a

[*] Traditionally, cryptography referred to the design of cryptosystems, cryptanalysis to methods of attacking them, and cryptology to both of these tasks. Increasingly, cryptography is replacing cryptology as the main descriptor of the field, and we will adhere to this convention.

[†] Like sets, strings can be empty; but since empty strings will be unusual in our work, our default assumption will be that strings are nonempty unless explicitly stated otherwise.

given day (over the past five years), we simply look up the temperature on that day in the table. Since functions are used throughout the subject of cryptography, we will now provide a formal definition.

Functions

Definition 1.1

A **function** (or **mapping**) **from a set *A*** (= the set of inputs) **to a set *B*** (= a set containing all possible outputs) is a rule, formula, or algorithm that assigns to each element $a \in A$ (an input) a unique element $f(a) \in B$ (the corresponding output). The element $f(a)$ is also called the **image of *a* under *f***, and if $f(a) = b$, we say that f **maps** a to b. The notation $f : A \to B$ is used to indicate that f is a function from the set A to the set B. The set A is called the **domain** of the function, and the set B is called the **codomain**. The set of all outputs of f is called the **range of *f*** and is denoted as $f(A)$. Note that the range is a subset of the codomain, that is, $f(A) \subseteq B$.

It is helpful to visualize a generic function with a diagram such as the one shown in Figure 1.2.[*]

In contrast with calculus courses, almost all of the functions that are dealt with in cryptography have domains and codomains that are either finite sets or discrete infinite sets.[†] For functions involving small domains and codomains, diagrams can easily be drawn describing their actions; the following example demonstrates this idea.

Example 1.1

Which of the three diagrams in Figure 1.3 represent(s) functions from the domain {a, b, c} to the set {1, 2, 3}?
Solution: The rule F is not a function since the input b is assigned to have two outputs. The other two rules specify functions, since each element of the domain {a, b, c} is assigned exactly one output in the codomain.

[*] In lower-level mathematics classes, students are sometimes taught that a function is just a formula such as $f(x) = x^2$. What is usually intended is that the domain is taken to be the largest possible subset A of real numbers for which the formula makes sense (in this case A = {real numbers}), and so $f: A \to$ {real numbers}.

[†] Unlike continuous infinite sets such as the set of real numbers that contains whole intervals of numbers, discrete infinite sets can be formed by taking a union of an infinite sequence of finite sets. For example, binary strings of any fixed length form a finite set. But the set of all binary strings of finite length, the union of all of the sets of binary strings of length 0, 1, 2, 3, and so forth, is an example of a discrete infinite set. Here is another distinction. It is always possible to represent any element of a discrete set with a finite string (in some alphabet), whereas for continuous infinite sets, this typically cannot be done. For example, there are many real numbers whose decimal expansions are nonending and nonrepeating, and would require an infinite string of decimal digits to write down.

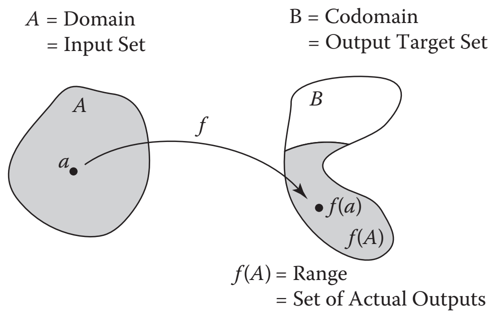


Figure 1.2 Schematic diagram of a function *f*: *A* → *B* with the output *f* (*a*) of an element in the domain *A.* The range *f* (*A*) (shaded on the right) is a subset of the codomain *B.*

We point out two important observations about the functions G and H of Example 1.1. First, note that $G(a) = G(b) = 1$, that is, two inputs are assigned a single output; a function is allowed to do this (but not the other way around). The function H does not do this: different outputs are assigned different inputs. Second, note that not every element of the codomain of G is an actual output: 3 does not occur as an output, but the function H does actually realize each element of its codomain as an output. These two properties of H are very important, and they are the given official designations in the following definition.

One-to-One and Onto Functions, Bijections

Definition 1.2

Suppose that $f : A \to B$ is a function.

(a) We say f is **one-to-one** if different inputs are always assigned different outputs; in other words, if two elements $x, y \in A$ have the same outputs (under f): $f(x) = f(y)$, then they must be the same: $x = y$.
(b) We say f is **onto** if every element of the codomain B occurs as an actual output; in other words, if b is an element of the set B, then there exists an element a of the domain such that the output of a is b: $f(a) = b$. In other words, the range equals the codomain, i.e., $f(A) = B$.
(c) We say f is **bijective**, or a **bijection**, if it is both one-to-one and onto.

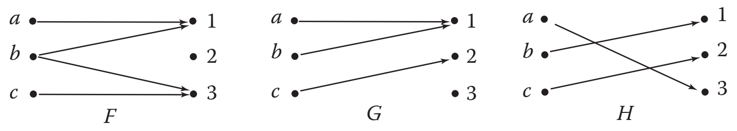


Figure 1.3 Three diagrams assigning elements of the set {*a*, *b*, *c*} to elements of the set {1, 2, 3}.

Note that the temperature function mentioned earlier (with domain being the days over the past five years, codomain being the set of integers $\{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$, and rule specified by a table giving the high temperature at the LAX airport each day) is not a one-to-one function since there are different days (over the past five years) that had the same high temperature (that is, two inputs share the same output). This function is also not onto (Why?).

Example 1.2

Determine whether the following functions are one-to-one and/or onto:

(a) $F : \{a, b, c, \cdots, x, y, z\} \to \{0, 1, 2, \cdots, 23, 24, 25\}$ defined by the rule

F(ith letter of the alphabet) = $i - 1$.

This rule is an abbreviation for writing out each of the 26 input/output relations:

$F(\text{a}) = 0,\ F(\text{b}) = 1,\ F(\text{c}) = 2,\ F(\text{d}) = 3$, etc.

(b) The function G : {length 4 binary strings} $\to$ {length 3 binary strings} defined by the rule

$G(b_1b_2b_3b_4) = b_1b_2b_4$

(i.e., the third binary digit is deleted from the input to produce the output; for example, $G(0010) = 000$).

(c) The function H : {length 3 binary strings} $\to$ {length 4 binary strings} defined by the rule

$H(b_1b_2b_3) = b_1b_2b_3b^*$

where the final bit of the output, b^*, is taken to be 0 if the first three bits add up to an even number, and 1 if they add up to an odd number. For example, if $b_1b_2b_3 = 101$, then $b_1 + b_2 + b_3 = 1 + 0 + 1 = 2$, which is even, so $b^* = 0$, and thus $H(101) = 1010$. Similarly, $H(100) = 1001$.

Solution: Part (a): This function F is both one-to-one and onto; it merely codes each letter (input) into its uniquely defined place in the alphabet, less one (output). Thus, no two letters are assigned the same output (one-to-one), and the codomain consists exactly of all of the outputs (onto).

Part (b): This function G is onto but not one-to-one. To see why it is onto, consider any length-3 binary string $c_1c_2c_3$ where $c_i = 0$, or 1 (that is, any element of the codomain), and notice that it will be the output of G applied to either of the length-4 binary strings $c_1c_2 0 c_3$ or $c_1c_2 1 c_3$. The fact that these two different inputs have the same output shows also that G is not one-to-one.

Part (c): This function H is one-to-one but not onto. To see that it is one-to-one is easy: if $H(b_1b_2b_3) = H(c_1c_2c_3)$, this means that $b_1b_2b_3b^* = c_1c_2c_3c^*$. But for two strings to be the same, each of the corresponding components must be equal. Just looking at the first three gives $b_1 = c_1$, $b_2 = c_2$, $b_3 = c_3$, which is tantamount to $b_1b_2b_3 = c_1c_2c_3$. To see that H is not onto, notice that since the fourth bit, b^*, of the output $H(b_1b_2b_3) = b_1b_2b_3b^*$ is completely determined by the input bits, only one of the two strings $b_1b_2b_3 0$, $b_1b_2b_3 1$ will be an output. For example, since $H(101) = 1010$, the string 1011 will not be an output.

Given any alphabet A, the set of all finite strings in A includes all strings with characters in A of length 1, 2, 3, and so on, and also includes a single **empty string** that contains no characters and so has length 0. We denote this empty string as $\varnothing$. Given two strings σ_1 and σ_2 in A, having respective length ℓ_1 and ℓ_2, their **concatenation** $\sigma_1 \cdot \sigma_2$ is the string of length $\ell_1 + \ell_2$ obtained by pasting the string σ_2 at the right end of string σ_1.

Exercise for the Reader 1.1

(a) Is function C: {finite length binary strings} $\to$ {finite length binary strings} defined by $C(\sigma) = 1010 \cdot \sigma$ one-to-one? Is it onto?

(b) Determine whether the following function is one-to-one:

D : {length 3 binary strings} $\to$ {length 3 binary strings} defined by $D(b_1b_2b_3) = d_1d_2d_3$, where $d_1 = b_1$ and

$$d_2 = \begin{cases} 1, & \text{if } b_1 + b_2 \text{ is odd} \\ 0, & \text{if } b_1 + b_2 \text{ is even} \end{cases} \quad \text{and} \quad d_3 = \begin{cases} 1, & \text{if } b_1 + b_2 + b_3 \text{ is odd} \\ 0, & \text{if } b_1 + b_2 + b_3 \text{ is even} \end{cases}$$

Inverse Functions

The one-to-one property of a function is very important when we use functions in cryptosystems because their processes can be reversed. Since there is only one input for each realized output, the association can be reversed; think of a function diagram as in Figure 1.2: when a function is one-to-one, the arrows can be reversed. If a function $f : A \to B$ is also onto (so a bijection), then every element of the codomain is an output that corresponds to a unique input, and so we can define a function from B to A by associating each element $b \in B$ the corresponding input under f whose output is b. We call this function the **inverse function of *f***, and it is denoted as $f^{-1} : B \to A$. Thus, $f^{-1}(b) = a$ if, and only if, $b = f(a)$. The inverse function simply "undoes" what the function does; see Figure 1.4.

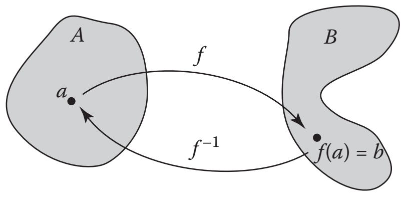


Figure 1.4 Illustration of the inverse function $f^{-1} : B \to A$ of a bijection $f : A \to B$.

Example 1.3

(a) Letting F be the bijection of Example 1.2(a), determine the string $F^{-1}(6)F^{-1}(4)F^{-1}(13)F^{-1}(4)F^{-1}(21)F^{-1}(0)$.

(b) Consider the function $F: \{1, 2, 3, 4\} \to \{1, 2, 3, 4\}$ defined by the rule that for each $a \in \{1, 2, 3, 4\}$, $F(a)$ is the remainder when a^3 is divided by 5.[*] Determine whether the inverse function F^{-1} exists, and if it does, explain how it works.

Solution: Part (a): It is helpful to draw a table for the values of F; see Table 1.3 later in this chapter (but change letters to lowercase). From such a table, we can easily identify the given string to be "geneva."

Part (b): The domain is small enough so that we can compute all of the values of F rather quickly: Since $1^3 = 1 = 5 \cdot 0 + 1$, we get $F(1) = 1$. From $2^3 = 8 = 5 \cdot 1 + 3$, we get $F(2) = 3$. Similarly, the equations $3^3 = 5 \cdot 5 + 2$ and $4^3 = 12 \cdot 5 + 4$ lead us to $F(3) = 2$ and $F(4) = 4$. We now can see that F is a bijection, and the inverse function $F^{-1}: \{1, 2, 3, 4\} \to \{1, 2, 3, 4\}$ can be described by reversing the inputs and outputs. But this clearly results in the same function, that is, $F^{-1} = F$.

Substitution Ciphers

We are nicely prepared to define our first cipher, known as a substitution cipher. Many of us have some experience with such ciphers going back to our days in elementary school when we wanted to pass notes to some of our classmates in such a way that if the note was intercepted by the teacher (or another unintended student), he or she would not be able to read it.

Definition 1.3

A **substitution cipher** is simply a function F from a plaintext alphabet P to a ciphertext alphabet C, that is both *one-to-one* and *onto*. Thus, for

[*] The remainder when we divide a positive integer b by 5 is the unique integer r, with $0 \le r < 5$, such that $b = 5q + r$, for some integer q. This is just the usual remainder in long division that one learned in grade school; we give a much more thorough account of this topic in the next chapter.

every plaintext letter $p \in P$, the function associates a unique ciphertext letter $c = F(p) \in C$, such that:

(i) *One-to-one condition*. Different plaintext letters will always be associated with different ciphertext letters; i.e., if $p_1 \neq p_2$ (two letters in P), then $F(p_1) \neq F(p_2)$ (two letters in C).

(ii) *Onto condition*. Every ciphertext letter is associated with a plaintext letter; i.e., if $c \in C$, then there is an associated plaintext letter $p \in P$, with $c = F(p)$.

The function F (that specifies the correspondence between plaintext and ciphertext letters) is called the *key* of the substitution cipher. More generally, a **key** in a certain cryptosystem is some parameter that is sufficient to completely describe the encryption and/or decryption mapping of any particular instance of the cryptosystem. In some situations, as with a general substitution cipher, the key and the encryption and/or the decryption mapping are synonymous, because it is difficult to describe a general substitution cipher with anything less than a specification of the encryption mapping. Once the key is known, it is straightforward to encode plaintext messages, which are strings in the plaintext alphabet P, into ciphertext, and to decode ciphertext messages back into plaintext. Thus, the key should be made available only to the sender of the messages (who needs it in order to encrypt the plaintext message to the ciphertext message) and the intended recipient (who needs it to decrypt the ciphertext message back to its original plaintext form).

We give a simple example of a substitution cipher in which both plaintext and ciphertext alphabets consist of the set of 26 English letters. For added clarity, we will let the plaintext alphabet be the set of lowercase letters: $P = \{a, b, c, \ldots, x, y, z\}$, and the ciphertext alphabet be the set of uppercase letters: $C = \{A, B, C, \ldots, X, Y, Z\}$.[*] In cases where the plaintext and ciphertext alphabets are (essentially) the same, a substitution cipher corresponds to a *rearrangement* (or *permutation*) of the letters of the alphabet. A special case is where each letter is shifted a certain number of letters down the alphabet (where the ciphertext letters A, B, C, … cycle back after Z). Such substitution ciphers are called **shift ciphers**. The following example describes a shift cipher that was used by the Roman emperor Julius Caesar (100 b.c.–44 b.c.), and has come to be known as the *Caesar cipher*.

Example 1.4: The Caesar Cipher

Consider the substitution cipher determined by the permutation of the 26 (uppercase) letters of the alphabet obtained by shifting each plaintext letter three letters down in the alphabet

[*] All of the ideas that we present would work equally well for larger alphabets, and in practice all contemporary encryption devices are able to deal with plaintext involving upper- and lowercase letters, numbers, punctuation marks, and other symbols. Most modern computer-based cryptosystems (that are discussed after Chapter 7 of this book) process plaintext and ciphertext as binary strings (sequences of zeros and ones), integers, or even objects in more abstract number systems.

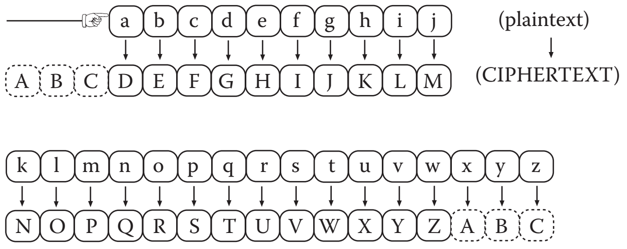

a b c d e f g h i j
(plaintext)
A B C D E F G H I J K L M
(CIPHERTEXT)
k l m n o p q r s t u v w x y z
N O P Q R S T U V W X Y Z A B C

Figure 1.5 Schematic diagram of the shift permutation associated with the Caesar cipher. Each (lowercase) plaintext letter is simply shifted three letters forward in the alphabet to obtain the corresponding ciphertext.

(and recycling back to the beginning of the alphabet when we pass Z). Thus, the ciphertext letters of a, b, and c are D, E, and F, respectively. This entire shift permutation is shown in Figure 1.5.

If Caesar used this cipher to encode his famous quote:

i came, i saw, i conquered

the corresponding ciphertext would be (omitting spaces and commas):[*]

LFDPHLVDZLFRQTXHUHG

To decrypt this ciphertext, an intended recipient would simply need to shift each ciphertext letter backward three letters (see Figure 1.5). The intended recipient would be privy to the cipher, and so would be easily able to perform the decryption.

Notice the key of the Caesar cipher—namely, the shift permutation shown in Figure 1.5—can be used to both encrypt plaintext messages into ciphertext and also to decrypt ciphertext messages back into plaintext. More generally, for any substitution cipher determined by a one-to-one and onto function $F : P \to C$, the decryption procedure simply uses the inverse substitution function $F^{-1} : C \to P$, a table that can be obtained by simply reversing the arrows of the table for F (or in Figure 1.5 by reversing the arrows). Notice that the resulting inverse function of the forward shift by three letters is a backward shift by three letters. If we know we are dealing with a shift cipher, the key can be abbreviated simply by giving the number $\kappa > 0$ of letters that we shift the plaintext letters down the alphabet to obtain the corresponding ciphertext letters. Thus, the key for the Caesar (shift) cipher is $\kappa = 3$.[†]

[*] Preserving spaces, either directly or by means of an additional ciphertext character, would be easily detected and would render any substitution cipher much less secure since it would convey complete information on word lengths of the plaintext.

[†] Note that when $\kappa = 13$, the shift cipher is its own inverse (i.e., it is self-decrypting). This is the famous "rot13" cipher that was used in the early days of the Internet. It was discovered in 1999 that this low-security system was actually used by a major international e-mail provider to store user passwords.

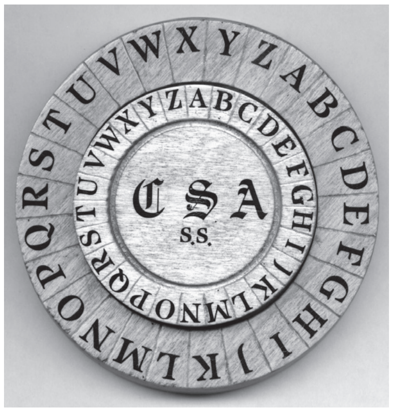


Figure 1.6 A Confederate cipher disk.

Shift ciphers and generalizations of it are most naturally described in terms of modular integers and their arithmetic, and these concepts are developed in the next chapter. More sophisticated ciphers and the ability to program them for computers rely on arithmetic in other number systems such as matrices (Chapter 4), various bases (Chapter 6), finite fields (Chapter 10), and most recently, an interesting arithmetic involving certain points that lie on special curves known as elliptic curves (Chapter 12).

Before the age of computers, mechanical devices and machines were created for the sole purpose of encrypting and decrypting messages with respect to particular cryptosystems. Thus, rather than simply exchanging keys for the cryptosystem, designated parties would all have the same cryptographic devices (which, of course, were kept very secure). A very simple shift cipher device was created by the Confederates for encryption/decryption during the U.S. Civil War, showing that devices such as Caesar's cipher remained in serious use for nearly two millennia. A photograph of a Confederate cipher disk is shown in Figure 1.6.[*] Only five such original devices are known to exist today; one is on display at the NSA museum in Fort Meade, Maryland. Such mechanical cryptosystems reached their pinnacle with the notorious German Enigma machines, which were extremely sophisticated mechanical and electric devices. We return to this interesting era of history in Chapter 3.

Exercise for the Reader 1.2

(a) Find the ciphertext for the plaintext message: "Meet the iceman at noon," using the shift cipher with a shift of 12 letters down the alphabet.

(b) The following ciphertext was encrypted using the shift cipher of part (a):

VQZWUZEUEMFGDZOAMF

Find the original plaintext message.

[*] We kindly acknowledge the Confederate Secret Service Camp 1710 for permission to include this photograph (http://home.earthlink.net/~cssscv/).

Attacks on Cryptosystems

Let us now briefly digress to the other side of the game. How would eavesdropper Eve be able to crack a substitution cipher? Generally, it is safe to assume that the intruder has some information about the type of cryptosystem used, for example, a substitution cipher. Depending on what else Eve knows, there are several different approaches. Some common approaches to eavesdropping, or **passive attacks**, in a cryptosystem are described in general in the following definition. We remind the reader that there are other ways that a cryptosystem can be compromised, for example, by attempting to modify messages or sending an encrypted message pretending to be from someone else. Such an intrusion would be called an **active attack**, since it attempts to change or corrupt the data, and would be done by a Mallory (rather than an Eve). We address active attacks later in this chapter.

Definition 1.4 Types of Passive Attacks on a Cryptosystem

We differentiate the various attacks that Eve can make depending on what information she has about the cryptosystem.

(a) If Eve has only a string (or strings) of ciphertext, her attack would be termed **ciphertext only**.
(b) If Eve has both a string (or strings) of ciphertext and the corresponding plaintext, it is called a **known plaintext** attack.
(c) In a **chosen plaintext** attack, Eve would have temporary access to the encryption system, be able to use it to encrypt some plaintext strings of her choice, and see the corresponding ciphertext strings.
(d) In a **chosen ciphertext** attack, Eve has temporary access to the decryption machine and could use it to decrypt some ciphertext strings of her choice (perhaps ones that she has previously intercepted).

Example 1.5: Passive Attacks on a Substitution Cipher

We discuss how each type of passive attack could be implemented on a substitution cipher.

Since substitution ciphers are **monoalphabetic** ciphers, meaning that each plaintext character is always encrypted to the same ciphertext character, a chosen plaintext or a chosen ciphertext attack could easily reveal the whole system. For example, in a chosen plaintext/ciphertext attack, if we simply encrypted the string “abcd … xyz,” we would have the entire key. If the system was a forward shift cipher (and we had this information), we would only need to encrypt/decrypt a single letter—say, “a”—to determine the key. We will soon introduce ciphers that are **polyalphabetic**, meaning that plaintext characters may encrypt to different ciphertext characters at different instances. Such a naïve approach as above will not suffice for a chosen plaintext/ciphertext attack on polyalphabetic ciphers.

For a general substitution cipher, a known plaintext attack would tell Eve exactly how the letters appearing in the known plaintext are encrypted. If we were dealing with a shift cipher, then, as above, the information about a single plaintext character would determine the entire key.

Finally, we move on to discuss ciphertext-only attacks, which are typically the most difficult. If it is known, however, that we are dealing with a shift cipher, then since there are only 26 keys (25 actually), the **brute-force approach** of simply trying each of them to decode a given ciphertext (until it produces something that makes sense) could easily be implemented (on a computer), and this would completely determine the key. For a general substitution cipher, however, there are too many possibilities to check for a brute-force approach to be feasible, even using supercomputers. Indeed, to see how many different one-to-one and onto substitution functions $F: P \to C$ there are from the 26-letter English plaintext alphabet P to another set C (the ciphertext alphabet) of the same size, we note that there are 26 choices for F(a), and after this is specified—say, $F(\text{a}) = \text{Q}$—there will then be 25 choices for F(b) (that is, all ciphertext letters except Q, since it was already used); once one is specified, there will be 24 choices for F(c), and so on, until we get to F(z), when there will be only one remaining choice. It follows from the multiplication principle,[*] that the total number of substitution functions $F: P \to C$ is $26 \cdot 25 \cdot 24 \cdot \cdots \cdot 3 \cdot 2 \cdot 1$. This product is abbreviated as 26! and is read as "26 *factorial*."[†] Since $26! = 4.0329\ldots \times 10^{26}$, even if we had a computer that could check 1 trillion permutations per second, since there are "only" 3.1536×10^7, seconds in a year, it would require over 10 billion years—over twice the age of the Earth, to have this (fast) computer check through all permutations.

A much more effective tool in a ciphertext-only attack, or to use after one has already made use of a known plaintext attack but still has not completely determined the cipher, is **statistical frequency counts**. The idea of statistical frequency counting methods relies on the fact that some letters tend to occur more frequently in written English than others. Many tables have been

[*] The multiplication principle is a very useful principle for counting. In its general form, it states that if we have a process involving a finite sequence of choices: choice #1 has k_1 possible options, choice #2 has k_2 possible options, choice #3 has k_3 possible options, and so on, then the total number of outcomes of this sequence of choices is the product of the numbers of options: $k_1 \cdot k_2 \cdot k_3 \cdot \cdots$.

[†] In general, if n is any positive integer, $n!$ (n factorial) is defined to be the product of all positive integers that are less than or equal to n; i.e., $n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 3 \cdot 2 \cdot 1$.

TABLE 1.1 Frequencies of the Letters of the English Alphabet

Letter	Probability	Letter	Probability
a	.082	n	.067
b	.015	o	.075
c	.028	p	.019
d	.043	q	.001
e	.127	r	.060
f	.022	s	.063
g	.020	t	.091
h	.061	u	.028
i	.070	v	.010
j	.002	w	.023
k	.008	x	.001
l	.040	y	.020
m	.024	z	.001

published on this; for example, Table 1.1 shows the frequencies that were computed by Beker and Piper [BePi-82].*

Thus, *e* is by far the most frequently appearing letter (nearly 13% of all characters encountered in written English tend to be *e*'s). Although it is possible to create exceptional passages which violate these frequencies,† they tend to be very useful in ciphertext-only attacks. Thus, in a long ciphertext (from a substitution cipher), if a certain character appeared most often—or better yet, close to the 12.7% frequency of *e*—we would predict that this letter is the encryption of *e*. We could continue "guessing letters" in this fashion. Setting it up like a game of hangman, we could sometimes guess new letters simply by completing words. Apart from single letters, we can also use the fact that certain two-letter and three-letter combinations occur more frequently than others. For example, the most common two-letter combinations are (more common items listed first) *th*, *he*, *in*, *er*, *an*, *re*, *ed*, *on*, *es*, *st*, *en*, *at*, and *to*, and the most common three-letter combinations are *the*, *ing*, *and*, *her*, *ere*, *ent*, *tha*, *nth*, and *was*. Larger portions of ciphertext tend to make such statistical methods more effective.

The first polyalphabetic ciphers were created in the 14th and 15th centuries. Since cryptosystems were in constant use for military and diplomatic issues, new developments were sometimes kept as carefully guarded secrets by the ruling governments. Scientists who worked in the

* Of course, there will be variations in frequencies depending on the text corpus being examined. For example, the distributions in e-mails, brief text messages, and computer codes would each have distinguishing characteristics. But for most written English that is not completely informal, the distribution given in Table 1.1 works remarkably well.

† In 1939 an entire novel, *Gadsby*, was written by Ernest Vincent Wright and did not contain the letter e; it had over 50,000 words. Unfortunately, Wright died (at age 66) on the day his book was published, so he never saw it in print.

Figure 1.7 Blaise de Vigenère (1523–1596), French diplomat and cryptographer.

field were, of course, made to understand that even if they were to make a groundbreaking discovery, they could not expect to enjoy any fame, let alone any public recognition for it.

The Vigenère Cipher

A prototypical story exhibiting such characteristics concerns the so-called **Vigenère cipher**. Blaise de Vigenère[*] (Figure 1.7) described this cipher in his authoritative book on cryptography, *Traicté des Chiffres ou Secrètes Manières d'Escrire*[†] (first published in 1586). In it he explained that in the development of his cipher, many of the ingredients came from prominent cryptographers of the recent past; the table was invented by German Johannes Trithemius (1462–1516) and the keyword idea was introduced in a 1553 pamphlet by Italian Giovanni Battista Bellaso. Vigenère's additional contribution to the method had to do with the way in which the key was implemented. Nonetheless, it was through Vigenère's influential book that the method became widely known and hence attributed to him. The Vigenère cipher was easy to implement and many practitioners became confident in its security; it was used extensively up through the mid-19th century. In fact, it earned the name *le chiffre indéchiffrable* ("the unbreakable cipher"). It took a full three centuries for the Vigenère cipher to finally meet its demise. We now explain how the cipher works. In Chapter 5 we show its vulnerability to an ingenious ciphertext-only attack.

[*] Vigenère was born in the town of Saint-Pourçain, the son of a French nobleman. He received his primary education in Paris, after which at age 17 he began his diplomatic career as an assistant to the secretary of state of Francis I. His interest in cryptography began during some long-term diplomatic visits to Italy, beginning at age 26, where he met several prominent Italian cryptographers and began reading books on the subject. After retiring as a diplomat at age 47, he spent much of his retirement working on cryptography and he wrote over 20 books on the subject.

[†] The title of Vigenère's book is in old French (before the Academie Française codified spelling), akin to Shakespearian English. Translation: *Treatise on Numerals and Secret Ways of Writing.*

Definition 1.5 The Vigenère Cipher

The Vigenère cipher is determined by a key that can be any string of letters of the English alphabet, along with the **Vigenère tableau**, which is shown in Table 1.2. To encode a plaintext message, we work our way from left to right. For each plaintext character, we use the corresponding character of the key, and locate the key character's row (the key row) of the Vigenère tableau. The corresponding ciphertext character will be directly below the plaintext character in this key row. If and when the key characters are used up, we recycle back to the start of the key and continue until the plaintext is encoded.

To decode a ciphertext message, we also work from left to right using one key character for each ciphertext character. This time, the key tells us the key row in which we locate the ciphertext character, and the corresponding plaintext character will be the letter on the top of the corresponding column.

Example 1.6

(a) Use the Vigenère cipher to encode the message "Vive la France," using the keyword "money."

(b) Given that the Vigenère cipher of part (a) was used to produce the ciphertext:

NFVREAIGXFQUHMJXCGMLQ

find the original plaintext message.

Solution: Part (a): To encode the first plaintext letter v, the key row would be the m-row (first letter of the key), and the corresponding ciphertext character would be directly below the v-column, that is, H. (This process is shaded in Table 1.2.) Similarly, to encode the second letter i of plaintext, we look in the o-row under i to get W. We continue in this fashion. Note that when we get to the sixth plaintext character a (and again at the 11th plaintext character c), we would recycle back to the beginning of the keyword (so use the m-row). The complete encryption is thus:

```
plaintext:  v i v e l a f r a n c e
keyword:    m o n e y m o n e y m o
ciphertext: H W I I J M T E E L O S
```

Notice that the repeated instances of plaintext letters v, e, and a encrypt to different letters; this is in sharp contrast to substitution ciphers!

Part (b): To decode the first ciphertext letter N, we search for the location of N in the m-row of Table 1.2 (m is the first letter of the key "money"). Since N appears in the b-column of Table 1.2, the first plaintext letter is b. In the same fashion since the second ciphertext letter F appears in the r-column of the o-row of Table 1.2,

TABLE 1.2 Vigenère Tableau

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

Note: The 26 columns (labeled a through z) correspond to plaintext characters, the 26 rows (labeled a through z) correspond to key characters, the uppercase letters inside are ciphertext letters. For example, the shaded v-column corresponds to the plaintext letter v, the shaded m-row corresponds to the key character m, and where they intersect gives the corresponding ciphertext character H.

the second plaintext letter is r. Continuing this process, recycling the keyword "money," we arrive at the following decryption:

ciphertext:	N F V R E A I G X F Q U H M J X C G M L Q
keyword:	m o n e y m o n e y m o n e y m o n e y m
plaintext:	b r i n g o u t t h e g u i l l o t i n e

Exercise 1.3

(a) Use the Vigenère cipher to encode the message "Code blue alert," using the keyword "dijon."

(b) Given that the Vigenère cipher of part (a) was used to produce the ciphertext:

EZNOXRCCOGPQMBVJPC

find the original plaintext message.

Exercise 1.4

(a) Explain how the decryption of a Vigenère cipher can be realized as an encryption of a Vigenère cipher with another keyword.

(b) Find the keyword for the Vigenère cipher corresponding to the decryption process of the Vigenère cipher of Example 1.6.

The Playfair Cipher

Our next example is the first historically documented example of what is known as a **block cipher**. In a block cipher, letters are grouped into same-sized blocks, and these plaintext blocks are processed together to form the corresponding blocks of ciphertext in a way that changing a single letter in a plaintext block can potentially change other letters in the corresponding ciphertext block. It was created in the mid-19th century by the British scientist Sir Charles Wheatstone. It is known as the **Playfair cipher**, after Baron Lyon Playfair, who helped to promote its use by the British government in its South African (Boer) wars. It continued to be used by the British military through World War I.*

* Sir Charles Wheatstone (1802–1875) was a British scientist and prolific inventor most famous for developing the Wheatstone bridge, a device for measuring resistances in electric circuits. He also invented a telegraph before Samuel Morse—an achievement for which he was knighted; a musical instrument (the concertina); and a three-dimensional image display device (the stereoscope). Cryptography was one of his hobbies that he shared with his friend Baron Lyon Playfair (1818–1898), who lived across London's Hammersmith Bridge. They took Sunday walks together where they worked on cracking codes. Their dispositions were quite different. Wheatstone was so extremely shy that, although appointed as a professor, he rarely gave public lectures. In contrast, Playfair, also a scientist, was a public figure who served in an assortment of official roles including as Speaker in the House of Commons and as president of the British Association of Advancement of Science. He had direct access to many policymakers and was able to convince them to adopt the Playfair cipher.

Definition 1.6 The Playfair Cipher

We start with a key, which can be any word. To illustrate, we use "basketball" for the key. Repeated letters of the key are removed; in our example, we get "basketl." The letters of the reduced word are then deployed into a 5×5 array (starting from the upper left and proceeding in reading order), and the remaining spaces of the array are filled with the remaining letters of the alphabet, except that i and j are treated as a single letter. In our example the array would be:

b	a	s	k	e
t	l	c	d	f
g	h	ij	m	n
o	p	q	r	u
v	w	x	y	z

Encryption scheme: Given a plaintext message, for example,

the iceman will arrive at midnight

we group the letters into adjacent pairs, but if any pair has the same two letters, we insert an x between them and regroup.

th ei ce ma nw il la rx ri ve at mi dn ig ht

In case there is an odd number of letters, we would append an additional x at the end to complete the last pair.

Each pair of letters is encrypted using the above 5×5 array depending on which of the following three cases is applicable:[*]

Case 1. The two letters are not in the same row or column of the array. In this case, we replace each letter with the letter in its row that is in the column of the other letter. In our example, the first pair *th* falls into this case, so *t* gets replaced by *l*, and *h* gets replaced by *g*, so the pair gets encrypted as *lg*.

Case 2. The two letters are in the same row. In this case, we replace each letter with the letter to its immediate right, cycling back to the beginning of the row if the letter is all the way on the right. In our example, the pair *ig* falls in this case, so it gets encrypted as *mh*.

Case 3. The two letters are in the same column. In this case, we replace each letter with the letter immediately below it, cycling back to the top of the column if the letter is all the way at the bottom. In our example, the pair *la* falls in this case, so it gets encrypted as *hl*.

[*] As in the Vigenère cipher, rows are horizontal segments of the table, and columns are vertical segments.

Continuing with the remaining pairs, we obtain the sequence:

lg sn fs hk hz hc hl qy qm z bbl nm fm mh gl

and thus the ciphertext is

LGSNFSHKHZHCHLQYQMZBBLNMFMMHGL

The decryption process is accomplished by reversing the above process. First go through the pairs of letters according to their cases (in cases 2 and 3, the replacements are done with the letters immediately to the left or above), then remove any redundant x's to recover the original message.

Exercise for the Reader 1.5

(a) The Playfair cipher is used with keyword "barcelona" to encrypt the message "Meet agent Yullov at the Auberge Restaurant." Find the ciphertext.

(b) The Playfair cipher of part (a) was used to produce the following ciphertext:

MAXHNVGLBERCCXSIHBXSGBBCACMRDERQRZ

Decode this message.

Although it is more secure than substitution ciphers, the Playfair cipher is susceptible to ciphertext-only attacks by doing statistical frequency counts of pairs of letters, since any pair of letters will always get encrypted in the same fashion. But since there are $26^2 = 676$ such ordered pairs of letters and the distinctions are less pronounced than those for single-letter statistics, a ciphertext-only attack would typically require significantly larger portions of ciphertext. Also, short keywords make the Playfair cipher much easier to crack (since the portion of the array after the keyword is much more predictable). For more details on the cryptanalysis of the Playfair cipher, the interested reader may consult [Gai-89]. More sophisticated block ciphers are often naturally developed in terms of matrices, and Chapter 4 presents all of the properties about matrices that we will need.

The 20th century saw a proliferation of ever more sophisticated block cryptosystems that required special mechanical and/or electric devices to use. These systems continued to evolve into the computer age. In Chapter 7 we develop the **Data Encryption Standard** (DES), which was a system adopted in 1977 by the U.S. government to address the growing cryptographic needs of business and industry. The encryption process of DES involved 16 complicated rounds of processing blocks consisting of binary strings (zeros and ones) of size 64. The details are quite complicated, involving various substitutions, permutations, and some other functions that we will explain later. This is a "computer-only" system that is unfeasible for hand calculations. The DES system has a high degree of

entropy, meaning that minor changes in the plaintext can produce radically different ciphertexts. The system was in widespread worldwide use for nearly 30 years. With increasing computer speeds and new cryptanalysis methods being developed, it started to become apparent that a more secure system was required, and this led to the **Advanced Encryption Standard** (AES) system in 2002. Whereas the DES, although quite complicated, relied on rather basic mathematical functions and operations, the AES is based on arithmetic in an abstract number system called a finite field. We discuss finite fields in Chapter 10 and develop the AES in Chapter 11.

All of the cryptosystems described above and all of the other ones that we did not mention dating before the 1970s shared the common disadvantage that they are so-called **symmetric key** (or **private key**) **cryptosystems**. This simply means that the decryption key and process are essentially the same as the encryption key process (perhaps with certain elements of the process being reversed). The ramification is that the key must be provided to both the sender and recipient so that secure communication can take place, and of course, the keys must be kept out of reach from any anticipated hackers.

Most experts had believed that there was no way around this symmetric key concept; in other words, if one knows the encryption scheme, then one should be able to figure out how to reverse the process and thus be able to decrypt any message sent under the same cryptosystem. One of the main drawbacks of all symmetric key cryptosystems is the fact that in order for such a system to be employed, the keys must be distributed to all participating parties before any secure communication can take place. This task by itself is often difficult or impractical. Such drawbacks can now be circumvented thanks to a remarkable revolution in cryptography known as **public key cryptography** or **asymmetric key cryptography** that occurred in the 1970s. The discovery was first published in a groundbreaking 1976 paper by American cryptographers Whit Diffie and Martin Hellman [DiHe-76].* Although Diffie and Hellman did not provide a complete practical implementation of a public key cryptosystem, they provided an important key exchange protocol (*the Diffie–Hellman key exchange*) by which two remote parties could establish a secure key using public (insecure) channels. Inspired by the Diffie–Hellman paper and the need for a practical cryptosystem implementation

* Merkle and Hellman later collaborated to develop one of the first public key cryptosystems; it is discussed, with others, in Chapter 10. Bailey Whitfield (Whit) Diffie went straight from earning his B.S. degree (1965) in mathematics at MIT to a job at the MITRE Corporation, where he became very interested in cryptography. This interest motivated him to accept a position four years later at Stanford's artificial intelligence laboratory. Martin E. Hellman earned his B.S., M.S., and Ph.D. degrees in electrical engineering from New York University. After completing postdoctoral positions at IBM and MIT, he moved on to take an academic position at Stanford in 1971, where he met Diffie. Both received numerous accolades for their pioneering work, including an honorary doctorate for Diffie from the Swiss Federal Institute of Technology. Hellman remained at Stanford until his retirement, where he had an illustrious career with continuous strong research activity and as an award-winning teacher. Diffie worked for most of the rest of his career in industry and currently serves as a vice president and chief security officer at Sun Microsystems.

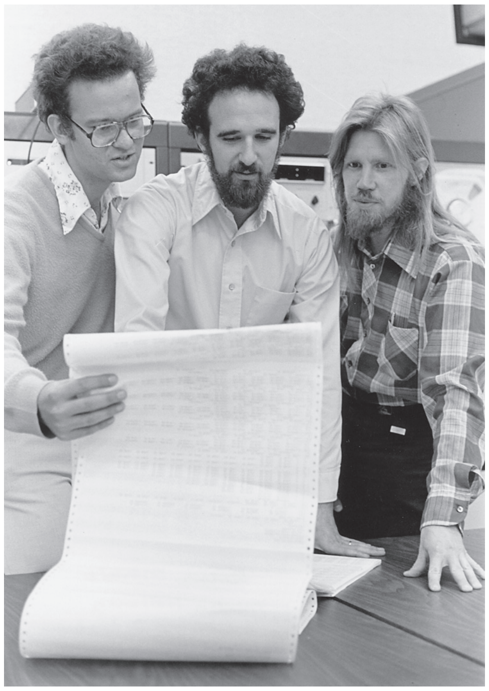


Figure 1.8 American cryptographers Martin Hellman (1945–) (middle), and Whit Diffie (1944–) (right), pictured with Ralph Merkle (1952–). With permission of Chuck Painter/Stanford News Service.

of their concept, MIT scientists Ronald Rivest, Adi Shamir, and Leonard Adleman invented their *RSA cryptosystem*[*] in 1978. This has turned out to be one of the most important and widely used public key cryptosystems, and for their ingenious achievement, the three were awarded the Turing Award in 2002. The Turing Award is often referred to as the Nobel Prize in computer science.

The concept of public key cryptography had actually been discovered by British cryptographer James Ellis (Figure 1.9)[†] in the late 1960s, and the

[*] It was in their RSA paper [RiShAd-78] that the characters "Alice" and "Bob" were introduced as permanent fixtures in the cryptography saga.

[†] James Ellis was born in Britain and studied physics at Imperial College in London. After college, his first job was with the Post Office Research Station (which had an active cryptography team), and he was subsequently recruited in 1952 by the GCHQ (which had previously been *Bletchley Park*). His discovery of public key cryptography was made in the late 1960s, apparently motivated by his reading of a World War II-era paper on the concepts of adding/subtracting random noise to encrypt voice communications. Ellis did not have a sufficient mathematical background to adapt his concept into a practical algorithm. Clifford Cocks had a very strong mathematical background to nicely complement Ellis's strengths. He won the silver medal at the International Mathematical Olympiad as a high school student and went on to study mathematics at Cambridge, and then to do graduate work in number theory at Oxford. As a graduate student, he was recruited by GCHQ in 1973, and after learning of Ellis's public key discovery, he invented, in his first year at GCHQ, the public key cryptosystem that was later known as RSA. It was not until 1997 that the GCHQ allowed information about these discoveries to be made public. This dissemination was made through a public lecture by Cocks in that same year. The timing was unfortunate, since Ellis had passed away one month before this talk.

Figure 1.9 James H. Ellis (1924–1997), British cryptographer.

RSA implementation of it by Clifford Cocks (Figure 1.10) in 1973, while they were employed at the *Government Communications Headquarters* (GCHQ), the British analogue of the United States' NSA. The latter scientists did not receive any recognition for their discoveries until 1997, when the British government decided to declassify the information. Such stories are typical of many of the unsung heroes of cutting-edge cryptography, who often are required (by their governments) to keep a tight lid on their discoveries as matters of national security.

We will enter into the technical details of public key cryptography in Chapter 9, but it will be helpful to first give a superficial overview: Each communicating party (or individual) has two keys, a **public key** and a **private key**. Unlike with symmetric key cryptography, it is not feasible to obtain the private key from knowledge of the public key. The directory of public keys is made available to the general public (including Eve and Mallory), while all parties keep their private keys only to themselves. When Alice sends a message to Bob, she encrypts the message using Bob's public key. Only Bob, who has the corresponding private key, will be able to decrypt Alice's message. Apart from removing the prerequisite key distribution issue,

Figure 1.10 Clifford C. Cocks (1950–), British cryptographer.

public key cryptography also greatly reduces the number of keys needed. For example, if we had a network of one million parties, with a symmetric key cryptosystem, each pair would need a separate key exchanged before communications could take place. This would amount to half a trillion keys, all of which need to be securely transmitted—this is a logistical nightmare. A public key system, on the other hand, would require only two million keys, none of which would need to be securely transmitted.

Basically, public key cryptography translates the difficulty of cracking into the system (by determining a private key from public key registries) into the difficulty of solving certain notoriously difficult mathematical problems, whose "inverse" problems are much easier to solve. For example, the RSA system, to be discussed in Chapter 9, is based on the difficulty of factoring large positive integers. The inverse problem is simply multiplying large positive integers, which has always been easy. We will learn much more about prime numbers and the associated number theory in Chapter 8, which also addresses the important practical problem of generating large prime numbers (since they are needed for many public key cryptosystems). Encryption is based on the easier inverse problem, whereas unauthorized decryption would be based on the computationally infeasible problem. Using such problems that have been well known and actively researched for a long time adds to the confidence of the security of such a system. Any of these public key systems are subject to faltering upon any new discovery of efficient algorithms for the intractable problems on which they are based. Although it has not been proved, for example, that an efficient algorithm for prime factorization cannot exist, it is the general consensus that this is the case. Other public key cryptosystems are based on a very special class of intractible problems known as NP complete problems.[*] We will introduce knapsack cryptosystems, which are based on the NP complete knapsack problem. It is interesting to point out that because of the increased importance that such problems now have due to the widespread use of cryptosystems that are based on them, the NSA strictly regulates certain areas of research relating to such problems. American scientists who make any novel discoveries in areas relating to public key cryptography need to clear them with the NSA before announcing them to the public (or publishing).

Several other public key cryptosystems are developed in Chapter 9. In addition to the confidentiality that is provided by symmetric key cryptosystems, public key cryptosystems all provide the following additional features:

[*] There are a very large number of computational problems where there is an "efficient" way to check whether a proposed answer is correct, in that it can be done in an amount of time that is bounded by a power of the input size (this is called "*in polynomial time*") but where no known algorithm has been designed to find the solution that will also work in polynomial time. A prototypical example is the prime factorization problem. It has been established that there is a plethora of such problems that are seemingly unrelated but if a polynomial time algorithm is discovered for one of them, then polynomial time algorithms can be produced for all of them! This latter class of problems is known as the *NP complete problems*, whereas problems that can be solved in polynomial time are called *P problems*. Most scientists believe that $NP \neq P$, but the conjecture remains one of the most famous unsolved problems in mathematics and computer science. For more details on the $P = NP$ problem, the interested reader is referred to the classic but authoritative reference by Garey and Johnson [GaJo-79]. Resolving this problem is one of the seven millennium problems for which the *Clay Foundation* (http://www.claymath.org/millennium/) is offering $1 million prizes.

- **Authentication**: The intended recipient of a message will be able to verify that it came from the indicated sender.
- **Nonrepudiation**: The sender of a message will not be able to deny that he or she was the sender.

These can be achieved by so-called **digital signature schemes**. Digital signatures, unlike ordinary signatures, are unique for each sender and cannot be forged.

With all of the added advantages and high security of public key cryptosystems, a natural question thus arises: Why even bother anymore with symmetric key cryptosystems? The answer is that symmetric key cryptosystems are significantly faster and more efficient than public key cryptosystems. Thus both types of cryptosystems can continue to live a productive coexistence: public key cryptosystems can be used to securely exchange private keys, after which the faster private key cryptosystems can be used.

In the mid-1980s, a new sort of public key cryptosystem was developed using a geometrically motivated (but analytically complicated) arithmetic of points with integer coordinates on certain planar curves known as *elliptic curves*. In spite of their name, these curves are not ellipses but a more diverse family of unbounded curves. The key sizes required for a given elliptic curve cryptosystem are significantly smaller than what would be required for other typically known public key cryptosystems with the same degree of security, and this fact has made elliptic curve cryptography one of the most promising and extensively studied branches of cryptography. Elliptic curve cryptography will be studied in Chapter 12.

The One-Time Pad, Perfect Secrecy

Circumstances and needs, as well as advances in technology, fuel the constant efforts to design (and attempts to crack) evermore sophisticated cryptosystems. The eminent scientist Claude Shannon* (Figure 1.11) wrote a number of seminal papers on cryptography in which he gave two important properties that cryptosystems should possess to avoid being compromised: *diffusion* and *confusion*. *Diffusion* means that changing just a single character in the plaintext should diffuse (spread out) to affect changes in several ciphertext letters (the more the better). *Confusion* means

* Claude Shannon grew up in Michigan. He earned a bachelor's degree with a double major in mathematics and electrical engineering from the University of Michigan–Ann Arbor. His landmark discovery of an effective symbolism for electric circuits actually came from his master's thesis at MIT: *A Symbolic Analysis of Relay and Switching Circuits*. This thesis has had a tremendous impact on industry by changing circuit design from an art to a science. Shannon went on to earn a doctorate at MIT and continued to make valuable contributions to the electronics and communications fields during his career working at Bell Labs, where his laboratory office ceiling was adorned with a rainbow of gowns from honorary doctorates that he had received. He developed a secure cryptosystem that was used by Roosevelt and Churchill for transoceanic communications during World War II. His work in this area motivated the development of the field of coding theory, for which he is considered the founder. Coding theory studies what are called error-correcting codes, which are used in everything from CDs to routine data transmissions. We have Shannon to thank, for example, when a scratched music CD will still play perfectly well.

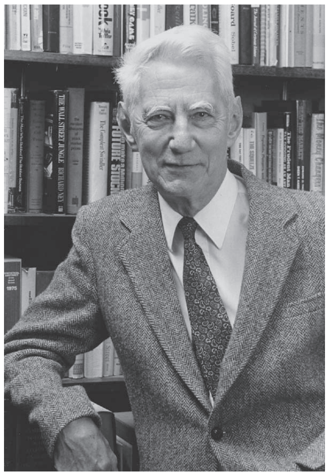


Figure 1.11 Claude E. Shannon (1916–2001), American applied mathematician.

that there should be no simple relationship between a cryptosystem's key and instances of its ciphertext. For example, any substitution cipher does not exhibit diffusion since changes in a single plaintext letter will affect only the corresponding ciphertext letter. Block ciphers are conceived to have good diffusion.

Another important contribution of Shannon was the concept of *perfect secrecy* that he introduced in 1949. This concept rigorously defines what it means for a cryptosystem to be "unbreakable," in the sense that seeing the ciphertext of any plaintext message (in a ciphertext-only attack) gives the hacker absolutely no information about the plaintext. There is actually a rather simple cryptosystem that exhibits perfect secrecy: the Vigenère cipher with a randomly generated key that is the same length as the plaintext; it is called a **one-time pad**. This cryptosystem is sometimes also called the **Vernam cipher**, after its inventor, Gilbert S. Vernam, a cryptographer with AT&T. It is not very practical to use because of the large keys, and the fact that once a key is used it must be thrown out. Although it had been conjectured for several decades that the one-time pad was perfectly secure, Shannon was the first to provide a rigorous proof. One-time pads have since been used for some of the most sensitive communication purposes; for example, Figure 1.12 shows a one-time pad system at the U.S. end of the Moscow–Washington hotline, in use during the Cold War era.

The next example shows how the one-time pad works.

Example 1.7: The One-Time Pad

The concept of a one-time pad involves *randomness*. By its very nature, any random process is unpredictable and this will be the key element that results in the system's being perfectly secure. There are 26 different shift operators, corresponding to the keys $\kappa = 0, 1, 2, 3, \cdots, 25$. The key for a one-time pad needs to

Figure 1.12 Photograph of the one-time pad machines (black) in use by the U.S. Signal Corps to support the Washington–Moscow hotline. The white machines were used to print and read plaintext messages. Photograph courtesy of the United States National Archives.

consist of a sequence of shift keys that are randomly selected from the list of 26 possible keys. Each key corresponds to how many letters down the alphabet the plaintext letter *a* (and hence all plaintext letters) gets shifted, see Table 1.3.

Suppose that we need to send a message that contains N characters. The one-time pad would require a key of length at least N. To produce the key, imagine that we label 26 identical balls with the possible key numbers 0–25 and place them in an urn; see Figure 1.13.

We shuffle the balls, randomly draw one ball, record its number, then replace it in the urn and reshuffle. We repeat this process N times to produce the one-time pad key. Although it seems contradictory, computer algorithms (which are programmed to follow a fixed set of instructions) have been designed to produce so-called *pseudorandom numbers*, which, for all practical purposes, can be assumed random.* The computer implementations given at the end of this chapter provide some schemes for producing such random numbers. For example, suppose that we needed to create a one-time pad cipher with keylength

* Of course, any computer algorithm runs on a specified set of instructions, so technically such a program cannot produce truly random numbers. Nonetheless, effective algorithms can be created that produce streams of numbers that satisfy all of the important statistical tests for randomness. Moreover, the programs can call on the computer clock to produce the “seed” of the generator so the algorithm will produce different streams at each call. For more details on such pseudorandom number generator algorithms, we refer the reader to Chapter 2 of [LePa-06] or Chapter 3 of [Knu-98].

TABLE 1.3 Key Values and Letters

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Note: The correspondence of key values κ (bottom row) and the ciphertext letter (top row) to which the plaintext letter "a" gets shifted to with a shift cipher. The value $\kappa = 0$ is not allowed as a key since it corresponds to the identity shift (that is, ciphertext letters would be identical to plaintext letters).

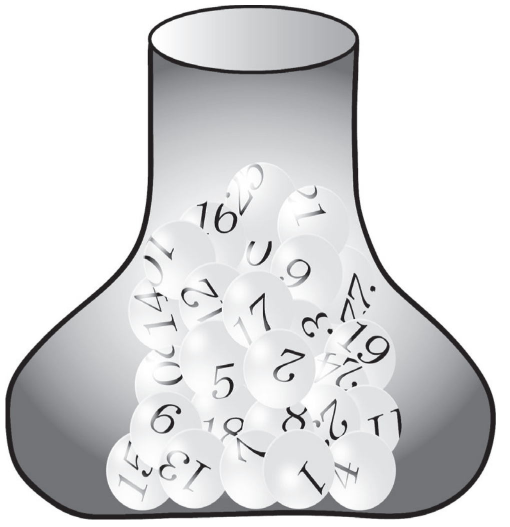


Figure 1.13 An urn containing 25 balls of identical size, weight, and texture can be used for the purpose of random number generation.

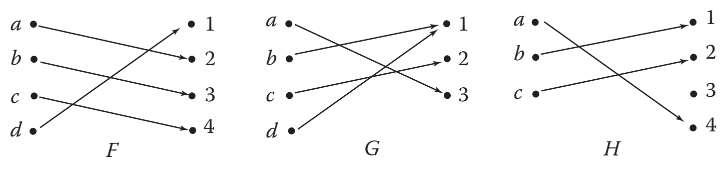
$N = 15$. Resorting to a random number generator, we obtained the following sequence that we will use as the key for the one-time pad:

$$\kappa = [21\ 23\ 4\ 23\ 16\ 3\ 7\ 14\ 24\ 25\ 4\ 25\ 9\ 13\ 21]$$

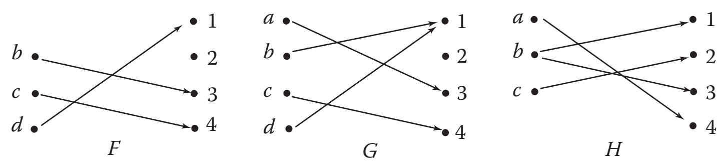
By consulting Table 1.3, we see that the resulting one-time pad will simply be the Vigenère cipher with keyword: vxexqdhoyzezjnv. Notice that we used lowercase letters although Table 1.3 had uppercase letters (Why?).

Chapter 1 Exercises

1. For the three diagrams shown below, indicate which specify functions. For each function, identify its domain, codomain, and range, and determine whether it is (a) one-to-one, (b) onto, or (c) bijective.



2. For these three diagrams, indicate which specify functions. For each function, identify its domain, codomain, and range, and determine whether it is (a) one-to-one, (b) onto, or (c) bijective.



3. Consider the function *f* : {a, b, c, y} $\to$ {length 2 binary strings} defined by *f*(a) = 00, *f*(b) = 01, *f*(c) = 10, *f*(y) = 11.
   (a) Is *f* one-to-one?
   (b) Is *f* onto?
   (c) Determine the binary string *f*(a) *f*(b) *f*(b) *f*(y).
   (d) Suppose that the binary string 1000010111 was produced by concatenating the outputs of *f* under a corresponding string of input values. Determine the input string.

4. Consider the function G : {a, b, e, f, l, t, y} $\to$ {length 3 binary strings} defined by G(a) = 000, G(b) = 001, G(e) = 010, G(f) = 011, G(l) = 100, G(t) = 101, G(y) = 110.
   (a) Is G one-to-one?
   (b) Is G onto?
   (c) Determine the binary string G(b) G(e) G(l) G(t).
   (d) Suppose that the binary string 100000101010 was produced by concatenating the outputs of G under a corresponding string of input values. Determine the input string.

5. (a) Suppose that $f : A \to B$ is a function, where A and B are finite sets, and that A has more elements than B. Does f necessarily have to be onto? Can f ever be one-to-one? Explain.
   (b) Suppose that $f : A \to B$ is a function, where A and B are finite sets, and that B has more elements than A. Does f necessarily have to be one-to-one? Can f ever be onto? Explain.

6. (a) Suppose that $f : A \to B$ is a one-to-one function, where A and B are finite sets, each containing the same number of elements. Explain why *f* is necessarily bijection.
   (b) Suppose that $f : A \to B$ is an onto function, where A and B are finite sets, each containing the same number of elements. Explain why *f* is necessarily bijection.

7. Provide an example of a function from the positive integers $\{1, 2, 3, \cdots\}$ to $\{1, 2, 3, \cdots\}$ that is:
   (a) Neither one-to-one nor onto.
   (b) One-to-one, but not onto.
   (c) Onto, but not one-to-one.
   (d) A bijection *f* such that $f(a) \neq a$, for each positive integer a.

8. Provide an example of a function from the set {finite length binary strings} to the set {finite length binary strings} that is:
   (a) Neither one-to-one nor onto.
   (b) One-to-one, but not onto.
   (c) Onto, but not one-to-one.
   (d) A bijection *f* such that $f(\sigma) \neq \sigma$, for finite length binary string σ.

9. Consider the suffix function G: {finite length binary strings} $\to$ {finite length binary strings} defined by $G(\sigma) = \sigma \cdot 1$; i.e., $G(\sigma)$ is the concatenation of σ with the length 1 string "1." For example, $G(1010) = 10101$. (In other words, G tacks a suffix "1" onto every string.)
   (a) Is G one-to-one?
   (b) Is G onto?
   (c) In case G is a bijection, determine the inverse function.

10. Consider the reversal function H: {finite length binary strings} $\to$ {finite length binary strings} defined by $H(b_1b_2\cdots b_{n-1}b_n) = b_nb_{n-1}\cdots b_2b_1$; i.e., the output of any binary string (under H) is the string of the same length, but with the bits given in the opposite order. For example, $H(1010) = 0101$.
   (a) Is H one-to-one?
   (b) Is H onto?
   (c) In case H is a bijection, determine the inverse function.

11. Consider the function f: {length 8 binary strings} $\to$ {length 8 binary strings} defined by $f(b_1b_2b_3b_4b_5b_6b_7b_8) = b_2b_4b_6b_8b_1b_3b_5b^*$, where $b^* = 1$ if $b_6 + b_7 + b_8$ is an even number; otherwise, $b^* = 0$. For example, $f(11110000) = 11001101$.
   (a) Is f one-to-one?
   (b) Is f onto?
   (c) In case f is a bijection, determine the inverse function.

12. Consider the function g: {length 4 binary strings} $\to$ {length 4 binary strings} defined by $g(b_1b_2b_3b_4) = c_1c_2b_1b_4$, where $c_1 = 1$ if $b_1 + b_2$ is an even number; otherwise, $c_1 = 0$, and while $c_2 = 1$ if $b_2 + b_4$ is an even number; otherwise, $c_2 = 0$. For example, $g(1111) = 0011$.
   (a) Is g one-to-one?
   (b) Is g onto?
   (c) In case g is a bijection, determine the inverse function.

13. (a) Use the Caesar cipher to encrypt the following strings of plaintext:
      (i) the shipment will arrive at noon
      (ii) lay low until friday
      (iii) always use the back door
      (iv) the phone is bugged
   (b) Decrypt each of the following ciphertexts that came from the Caesar cipher:
      (i) EULQJWKHLWHPWRMHQNLQV
      (ii) VHQGDJHQWSRONDVLJQDO
      (iii) LQWHUFHSWWKHLUFDVHZRUNHU
      (iv) FKHFNLQWRWKHKRWHO

14. (a) Use the Caesar cipher to encrypt the following strings of plaintext:
      (i) two minutes until alarm sounds
      (ii) spread out your team
      (iii) reconnaissance is on schedule
      (iv) this hotel is safe
   (b) Decrypt each of the following ciphertexts that came from the Caesar cipher:

(i) OHDYHPRQHBLQVZLVVDFFRXQW
(ii) VWDOOWKHPIRUWKUHHKRXV
(iii) GRQWOHDYHZLWKRXWDJHQWGXFKRYVNL
(iv) ERRNDIOLJKWWRSUDJXHXQGHUDOLDV

15. (a) Use the shift cipher with key $\kappa = 22$ (i.e., encryption is accomplished by shifting 22 letters down the alphabet) to encrypt each of the strings of plaintext of Exercise 13, part (a).
(b) Decrypt each of the following ciphertexts that came from the shift cipher with key $\kappa = 18$:
(i) OSALAFYXGJAFKLJMULAGFK
(ii) KMTBWULZSKTGSJVWVHDSFW
(iii) ESCWAFALASDUGFLSULSKSTMKAFWKKESF
(iv) GHWJSLAGFZSKTWWFUGEHJGEARWV

16. (a) Use the shift cipher with key $\kappa = 6$ (i.e., encryption is accomplished by shifting six letters down the alphabet) to encrypt each of the strings of plaintext of Exercise 14, part (a).
(b) Decrypt each of the following ciphertexts that came from the shift cipher with key $\kappa = 1$.
(i) SFUVSOUPGJFMEPGGJDFOPX
(ii) BTTFNCMFZPVSTUSJLFUJNFCZNJEOJHIU
(iii) TFOEGPSBEEJUJPOBMBHFOUT
(iv) JOGPSNBMMMPDBMDBTFXPSLFSTPGUIFQMBO

17. (a) Use the Vigenère cipher with key `rocket` to encrypt each of the strings of plaintext of Exercise 13, part (a).
(b) Decrypt each of the following ciphertexts that came from the Vigenère cipher with key `bluefog`:
(i) `ILLVJZRXTFPGSCBTNMSULPCSSZ`
(ii) `USYQJHZJYAANHNXLNWTBOTLMIYIV`
(iii) `DZGIFZUOPVYYPXJYACTIXQTYGJ`
(iv) `SPHXFFUPXCRYVKIZNIQAGSTARTBOOEBIK WLUSUVWCTETMIRSTU`

18. (a) Use the Vigenère cipher with key `mole` to encrypt each of the strings of plaintext of Exercise 14, part (a).
(b) Decrypt each of the following ciphertexts that came from the Vigenère cipher with key `timbucktu`:
(i) `VWPFABXYG`
(ii) `TTXTSCMYFAMSYEIUGLDFUNRNHZGO`
(iii) `UZUOAIHOKVUHBDOCLQAOAYZAEME`
(iv) `RWGSUVBULXMTMZHLMQEXUSMCGORPLIHO`

19. (a) Use the Playfair cipher with key `diskjockey` to encrypt each of the strings of plaintext of Exercise 13, part (a).
(b) Decrypt each of the following ciphertexts that came from the Playfair cipher of part (a):
(i) `RBIABDIGTPSZ`
(ii) `QMBGDTYASKCZXKPKCIDUICTPYBQM`
(iii) `REBSLUMNGYXYNBLFCR`
(iv) `QTBPCPSCDZLXYBQTDMYIKDTKUFGEQD SIYEITBQGYGDGAKW`

20. (a) Use the Playfair cipher with key `crimson` to encrypt each of the strings of plaintext of Exercise 14, part (a).
    (b) Decrypt each of the following ciphertexts that came from the Playfair cipher of part (a):
    (i) `KFMCVFNIRAQGCFASOIEFQY`
    (ii) `EFFLDINGKOMCQBORGV`
    (iii) `YTFCGCIDIOCHINRAYTFCKCPMAVBC`
    (iv) `OHXNCFNERDRQFCCDBPKFIOYTKOIN PCAVNELBQW`

21. Explain how a known plaintext attack on the Vigenère cipher would work. How much plaintext would be required for the attack to work?

22. (a) Explain how a chosen plaintext attack on the Vigenère cipher would work. How much plaintext would be required for the attack to work?
    (b) Explain how a chosen ciphertext attack on the Vigenère cipher would work. How much ciphertext would be required for the attack to work?

ADFGVX Cipher

A cipher that is similar to the Playfair cipher, known as the **ADFGVX cipher**, was used by the Germans during the First World War. The ciphertexts involve only these six letters, which were chosen because of their easy distinctions in Morse code (which through telegraphs and radio was the primary means of military communications). We explain how this cipher works through a specific example. First, the method begins by randomly arranging the 26 letters of the alphabet along with the 10 digits into a 6×6 array with the rows and columns labeled with the letters ADFGVX. Table 1.4 shows such a table.

TABLE 1.4 ADFGVX Table

	A	**D**	**F**	**G**	**V**	**X**
A	8	p	3	d	1	n
D	1	t	4	o	a	h
F	7	k	b	c	5	z
G	j	u	6	w	g	m
V	x	s	v	i	r	2
X	9	e	y	0	f	q

Encryption: Suppose that we are given a plaintext, such as “Ambush at the Rhein.”

Step 1. Replace each plaintext letter with the pair of letters in the ADFGVX table (Table 1.4) that label the plaintext letter’s row and column. So *a* is replaced by *DV*, *t* by *DD*, and so on.

```
plaintext:  a  m  b  u  s  h  a  t  t  h  e  r  h  e  i  n
Step 1:     DV GX FF GD VD DX DV DD DD DX XD VV DX XD VG AX
```

At this point, we have a substitution cipher, which at the time of the First World War would have certainly been long-outdated technology and easily hacked. The second and final step makes the plaintext much more difficult to hack.

Step 2. This part, which depends on a keyword, will permute the output string of Step 1. In this example, we use the keyword MAGIC. We create a new table with columns labeled by the keyword, and fill in the cells below it in reading order, row, by row. After this is done, we rearrange the columns of this table, so the keyword letters are in alphabetical order. The ciphertext is obtained by taking the letters of each column, from top to bottom, and taking the alphabetized columns in order.

M	**A**	**G**	**I**	**C**
D	V	G	X	F
F	G	D	V	D
D	X	D	V	D
D	D	D	D	X
X	D	V	V	D
X	X	D	V	G
`A	X			

A	**C**	**G**	**I**	**M**
V	F	G	X	D
G	D	D	V	F
X	D	D	V	D
D	X	D	D	D
D	D	V	V	X
X	G	D	V	X
X				A

Reading down the columns of the second (column permuted table) gives us the ciphertext:

Ciphertext: `VGXDDXXFDDXDGGDDDVDXVVDVVDFDDXXA`

Decryption is performed by reversing the encryption process. Note that in addition to the keyword, the ADFGVX table (Table 1.4) is also part of the key, since it depends on how the letters and digits were randomly deployed in the 36 cells.

Historical Aside: By the time of the First World War, the French had assembled a very strong cryptography team, after having suffered an embarrassing defeat where they had lost the provinces of Alsace and Lorraine in the Franco-Prussian War of 1870. This defeat would most probably have been avoided if the French had better intelligence. Soon after the Germans began confidently using the ADFGVX cipher in 1918, as they were making plans to take over Paris, the French put their most prized cryptographer, Lieutenant Georges Painvin (Figure 1.14), to work on decrypting this new cipher. Painvin

Figure 1.14 Georges Painvin (1886–1980), French cryptographer.

worked day and night to crack it and was able to succeed with three months of hard work. His efforts were so consuming, though, that they affected his health; he lost 30 pounds in the process. Readers interested in learning more details about Painvin's ingenious attack may refer to [Kah-96].

23. (a) Use the ADFGVX cipher with key `PARIS` to encrypt each of the strings of plaintext of Exercise 13, part (a).
    (b) Decrypt each of the following ciphertexts that came from the ADFGVX cipher of part (a):
      (i) `VVVDXDVDDXVDDD`
      (ii) `XXDDGADAXVVXGGXVXXGVXGXGVGGD`
      `DXDAGDGDDADAXAGAVAFVXVGVDXGDXA`
      (iii) `DVDGVGDGDFDDVDFVVXGVGDVDGDX`
      `VGDXVDDGDVD`
      (iv) `XFDDDDAXDDGXXDVVVFDDADXXDGD`
      `VADVDVXAVAAXXDGFDXDAGAFDGD`
      `DDDVGDFDG`

24. (a) Use the ADFGVX cipher with key `CRIMSON` to encrypt each of the strings of plaintext of Exercise 14, part (a).
    (b) Decrypt each of the following ciphertexts that came from the ADFGVX cipher of part (a):
      (i) `DVDDAAXDVGFGDDDXGFXFVVADVXVGFA`
      (ii) `VVADXDDGXGDDDDVGDADDXXVDGX`
      `VGVDXVXXGXXVVXVDVGGGXDVDDA`
      (iii) `DXGDXVDDVVXVGXXGVVGXFDFXGDGD`
      `FVDDFDDDAFAGXXGFVD`
      (iv) `DFAVVXDADVDDDDVVDGXXXDDVD`
      `DXXDDXVDVADDDDDDGDGXDGXDAXVD`
      `DDVDAXADDVDXDAD`

25. Do identical adjacent pairs of plaintext typically encode to the same four-letter ciphertext strings under the ADFGVX cipher? Explain your answer.

26. Are there any problems with procedure and/or loss of security with the ADFGVX cipher if one were to use a keyword with duplicated letters (such as LONDON)? Explain your answer.

27. (a) Do we gain any new ciphers by allowing the shift ciphers to shift to the left (rather than just to the right)? Explain your answer.
    (b) Do we gain any new ciphers by allowing the shift ciphers to shift more than 25 letters to the right? Explain.

28. Suppose that we construct a cryptosystem consisting of a Vigenère cipher, followed by another Vigenère cipher, where the keywords of each have the same length. Explain how much additional security, if any, such a system would provide over a single Vigenère cipher.

29. Suppose that $n > m$ are positive integers. Discuss the differences in security of the following two cryptosystems:
    (i) Use a Vigenère cipher with a keyword of length nm.
    (ii) Use a Vigenère cipher with a keyword of length n followed by another Vigenère cipher of keyword length m.

30. (a) List all binary strings of length 0, 1, 2, and 3.
    (b) Use the multiplication principle to compute the number of binary strings of length n, where n is any positive integer.
    (c) Letting B_n denote the binary strings of length n, explain why every string in B_{n+1} can be uniquely expressed as either $0 \cdot \sigma$ or $1 \cdot \sigma$ for some length-n binary string σ.
    (d) Use the result of part (c) to give another proof of the result of part (b) using mathematical induction.

31. Discuss the secrecy of a substitution cipher that is used to send a plaintext message that consists of just a single letter.

Chapter 1 Computer Implementations and Exercises

Note: Some of the exercises below ask the reader to write programs that may not be feasible on some computing platforms or that require knowledge of certain sorts of data structures that will not be essential in later developments in this book. For example, most of the cryptosystems that we will develop after this chapter are designed to work directly on either strings or ordered lists (vectors) of numbers. The numbers will most often be integers or binary numbers (zeros and ones). Later, we will essentially assume that plaintexts will be presented in this form. In cases where the programming for particular exercises is not feasible or not important for a particular platform or use, such an exercise may be suitably improvised or even skipped without any loss of continuity.

Vector/String Conversions

Oftentimes in computer implementations of cryptosystems, it is more convenient to work with **vectors** rather than strings. A vector is simply an ordered list. This will be the case, for example, in our development of DES in Chapter 8. On the other hand, it is often more aesthetic to display binary strings rather than binary vectors. For example, the binary vector corresponding to the binary string `101100011101` might display (depending on your particular computing platform) as

```
[1 0 1 1 0 0 0 1 1 1 0 1]
```

or as

```
[1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1]
```

Vectors are more versatile data structures than strings, since elements could be digits or any numbers. For example, the vector [32, 5] could not be so unambiguously represented as a string (325 would not do). The first two exercises below ask you to create conversion programs to pass between strings and vectors. If you need to work with strings of digits (such as binary strings), you need to know the syntax by which to enter

TABLE 1.3 Key Values and Letters

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Note: The correspondence of key values κ (bottom row) and the ciphtertext letter (top row) to which the plaintext letter a gets shifted to with a shift cipher. The value $\kappa = 0$ is not allowed as a key since it corresponds to the identity shift (that is, ciphertext letters would be identical to plaintext letters).

them into your computing platform. For example, the number 101 is a different data structure than the binary string 101.

1. *Program for Converting Strings of Digits to Vectors of Digits.* Write a program `Vec = String2Vec(Str)` that inputs a string `Str` of digits (binary or decimal) and outputs the corresponding vector `Vec`. Thus, for example, the command `String2Vec (24821)` should produce the output `[2 4 8 2 1]`. Run your program with the following inputs, and record the outputs:
   (a) 110101111
   (b) 22953688
   (c) 9876543210

2. *Program for Converting Vectors of Digits to Strings of Digits.* Write a program `Str = Vec2String(Vec)` that inputs a vector `Vec` of digits (binary or decimal) and outputs the corresponding string `Str`. Thus, for example, the command `Vec2String([2 4 8 2 1])` should produce the output `24821`. Run your program with the following inputs, and record the outputs:
   (a) `[1 0 1]`
   (b) `[1 0 1 1 0 0 0 1 1 1 0 1]`
   (c) `[9 8 7 6 5 4 3 2 1 0]`

Integer/Text Conversions

The next four exercises ask you to develop programs that will make conversions between the integer/text correspondence of Table 1.3.

Since vectors tend to be easier to work with than strings, it is probably best (depending on your particular platform) to have programs work internally with vectors but accept inputs and/or display outputs as strings. In order to achieve conversions relating to Table 1.3, it is most obvious to first think of employing a simple lookup type code:

(using a FOR loop to go through each inputted symbol `Let`, and then)

```
IF Let == A
      SET Code = 0
ELSE IF Let ==B
      SET Code = 1
ELSE IF Let == C
      SET Code = 2
(…etc…)
```

Although this would certainly work, it would be more efficient to make use of any built-in text conversion programs that your platform may have available. Most platforms have a conversion program that converts any of the 256 standard ASCII symbols (including upper- and lowercase letters, punctuation marks, and so forth) into its unique representative as an integer from 0 to 255. The 26 uppercase/lowercase letters should be mapped to contiguous blocks of integers, so you would simply need to find out where A (or a) gets mapped in order to create a very simple program. For example, if A gets mapped to 65 (so Z would get mapped to 90), you could simply take the output of this built-in mapping function and subtract 65 to arrive at the letter-to-integer conversion of Table 1.3.

In case your platform's program can convert a whole string at once (into a vector of integers), your program could be accomplished in a single line of code.

3. *Program for Converting Uppercase Text to Integers*. Write a program `Vec = UCText2Int(STR)` that inputs a string `STR` of uppercase English letters, and outputs the corresponding vector `Vec` of integers as per Table 1.3. Thus, for example, the command `UCText2Int(CATBIRD)` should produce the output [2 0 19 1 8 17 3]. Run your program with the following inputs, and record the outputs:
   (a) JUSTDOIT
   (b) ROADTRIPTHISWEEKEND
   (c) HIGHSTAKESGAME

4. *Program for Converting Lowercase Text to Integers*. Write a program `Vec = LCText2Int(str)` that inputs a string `str` of lowercase English letters, and outputs the corresponding vector `Vec` of integers as per Table 1.3. Thus, for example, the command `Text2Int(catbird)` should produce the output [2 0 19 1 8 17 3]. Run your program with the following inputs, and record the outputs:
   (a) longlivetheking
   (b) letsgotoamovie
   (c) dinnerpartytonite

5. *Program for Converting Integers to Uppercase Text*. Write a program `STR = Int2UCText(Vec)` that inputs a vector `Vec` of integers in the range 0 to 26, and outputs the corresponding string `STR` of uppercase English letters, as per Table 1.3. This is simply the inverse function of the function of Computer Exercise 3. First run this program on the outputs for `UCText2Int` when applied to the inputs of parts (a), (b), and (c) of Computer Exercise 3 to check that your new function is really the inverse of `UCText2Int`. Next, run your program with the following inputs, and record the outputs:
   (a) [2 7 0 12 15 0 6 13 4]
   (b) [5 8 11 4 19 12 8 6 13 14 13]
   (c) [2 7 14 2 14 11 0 19 4 12 14 20 18 18 4]

6. *Program for Converting Integers to Lowercase Text*. Write a program `str = Int2LCText(Vec)` that inputs a vector `Vec` of integers in the range 0 to 26, and outputs the corresponding string `str` of lowercase English letters, as per Table 1.3 (but with lowercase letters). This is simply the inverse function of the program `LCText2Int` of Computer Exercise 4. What happens if you apply this program to the output of the program `UCText2Int` of Computer Exercise 5 to a string of uppercase letters? Check your conclusion by the evaluation of `Int2LC Text(UCText2Int(CATBIRD))`. Run your program with the following inputs, and record the outputs:
   (a) [15 8 2 10 20 15 19 7 4 15 8 4 2 4 18]
   (b) [0 1 14 17 19 19 7 4 12 8 18 18 8 14 13 13 14 22]
   (c) [15 17 14 2 4 4 3 22 8 19 7 2 14 13 19 8 13 6 4 13 2 24 15 11 0 13 19 22 14]

Programming Basic Ciphers with Integer Arithmetic

The programs of the preceding computer exercises should facilitate writing encryption/decryption programs for most of the basic ciphers that were introduced in this section. The basic idea to consider is that it is much simpler to work with integers rather than the letters they correspond to in Table 1.3. This simplicity will be further enhanced as we introduce new forms of arithmetic. For example, modular arithmetic of the next chapter is particularly suitable for implementing shift and related ciphers. For now, if we wanted to implement a shift cipher, say the Caesar cipher, using the integer representation of Table 1.3, we would simply add 3 (the key) to a given plaintext representative, as long as the result is less than 26. For example, the plaintext letter f is represented by 5 (in Table 1.3), adding 3 gives 8, the corresponding representative for the ciphertext letter I (in Table 1.3). In case adding 3 gives an integer greater than 25, we would subtract 26 from the result, as this would have the same effect as cycling back to the beginning of the alphabet. For example, the plaintext letter y corresponds to 24, adding 3 gives 27, and since this is greater than 25, we subtract 26 to get 1, which is the representative of the corresponding ciphertext letter B.

7. *Program for Shift Cipher.* Write a program `StrOut = ShiftCrypt(str,kappa)` that inputs a string `str` of plaintext in lowercase English letters, and an integer `kappa` mod 26. The output `StrOut` should be the corresponding ciphertext (in uppercase letters) after the shift operator with key `kappa` is applied to the plaintext. Then use your program to redo the computations of Chapter Exercises 13 and 15.

   Note: In the decrypting parts, you will need to change your ciphertexts to lowercase (and choose the correct shift parameter).

   Suggestion: The programs of some of the preceding computer exercises should be useful here.

8. *Ciphertext-Only Attack on the Shift Cipher.* It is known that the following ciphertexts were encrypted using (perhaps different) shift ciphers. Decrypt these messages and determine the corresponding keys that were used.
   (a) `HXDALJAANBNAEJCRXWRBDWMNACQNWJVN SXWNB`
   (b) `BCJHJCCQNARCIKDCYJATHXDALJAJCCQN FJUMXAO`
   (c) `DWQYIDMCIFBSKGDODSFOHHVSTFCBHRSGYI BRSFHVSBOASXCBSG`
   (d) `XLIHIXEMPWSJCSYVQIXXMRKAMPPFISRXLI WXSGOTEKIAVMXXIRMRGSHI`

9. *Program for Vigenère Cipher.* Write a program `StrOut = VigenereCrypt(str,keystr)` that inputs a string `str` of plaintext in lowercase English letters and another such string `keystr` representing a key. The output `StrOut` should be the corresponding ciphertext (in uppercase letters) after the Vigenère

cipher with key `keystr` is applied to the plaintext. Then use your program to redo the computations of Chapter Exercise 17, part (a).

Note: In the decrypting parts, you will need to change your ciphertexts to lowercase (and choose the correct key).

Suggestion: The programs of some of the preceding computer exercises should be useful here. Your program should proceed character by character, using a FOR loop.

10. *Program for Decryption of Vigenère Cipher.* Write a program `strOut = VigenereDeCrypt(STR,keystr)` that inputs a string `STR` of ciphertext in uppercase English letters and another such string `keystr` representing a key. The output `strOut` should be the corresponding plaintext (in lowercase letters) before the Vigenère cipher with key `keystr` is applied to produce the ciphertext. Then use your program to redo the computations of Chapter Exercise 17, part (b).

    Suggestion: Modify your program `VigenereCrypt` by changing each individual shift to its inverse shift.

11. *Program for Playfair Cipher.* Write a program `StrOut = PlayfairCrypt(str,keystr)` that inputs a string `str` of plaintext in lowercase English letters and another such string `keystr` representing a key. The output `StrOut` should be the corresponding ciphertext (in uppercase letters) after the Playfair cipher when key `keystr` is applied to the plaintext. Then use your program to redo the computations of Chapter Exercise 19, part (a).

12. *Program for Decryption of the Playfair Cipher.* Write a program `strOut = PlayfairDeCrypt(STR,keystr)` that inputs a string `STR` of ciphertext in uppercase English letters and another lowercase string `keystr` representing a key. The output `strOut` should be the corresponding plaintext (in lowercase letters) before the Playfair cipher with key `keystr` is applied to it. Then use your program to redo the computations of Chapter Exercise 19, part (b).

Computer-Generated Random Numbers

Most computing platforms feature built-in "random number generators" that are of production quality. Recall that the text cites references that provide detailed developments of such programs, and the interested reader may wish to pursue these, but our approach will be to make the following convention.

Convention: We assume that a random number generator is available on our computing platform. We denote it by `rand`, and assume that it functions as follows: Each time `rand` is called, the output will be a pseudorandom real number (with decimals) from the interval (0,1); that is, 0 < `rand` < 1.

In the language of statistics, we say that rand is uniformly distributed in the interval (0,1). This means that each time `rand` is called to generate a random number, the probability that `rand` will lie in any subinterval of (0,1) will equal the length of that subinterval. For example, the probability that `rand` (on any given call) be less than 1/2—that is, 0 < `rand` < 1/2—is 1/2, and the probability that `rand` will be greater than 7/8—that is, 7/8 < `rand` < 1—is 1/8 [the length of the interval (7/8,1)]. This `rand` function may also have ways to reset its "seed" from its default value so that it will start off differently whenever the program is restarted; linking the seed to the computer's clock is usually a good way to accomplish this. Additional features of the `rand` function may include options that will allow it to produce ordered lists (vectors) of such random numbers,[*] and such a feature is particularly convenient for generating one-time pads. Although `rand` produces real numbers (with decimals) in the special range (0,1), we often need to generate random integers in a specified range. This can be done using the Algorithm 1.1, which is based on the following simple fact.

Fact: Since `rand` is uniformly distributed in (0,1), if N is any positive integer, then N rand will be uniformly distributed in the interval $(0,N)$.

In order to convert real numbers to integers, we use the **floor** function (built in to most computer platforms). This is a function mapping the real numbers to the integers, which operates as follows: For any real number x, floor(x) will be the greatest integer that is less than or equal to x. For example, floor(2.1) = 2 = floor(2) = floor(2.999), floor(π) = 3, and floor(–2.6) = –3.

Algorithm 1.1: Generating Random Integers Using `rand`

Given two integers $\ell < k$, the number $J = \ell + \text{floor}([k - \ell + 1] \times \texttt{rand})$ will be a random integer in the range $\ell \le J \le k$.

To help better understand this algorithm (in a very relevant situation), suppose we take $\ell = 0, k = 25$. Then since $k - \ell + 1 = 26$, the fact mentioned above tells us that $[k - \ell + 1] \times \texttt{rand} = 26 \times \texttt{rand}$ is a real number that is uniformly distributed in the interval (0, 26). When we take the floor: $\text{floor}([k - \ell + 1] \times \texttt{rand})$, the possible integers that can arise are the integers from 0 to 25 (inclusive) and since each of these integers will occur if the previous number lies in an interval of length 1 (in the total interval of length 26), it follows that each of these 26 integers has a 1/26 chance of occurring.

13. *Program for Creation of Keys for One-Time Pads.*
    (a) Write a program `key = OneTimePadKeyMaker (keylength)` that inputs a positive integer `keylength`, and outputs a vector having keylength randomly chosen integers from the range {0, 1, …, 25}. `strOut` should be the corresponding plaintext (in lowercase letters) before the Vigenère cipher with key `keystr` is applied to it. Use your program to produce a length-12 key.
    (b) Write a program having syntax `LetterStr = OneTime PadKeyMaker(keylength)` that functions like the one in part (a) except that the output will be a string (rather than a vector) of lowercase English letters that are determined by Table 1.3 (from the random integers that are generated). Use your program to produce a random key of length 12.

[*] If this feature is not available, ordered lists can easily be produced by using a FOR loop.

14. *Program for Random Integer Generator.*
    (a) Write a program `Vec = RandIntGen(ell, k, length)` that inputs three integers, the first two need only satisfy `ell < k`, and the third, `length`, is any positive integer. The output, `Vec`, is a vector with length elements consisting of randomly generated integers from the range `ell` $\le J \le$ `k`. The program should be based on Algorithm 1.1.
    (b) Use your program to produce a length-20 vector of random integers from the range $26 \le J \le 30$. Print out this vector.
    (c) Use your program to produce a length-1000 vector of binary digits (0s and 1s). Do not print this vector, but (get your computer to) count how many of the entries are 0s and write this down. Repeat this and record the new count of the zeros.

Appendix C: Solutions to All Exercises for the Reader

Chapter 1: An Overview of the Subject

EFR 1.1

(a) The function ***C*** **is not onto** since any string whose first two bits are different from 10 will not be in the range. The function ***C*** **is one-to-one**, since $C(\sigma) = C(\sigma')$ implies $1010 \cdot \sigma = 1010 \cdot \sigma'$ and by ignoring the first four bits, we get that $\sigma = \sigma'$.

(b) Suppose that $D(b_1b_2b_3) = D(b_1'b_2'b_3')$. Let $d_1d_2d_3 = D(b_1b_2b_3)$, $d_1'd_2'd_3' = D(b_1'b_2'b_3')$. Equating first bits: $d_1 = d_1'$, the definition of D tells us that $b_1 = b_1'$. Next, since $d_2 = d_2'$, the definition of D tells us that $b_1 + b_2$, $b_1' + b_2'$ are either both even, or both odd. But since we already know that $b_1 = b_1'$, this means that b_2, b_2' are either both even, or both odd. Since these bits can only be 0 or 1, this forces them to be equal, i.e., $b_2 = b_2'$. Finally, since $d_3 = d_3'$, a similar argument shows $b_3 = b_3'$. We have thus shown that $D(b_1b_2b_3) = D(b_1'b_2'b_3')$ implies $b_1b_2b_3 = b_1'b_2'b_3'$, i.e., ***D*** **is one-to-one**.

EFR 1.2

(a) YQQFFTQUOQYMZMFZAAZ

(b) Jenkins is a turncoat

EFR 1.3

(a) FWMSOOCNOYHZC

(b) Break out at midnight

EFR 1.4

(a) First we state the procedure using the Vigenère tableau (Table 1.2) and then we explain why it works.

Procedure: For each keyword letter, look in the corresponding *row* of the Vigenère tableau for the ciphertext letter A; the column letter where *A* is found will be the corresponding letter for the decryption keyword, if Vigenère encryption is used. For example, the first letter of the Vigenère keyword *money* is *m*, and we find that in the *m*-row of the Vigenère tableau, the letter *A* appears in the *o*-column. So the first letter of the Vigenère decryption keyword is *o*.

Why This Works: In the Vigenère encryption process, each letter of the keyword corresponds to a substitution shift cipher where *a* gets shifted to the keyword letter. For example, if the first keyword letter is *m*, then the corresponding shift would shift the plaintext letter *a* to the ciphertext letter *M*, and all letters are shifted 12 letters down (looking at Table 1.3 will be helpful). In order to reverse this shift, we could either shift the ciphertext letters 12 units up the alphabet, or shift them the complementary number 26 – 12 = 14 units down, corresponding to the shift where *a* goes to *O*. (Because with the latter option, applying both the original shift and the latter shift would result in a shift of 12 + (26 – 12) = 26 letters down the alphabet, which simply brings the plaintext letters back to themselves.) In summary, Vigenère decryption can be achieved by using the Vigenère encryption process on the modified keyword by taking each letter of the original keyword, and using instead the letter that is obtained by shifting *a* in the opposite direction by the same amount, or the complementary number of letters down. The Vigenère tableau is organized in such a way that these reverse shifts are readily obtained by the indicated lookup procedure.

(b) Using the procedure of part (a), the corresponding Vigenère decryption keyword would be *omnwc*.

Note: Here is an explanation in terms of shift ciphers: The Vigenère encryption keyword *money* corresponds to shifts of 12, 14, 13, 4, 24 down the alphabet (looking at Figure 1.3 of the text might be helpful), the corresponding inverse shifts would be 26 – 12, 26 – 14, 26 – 13, 26 – 4, 26 – 24 = 14, 12, 13, 22, 2, which correspond to the keyword *omnwc*.

EFR 1.5

(a) Removing the duplicated letter *a*, the modified keyword *barcelon* results in the Playfair array:

b	a	r	c	e
l	o	n	d	f
g	h	ij	k	m
p	q	s	t	u
v	w	x	y	z

Inserting x's between double letters of the plaintext, and pairing off the letters gives us:

me et ag en ty ul lo va tx th ea ub er ge re st au ra nt

Encrypting each pair according to the applicable case 1, 2, or 3 produces:

uf cu bh rf yc pf on wb sy qk br pe bc mb cb tu eq cr ds

and thus the following ciphertext:

(b) Breaking off the ciphertext into pairs (and putting it in lowercase) gives:

ma xh nv gl be rc cx si hb xs gb bc ac mr de rq rz

Using the array of part (a), and reversing each of appropriate cases 1, 2, or 3 of the Playfair encryption cipher produces the following:

he wi lx lb ec ar ry in ga si lv er br ie fc as ex

Putting the words together and removing redundant *x*'s gives the original message: "He will be carrying a silver briefcase."

Appendix D: Answers and Brief Solutions to Selected Odd-Numbered Exercises

Chapter 1

1. All three are functions.

(a) Domains of F and G are $\{a, b, c, d\}$, domain of H is $\{a, b, c\}$. Codomains of F and H are $\{1, 2, 3, 4\}$, codomain of G is $\{1, 2, 3\}$. Range of F is $\{1, 2, 3, 4\}$, range of G is $\{1, 2, 3\}$, range of H is $\{1, 2, 4\}$,

(b) F and H are one-to-one, G is not.

(c) F and G are onto, H is not.

(d) Only F is bijective (both one-to-one and onto).

3. (a) Yes

(b) Yes

(c) 00010111

(d) cabby

5. (a) *Such a function need not be onto*. For example, the set $A = \{1, 2\}$ has more elements than the set $B = \{1\}$, but the function f from A to B defined by $f(1) = 1 = f(2)$ is not onto. *Such a function can never be one-to-one*. Reason: Since one-to-one functions can never have duplicated outputs, the range must be the same size as the domain. But the range is a subset of the codomain, so for a one-to-one function, the size of the codomain must be at least as large as the domain.

(b) *Such a function need not be one-to-one*. For example, the set $A = \{1, 2\}$ has fewer elements than the set $B = \{1, 2, 3\}$, but the function f from A to B defined by $f(1) = 1 = f(2)$ is not one-to-one. *Such a function can never be onto*. Reason: The range is at most as large as the domain A, which is assumed to be smaller than the codomain B.

7. In each of the examples below, we will specify a function $f:\{1,2,3,\cdots\}\rightarrow\{1,2,3,\cdots\}$.

(a) Any constant function, such as $f(i) = 1$, for each $i \in \{1,2,3,\cdots\}$ is neither one-to-one nor onto.

(b) The right shift function $f(i) = i + 1$ for each $i \in \{1,2,3,\cdots\}$ is one-to-one but not onto since 1 is not in the range.

(c) The function defined by $f(1) = 1$ and $f(i) = i - 1$, for each $i \in \{2,3,\cdots\}$, is onto but not one-to-one since $f(1) = 1 = f(2)$.

(d) The function defined by taking each even integer to the odd integer right before it, and each odd integer to the even integer right after it, is bijective and satisfies the indicated condition (an output never equals its input). Here is a formula for this function: $f(i) = i - 1$ if i is even, and $f(i) = i + 1$ if i is odd.

9. (a) Yes
(b) No. Reason: No string ending in a zero is in the range.
(c) No

11. (a) Yes. Reason: If $f(b_1b_2b_3b_4b_5b_6b_7b_8) = f(b_1'b_2'b_3'b_4'b_5'b_6'b_7'b_8')$, this means that $b_2b_4b_6b_8b_1b_3b_5b^* = b_2'b_4'b_6'b_8'b_1'b_3'b_5'b'^*$ so equating bits gives us that $b_i = b_i'$ for all indices i except $i = 7$. But since $b^* = b'^*$, we also must have (according to the definition of f) that $b_6 + b_7 + b_8$, $b_6' + b_7' + b_8'$ are both even or both odd, and since we already know that the first and third of these three terms are the same, it follows that b_7, b_7' are both even or both odd. Since they can only be 0 or 1, they must be the same.
(b) Yes. Reason: Given any length-8 string $d_1d_2d_3d_4d_5d_6d_7d_8$, we need to find an input string $b_1b_2b_3b_4b_5b_6b_7b_8$ that will give us this output under f, i.e., satisfying $b_2b_4b_6b_8b_1b_3b_5b^* = d_1d_2d_3d_4d_5d_6d_7d_8$. From this latter equation, it follows that we must set $b_2 = d_1$, $b_4 = d_2$, $b_6 = d_3$, $b_8 = d_4, b_1 = d_5, b_3 = d_6, b_5 = d_7$, so the only bit left to specify in the input is b_7. Since b_6, b_8 are already specified as d_3, d_4, in order to have $b^* = d_8$, we will need to choose b_7 so that if $d_8 = 1$, then $b_7 + d_3 + d_4 \ (= b_6 + b_7 + b_8)$ is even, whereas if $d_8 = 0$, then $b_7 + d_3 + d_4$ is odd. In either case, notice that we must have $b_7 + d_3 + d_4 + d_8$ be odd. This can clearly be done (in only one way) as follows: if $d_3 + d_4 + d_8$ is odd, we must have $b_7 = 0$, while if $d_3 + d_4 + d_8$ is even, we must have $b_7 = 1$. Alternatively, the fact that f is onto follows from the result of Exercise 6(a) and the fact that f is one-to-one (which was proved in part(a)).
(c) The inverse function's formula was determined in part (b) in the process of showing f is onto. Here is the summary formula of the inverse function: $f^{-1}(d_1d_2d_3d_4d_5d_6d_7d_8) = d_5d_1d_6d_2d_7d_3d^{**}d_4 \ (= b_1b_2b_3b_4b_5b_6b_7b_8)$, where $d^{**} = 0$, if $d_3 + d_4 + d_8$ is odd, and otherwise $d^{**} = 1$.

13. (a) (i) WKHVKLSPHQWZLOODUULYHDWQRRQ
(ii) ODBORZXQWLOIULGDB
(iii) DOZDBVXVHWKHEDFNGRRU
(iv) WKHSKRQHLVEXJJHG

(b) (i) Bring the item to Jenkins
(ii) Send Agent Polk a signal
(iii) Intercept their case worker
(iv) Check in to the hotel

15. (a) (i) PDAODELIAJPSEHHWNNERAWPJKKJ
(ii) HWUHKSQJPEHBNEZWU
(iii) WHSWUOQOAPDAXWYGZKKN
(iv) PDALDKJAEOXQCCAZ

(b) (i) Waiting for instructions
(ii) Subject has boarded plane

(iii) Make initial contact as a businessman
(iv) Operation has been compromised

17. (a) (i) KVGCLBGAGXXPZZNKVKZJGKXGFCP
(ii) COAVSPLBVSPYIWFKC
(iii) RZYKCLLGGDLXSOEUHHFF
(iv) KVGZLHESKCFNXUGN

(b) (i) Harrell will be waiting for you
(ii) The meeting with Watson is as set up
(iii) Come alone but bring your piece
(iv) Rent a room in the Hotel Marignon in the Fifth Arrondisement

19. (a) (i) QMYIGSTGCPQZKGRLXKPKIGBRTDDT
(ii) RLAHSZDUPOMGPKKCAW
(iii) LRXYHYWDBPGYCBADIDKT
(iv) QMGVMSPCSKCZLVPGKU

(b) (i) Take cover now
(ii) The money is buried underneath
(iii) Pay off the watchman
(iv) Pretend you are a professor; once inside, copy the files

21. If we have any string of plaintext and the corresponding string of ciphertext, for each matched letters in the strings, the corresponding keyword shift letter is specified by the number of letters that the plaintext letter gets shifted down the alphabet to get the corresponding ciphertext letter. If we shift 0 letters down, the keyword letter is *a*; if we shift 1 letter down, the keyword letter is *b*, etc. (see Table 1.3). Thus, for example, if we knew that a Vigenère cipher was used to convert the plaintext "theyhavegrenades" into the ciphertext POMQRETANZWXEBAZ, since the first letter *t* goes to *P*, which is 26 – (19 – 15) = 22 letters down the alphabet, we get the first keyword letter must be *w*. (To see this, refer to Table 1.3, and use the fact that since *P* is to the left of *t*, the shift must have cycled back after passing *z*.) Similarly, since the next plaintext letter *h* goes to *O*, which is 14 – 7 = 7 letters down the alphabet, the second keyword letter must be *h*. Continuing in this fashion, the given plaintext/ciphertext correspondence produces the following keyword sequence "whiskeywhiskeywh," so it appears that the keyword is *whiskey*. For such an attack to completely determine the keyword, the known strings must be at least as long as the (unknown) keyword.

23. (a) (i) DVGXDDVVDAGXDDAGDVVVDXDDVXDGDVXDDDDADGAVGDXAXXGXVAVFDG
(ii) ADGDAVVVDDVVAFDFGXDVDDAGDXGXXGAGVG
(iii) VDDXXDDDAXDDDFGVDGVDDFFGDVGDXVAGGFVDFGDV
(iv) DAGVFGGXDXVDXDDDDFVADDAGGVXXXDGD

(b) (i) retreat
(ii) more munitions needed in Normandy
(iii) strike tomorrow at 4 am
(iv) Metz is a lost battle redeploy in Lyon

25. Each plaintext letter gives rise to two ciphertext letters. In Step 1, the plaintext letters are assigned to unique pairs of ciphertext, but into Step 2 these pairs are broken as the letters are put row by row into an array, and after mixing up the columns, the letters are processed column by column. So it is very unlikely that identical adjacent pairs of letters will give rise to the same four-letter ciphertext passages. Here is a specific example. Under the keyword PARIS, the ADFGVX ciphertexts for "abc" and "cab" are VFDGFF and GFFVFD, respectively. In the first, "ab" corresponds to VFDG, while in the second it corresponds to FVFD.

27. No. Shifting to the left by k letters is the same as shifting to the right by $26 - k$ letters.

29. Generally (i) is more secure. This makes sense since the key needed to describe (i) has length nm, while the key needed to describe (ii) has length $n + m$ (which is usually smaller than nm). For an extreme case, consider what happens when m is a factor of n (or the other way around). Then the cipher (ii) is equivalent to a Vigenère cipher with keylength n, so the additional Vigenère cipher of keylength m adds no additional security. Here is a specific example: if in (ii) $n = 4$ and $m = 2$, and the corresponding keywords are *gold* and *be*, then the cipher (ii) is just the Vigenère cipher whose keyword is *hsmh* (this is simply the ciphertext when the Vigenère cipher with keyword *be* is applied to the plaintext *gold*), which is much less secure than a Vigenère cipher with a key of length $nm = 8$. In general, the effective keylength of the cipher (ii) will be the least common multiple of n and m, so (ii) will compare better with (i), in terms of security, in cases where n and m do not share many common factors. But even when n and m have no common factors (other than 1), so that the effective single Vigenère cipher keylength in (ii) is nm, the system (i) has many more actual Vigenère cipher keys than the effective length mn keys resulting from (ii). In particular, the individual characters in (i) can be chosen randomly, but those in (ii) cannot be made to have a random pattern.

31. Such a system has perfect secrecy. With the knowledge of just one letter of ciphertext, any of the 26 possible plaintext letters is equally possible.

Chapter 2

1. (a) False
(b) True
(c) False