MODULE 1

1.8 A MODEL FOR NETWORK SECURITY

A model for much of what we will be discussing is captured, in very general terms, in Figure 1.5. A message is to be transferred from one party to another across some sort of Internet service. The two parties, who are the *principals* in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the Internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

Security aspects come into play when it is necessary or desirable to protect the information transmission from an opponent who may present a threat to confidentiality, authenticity, and so on. All the techniques for providing security have two components:

- A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of a code based on the contents of the message, which can be used to verify the identity of the sender.
- Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception.[6]

A trusted third party may be needed to achieve secure transmission. For example, a third party may be responsible for distributing the secret information

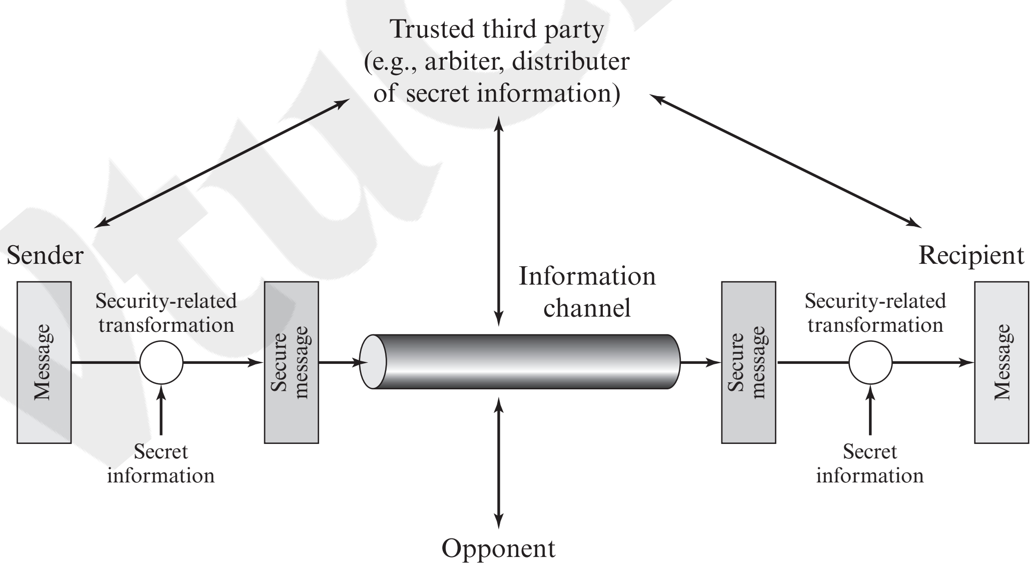


Figure 1.5 Model for Network Security

[6]Part Two discusses a form of encryption, known as a symmetric encryption, in which only one of the two principals needs to have the secret information.

to the two principals while keeping it from any opponent. Or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.

This general model shows that there are four basic tasks in designing a particular security service:

1. Design an algorithm for performing the security-related transformation. The algorithm should be such that an opponent cannot defeat its purpose.
2. Generate the secret information to be used with the algorithm.
3. Develop methods for the distribution and sharing of the secret information.
4. Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

Parts One through Five of this book concentrate on the types of security mechanisms and services that fit into the model shown in Figure 1.5. However, there are other security-related situations of interest that do not neatly fit this model but are considered in this book. A general model of these other situations is illustrated in Figure 1.6, which reflects a concern for protecting an information system from unwanted access. Most readers are familiar with the concerns caused by the existence of hackers, who attempt to penetrate systems that can be accessed over a network. The hacker can be someone who, with no malign intent, simply gets satisfaction from breaking and entering a computer system. The intruder can be a disgruntled employee who wishes to do damage or a criminal who seeks to exploit computer assets for financial gain (e.g., obtaining credit card numbers or performing illegal money transfers).

Another type of unwanted access is the placement in a computer system of logic that exploits vulnerabilities in the system and that can affect application programs as well as utility programs, such as editors and compilers. Programs can present two kinds of threats:

- **Information access threats:** Intercept or modify data on behalf of users who should not have access to that data.
- **Service threats:** Exploit service flaws in computers to inhibit use by legitimate users.

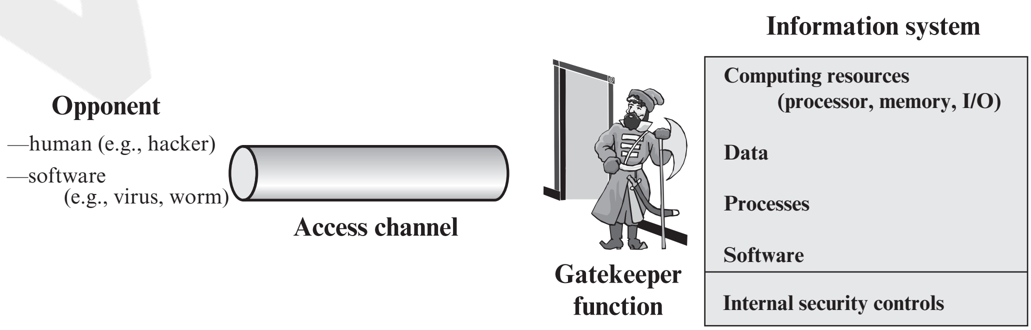


Figure 1.6 Network Access Security Model

Viruses and worms are two examples of software attacks. Such attacks can be introduced into a system by means of a disk that contains the unwanted logic concealed in otherwise useful software. They can also be inserted into a system across a network; this latter mechanism is of more concern in network security.

The security mechanisms needed to cope with unwanted access fall into two broad categories (see Figure 1.6). The first category might be termed a gatekeeper function. It includes password-based login procedures that are designed to deny access to all but authorized users and screening logic that is designed to detect and reject worms, viruses, and other similar attacks. Once either an unwanted user or unwanted software gains access, the second line of defense consists of a variety of internal controls that monitor activity and analyze stored information in an attempt to detect the presence of unwanted intruders. These issues are explored in Part Six.

3.1 SYMMETRIC CIPHER MODEL

A symmetric encryption scheme has five ingredients (Figure 3.1):

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.

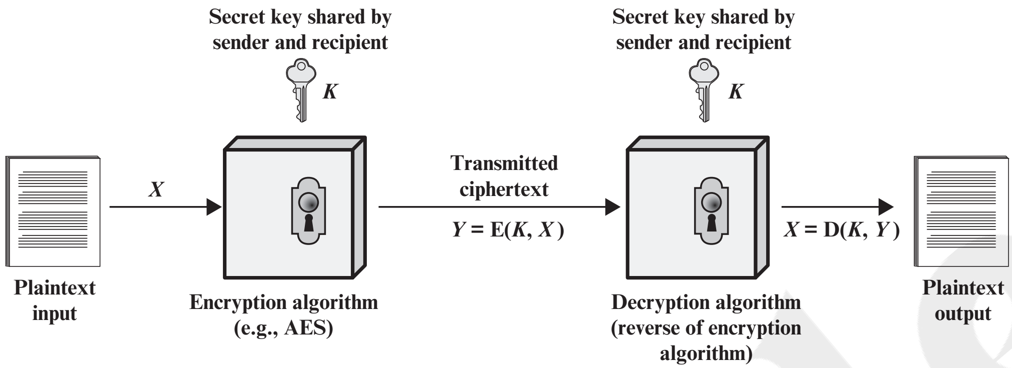


Figure 3.1 Simplified Model of Symmetric Encryption

- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the ciphertext *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 3.2. A source produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays,

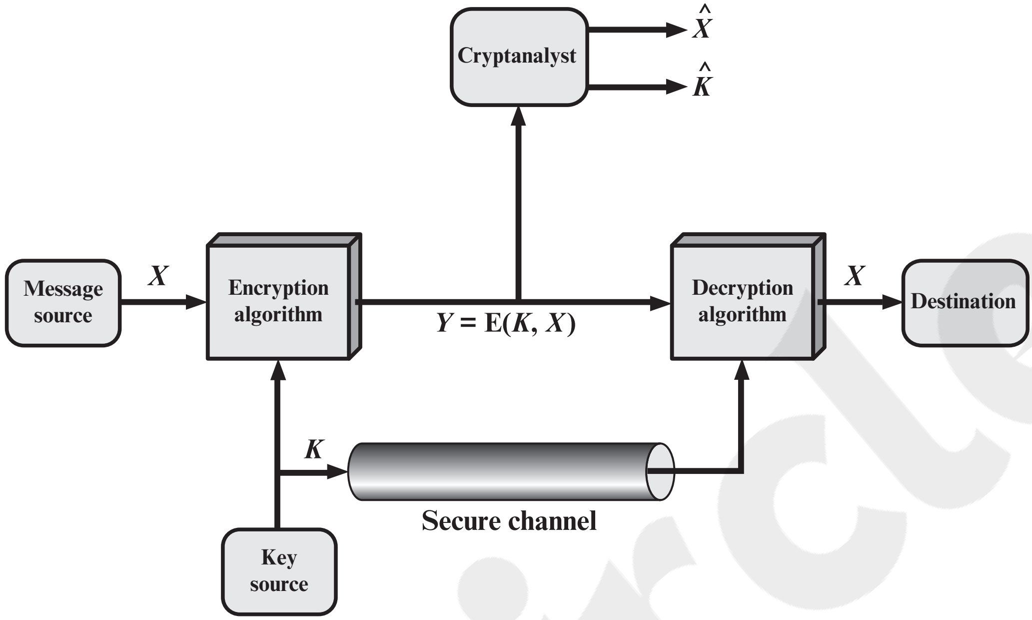


Figure 3.2 Model of Symmetric Cryptosystem

the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K_1, K_2, \ldots, K_J]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

With the message X and the encryption key K as input, the encryption algorithm forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$. We can write this as

$$Y = \mathrm{E}(K, X)$$

This notation indicates that Y is produced by using encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key K.

The intended receiver, in possession of the key, is able to invert the transformation:

$$X = \mathrm{D}(K, Y)$$

An opponent, observing Y but not having access to K or X, may attempt to recover X or K or both X and K. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover X by generating a plaintext estimate $\hat{X}$. Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover K by generating an estimate $\hat{K}$.

Cryptography

Cryptographic systems are characterized along three independent dimensions:

1. **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (i.e., that all operations are reversible). Most systems, referred to as *product systems*, involve multiple stages of substitutions and transpositions.
2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **The way in which the plaintext is processed.** A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis and Brute-Force Attack

Typically, the objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext–ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.
- **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

We first consider cryptanalysis and then discuss brute-force attacks.

Table 3.1 summarizes the various types of **cryptanalytic attacks** based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general, we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the ciphertext itself, generally applying various statistical tests to it. To use this

Table 3.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext Only	■ Encryption algorithm ■ Ciphertext
Known Plaintext	■ Encryption algorithm ■ Ciphertext ■ One or more plaintext–ciphertext pairs formed with the secret key
Chosen Plaintext	■ Encryption algorithm ■ Ciphertext ■ Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen Ciphertext	■ Encryption algorithm ■ Ciphertext ■ Ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen Text	■ Encryption algorithm ■ Ciphertext ■ Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key ■ Ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an EXE file, a Java source listing, an accounting file, and so on.

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by Corporation X might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. An example of this strategy is differential cryptanalysis, explored in Appendix S.

In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 3.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Two more definitions are worthy of note. An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext simply because the required information is not there. With the exception of a scheme known as the one-time pad (described later in this chapter), there is no encryption algorithm that is unconditionally secure. Therefore, all that the users of an encryption algorithm can strive for is an algorithm that meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

An encryption scheme is said to be **computationally secure** if either of the foregoing two criteria are met. Unfortunately, it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully.

All forms of cryptanalysis for symmetric encryption schemes are designed to exploit the fact that traces of structure or pattern in the plaintext may survive encryption and be discernible in the ciphertext. This will become clear as we examine various symmetric encryption schemes in this chapter. We will see in Part Two that cryptanalysis for public-key schemes proceeds from a fundamentally different premise, namely, that the mathematical properties of the pair of keys may make it possible for one of the two keys to be deduced from the other.

A **brute-force attack** involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. That is, if there are X different keys, on average an attacker would discover the actual key after $X/2$ tries. It is important to note that there is more to a brute-force attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed.

3.2 SUBSTITUTION TECHNIQUES

In this section and the next, we examine a sampling of what might be called classical encryption techniques. A study of these techniques enables us to illustrate the basic approaches to symmetric encryption used today and the types of cryptanalytic attacks that must be anticipated.

The two basic building blocks of all encryption techniques are substitution and transposition. We examine these in the next two sections. Finally, we discuss a system that combines both substitution and transposition.

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols.[1] If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

Caesar Cipher

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

```
plain:  meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

```
plain:  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
cipher: 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
```

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Then the algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:[2]

$$C = \mathrm{E}(3, p) = (p + 3) \bmod 26$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = \mathrm{E}(k, p) = (p + k) \bmod 26 \tag{3.1}$$

[1]When letters are involved, the following conventions are used in this book. Plaintext is always in lowercase; ciphertext is in uppercase; key values are in italicized lowercase.

[2]We define a mod n to be the remainder when a is divided by n. For example, $11 \bmod 7 = 4$. See Chapter 2 for a further discussion of modular arithmetic.

where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$$p = \mathrm{D}(k, C) = (C - k) \bmod 26 \tag{3.2}$$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: simply try all the 25 possible keys. Figure 3.3 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

Three important characteristics of this problem enabled us to use a brute-force cryptanalysis:

1. The encryption and decryption algorithms are known.
2. There are only 25 keys to try.
3. The language of the plaintext is known and easily recognizable.

In most networking situations, we can assume that the algorithms are known. What generally makes brute-force cryptanalysis impractical is the use of an algorithm that employs a large number of keys. For example, the triple DES algorithm,

	PHHW PH DIWHU WKH WRJD SDUWB
KEY	
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Figure 3.3 Brute-Force Cryptanalysis of Caesar Cipher

```
~+Wµ"— Ω-O)≤4{∞‡¸ ë~Ω%ràu·¯Í ◊¯Z-
Ú≠2Ò#Åæ∂ œ«q7¸Ωn·®3N◊Ú Œz'Y-ƒ∞Í[±Û_ èΩ,<NO¬±«˘xã  Åäfèü3Å
x}ö§kºÂ
_yÍ ^ΔÉ] ˛¤ J/˚iTê&ı 'c<uΩ-
 ÄD(G WÄC~y_ïõÄW PÔı«ÎÜ†ç],¤¡˘Ì^üÑπ˘≈˘L˘9Ogﬂ0˘&Œ≤ ¬≤ ØÔ§˝:
 ˘Œ!SGqèvo^ ú\¸S>h<-*6ø‡%x´˝|ﬁÓ#≈˜my‰˘≥ñP<,ƒi Áj  Å◊¿˝Zù-
Ω¨Õ¯6Œÿ{% „ΩÊó ¸ï π÷Áî˚úO2çSÿ´O-
2Äﬂßi /@^"∏Kºª PŒπ¸úé^´3∑˘ö˘ÔZÌ"Y¬ŸΩœY> Ω+eô/˙<Kf¿*÷˜"≤û˜
B ZøK˜Qßÿüƒ˛!Òﬂ ÎzsS/]>ÈQ ü
```

Figure 3.4 Sample of Compressed Text

examined in Chapter 7, makes use of a 168-bit key, giving a key space of 2^{168} or greater than 3.7×10^{50} possible keys.

The third characteristic is also significant. If the language of the plaintext is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recognition difficult. For example, Figure 3.4 shows a portion of a text file compressed using an algorithm called ZIP. If this file is then encrypted with a simple substitution cipher (expanded to include more than just 26 alphabetic characters), then the plaintext may not be recognized when it is uncovered in the brute-force cryptanalysis.

Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Before proceeding, we define the term *permutation*. A **permutation** of a finite set of elements S is an ordered sequence of all the elements of S, with each element appearing exactly once. For example, if $S = \{a, b, c\}$, there are six permutations of S:

abc, acb, bac, bca, cab, cba

In general, there are $n!$ permutations of a set of n elements, because the first element can be chosen in one of n ways, the second in $n - 1$ ways, the third in $n - 2$ ways, and so on.

Recall the assignment for the Caesar cipher:

```
plain:  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
cipher: 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
```

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4×10^{26} possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message.

There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language. To see how such a cryptanalysis might proceed, we give a partial example here that is adapted from one in [SINK09]. The ciphertext to be solved is

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
```

As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 3.5 (based on [LEWA00]). If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match. In any case, the relative frequencies of the letters in the ciphertext (in percentages) are as follows:

P	13.33	H	5.83	F	3.33	B	1.67	C	0.00
Z	11.67	D	5.00	W	3.33	G	1.67	K	0.00
S	8.33	E	5.00	Q	2.50	Y	1.67	L	0.00
U	8.33	V	4.17	T	2.50	I	0.83	N	0.00
O	7.50	X	4.17	A	1.67	J	0.83	R	0.00
M	6.67								

Comparing this breakdown with Figure 3.5, it seems likely that cipher letters P and Z are the equivalents of plain letters e and t, but it is not certain which is which. The letters S, U, O, M, and H are all of relatively high frequency and probably

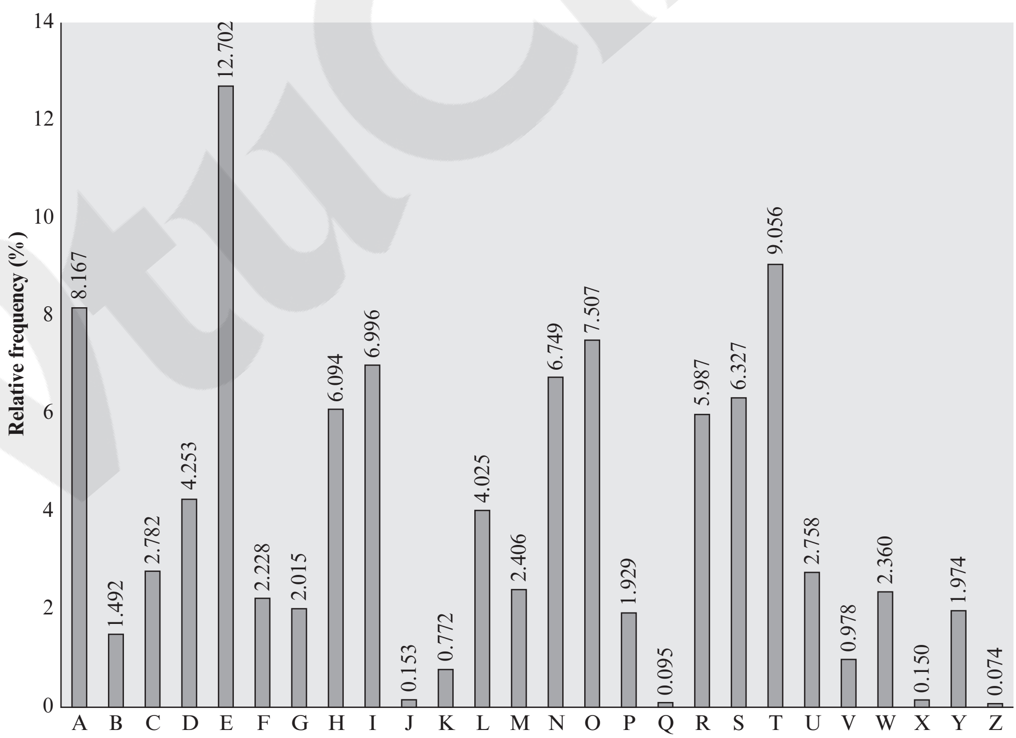


Figure 3.5 Relative Frequency of Letters in English Text

correspond to plain letters from the set {a, h, i, n, o, r, s}. The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {b, j, k, q, v, x, z}.

There are a number of ways to proceed at this point. We could make some tentative assignments and start to fill in the plaintext to see if it looks like a reasonable "skeleton" of a message. A more systematic approach is to look for other regularities. For example, certain words may be known to be in the text. Or we could look for repeating sequences of cipher letters and try to deduce their plaintext equivalents.

A powerful tool is to look at the frequency of two-letter combinations, known as **digrams**. A table similar to Figure 3.5 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our ciphertext, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the ciphertext, and we can translate that sequence as "the." This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track.

Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th_t. If so, S equates with a.

So far, then, we have

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
 t a        e  e te  a that e e a       a
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
   e t   ta t ha e ee  a e  th    t  a
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
 e  e e tat  e   the   t
```

Only four letters have been identified, but already we have quite a bit of the message. Continued analysis of frequencies plus trial and error should easily yield a solution from this point. The complete plaintext, with spaces added between words, follows:

```
it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow
```

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet. A countermeasure is to provide multiple substitutes, known as homophones, for a single letter. For example, the letter e could be assigned a number of different cipher symbols, such as 16, 74, 35, and 21, with each homophone assigned to a letter in rotation or randomly. If the number of symbols assigned to each letter is proportional to the relative frequency of that letter, then single-letter frequency information is completely obliterated. The great mathematician Carl Friedrich Gauss believed that he had devised an unbreakable cipher using homophones. However, even with homophones, each element of plaintext affects only one element of ciphertext, and multiple-letter patterns (e.g., digram frequencies) still survive in the ciphertext, making cryptanalysis relatively straightforward.

Two principal methods are used in substitution ciphers to lessen the extent to which the structure of the plaintext survives in the ciphertext: One approach is to encrypt multiple letters of plaintext, and the other is to use multiple cipher alphabets. We briefly examine each.

Playfair Cipher

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams.[3]

The Playfair algorithm is based on the use of a 5×5 matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase:*[4]

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

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.
2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.
3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.
4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are $26 \times 26 = 676$ digrams,

[3]This cipher was actually invented by British scientist Sir Charles Wheatstone in 1854, but it bears the name of his friend Baron Playfair of St. Andrews, who championed the cipher at the British foreign office.

[4]The book provides an absorbing account of a probable-word attack.

so that identification of individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable. It was used as the standard field system by the British Army in World War I and still enjoyed considerable use by the U.S. Army and other Allied forces during World War II.

Despite this level of confidence in its security, the Playfair cipher is relatively easy to break, because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient.

One way of revealing the effectiveness of the Playfair and other ciphers is shown in Figure 3.6. The line labeled *plaintext* plots a typical frequency distribution of the 26 alphabetic characters (no distinction between upper and lower case) in ordinary text. This is also the frequency distribution of any monoalphabetic substitution cipher, because the frequency values for individual letters are the same, just with different letters substituted for the original letters. The plot is developed in the following way: The number of occurrences of each letter in the text is counted and divided by the number of occurrences of the most frequently used letter. Using the results of Figure 3.5, we see that e is the most frequently used letter. As a result, e has a relative frequency of 1, t of $9.056/12.702 \approx 0.72$, and so on. The points on the horizontal axis correspond to the letters in order of decreasing frequency.

Figure 3.6 also shows the frequency distribution that results when the text is encrypted using the Playfair cipher. To normalize the plot, the number of occurrences of each letter in the ciphertext was again divided by the number of occurrences of e in the plaintext. The resulting plot therefore shows the extent to which the frequency distribution of letters, which makes it trivial to solve substitution

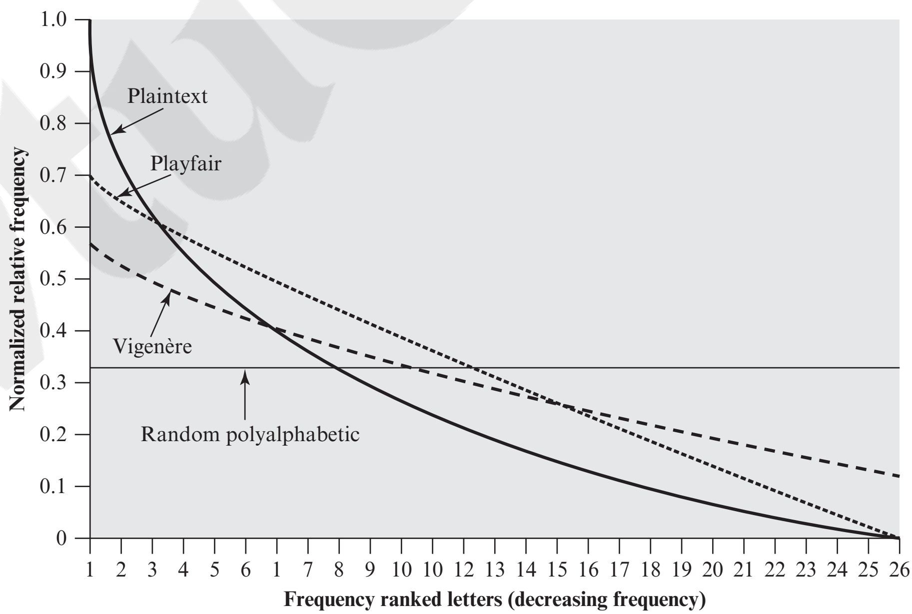


Figure 3.6 Relative Frequency of Occurrence of Letters

ciphers, is masked by encryption. If the frequency distribution information were totally concealed in the encryption process, the ciphertext plot of frequencies would be flat, and cryptanalysis using ciphertext only would be effectively impossible. As the figure shows, the Playfair cipher has a flatter distribution than does plaintext, but nevertheless, it reveals plenty of structure for a cryptanalyst to work with. The plot also shows the Vigenère cipher, discussed subsequently. The Hill and Vigenère curves on the plot are based on results reported in [SIMM93].

Hill Cipher[5]

Another interesting multiletter cipher is the Hill cipher, developed by the mathematician Lester Hill in 1929.

Concepts from Linear Algebra Before describing the Hill cipher, let us briefly review some terminology from linear algebra. In this discussion, we are concerned with matrix arithmetic modulo 26. For the reader who needs a refresher on matrix multiplication and inversion, see Appendix E.

We define the inverse $\mathbf{M}^{-1}$ of a square matrix $\mathbf{M}$ by the equation $\mathbf{M}(\mathbf{M}^{-1}) = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. $\mathbf{I}$ is a square matrix that is all zeros except for ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation. For example,

$$\mathbf{A} = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \qquad \mathbf{A}^{-1} \bmod 26 = \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}$$

$$\mathbf{AA}^{-1} = \begin{pmatrix} (5 \times 9) + (8 \times 1) & (5 \times 2) + (8 \times 15) \\ (17 \times 9) + (3 \times 1) & (17 \times 2) + (3 \times 15) \end{pmatrix}$$

$$= \begin{pmatrix} 53 & 130 \\ 156 & 79 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

To explain how the inverse of a matrix is computed, we begin with the concept of determinant. For any square matrix ($m \times m$), the **determinant** equals the sum of all the products that can be formed by taking exactly one element from each row and exactly one element from each column, with certain of the product terms preceded by a minus sign. For a 2×2 matrix,

$$\begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

the determinant is $k_{11}k_{22} - k_{12}k_{21}$. For a 3×3 matrix, the value of the determinant is $k_{11}k_{22}k_{33} + k_{21}k_{32}k_{13} + k_{31}k_{12}k_{23} - k_{31}k_{22}k_{13} - k_{21}k_{12}k_{33} - k_{11}k_{32}k_{23}$. If a square

[5]This cipher is somewhat more difficult to understand than the others in this chapter, but it illustrates an important point about cryptanalysis that will be useful later on. This subsection can be skipped on a first reading.

matrix **A** has a nonzero determinant, then the inverse of the matrix is computed as $[\mathbf{A}^{-1}]_{ij} = (\det \mathbf{A})^{-1}(-1)^{i+j}(\mathrm{D}_{ji})$, where (D_{ji}) is the subdeterminant formed by deleting the jth row and the ith column of **A**, det(**A**) is the determinant of **A**, and $(\det \mathbf{A})^{-1}$ is the multiplicative inverse of (det **A**) mod 26.

Continuing our example,

$$\det\begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} = (5 \times 3) - (8 \times 17) = -121 \bmod 26 = 9$$

We can show that $9^{-1} \bmod 26 = 3$, because $9 \times 3 = 27 \bmod 26 = 1$ (see Chapter 2 or Appendix E). Therefore, we compute the inverse of **A** as

$$\mathbf{A} = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix}$$

$$\mathbf{A}^{-1} \bmod 26 = 3\begin{pmatrix} 3 & -8 \\ -17 & 5 \end{pmatrix} = 3\begin{pmatrix} 3 & 18 \\ 9 & 5 \end{pmatrix} = \begin{pmatrix} 9 & 54 \\ 27 & 15 \end{pmatrix} = \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}$$

The Hill Algorithm This encryption algorithm takes m successive plaintext letters and substitutes for them m ciphertext letters. The substitution is determined by m linear equations in which each character is assigned a numerical value $(a = 0, b = 1, \ldots, z = 25)$. For $m = 3$, the system can be described as

$$\begin{aligned} c_1 &= (k_{11}p_1 + k_{21}p_2 + k_{31}p_3) \bmod 26 \\ c_2 &= (k_{12}p_1 + k_{22}p_2 + k_{32}p_3) \bmod 26 \\ c_3 &= (k_{13}p_1 + k_{23}p_2 + k_{33}p_3) \bmod 26 \end{aligned}$$

This can be expressed in terms of row vectors and matrices:[6]

$$(c_1\ c_2\ c_3) = (p_1\ p_2\ p_3)\begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \bmod 26$$

or

$$\mathbf{C} = \mathbf{PK} \bmod 26$$

where **C** and **P** are row vectors of length 3 representing the plaintext and ciphertext, and **K** is a 3×3 matrix representing the encryption key. Operations are performed mod 26.

[6]Some cryptography books express the plaintext and ciphertext as column vectors, so that the column vector is placed after the matrix rather than the row vector placed before the matrix. Sage uses row vectors, so we adopt that convention.

For example, consider the plaintext "paymoremoney" and use the encryption key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

The first three letters of the plaintext are represented by the vector (15 0 24). Then (15 0 24)**K** = (303 303 531) mod 26 = (17 17 11) = RRL. Continuing in this fashion, the ciphertext for the entire plaintext is RRLMWBKASPDH.

Decryption requires using the inverse of the matrix **K**. We can compute det **K** = 23, and therefore, $(\det \mathbf{K})^{-1}$ mod 26 = 17. We can then compute the inverse as[7]

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}\begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix $\mathbf{K}^{-1}$ is applied to the ciphertext, then the plaintext is recovered.

In general terms, the Hill system can be expressed as

$$\mathbf{C} = \mathrm{E}(\mathbf{K}, \mathbf{P}) = \mathbf{PK} \bmod 26$$

$$\mathbf{P} = \mathrm{D}(\mathbf{K}, \mathbf{C}) = \mathbf{CK}^{-1} \bmod 26 = \mathbf{PKK}^{-1} = \mathbf{P}$$

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger matrix hides more frequency information. Thus, a 3 × 3 Hill cipher hides not only single-letter but also two-letter frequency information.

Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack. For an $m \times m$ Hill cipher, suppose we have m plaintext–ciphertext pairs, each of length m. We label the pairs $\mathbf{P}_j = (p_{1j}p_{1j} \ldots p_{mj})$ and $\mathbf{C}_j = (c_{1j}c_{1j} \ldots c_{mj})$ such that $\mathbf{C}_j = \mathbf{P}_j\mathbf{K}$ for $1 \leq j \leq m$ and for some unknown key matrix **K**. Now define two $m \times m$ matrices $\mathbf{X} = (p_{ij})$ and $\mathbf{Y} = (c_{ij})$. Then we can form the matrix equation $\mathbf{Y} = \mathbf{XK}$. If **X** has an inverse, then we can determine $\mathbf{K} = \mathbf{X}^{-1}\mathbf{Y}$. If **X** is not invertible, then a new version of **X** can be formed with additional plaintext–ciphertext pairs until an invertible **X** is obtained.

Consider this example. Suppose that the plaintext "hillcipher" is encrypted using a 2 × 2 Hill cipher to yield the ciphertext HCRZSSXNSP. Thus, we know that (7 8)**K** mod 26 = (7 2); (11 11)**K** mod 26 = (17 25); and so on. Using the first two plaintext-ciphertext pairs, we have

[7]The calculations for this example are provided in detail in Appendix E.

$$\begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix} \mathbf{K} \bmod 26$$

The inverse of **X** can be computed:

$$\begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix}^{-1} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix}$$

so

$$\mathbf{K} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix} \begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 549 & 600 \\ 398 & 577 \end{pmatrix} \bmod 26 = \begin{pmatrix} 3 & 2 \\ 8 & 5 \end{pmatrix}$$

This result is verified by testing the remaining plaintext–ciphertext pairs.

Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is **polyalphabetic substitution cipher**. All these techniques have the following features in common:

1. A set of related monoalphabetic substitution rules is used.
2. A key determines which particular rule is chosen for a given transformation.

Vigenère Cipher The best known, and one of the simplest, polyalphabetic ciphers is the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter a. Thus, a Caesar cipher with a shift of 3 is denoted by the key value 3.[8]

We can express the Vigenère cipher in the following manner. Assume a sequence of plaintext letters $P = p_0, p_1, p_2, \ldots, p_{n-1}$ and a key consisting of the sequence of letters $K = k_0, k_1, k_2, \ldots, k_{m-1}$, where typically $m < n$. The sequence of ciphertext letters $C = C_0, C_1, C_2, \ldots, C_{n-1}$ is calculated as follows:

$$\begin{aligned} C &= C_0, C_1, C_2, \ldots, C_{n-1} = \mathrm{E}(K, P) = \mathrm{E}[(k_0, k_1, k_2, \ldots, k_{m-1}), (p_0, p_1, p_2, \ldots, p_{n-1})] \\ &= (p_0 + k_0) \bmod 26, (p_1 + k_1) \bmod 26, \ldots, (p_{m-1} + k_{m-1}) \bmod 26, \\ &\quad (p_m + k_0) \bmod 26, (p_{m+1} + k_1) \bmod 26, \ldots, (p_{2m-1} + k_{m-1}) \bmod 26, \ldots \end{aligned}$$

Thus, the first letter of the key is added to the first letter of the plaintext, mod 26, the second letters are added, and so on through the first m letters of the plaintext. For the next m letters of the plaintext, the key letters are repeated. This process

[8]To aid in understanding this scheme and also to aid in it use, a matrix known as the Vigenère tableau is often used. This tableau is discussed in a document at box.com/Crypto7e.

continues until all of the plaintext sequence is encrypted. A general equation of the encryption process is

$$C_i = (p_i + k_{i \bmod m}) \bmod 26 \tag{3.3}$$

Compare this with Equation (3.1) for the Caesar cipher. In essence, each plaintext character is encrypted with a different Caesar cipher, depending on the corresponding key character. Similarly, decryption is a generalization of Equation (3.2):

$$p_i = (C_i - k_{i \bmod m}) \bmod 26 \tag{3.4}$$

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as

```
key:          deceptivedeceptivedeceptive
plaintext:    wearediscoveredsaveyourself
ciphertext:   ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

Expressed numerically, we have the following result.

key	3	4	2	4	15	19	8	21	4	3	4	2	4	15
plaintext	22	4	0	17	4	3	8	18	2	14	21	4	17	4
ciphertext	25	8	2	21	19	22	16	13	6	17	25	6	21	19

key	19	8	21	4	3	4	2	4	15	19	8	21	4
plaintext	3	18	0	21	4	24	14	20	17	18	4	11	5
ciphertext	22	0	21	25	7	2	16	24	6	11	12	6	9

The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 3.6 shows the frequency distribution for a Vigenère cipher with a keyword of length 9. An improvement is achieved over the Playfair cipher, but considerable frequency information remains.

It is instructive to sketch a method of breaking this cipher, because the method reveals some of the mathematical principles that apply in cryptanalysis.

First, suppose that the opponent believes that the ciphertext was encrypted using either monoalphabetic substitution or a Vigenère cipher. A simple test can be made to make a determination. If a monoalphabetic substitution is used, then the statistical properties of the ciphertext should be the same as that of the language of the plaintext. Thus, referring to Figure 3.5, there should be one cipher letter with a relative frequency of occurrence of about 12.7%, one with about 9.06%, and so on. If only a single message is available for analysis, we would not expect an exact match of this small sample with the statistical profile of the plaintext language. Nevertheless, if the correspondence is close, we can assume a monoalphabetic substitution.

If, on the other hand, a Vigenère cipher is suspected, then progress depends on determining the length of the keyword, as will be seen in a moment. For now, let us concentrate on how the keyword length can be determined. The important insight that leads to a solution is the following: If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length, they will generate identical ciphertext sequences. In the foregoing example, two instances of the sequence "red" are separated by nine character positions. Consequently, in both cases, r is encrypted using key letter *e*, e is encrypted using key letter *p*, and d is encrypted using key letter *t*. Thus, in both cases, the ciphertext sequence is VTW. We indicate this above by underlining the relevant ciphertext letters and shading the relevant ciphertext numbers.

An analyst looking at only the ciphertext would detect the repeated sequences VTW at a displacement of 9 and make the assumption that the keyword is either three or nine letters in length. The appearance of VTW twice could be by chance and may not reflect identical plaintext letters encrypted with identical key letters. However, if the message is long enough, there will be a number of such repeated ciphertext sequences. By looking for common factors in the displacements of the various sequences, the analyst should be able to make a good guess of the keyword length.

Solution of the cipher now depends on an important insight. If the keyword length is m, then the cipher, in effect, consists of m monoalphabetic substitution ciphers. For example, with the keyword DECEPTIVE, the letters in positions 1, 10, 19, and so on are all encrypted with the same monoalphabetic cipher. Thus, we can use the known frequency characteristics of the plaintext language to attack each of the monoalphabetic ciphers separately.

The periodic nature of the keyword can be eliminated by using a nonrepeating keyword that is as long as the message itself. Vigenère proposed what is referred to as an **autokey system**, in which a keyword is concatenated with the plaintext itself to provide a running key. For our example,

```
key:          deceptivewearediscoveredsav
plaintext:    wearediscoveredsaveyourself
ciphertext:   ZICVTWQNGKZEIIGASXSTSLVVWLA
```

Even this scheme is vulnerable to cryptanalysis. Because the key and the plaintext share the same frequency distribution of letters, a statistical technique can be applied. For example, e enciphered by *e*, by Figure 3.5, can be expected to occur with a frequency of $(0.127)^2 \approx 0.016$, whereas t enciphered by *t* would occur only about half as often. These regularities can be exploited to achieve successful cryptanalysis.[9]

VERNAM CIPHER The ultimate defense against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT&T engineer named Gilbert Vernam in 1918.

---

[9]Although the techniques for breaking a Vigenère cipher are by no means complex, a 1917 issue of *Scientific American* characterized this system as "impossible of translation." This is a point worth remembering when similar claims are made for modern algorithms.

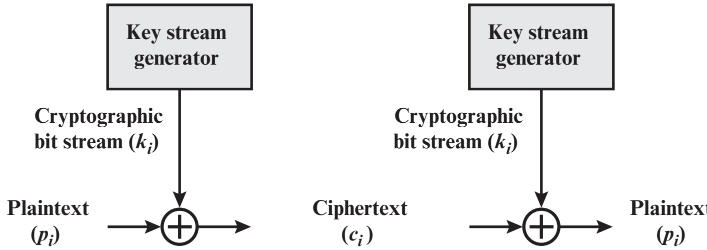


Figure 3.7 Vernam Cipher

His system works on binary data (bits) rather than letters. The system can be expressed succinctly as follows (Figure 3.7):

$$c_i = p_i \oplus k_i$$

where

p_i = ith binary digit of plaintext
k_i = ith binary digit of key
c_i = ith binary digit of ciphertext
$\oplus$ = exclusive-or (XOR) operation

Compare this with Equation (3.3) for the Vigenère cipher.

Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$$p_i = c_i \oplus k_i$$

which compares with Equation (3.4).

The essence of this technique is the means of construction of the key. Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword. Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.

One-Time Pad

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that is as long as the message, so that the key need not be repeated. In addition, the key is to be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. Such a scheme, known as a **one-time pad**, is unbreakable. It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code.

An example should illustrate our point. Suppose that we are using a Vigenère scheme with 27 characters in which the twenty-seventh character is the space character, but with a one-time key that is as long as the message. Consider the ciphertext

```
ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
```

We now show two different decryptions using two different keys:

```
ciphertext:  ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:         pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih
plaintext:   mr mustard with the candlestick in the hall

ciphertext:  ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:         pftgpmiydgaxgoufhklllmhsqdqogtewbqfgyovuhwt
plaintext:   miss scarlet with the knife in the library
```

Suppose that a cryptanalyst had managed to find these two keys. Two plausible plaintexts are produced. How is the cryptanalyst to decide which is the correct decryption (i.e., which is the correct key)? If the actual key were produced in a truly random fashion, then the cryptanalyst cannot say that one of these two keys is more likely than the other. Thus, there is no way to decide which key is correct and therefore which plaintext is correct.

In fact, given any plaintext of equal length to the ciphertext, there is a key that produces that plaintext. Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which was the intended plaintext. Therefore, the code is unbreakable.

The security of the one-time pad is entirely due to the randomness of the key. If the stream of characters that constitute the key is truly random, then the stream of characters that constitute the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext.

In theory, we need look no further for a cipher. The one-time pad offers complete security but, in practice, has two fundamental difficulties:

1. There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.
2. Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

Because of these difficulties, the one-time pad is of limited utility and is useful primarily for low-bandwidth channels requiring very high security.

The one-time pad is the only cryptosystem that exhibits what is referred to as *perfect secrecy*. This concept is explored in Appendix F.

3.5 STEGANOGRAPHY

We conclude with a discussion of a technique that (strictly speaking), is not encryption, namely, **steganography**.

A plaintext message may be hidden in one of two ways. The methods of **steganography** conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.[11]

A simple form of steganography, but one that is time-consuming to construct, is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message. For example, the sequence of first letters of each word of the overall message spells out the hidden message. Figure 3.9 shows an example in which a subset of the words of the overall message is used to convey the hidden message. See if you can decipher this; it's not too hard.

Various other techniques have been used historically; some examples are the following [MYER91]:

- **Character marking:** Selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held at an angle to bright light.
- **Invisible ink:** A number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
- **Pin punctures:** Small pin punctures on selected letters are ordinarily not visible unless the paper is held up in front of a light.
- **Typewriter correction ribbon:** Used between lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

[11] *Steganography* was an obsolete word that was revived by David Kahn and given the meaning it has today [KAHN96].

3rd March

Dear George,

Greetings to all at Oxford. Many thanks for your
letter and for the Summer examination package.
All Entry Forms and Fees Forms should be ready
for final despatch to the Syndicate by Friday
20th or at the very latest, I'm told. by the 21st.
Admin has improved here, though there's room
for improvement still; just give us all two or three
more years and we'll really show you! Please
don't let these wretched 16+ proposals destroy
your basic O and A pattern. Certainly this
sort of change, if implemented immediately,
would bring chaos.

Sincerely yours.

Figure 3.9 A Puzzle for Inspector Morse
(*From* The Silent World of Nicholas Quinn, *by Colin Dexter*)

Although these techniques may seem archaic, they have contemporary equivalents. [WAYN09] proposes hiding a message by using the least significant bits of frames on a CD. For example, the Kodak Photo CD format's maximum resolution is 3096×6144 pixels, with each pixel containing 24 bits of RGB color information. The least significant bit of each 24-bit pixel can be changed without greatly affecting the quality of the image. The result is that you can hide a 130-kB message in a single digital snapshot. There are now a number of software packages available that take this type of approach to steganography.

Steganography has a number of drawbacks when compared to encryption. It requires a lot of overhead to hide a relatively few bits of information, although using a scheme like that proposed in the preceding paragraph may make it more effective. Also, once the system is discovered, it becomes virtually worthless. This problem, too, can be overcome if the insertion method depends on some sort of key (e.g., see Problem 3.22). Alternatively, a message can be first encrypted and then hidden using steganography.

The advantage of steganography is that it can be employed by parties who have something to lose should the fact of their secret communication (not necessarily the content) be discovered. Encryption flags traffic as important or secret or may identify the sender or receiver as someone with something to hide.

4.1 TRADITIONAL BLOCK CIPHER STRUCTURE

Several important symmetric block encryption algorithms in current use are based on a structure referred to as a Feistel block cipher [FEIS73]. For that reason, it is important to examine the design principles of the Feistel cipher. We begin with a comparison of stream ciphers and block ciphers. Then we discuss the motivation for the Feistel block cipher structure. Finally, we discuss some of its implications.

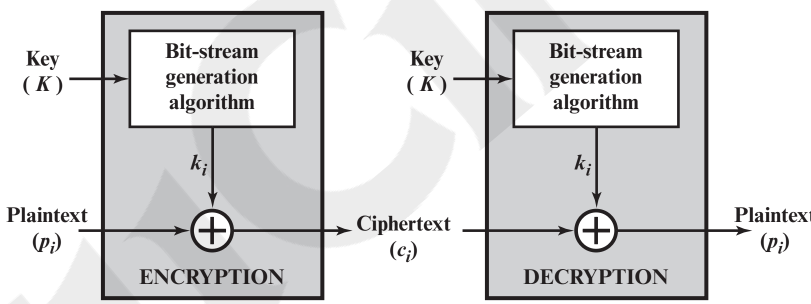
[1]However, you may safely skip Appendix G, at least on a first reading. If you get lost or bogged down in the details of DES, then you can go back and start with simplified DES.

Stream Ciphers and Block Ciphers

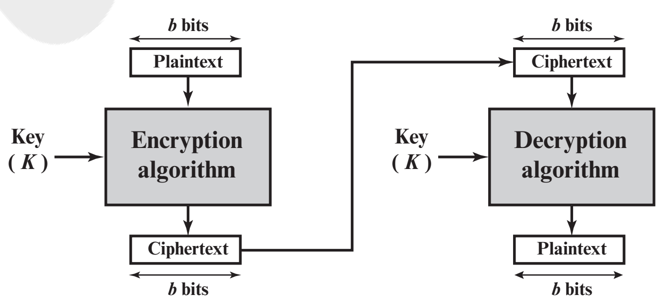
A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher. In the ideal case, a one-time pad version of the Vernam cipher would be used (Figure 3.7), in which the keystream (k_i) is as long as the plaintext bit stream (p_i). If the cryptographic keystream is random, then this cipher is unbreakable by any means other than acquiring the keystream. However, the keystream must be provided to both users in advance via some independent and secure channel. This introduces insurmountable logistical problems if the intended data traffic is very large.

Accordingly, for practical reasons, the bit-stream generator must be implemented as an algorithmic procedure, so that the cryptographic bit stream can be produced by both users. In this approach (Figure 4.1a), the bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong. That is, it must be computationally impractical to predict future portions of the bit stream based on previous portions of the bit stream. The two users need only share the generating key, and each can produce the keystream.

A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or



(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

Figure 4.1 Stream Cipher and Block Cipher

128 bits is used. As with a stream cipher, the two users share a symmetric encryption key (Figure 4.1b). Using some of the modes of operation explained in Chapter 7, a block cipher can be used to achieve the same effect as a stream cipher.

Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based symmetric cryptographic applications make use of block ciphers. Accordingly, the concern in this chapter, and in our discussions throughout the book of symmetric encryption, will primarily focus on block ciphers.

Motivation for the Feistel Cipher Structure

A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits. There are 2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformations for $n = 2$.

Reversible Mapping		**Irreversible Mapping**	
Plaintext	**Ciphertext**	**Plaintext**	**Ciphertext**
00	11	00	11
01	10	01	10
10	00	10	01
11	01	11	01

In the latter case, a ciphertext of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is $2^n!$.[2]

Figure 4.2 illustrates the logic of a general substitution cipher for $n = 4$. A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique one of 16 possible output states, each of which is represented by 4 ciphertext bits. The encryption and decryption mappings can be defined by a tabulation, as shown in Table 4.1. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext. Feistel refers to this as the *ideal block cipher*, because it allows for the maximum number of possible encryption mappings from the plaintext block [FEIS75].

But there is a practical problem with the ideal block cipher. If a small block size, such as $n = 4$, is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible.

---

[2]The reasoning is as follows: For the first plaintext, we can choose any of 2^n ciphertext blocks. For the second plaintext, we choose from among $2^n - 1$ remaining ciphertext blocks, and so on.

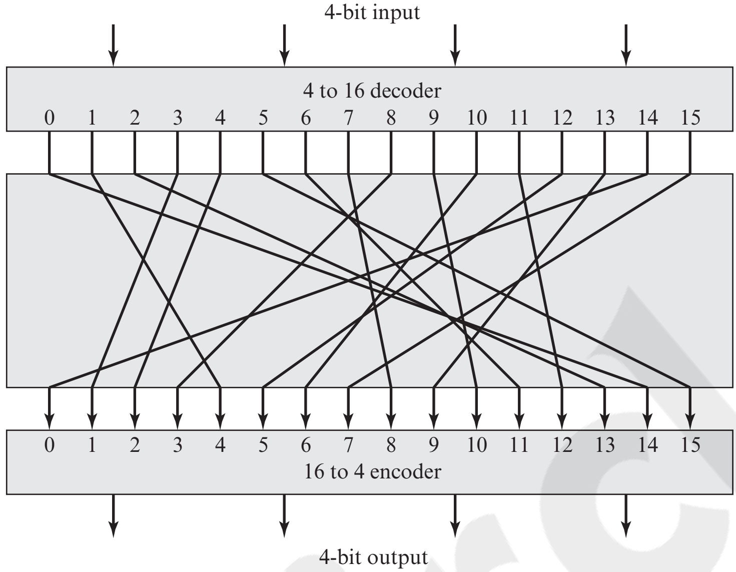


Figure 4.2 General n-bit-n-bit Block Substitution (shown with $n = 4$)

An arbitrary reversible substitution cipher (the ideal block cipher) for a large block size is not practical, however, from an implementation and performance point of view. For such a transformation, the mapping itself constitutes the key. Consider again Table 4.1, which defines one particular reversible mapping from

Table 4.1 Encryption and Decryption Tables for Substitution Cipher of Figure 4.2

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

plaintext to ciphertext for $n = 4$. The mapping can be defined by the entries in the second column, which show the value of the ciphertext for each plaintext block. This, in essence, is the key that determines the specific mapping from among all possible mappings. In this case, using this straightforward method of defining the key, the required key length is (4 bits) $\times$ (16 rows) = 64 bits. In general, for an n-bit ideal block cipher, the length of the key defined in this fashion is $n \times 2^n$ bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length is $64 \times 2^{64} = 2^{70} \approx 10^{21}$ bits.

In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal block cipher system for large n, built up out of components that are easily realizable [FEIS75]. But before turning to Feistel's approach, let us make one other observation. We could use the general block substitution cipher but, to make its implementation tractable, confine ourselves to a subset of the $2^n!$ possible reversible mappings. For example, suppose we define the mapping in terms of a set of linear equations. In the case of $n = 4$, we have

$$y_1 = k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4$$
$$y_2 = k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4$$
$$y_3 = k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4$$
$$y_4 = k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4$$

where the x_i are the four binary digits of the plaintext block, the y_i are the four binary digits of the ciphertext block, the k_{ij} are the binary coefficients, and arithmetic is mod 2. The key size is just n^2, in this case 16 bits. The danger with this kind of formulation is that it may be vulnerable to cryptanalysis by an attacker that is aware of the structure of the algorithm. In this example, what we have is essentially the Hill cipher discussed in Chapter 3, applied to binary data rather than characters. As we saw in Chapter 3, a simple linear system such as this is quite vulnerable.

The Feistel Cipher

Feistel proposed [FEIS73] that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with a key length of k bits and a block length of n bits, allowing a total of 2^k possible transformations, rather than the $2^n!$ transformations available with the ideal block cipher.

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:

- **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.
- **Permutation:** A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

In fact, Feistel's is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions [SHAN49].[3] We look next at these concepts of diffusion and confusion and then present the Feistel cipher. But first, it is worth commenting on this remarkable fact: The Feistel cipher structure, which dates back over a quarter century and which, in turn, is based on Shannon's proposal of 1945, is the structure used by a number of significant symmetric block ciphers currently in use. In particular, the Feistel structure is used for Triple Data Encryption Algorithm (TDEA), which is one of the two encryption algorithms (along with AES), approved for general use by the National Institute of Standards and Technology (NIST). The Feistel structure is also used for several schemes for format-preserving encryption, which have recently come into prominence. In addition, the Camellia block cipher is a Feistel structure; it is one of the possible symmetric ciphers in TLS and a number of other Internet security protocols. Both TDEA and format-preserving encryption are covered in Chapter 7.

Diffusion and Confusion The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system [SHAN49]. Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the ciphertext, the cryptanalyst may be able to deduce the encryption key, part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the ciphertext are independent of the particular key used. The arbitrary substitution cipher that we discussed previously (Figure 4.2) is such a cipher, but as we have seen, it is impractical.[4]

Other than recourse to ideal systems, Shannon suggests two methods for frustrating statistical cryptanalysis: diffusion and confusion. In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits. An example of diffusion is to encrypt a message $M = m_1, m_2, m_3, \ldots$ of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

[3]The paper is available at box.com/Crypto7e. Shannon's 1949 paper appeared originally as a classified report in 1945. Shannon enjoys an amazing and unique position in the history of computer and information science. He not only developed the seminal ideas of modern cryptography but is also responsible for inventing the discipline of information theory. Based on his work in information theory, he developed a formula for the capacity of a data communications channel, which is still used today. In addition, he founded another discipline, the application of Boolean algebra to the study of digital circuits; this last he managed to toss off as a master's thesis.

[4]Appendix F expands on Shannon's concepts concerning measures of secrecy and the security of cryptographic algorithms.

adding k successive letters to get a ciphertext letter y_n. One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of ciphertext.[5]

Every block cipher involves a transformation of a block of plaintext into a block of ciphertext, where the transformation depends on the key. The mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key. On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion.

As [ROBS95b] points out, so successful are diffusion and confusion in capturing the essence of the desired attributes of a block cipher that they have become the cornerstone of modern block cipher design.

FEISTEL CIPHER STRUCTURE The left-hand side of Figure 4.3 depicts the encryption structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K. The plaintext block is divided into two halves, LE_0 and RE_0. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs LE_{i-1} and RE_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys K_i are different from K and from each other. In Figure 4.3, 16 rounds are used, although any number of rounds could be implemented.

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey K_i. Another way to express this is to say that F is a function of right-half block of w bits and a subkey of y bits, which produces an output value of length w bits: $F(RE_i, K_{i+1})$. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.[6] This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

---

[5]Some books on cryptography equate permutation with diffusion. This is incorrect. Permutation, *by itself*, does not change the statistics of the plaintext at the level of individual letters or permuted blocks. For example, in DES, the permutation swaps two 32-bit blocks, so statistics of strings of 32 bits or less are preserved.

[6]The final round is followed by an interchange that undoes the interchange that is part of the final round. One could simply leave both interchanges out of the diagram, at the sacrifice of some consistency of presentation. In any case, the effective lack of a swap in the final round is done to simplify the implementation of the decryption process, as we shall see.

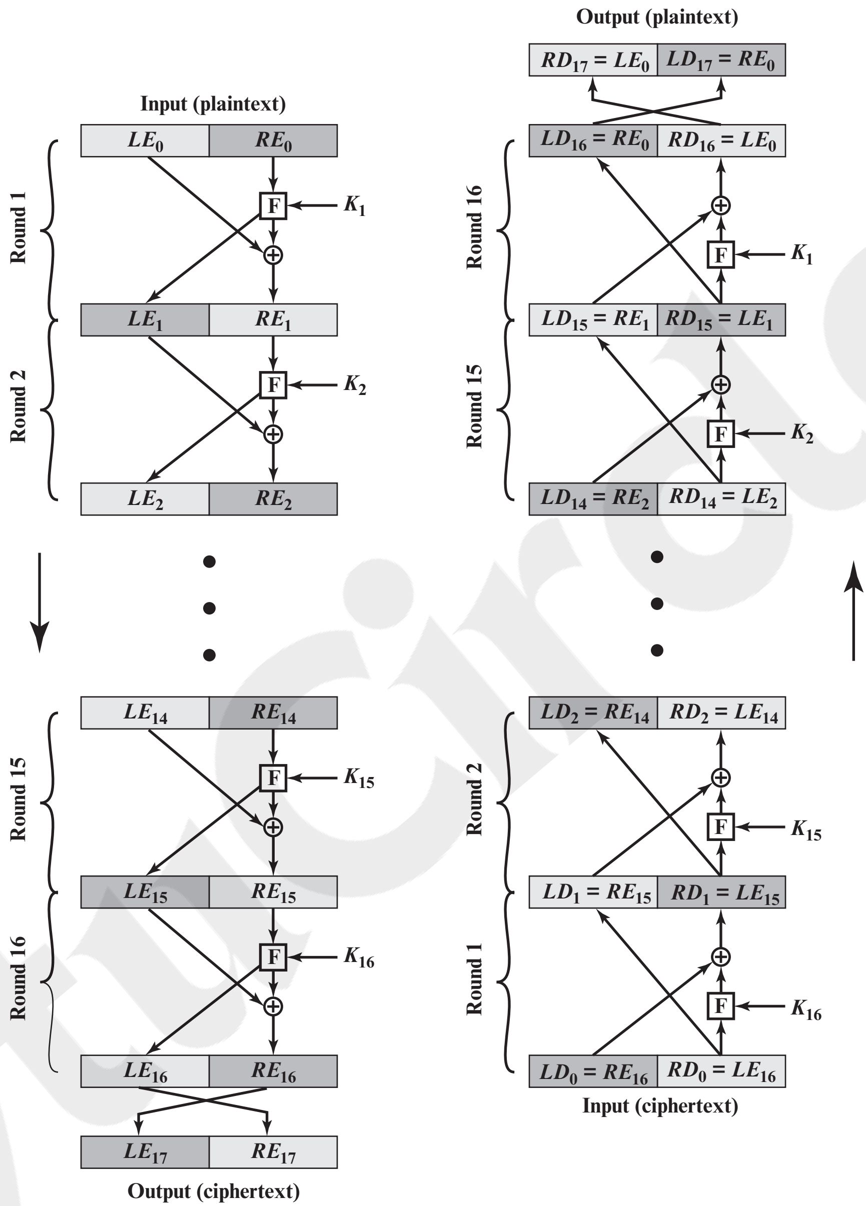


Figure 4.3 Feistel Encryption and Decryption (16 rounds)

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
- **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

FEISTEL DECRYPTION ALGORITHM The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on, until K_1 is used in the last round. This is a nice feature, because it means we need not implement two different algorithms; one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, Figure 4.3 shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm. For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the ith encryption round be $LE_i \| RE_i$ (LE_i concatenated with RE_i). Then the corresponding output of the $(16 - i)$th decryption round is $RE_i \| LE_i$ or, equivalently, $LD_{16-i} \| RD_{16-i}$.

Let us walk through Figure 4.3 to demonstrate the validity of the preceding assertions. After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is $RE_{16} \| LE_{16}$. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is $RE_{16} \| LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that

$$\begin{aligned} LE_{16} &= RE_{15} \\ RE_{16} &= LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16}) \end{aligned}$$

On the decryption side,

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus \mathrm{F}(RD_0, K_{16}) \\ &= RE_{16} \oplus \mathrm{F}(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})] \oplus \mathrm{F}(RE_{15}, K_{16}) \end{aligned}$$

The XOR has the following properties:

$$\begin{aligned} [A \oplus B] \oplus C &= A \oplus [B \oplus C] \\ D \oplus D &= 0 \\ E \oplus 0 &= E \end{aligned}$$

Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$. Therefore, the output of the first round of the decryption process is $RE_{15} \| LE_{15}$, which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the ith iteration of the encryption algorithm,

$$\begin{aligned} LE_i &= RE_{i-1} \\ RE_i &= LE_{i-1} \oplus \mathrm{F}(RE_{i-1}, K_i) \end{aligned}$$

Rearranging terms:

$$\begin{aligned} RE_{i-1} &= LE_i \\ LE_{i-1} &= RE_i \oplus \mathrm{F}(RE_{i-1}, K_i) = RE_i \oplus \mathrm{F}(LE_i, K_i) \end{aligned}$$

Thus, we have described the inputs to the ith iteration as a function of the outputs, and these equations confirm the assignments shown in the right-hand side of Figure 4.3.

Finally, we see that the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.

Note that the derivation does not require that F be a reversible function. To see this, take a limiting case in which F produces a constant output (e.g., all ones) regardless of the values of its two arguments. The equations still hold.

To help clarify the preceding concepts, let us look at a specific example (Figure 4.4 and focus on the fifteenth round of encryption, corresponding to the second round of decryption. Suppose that the blocks at each stage are 32 bits (two 16-bit halves) and that the key size is 24 bits. Suppose that at the end of encryption round fourteen, the value of the intermediate block (in hexadecimal) is DE7F03A6. Then LE_{14} = DE7F and RE_{14} = 03A6. Also assume that the value of K_{15} is 12DE52. After round 15, we have LE_{15} = 03A6 and RE_{15} = F(03A6, 12DE52) $\oplus$ DE7F.

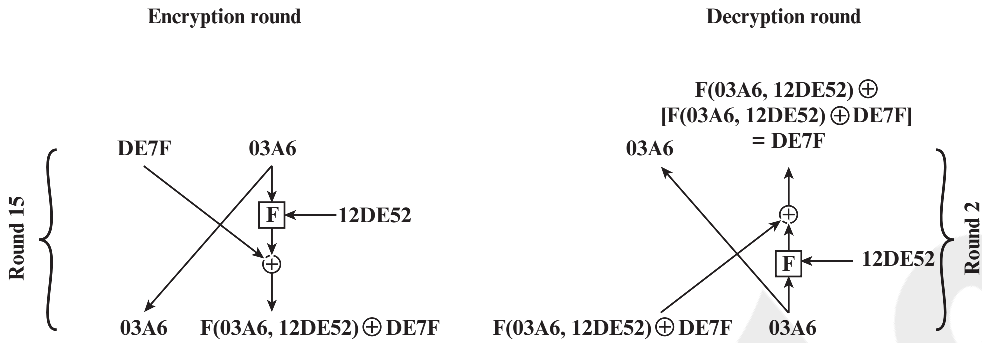


Figure 4.4 Feistel Example

Now let's look at the decryption. We assume that $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$, as shown in Figure 4.3, and we want to demonstrate that $LD_2 = RE_{14}$ and $RD_2 = LE_{14}$. So, we start with $LD_1 = \text{F}(03\text{A}6, 12\text{DE}52) \oplus \text{DE}7\text{F}$ and $RD_1 = 03\text{A}6$. Then, from Figure 4.3, $LD_2 = 03\text{A}6 = RE_{14}$ and $RD_2 = \text{F}(03\text{A}6, 12\text{DE}52) \oplus [\text{F}(03\text{A}6, 12\text{DE}52) \oplus \text{DE}7\text{F}] = \text{DE}7\text{F} = \text{LE}14$.

4.2 THE DATA ENCRYPTION STANDARD

Until the introduction of the Advanced Encryption Standard (AES) in 2001, the Data Encryption Standard (DES) was the most widely used encryption scheme. DES was issued in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).[7] For DEA, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

Over the years, DES became the dominant symmetric encryption algorithm, especially in financial applications. In 1994, NIST reaffirmed DES for federal use for another five years; NIST recommended the use of DES for applications other than the protection of classified information. In 1999, NIST issued a new version of its standard (FIPS PUB 46-3) that indicated that DES should be used only for legacy systems and that triple DES (which in essence involves repeating the DES algorithm three times on the plaintext using two or three different keys to produce the ciphertext) be used. We study triple DES in Chapter 7. Because the underlying encryption and decryption algorithms are the same for DES and triple DES, it remains important to understand the DES cipher. This section provides an overview.For the interested reader, Appendix S provides further detail.

[7]The terminology is a bit confusing. Until recently, the terms *DES* and *DEA* could be used interchangeably. However, the most recent edition of the DES document includes a specification of the DEA described here plus the triple DEA (TDEA) described in Chapter 7. Both DEA and TDEA are part of the Data Encryption Standard. Further, until the recent adoption of the official term *TDEA*, the triple DEA algorithm was typically referred to as *triple DES* and written as 3DES. For the sake of convenience, we will use the term 3DES.

DES Encryption

The overall scheme for DES encryption is illustrated in Figure 4.5. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.[8]

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.

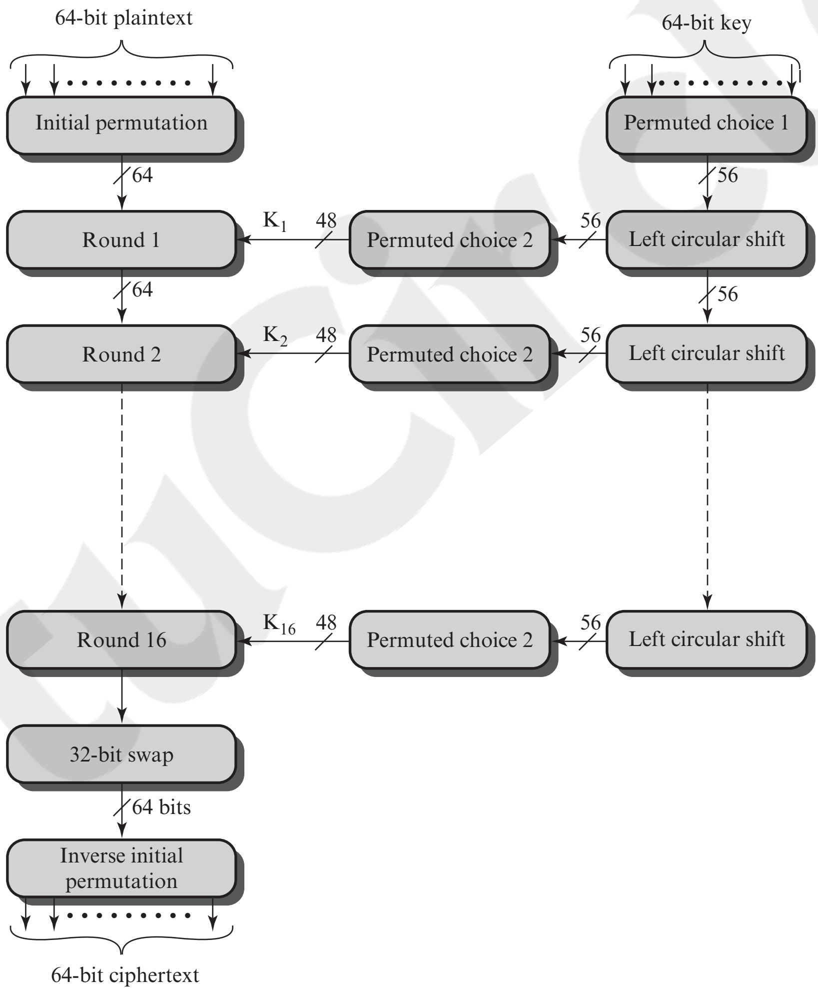


Figure 4.5 General Depiction of DES Encryption Algorithm

[8]Actually, the function expects a 64-bit key as input. However, only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily.

This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput. Finally, the preoutput is passed through a permutation $[IP^{-1}]$ that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 4.3.

The right-hand portion of Figure 4.5 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a *subkey* (K_i) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed. Additionally, the initial and final permutations are reversed.

4.3 A DES EXAMPLE

We now work through an example and consider some of its implications. Although you are not expected to duplicate the example by hand, you will find it informative to study the hex patterns that occur from one step to the next.

For this example, the plaintext is a hexadecimal palindrome. The plaintext, key, and resulting ciphertext are as follows:

Plaintext:	`02468aceeca86420`
Key:	`0f1571c947d9e859`
Ciphertext:	`da02ce3a89ecac3b`

Results

Table 4.2 shows the progression of the algorithm. The first row shows the 32-bit values of the left and right halves of data after the initial permutation. The next 16 rows show the results after each round. Also shown is the value of the 48-bit subkey generated for each round. Note that $L_i = R_{i-1}$. The final row shows the left- and right-hand values after the inverse initial permutation. These two values combined form the ciphertext.

The Avalanche Effect

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce

Table 4.2 DES Example

Round	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP^{-1}		da02ce3a	89ecac3b

Note: DES subkeys are shown as eight 6-bit values in hex format

a change in many bits of the ciphertext. This is referred to as the avalanche effect. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

Using the example from Table 4.2, Table 4.3 shows the result when the fourth bit of the plaintext is changed, so that the plaintext is **12468aceeca86420**. The second column of the table shows the intermediate 64-bit values at the end of each round for the two plaintexts. The third column shows the number of bits that differ between the two intermediate values. The table shows that, after just three rounds, 18 bits differ between the two blocks. On completion, the two ciphertexts differ in 32 bit positions.

Table 4.4 shows a similar test using the original plaintext of with two keys that differ in only the fourth bit position: the original key, **0f1571c947d9e859**, and the altered key, **1f1571c947d9e859**. Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

Table 4.3 Avalanche Effect in DES: Change in Plaintext

Round		δ	Round		δ
	02468aceeca86420 12468aceeca86420	1	9	c11bfc09887fbc6c 99f911532eed7d94	32
1	3cf03c0fbad22845 3cf03c0fbad32845	1	10	887fbc6c600f7e8b 2eed7d94d0f23094	34
2	bad2284599e9b723 bad3284539a9b7a3	5	11	600f7e8bf596506e d0f23094455da9c4	37
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18	12	f596506e738538b8 455da9c47f6e3cf3	31
4	0bae3b9e42415649 171cb8b3ccaca55e	34	13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
5	4241564918b3fa41 ccaca55ed16c3653	37	14	c6a62c4e56b0bd75 4bc1a8d91e07d409	33
6	18b3fa419616fe23 d16c3653cf402c68	33	15	56b0bd7575e8fd8f 1e07d4091ce2e6dc	31
7	9616fe2367117cf2 cf402c682b2cefbc	32	16	75e8fd8f25896490 1ce2e6dc365e5f59	32
8	67117cf2c11bfc09 2b2cefbc99f91153	33	IP^{-1}	da02ce3a89ecac3b 057cde97d7683f2a	32

Table 4.4 Avalanche Effect in DES: Change in Key

Round		δ	Round		δ
	02468aceeca86420 02468aceeca86420	0	9	c11bfc09887fbc6c 548f1de471f64dfd	34
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3	10	887fbc6c600f7e8b 71f64dfd4279876c	36
2	bad2284599e9b723 9ad628c59939136b	11	11	600f7e8bf596506e 4279876c399fdc0d	32
3	99e9b7230bae3b9e 9939136b768067b7	25	12	f596506e738538b8 399fdc0d6d208dbb	28
4	0bae3b9e42415649 768067b75a8807c5	29	13	738538b8c6a62c4e 6d208dbbb9bdeeaa	33
5	4241564918b3fa41 5a8807c5488dbe94	26	14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
6	18b3fa419616fe23 488dbe94aba7fe53	26	15	56b0bd7575e8fd8f d2c3a56f2765c1fb	27
7	9616fe2367117cf2 aba7fe53177d21e4	27	16	75e8fd8f25896490 2765c1fb01263dc4	30
8	67117cf2c11bfc09 177d21e4548f1de4	32	IP^{-1}	da02ce3a89ecac3b ee92b50606b62b0b	30

4.4 THE STRENGTH OF DES

Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16} keys. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond [DIFF77]. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about $20 million in 1977 dollars.

With current technology, it is not even necessary to use special, purpose-built hardware. Rather, the speed of commercial, off-the-shelf processors threaten the security of DES. A recent paper from Seagate Technology [SEAG08] suggests that a rate of 1 billion (10^9) key combinations per second is reasonable for today's multicore computers. Recent offerings confirm this. Both Intel and AMD now offer hardware-based instructions to accelerate the use of AES. Tests run on a contemporary multicore Intel machine resulted in an encryption rate of about half a billion encryptions per second [BASU12]. Another recent analysis suggests that with contemporary supercomputer technology, a rate of 10^{13} encryptions per second is reasonable [AROR12].

With these results in mind, Table 4.5 shows how much time is required for a brute-force attack for various key sizes. As can be seen, a single PC can break DES in about a year; if multiple PCs work in parallel, the time is drastically shortened. And today's supercomputers should be able to find a key in about an hour. Key sizes of 128 bits or greater are effectively unbreakable using simply a brute-force approach. Even if we managed to speed up the attacking system by a factor of 1 trillion (10^{12}), it would still take over 100,000 years to break a code using a 128-bit key.

Fortunately, there are a number of alternatives to DES, the most important of which are AES and triple DES, discussed in Chapters 6 and 7, respectively.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration (described in Appendix S). Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows

Table 4.5 Average Time Required for Exhaustive Key Search

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 Decryptions/s	Time Required at 10^{13} Decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns $= 1.125$ years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns $= 5.3 \times 10^{21}$ years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns $= 5.8 \times 10^{33}$ years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns $= 9.8 \times 10^{40}$ years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns $= 1.8 \times 10^{60}$ years	1.8×10^{56} years
26 characters (permutation)	Monoalphabetic	$2! = 4 \times 10^{26}$	2×10^{26} ns $= 6.3 \times 10^{9}$ years	6.3×10^{6} years

the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.[9]

Timing Attacks

We discuss timing attacks in more detail in Part Two, as they relate to public-key algorithms. However, the issue may also be relevant for symmetric ciphers. In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. [HEVI99] reports on an approach that yields the Hamming weight (number of bits equal to one) of the secret key. This is a long way from knowing the actual key, but it is an intriguing first step. The authors conclude that DES appears to be fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

4.5 BLOCK CIPHER DESIGN PRINCIPLES

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed all that much since the work of Feistel and the DES design team in the early 1970s. In this section we look at three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling.

---

[9]At least, no one has publicly acknowledged such a discovery.

Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds.

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES. Schneier [SCHN96] observes that for 16-round DES, a differential cryptanalysis attack is slightly less efficient than brute force: The differential cryptanalysis attack requires $2^{55.1}$ operations,[10] whereas brute force requires 2^{55}. If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than a brute-force key search.

This criterion is attractive, because it makes it easy to judge the strength of an algorithm and to compare different algorithms. In the absence of a cryptanalytic breakthrough, the strength of any algorithm that satisfies the criterion can be judged solely on key length.

Design of Function F

The heart of a Feistel block cipher is the function F, which provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F. One obvious criterion is that F be nonlinear, as we discussed previously. The more nonlinear F, the more difficult any type of cryptanalysis will be. There are several measures of nonlinearity, which are beyond the scope of this book. In rough terms, the more difficult it is to approximate F by a set of linear equations, the more nonlinear F is.

Several other criteria should be considered in designing F. We would like the algorithm to have good avalanche properties. Recall that, in general, this means that a change in one bit of the input should produce a change in many bits of the output. A more stringent version of this is the **strict avalanche criterion (SAC)** [WEBS86], which states that any output bit j of an S-box (see Appendix S for a discussion of S-boxes) should change with probability 1/2 when any single input bit i is inverted for all i, j. Although SAC is expressed in terms of S-boxes, a similar criterion could be applied to F as a whole. This is important when considering designs that do not include S-boxes.

Another criterion proposed in [WEBS86] is the **bit independence criterion (BIC)**, which states that output bits j and k should change independently when any single input bit i is inverted for all $i, j,$ and k. The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

[10] Differential cryptanalysis of DES requires 2^{47} *chosen* plaintext. If all you have to work with is known plaintext, then you must sort through a large quantity of known plaintext–ciphertext pairs looking for the useful ones. This brings the level of effort up to $2^{55.1}$.

Key Schedule Algorithm

With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key. No general principles for this have yet been promulgated.

Adams suggests [ADAM94] that, at minimum, the key schedule should guarantee key/ciphertext Strict Avalanche Criterion and Bit Independence Criterion.