CHAPTER 4

Block Ciphers and the Data Encryption Standard

4.1 Traditional Block Cipher Structure

Stream Ciphers and Block Ciphers
Motivation for the Feistel Cipher Structure
The Feistel Cipher

4.2 The Data Encryption Standard

DES Encryption
DES Decryption

4.3 A DES Example

Results
The Avalanche Effect

4.4 The Strength of DES

The Use of 56-Bit Keys
The Nature of the DES Algorithm
Timing Attacks

4.5 Block Cipher Design Principles

Number of Rounds
Design of Function F
Key Schedule Algorithm

4.6 Key Terms, Review Questions, and Problems

LEARNING OBJECTIVES

After studying this chapter, you should be able to

- Understand the distinction between stream ciphers and block ciphers.
- Present an overview of the Feistel cipher and explain how decryption is the inverse of encryption.
- Present an overview of Data Encryption Standard (DES).
- Explain the concept of the avalanche effect.
- Discuss the cryptographic strength of DES.
- Summarize the principal block cipher design principles.

The objective of this chapter is to illustrate the principles of modern symmetric ciphers. For this purpose, we focus on the most widely used symmetric cipher: the Data Encryption Standard (DES). Although numerous symmetric ciphers have been developed since the introduction of DES, and although it is destined to be replaced by the Advanced Encryption Standard (AES), DES remains the most important such algorithm. Furthermore, a detailed study of DES provides an understanding of the principles used in other symmetric ciphers.

This chapter begins with a discussion of the general principles of symmetric block ciphers, which are the principal type of symmetric ciphers studied in this book. The other form of symmetric ciphers, stream ciphers, are discussed in Chapter 8. Next, we cover full DES. Following this look at a specific algorithm, we return to a more general discussion of block cipher design.

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.

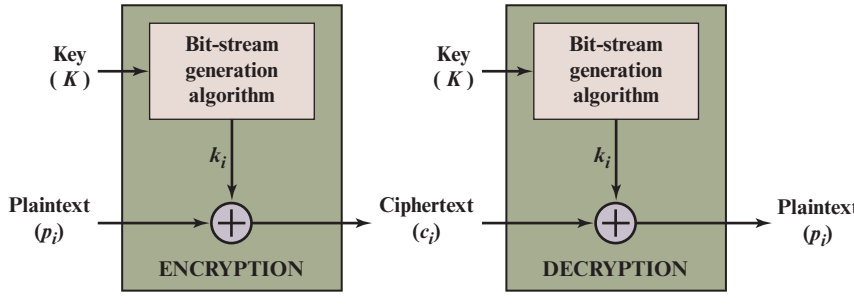
Stream Ciphers and Block Ciphers

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

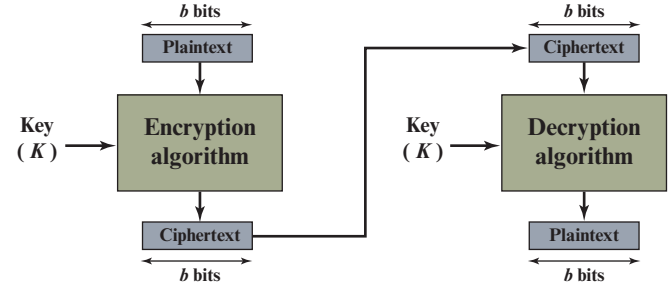
plaintext bit stream (p_i). If the cryptographic keystream is random, then this cipher is unbreakable by any means other than acquiring the keystream. However, the keystream must be provided to both users in advance via some independent and secure channel. This introduces insurmountable logistical problems 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 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.



(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

Figure 4.1 Stream Cipher and Block 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 $2^n!$.[1]

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.

[1]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.

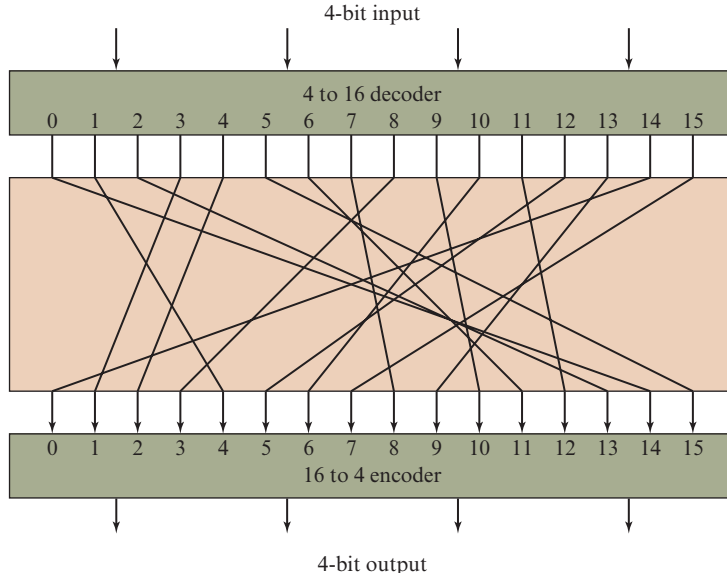


Figure 4.2 General n-bit-n-bit Block Substitution (shown with $n = 4$)

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

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

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].[2] 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.[3]

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:

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

[2]The paper is available at box.com/Crypto8e. 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.

[3]Appendix B 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.[4]

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.[5] This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

---

[4]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.

[5]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.

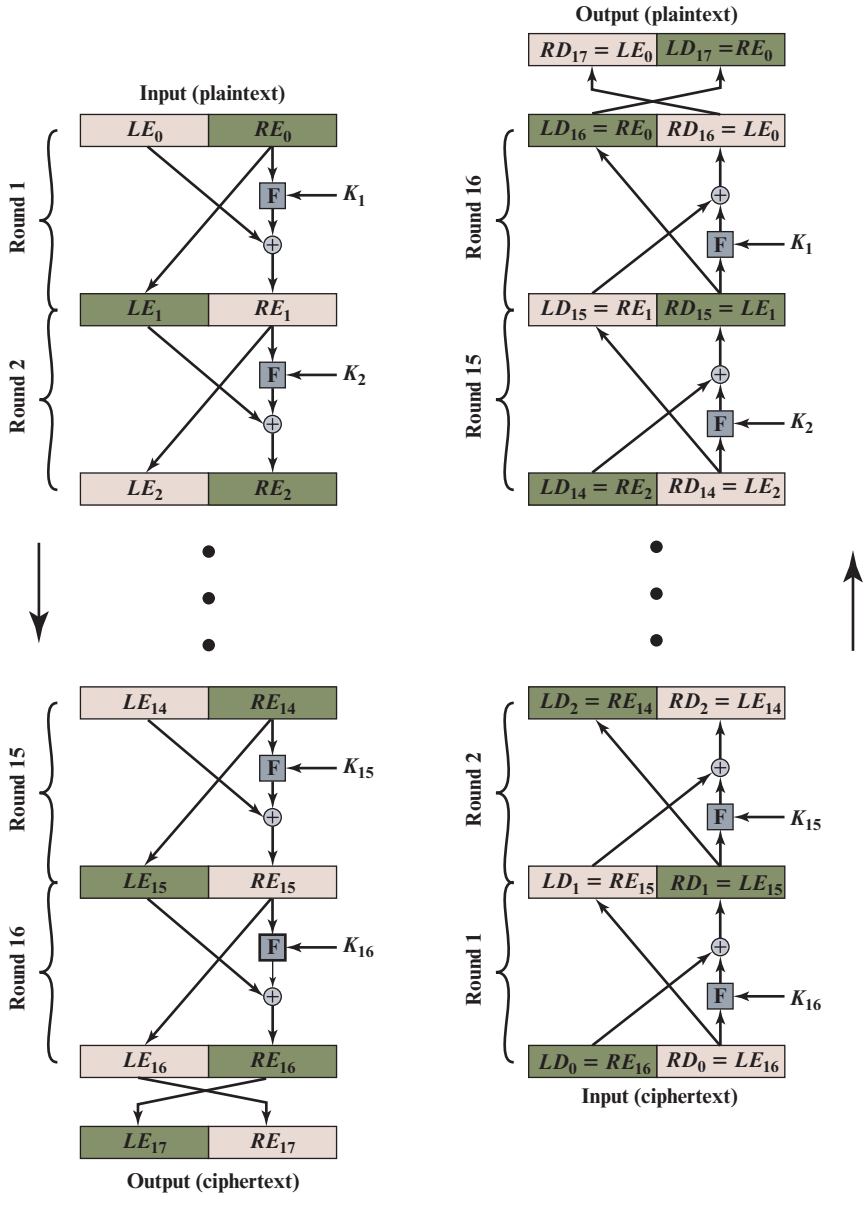


Figure 4.3 Feistel Encryption and Decryption (16 rounds)

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.

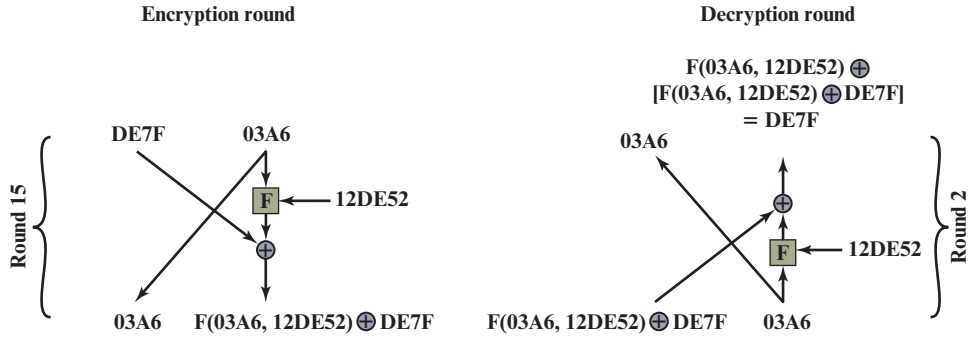


Figure 4.4 Feistel Example

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 $LD_1 = \text{F(03A6, 12DE52)} \oplus \text{DE7F}$ and $RD_1 = \text{03A6}$. Then, from Figure 4.3, $LD_2 = \text{03A6} = RE_{14}$ and $RD_2 = \text{F(03A6, 12DE52)} \oplus [\text{F(03A6, 12DE52)} \oplus \text{DE7F}] = \text{DE7F} = \text{LE14}$.

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).[6] 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 C provides further detail.

[6]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.[7]

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*.

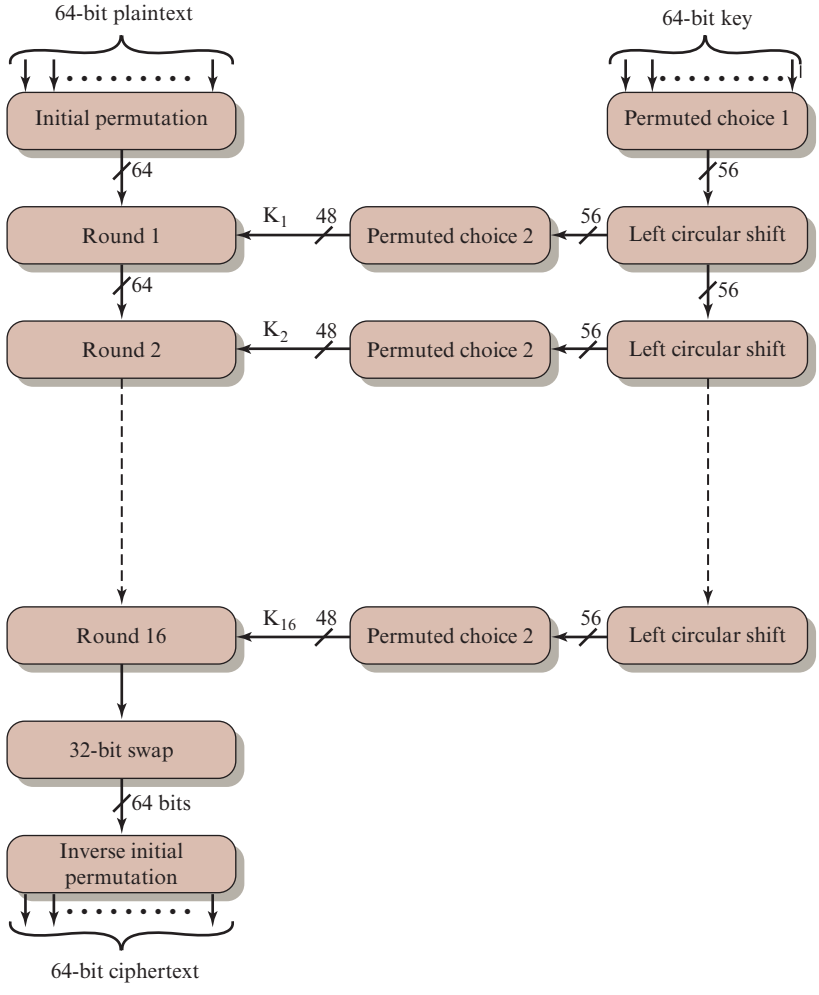


Figure 4.5 General Depiction of DES Encryption Algorithm

[7]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 2008 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 C). 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 the weaknesses in

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 = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years
26 characters (permutation)	Monoalphabetic	$2! = 4 \times 10^{26}$	2×10^{26} ns = 6.3×10^{9} years	6.3×10^{6} years

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.[8]

Timing Attacks

We discuss timing attacks in more detail in Part Three, 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.

[8] 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,[9] 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 C 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.

---

[9] 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.

4.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

avalanche effect	irreversible mapping	round function
block cipher	permutation	stream cipher
confusion	product cipher	subkey
diffusion	reversible mapping	substitution
Feistel cipher	round	

Review Questions

4.1 Briefly define a nonsingular transformation.

4.2 What is the difference between a block cipher and a stream cipher?

4.3 Why is it not practical to use an arbitrary reversible substitution cipher of the kind shown in Table 4.1?

4.4 Briefly define the terms substitution and permutation.

4.5 What is the strict avalanche criterion for the Feistel F function?

4.6 Which parameters and design choices determine the actual algorithm of a Feistel cipher?

4.7 What are the critical aspects of Feistel cipher design?

Problems

4.1 **a.** In Section 4.1, under the subsection on Feistel decryption algorithm, it is mentioned that the decryption algorithm is essentially the same as the encryption algorithm except that the subkeys are used in reverse order. Justify this statement.

b. In the same discussion, it was stated that the F function is not required to be reversible for the correctness of the algorithm. The statement holds for an F function that always produces a constant output. Will the claim still hold if F does not always produce a constant output? Justify your answer.

4.2 Consider a Feistel cipher composed of sixteen rounds with a block length of 128 bits and a key length of 128 bits. Suppose that, for a given k, the key scheduling algorithm determines values for the first eight round keys, $k_1, k_2, \ldots k_8$, and then sets

$$k_9 = k_8, k_{10} = k_7, k_{11} = k_6, \ldots, k_{16} = k_1$$

Suppose you have a ciphertext c. Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.)

4.3 Let π be a permutation of the integers $0, 1, 2, \ldots, (2^n - 1)$, such that $\pi(m)$ gives the permuted value of m, $0 \leq m < 2^n$. Put another way, π maps the set of n-bit integers onto itself, and no two integers map into the same integer. DES is such a permutation for 64-bit integers. We say that π has a fixed point at m if $\pi(m) = m$. That is, if π is an encryption mapping, then a fixed point corresponds to a message that encrypts to itself. We are interested in the number of fixed points in a randomly chosen permutation π. Show the somewhat unexpected result that the number of fixed points for π is 1 on an average, and this number is independent of the size of the permutation.

4.4 Consider a block encryption algorithm that encrypts blocks of length n, and let $N = 2^n$. Say we have t plaintext–ciphertext pairs $P_i, C_i = \mathrm{E}(K, P_i)$, where we assume that the key K is a randomly chosen m-bit string. Imagine that we wish to find K by exhaustive search. We could generate key K' and test whether $C_i = \mathrm{E}(K', P_i)$ for $1 \ldots i \ldots t$. If K' encrypts each P_i to its proper C_i, then we have evidence that $K = K'$. However, it may be the case that the mappings $\mathrm{E}(K, \cdot)$ and $\mathrm{E}(K', \cdot)$ exactly agree on the t plaintext–ciphertext pairs P_i, C_i and agree on no other pairs. Such keys are called *spurious keys*.

a. What is the probability that $\mathrm{E}(K, \cdot)$ and $\mathrm{E}(K', \cdot)$ agree on exactly t plaintext-ciphertext pairs?

b. Find the expected number of spurious keys when $\mathrm{E}(K, \cdot)$ and $\mathrm{E}(K', \cdot)$ agree on exactly t plaintext-ciphertext pairs.

4.5 For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 256-bit blocks of plaintext into 256-bit blocks of ciphertext. Let $\mathrm{EL}(k, m)$ denote the encryption of a 256-bit message m under a key k (the actual bit length of k is irrelevant). Thus,

$$\mathrm{EL}(k, [m_1 \oplus m_2]) = \mathrm{EL}(k, m_1) \oplus \mathrm{EL}(k, m_2) \text{ for all 128-bit patterns } m_1, m_2.$$

Describe how, with 256 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k. (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 256 plaintext/ciphertext pairs to work with, and you have the ability to choose the value of the ciphertexts.)

4.6 Suppose the DES F function mapped every 32-bit input R, regardless of the value of the input K, to

a. a 32-bit string of zero, and

b. R.

Then:

1. What function would DES compute?

2. What would the decryption look like?

Hint: Use the following properties of the XOR operation:

$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

$$(A \oplus A) = 0$$
$$(A \oplus 0) = A$$
$$A \oplus 1 = \text{bitwise complement of } A$$

4.7 Show that DES decryption is, in fact, the inverse of DES encryption.

4.8 The 32-bit swap after the sixteenth iteration of the DES algorithm is needed to make the encryption process invertible by simply running the ciphertext back through the algorithm with the key order reversed. This was demonstrated in the preceding problem. However, it still may not be entirely clear why the 32-bit swap is needed. To demonstrate why, solve the following exercises. First, some notation:

$A \| B$ = the concatenation of the bit strings A and B

$T_i(R \| L)$ = the transformation defined by the ith iteration of the encryption algorithm for $1 \leq I \leq 16$

$TD_i(R \| L)$ = the transformation defined by the ith iteration of the decryption algorithm for $1 \leq I \leq 16$

$T_{17}(R \| L)$ = $L \| R$, where this transformation occurs after the sixteenth iteration of the encryption algorithm

a. Show that the composition $TD_1(IP(IP^{-1}(T_{17}(T_{16}(L_{15} \| R_{15})))))$ is equivalent to the transformation that interchanges the 32-bit halves, L_{15} and R_{15}. That is, show that

$$TD_1(IP(IP^{-1}(T_{17}(T_{16}(L_{15} \| R_{15}))))) = R_{15} \| L_{15}$$

b. Now suppose that we did away with the final 32-bit swap in the encryption algorithm. Then we would want the following equality to hold:

$$TD_1(\text{IP}(IP^{-1}(T_{16}(L_{15} \| R_{15})))) = L_{15} \| R_{15}$$

Does it?

Note: **The following problems refer to details of DES that are described in Appendix C.**

4.9 Consider the substitution defined by row 1 of S-box S_1 in Table C.2. Show a block diagram similar to Figure 4.2 that corresponds to this substitution.

4.10 Compute the bits number 4, 17, 41, and 45 at the output of the first round of the DES decryption, assuming that the ciphertext block is composed of all ones, and the external key is composed of all ones.

4.11 This problem provides a numerical example of encryption using a one-round version of DES. We start with the same bit pattern for the key K and the plaintext, namely:

Hexadecimal notation:	0 1 2 3 4 5 6 7 8 9 A B C D E F
Binary notation:	0000 0001 0010 0011 0100 0101 0110 0111
	1000 1001 1010 1011 1100 1101 1110 1111

a. Derive K_1, the first-round subkey.
b. Derive L_0, R_0.
c. Expand R_0 to get $\text{E}[R_0]$, where $\text{E}[\cdot]$ is the expansion function of Table C.1.
d. Calculate $A = \text{E}[R_0] \oplus K_1$.
e. Group the 48-bit result of (d) into sets of 6 bits and evaluate the corresponding S-box substitutions.
f. Concatenate the results of (e) to get a 32-bit result, B.

g. Apply the permutation to get P(B).
h. Calculate $R_1 = \text{P}(B) \oplus L_0$.
i. Write down the ciphertext.

4.12 Analyze the amount of left shifts in the DES key schedule by studying Table C.3 (d). Is there a pattern? What could be the reason for the choice of these constants?

4.13 Suppose that a modern multi-core computer can process 10^9 key combinations per second. How much time will it take to search the key space of an encryption algorithm that has a 56-bit key? If the key size is increased to 60 bits but the CPU speed is also doubled, then how much time will the key search take on the new computer?

4.14 **a.** Let X' be the bitwise complement of X. Prove that if the complement of the plaintext block is taken and the complement of an encryption key is taken, then the result of DES encryption with these values is the complement of the original ciphertext. That is,

$$\begin{array}{lccc} \text{If} & Y & = & \text{E}(K, X) \\ \text{Then} & Y' & = & \text{E}(K', X') \end{array}$$

Hint: Begin by showing that for any two bit strings of equal length, A and B, $(A \oplus B)' = A' \oplus B$.

b. It has been said that a brute-force attack on DES requires searching a key space of 2^{56} keys. Does the result of part (a) change that?

4.15 **a.** We say that a DES key K is weak if DES_K is an involution. Exhibit four weak keys for DES.

b. We say that a DES key K is semi-weak if it is not weak and if there exists a key K' such that $\text{DES}_K^{-1} = \text{DES}_{K'}$. Exhibit four semi-weak keys for DES.

Programming Problems

4.1 Create software that can encrypt and decrypt using a general substitution block cipher.

4.2 Create software that can encrypt and decrypt using S-DES.

APPENDIX C

DATA ENCRYPTION STANDARD

The overall scheme for DES encryption is illustrated in Figure C.1, which repeats 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.[1]

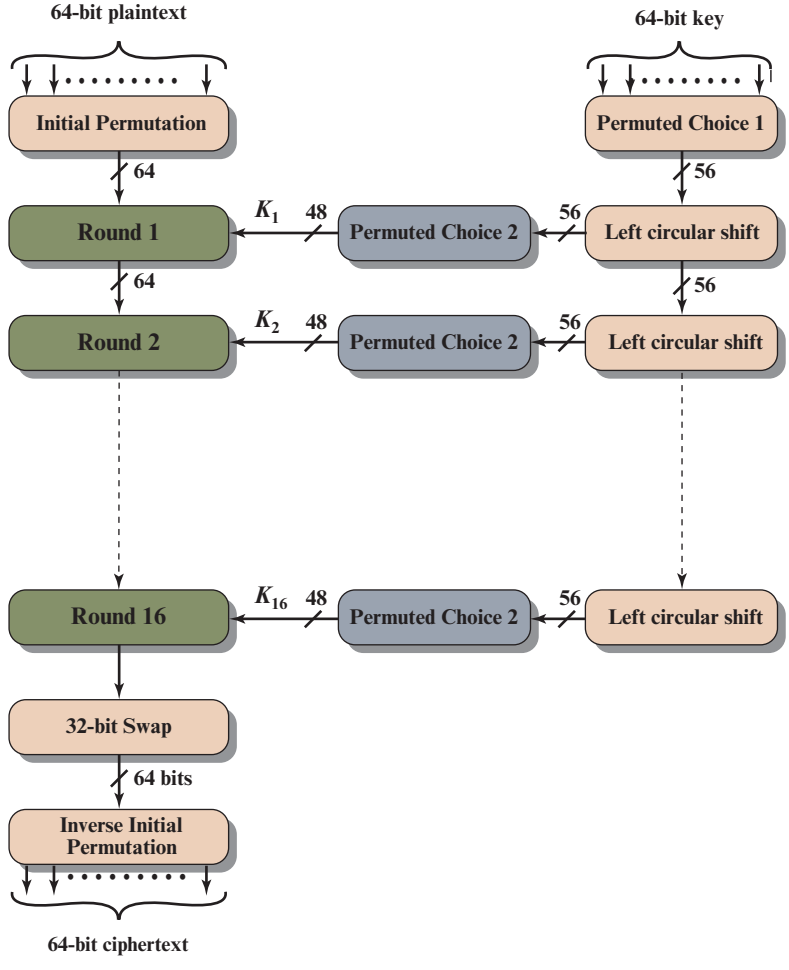


Figure C.1 General Depiction of DES Encryption Algorithm

[1]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.

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*. This is followed by a phase consisting of 16 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 C.1 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 16 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.

Initial Permutation

The initial permutation and its inverse are defined by tables, as shown in Tables C.1a and C.1b, respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Table C.1 Permutation Tables for DES

(a) Initial Permutation (IP)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input M:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}
M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}
M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}
M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

where *Mi* is a binary digit. Then the permutation $X = \text{IP}(M)$ is as follows:

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

If we then take the inverse permutation $Y = \text{IP}^{-1}(X) = \text{IP}^{-1}(\text{IP}(M))$, it can be seen that the original ordering of the bits is restored.

Details of Single Round

Figure C.2 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right).

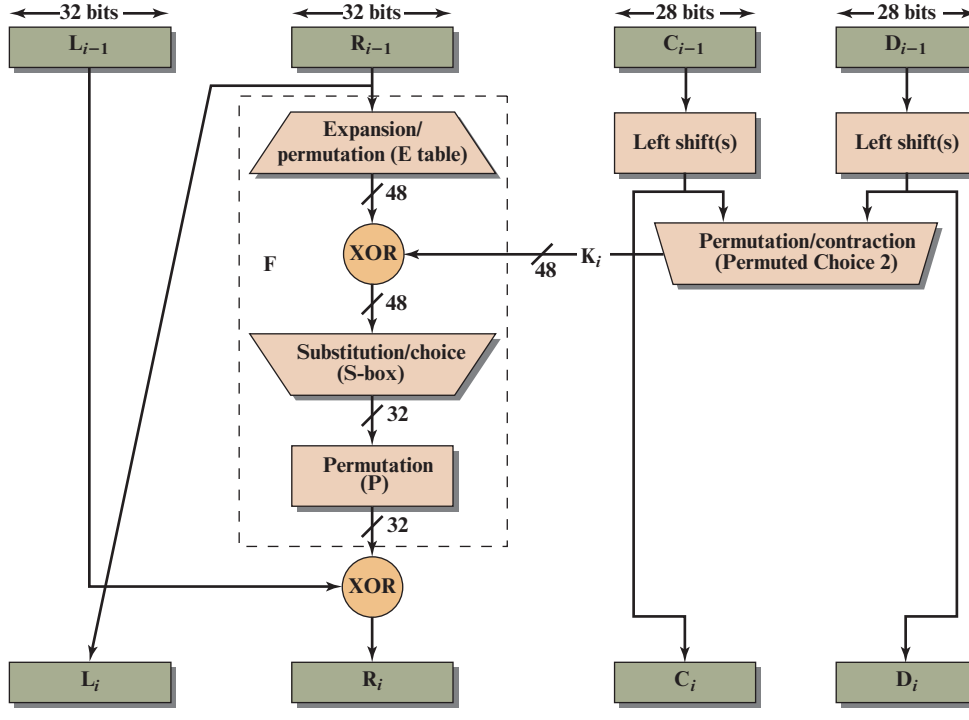


Figure C.2 Single Round of DES Algorithm

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus \mathrm{F}(R_{i-1}, K_i)$$

The round key K_i is 48 bits. The R input is 32 bits. This R input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the R bits (Table C.1c). The resulting 48 bits are XORed with K_i. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table C.1d.

The role of the S-boxes in the function F is illustrated in Figure C.3. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table C.2, which is interpreted as follows: The first and last bits of the input to box S_i form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S_i. The middle four bits select one of the 16 columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S_1, for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

Each row of an S-box defines a general reversible substitution. Figure 4.2 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S_1.

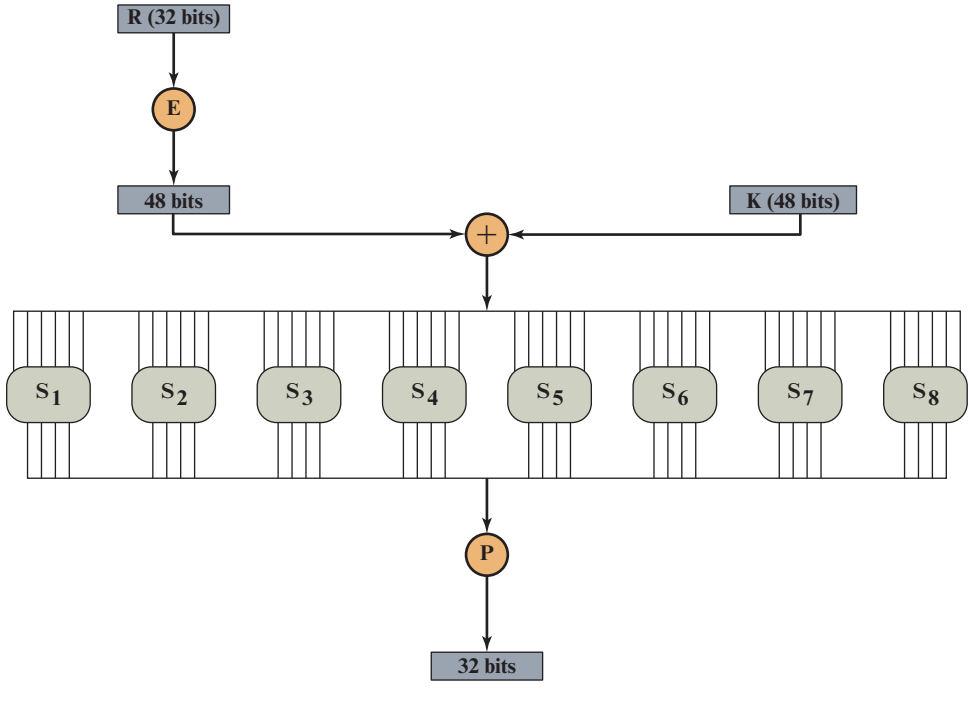


Figure C.3 Calculation of F(R, K)

Table C.2 Definition of DES S-Boxes

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (K_i). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

```
... efgh ijkl mnop ...
```

this becomes

```
... defghi hijklm lmnopq ...
```

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S-box immediately affects as many others as possible.

Key Generation Returning to Figures C.1 and C.2, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table C.3a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table C.3b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table C.3d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table C.3c), which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

Table C.3 DES Key Schedule Calculation

(a) Input Key							
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b) Permuted Choice One (PC-1)						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts															
Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.