Chapter Three
Block Ciphers and the Data Encryption Standard

3.1. Block Cipher Principles

Most 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.

3.1.1. 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. 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. Using some of the modes of operation explained in next Chapter, 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 focus on block ciphers.

3.1.2 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 transformation for n = 2.

Reversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Mapping	
Plaintext	**Ciphertext**
00	11
01	10
10	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 $2^n!$.

Figure 3.1 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 3.1. 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].

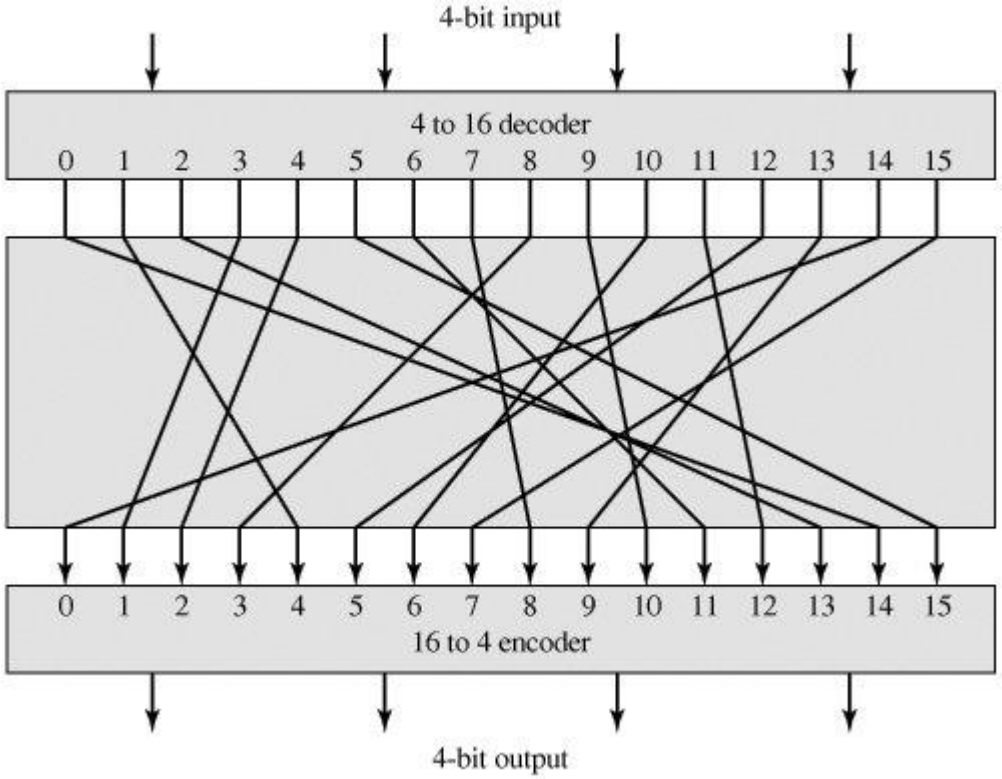

Figure 3.1. General n-bit-n-bit Block Substitution (shown with n = 4)

Table 3.1. Encryption and Decryption Tables for Substitution Cipher of Figure 3.4

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010

Table 3.1. Encryption and Decryption Tables for Substitution Cipher of Figure 3.4\	
Plaintext	**Ciphertext**
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111
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

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.

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 3.1, which defines one particular reversible mapping from 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) x (16 rows) = 64 bits. In general, for an n-bit ideal block cipher, the length of the key defined in this fashion is n x 2^n bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length is 64 x $2^{64} = 2^{70} \approx 10^{21}$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 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 2, applied to binary data rather than characters. As we saw in Chapter 2, a simple linear system such as this is quite vulnerable.

3.1.3. 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. In fact, this is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates confusion and diffusion functions [SHAN49]. 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 many significant symmetric block ciphers currently in use.

3.1.4 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, or 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 3.1) is such a cipher, but as we have seen, is impractical.
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 $M = m_1, m_2, m_3,...$ of characters with an averaging operation:

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

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.
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.

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.

3.1.5. Feistel Cipher Structure

Figure 3.2 depicts the 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, L_0 and R_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 L_{i-1} and R_{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.
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. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

Note: 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.

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.

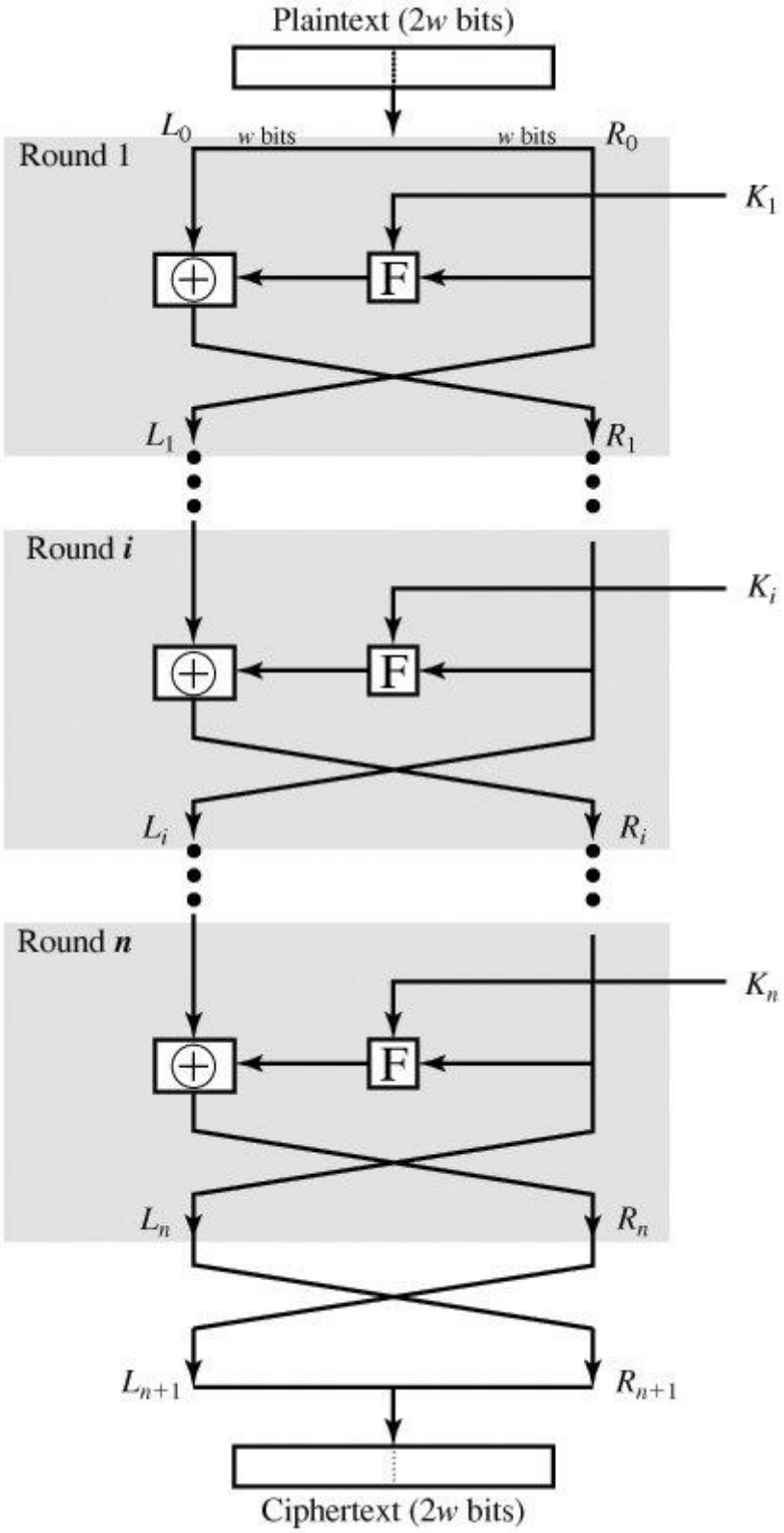

Figure 3.2. Classical Feistel Network

- 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: 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.

3.1.6. 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, consider Figure 3.3, which 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 (the result would be the same for any number of rounds). 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 $LE_i||RE_i$ (L_i concatenated with R_i). Then the corresponding input to the (16 i)th decryption round is $RE_i||LE_i$ or, equivalently, $RD_{16-i}||LD_{16-i}$.

Let us walk through Figure 3.3 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.

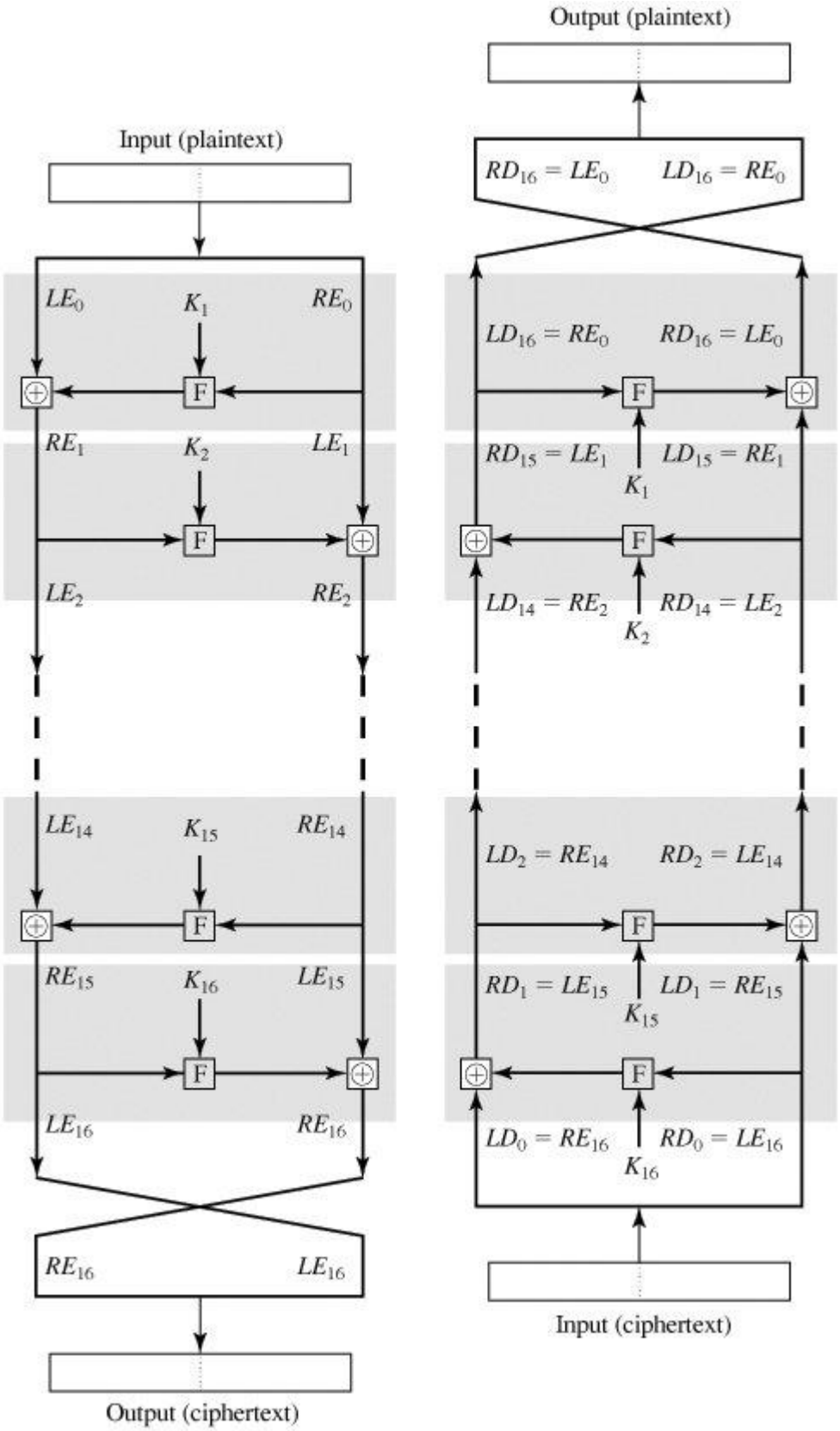

Figure 3.3. Feistel Encryption and Decryption

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

$$LE_{16} = RE_{15}$$
$$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16})$$

On the decryption side,

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

The XOR has the following properties:

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$D \oplus D = 0$$
$$E \oplus 0 = E$$

Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$. Therefore, the output of the first round of the decryption process is $LE_{15}||RE_{15}$, 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,

$$LE_i = RE_{i-1}$$
$$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

Rearranging terms,

$$RE_{i-1} = LE_i$$
$$LE_{i-1} = RE_i \oplus F(RE_{i-1}, K_{i2} = RE_i \oplus F(LE_i, K_i)$$

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 3.3.

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.

3.2. The Data Encryption Standard

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted 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). For DES, 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.

The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.

In the late 1960s, IBM set up a research project in computer cryptography led by Horst Feistel. The project concluded in 1971 with the development of an algorithm with the designation LUCIFER [FEIS73], which was sold to Lloyd's of London for use in a cash-dispensing system, also developed by IBM. LUCIFER is a Feistel block cipher that operates on blocks of 64 bits, using a key size of 128 bits. Because of the promising results produced by the LUCIFER project, IBM embarked on an effort to develop a marketable commercial encryption product that ideally could be implemented on a single chip. The effort was headed by Walter Tuchman and Carl Meyer, and it involved not only IBM researchers but also outside consultants and technical advice from NSA. The outcome of this effort was a refined version of LUCIFER that was more resistant to cryptanalysis but that had a reduced key size of 56 bits, to fit on a single chip.

In 1973, the National Bureau of Standards (NBS) issued a request for proposals for a national cipher standard. IBM submitted the results of its Tuchman-Meyer project. This was by far the best algorithm proposed and was adopted in 1977 as the Data Encryption Standard.

Before its adoption as a standard, the proposed DES was subjected to intense criticism, which has not subsided to this day. Two areas drew the critics' fire. First, the key length in IBM's original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits, an enormous reduction in key size of 72 bits. Critics feared that this key length was too short to withstand brute-force attacks. The second area of concern was that the design criteria for the internal structure of DES, the S-boxes, were classified. Thus, users could not be sure that the internal structure of DES was free of any hidden weak points that would enable NSA to decipher messages without benefit of the key. Subsequent events, particularly the recent work on differential cryptanalysis, seem to indicate that DES has a very strong internal structure. Furthermore, according to IBM participants, the only changes that were made to the proposal were changes to the S-boxes, suggested by NSA, that removed vulnerabilities identified in the course of the evaluation process.

Whatever the merits of the case, DES has flourished and is widely used, 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 only be used 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. Because the underlying encryption and decryption algorithms are the same for DES and triple DES, it remains important to understand the DES cipher.

3.2.1 DES Encryption

The overall scheme for DES encryption is illustrated in Figure 3.4. 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.
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.

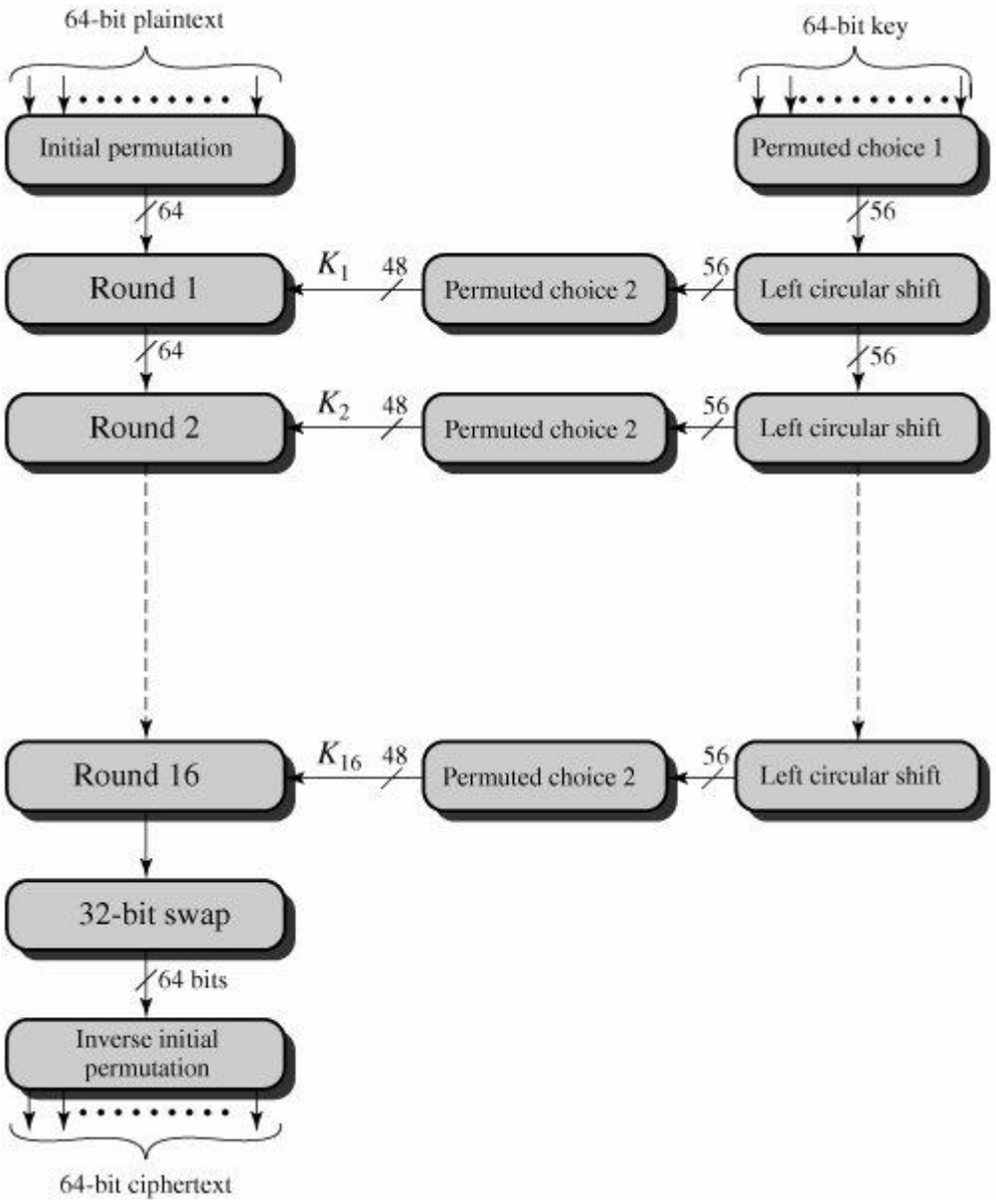

Figure 3.4. General Depiction of DES Encryption Algorithm

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 3.2.

The right-hand portion of Figure 3.4 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.

3.2.2. Initial Permutation

The initial permutation and its inverse are defined by tables, as shown in Tables 3.2a and 3.2b, 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 3.2. 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 M_i is a binary digit. Then the permutation X = 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 $Y = IP^{-1}(X) = IP^{-1}(IP(M))$, it can be seen that the original ordering of the bits is restored.

3.2.3. Details of Single Round

Figure 3.5 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). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

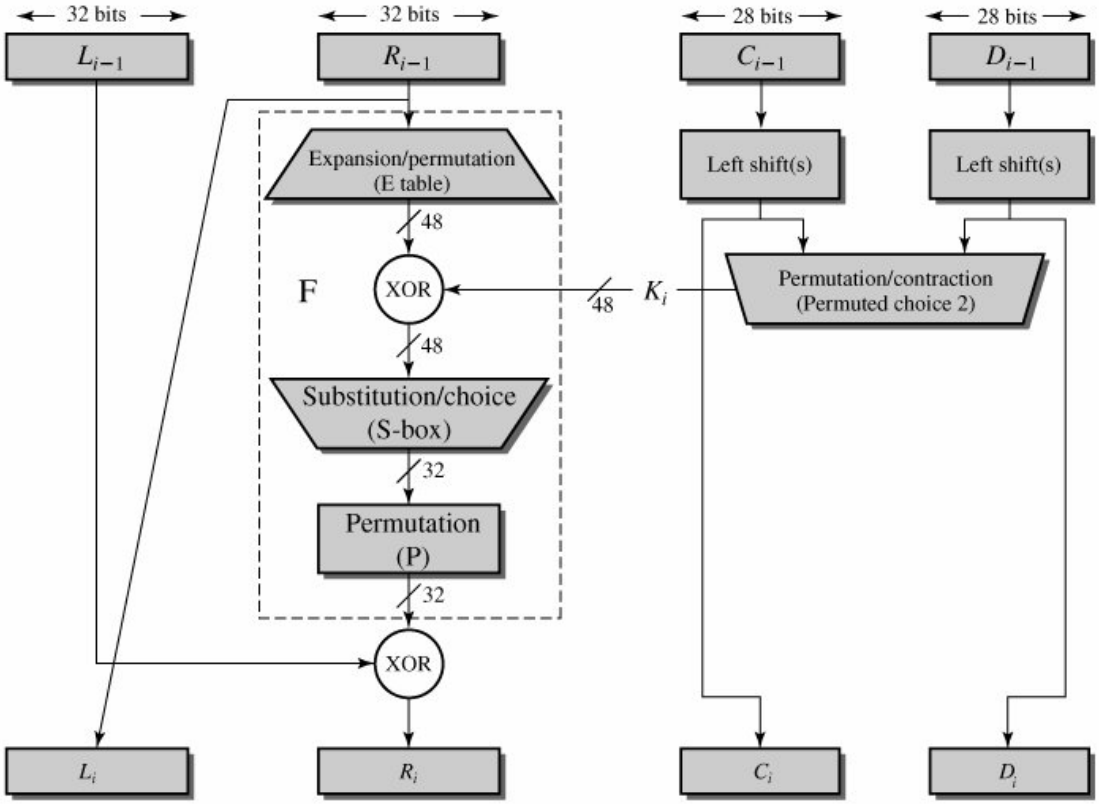

Figure 3.5. Single Round of DES Algorithm

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 3.2c). 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 3.2d.

The role of the S-boxes in the function F is illustrated in Figure 3.6. 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 3.3, 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 sixteen 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.

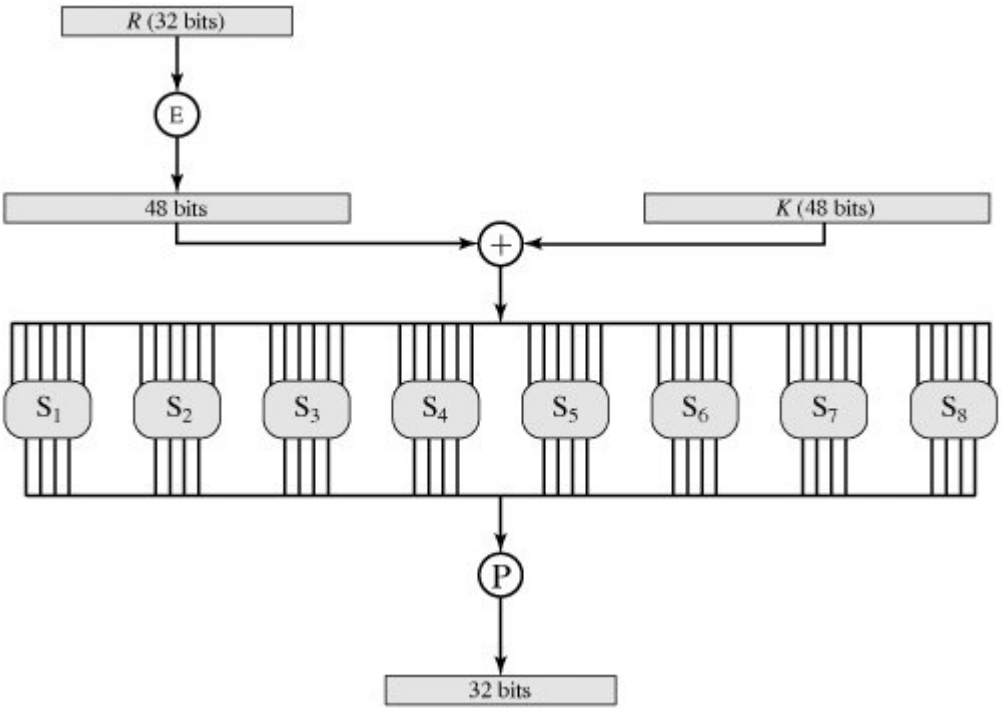

Figure 3.6. Calculation of F(R, K)

Each row of an S-box defines a general reversible substitution. Figure 3.1 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S_1.

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.

Table 3.3. 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

3.2.4. Key Generation

Returning to Figures 3.4 and 3.5, 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 3.4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b). 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 3.4d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

Table 3.4. 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

3.2.5. DES Decryption

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

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

DES exhibits a strong avalanche effect. Table 3.5 shows some results taken from [KONH81]. In Table 3.5a, two plaintexts that differ by one bit were used:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

with the key

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

Table 3.5. Avalanche Effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	**Number of bits that differ**	**Round**	**Number of bits that differ**
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30

6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

The Table 3.5a shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions.
Table 3.5b shows a similar test in which a single plaintext is input:
01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100
with two keys that differ in only one bit position:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100
0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

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.

3.3. 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.

3.3.1. 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}. 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 (see Table 2.2) 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.

DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker [EFF98]. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that there is more to a key-search 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. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

3.3.2. 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. 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 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

3.4. 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. It is useful to begin this discussion by looking at the published design criteria used in the DES effort. Then we look at three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling.

3.4.1. DES Design Criteria

The criteria used in the design of DES, as reported in [COPP94], focused on the design of the S-boxes and on the P function that takes the output of the S boxes (Figure 3.6). The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.
2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.
3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.
4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.
5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.
6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
7. This is a criterion similar to the previous one, but for the case of three S-boxes.

Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES. If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows:

1. The four output bits from each S-box at round i are distributed so that two of them affect (provide input for) "middle bits" of round (i + 1) and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.
2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.
3. For two S-boxes j, k, if an output bit from S_j affects a middle bit of S_k on the next round, then an output bit from S_k cannot affect a middle bit of S_j. This implies that for j = k, an output bit from S_j must not affect a middle bit of S_j. These criteria are intended to increase the diffusion of the algorithm.

3.4.2. 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 brute-force key search.

[9] Recall that 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 $2^{55.1}$.

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.

3.4.3. Design of Function F

The heart of a Feistel block cipher is the function F. As we have seen, in DES, this function relies on the use of S-boxes. This is also the case for most other symmetric block ciphers, as we shall see in Chapter 4. However, we can make some general comments about the criteria for designing F. After that, we look specifically at S-box design.

3.4.4. Design Criteria for F

The function F 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 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.

3.4.5. S-Box Design

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design. The papers are almost too numerous to count. Here we mention some general principles. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions.

A good summary of S-box design studies through early 1996 can be found in [SCHN96].

One obvious characteristic of the S-box is its size. An n x m S-box has n input bits and m output bits. DES has 6 x 4 S-boxes. Blowfish, has 8 x 32 S-boxes. Larger S-boxes, by and large, are more resistant to differential and linear cryptanalysis [SCHN96]. On the other hand, the larger the dimension n, the (exponentially) larger the lookup table. Thus, for practical reasons, a limit of n equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly.

S-boxes are typically organized in a different manner than used in DES. An n x m S-box typically consists of 2^n rows of m bits each. The n bits of input select one of the rows of the S-box, and the m bits in that row are the output. For example, in an 8 x 32 S-box, if the input is 00001001, the output consists of the 32 bits in row 9 (the first row is labeled row 0).

Mister and Adams [MIST96] propose a number of criteria for S-box design. Among these are that the S-box should satisfy both SAC and BIC. They also suggest that all linear combinations of S-box columns should be bent. Bent functions are a special class of Boolean functions that are highly nonlinear according to certain mathematical criteria [ADAM90]. There has been increasing interest in designing and analyzing S-boxes using bent functions.

A related criterion for S-boxes is proposed and analyzed in [HEYS95]. The authors define the guaranteed avalanche (GA) criterion as follows: An S-box satisfies GA of order π if, for a 1-bit input change, at least π output bits change. The authors conclude that a GA in the range of order 2 to order 5 provides strong diffusion characteristics for the overall encryption algorithm.

For larger S-boxes, such as 8×32, the question arises as to the best method of selecting the S-box entries in order to meet the type of criteria we have been discussing. Nyberg, who has written a lot about the theory and practice of S-box design, suggests the following approaches (quoted in [ROBS95b]):

- Random: Use some pseudorandom number generation or some table of random digits to generate the entries in the S-boxes. This may lead to boxes with undesirable characteristics for small sizes (e.g., 6 x 4) but should be acceptable for large S-boxes (e.g., 8 x 32).
- Random with testing: Choose S-box entries randomly, then test the results against various criteria, and throw away those that do not pass.
- Human-made: This is a more or less manual approach with only simple mathematics to support it. It is apparently the technique used in the DES design. This approach is difficult to carry through for large S-boxes.
- Math-made: Generate S-boxes according to mathematical principles. By using mathematical construction, S-boxes can be constructed that offer proven

security against linear and differential cryptanalysis, together with good diffusion.

A variation on the first technique is to use S-boxes that are both random and key dependent. An example of this approach is Blowfish, which starts with S-boxes filled with pseudorandom digits and then alters the contents using the key. A tremendous advantage of key-dependent S-boxes is that, because they are not fixed, it is impossible to analyze the S-boxes ahead of time to look for weaknesses.

3.4.6. Key Schedule Algorithm

A final area of block cipher design, and one that has received less attention than S-box design, is the 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.

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

3.5. Block Cipher Modes of Operation

A block cipher algorithm is a basic building block for providing data security. To apply a block cipher in a variety of applications, four "modes of operation" have been defined by NIST (FIPS 81). In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The four modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used. As new applications and requirements have appeared, NIST has expanded the list of recommended modes to five in Special Publication 800-38A. These modes are intended for use with any symmetric block cipher, including triple DES and AES. The modes are summarized in Table 3.6 and described briefly in the remainder of this section.

3.5.1. Electronic Codebook Mode

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 3.7). The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure 3.7, the plaintext (padded as necessary) consists of a sequence of b-bit blocks, P_1, P_2,..., P_N; the corresponding sequence of ciphertext blocks is C_1, C_2,..., C_N.

Table 3.6. Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed j bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

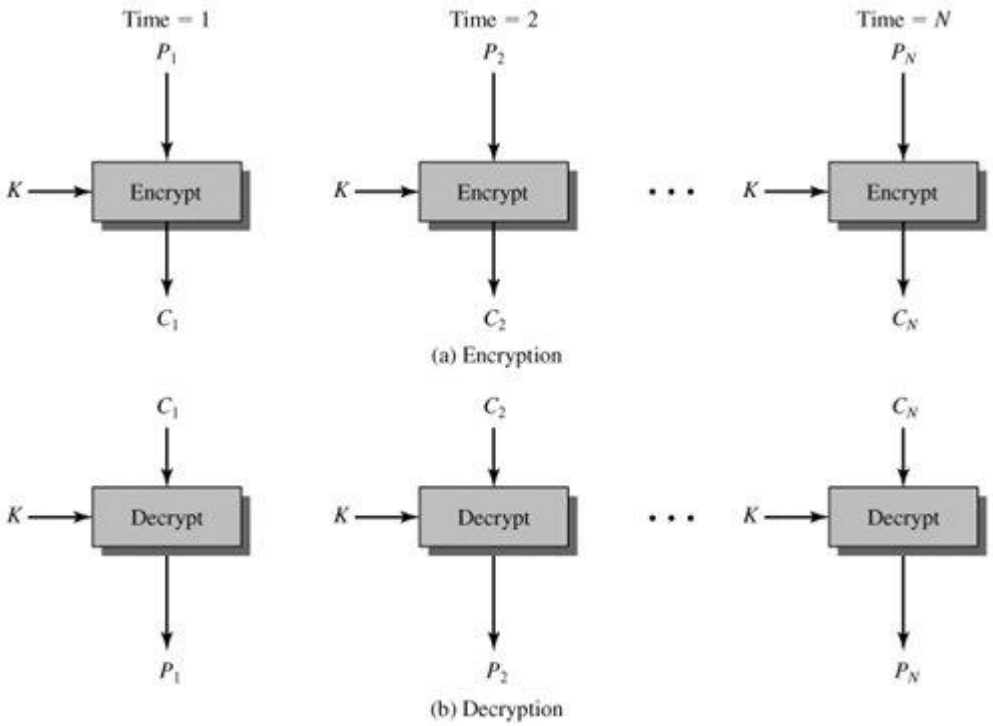

Figure 3.7. Electronic Codebook (ECB) Mode

The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use.

The most significant characteristic of ECB is that the same b-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with. If the message has repetitive elements, with a period of repetition a multiple of b bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

3.5.2. Cipher Block Chaining Mode

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode (Figure 3.8). In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

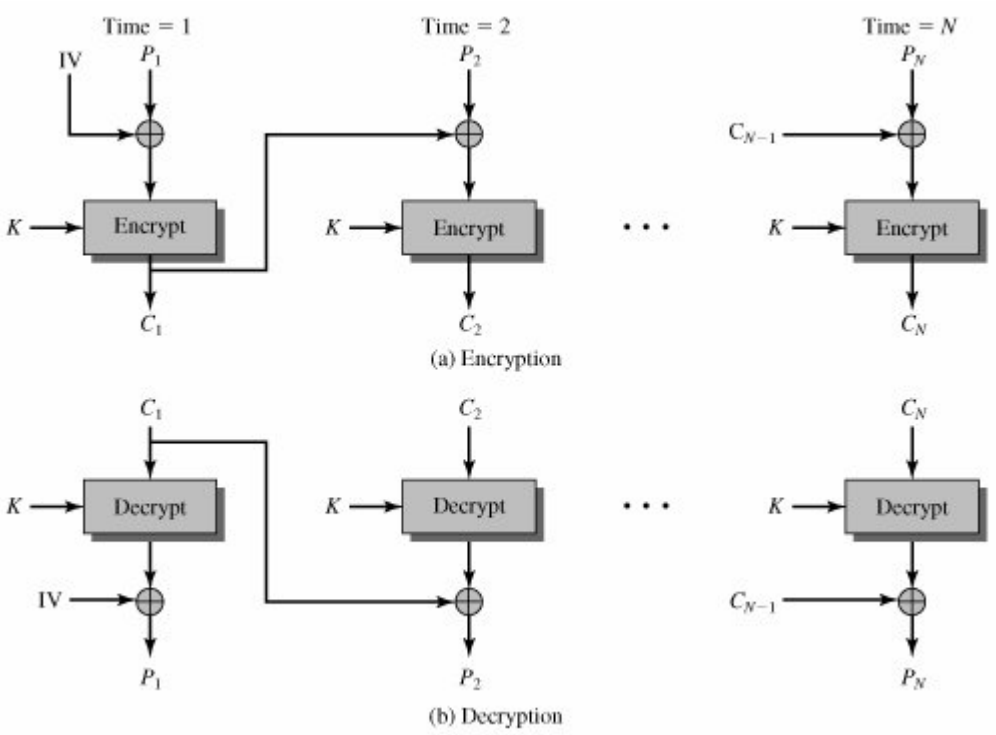

Figure 3.8. Cipher Block Chaining (CBC) Mode

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_j = E(K, [C_{j-1} \oplus P_j])$$

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j]))$$

$$D(K, C_j) = C_{j-1} \oplus P_j$$

$$C_{j-1} \oplus D(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block.

The IV must be known to both the sender and receiver but be unpredictable by a third party. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider the following:

$$C_1 = E(K, [IV \oplus P_1])$$

$$P_1 = IV \oplus D(K, C_1)$$

Now use the notation that X[i] denotes the ith bit of the b-bit quantity X. Then

$$P_1[i] = IV[i] \oplus D(K, C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$$

where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed.

In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than b bits.

In addition to its use to achieve confidentiality, the CBC mode can be used for authentication..

3.5.3. Cipher Feedback Mode

The DES scheme is essentially a block cipher technique that uses b-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a cipher text output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

Figure 3.9 depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than units of b bits, the plaintext is divided into segments of s bits.

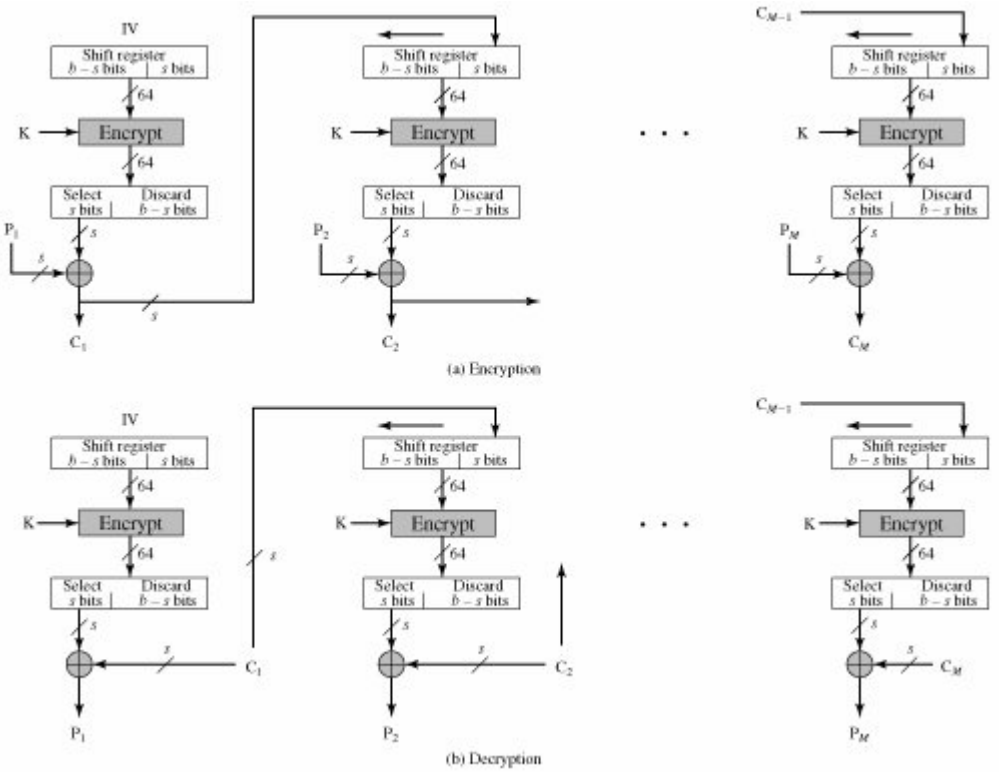

Figure 3.9. s-bit Cipher Feedback (CFB) Mode

First, consider encryption. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let $S_s(X)$ be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus S_s[E(K, IV)]$$

Therefore,

$$P_1 = C_1 \oplus S_s[E(K, IV)]$$

The same reasoning holds for subsequent steps in the process.

3.5.4. Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in Figure 3.10. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.

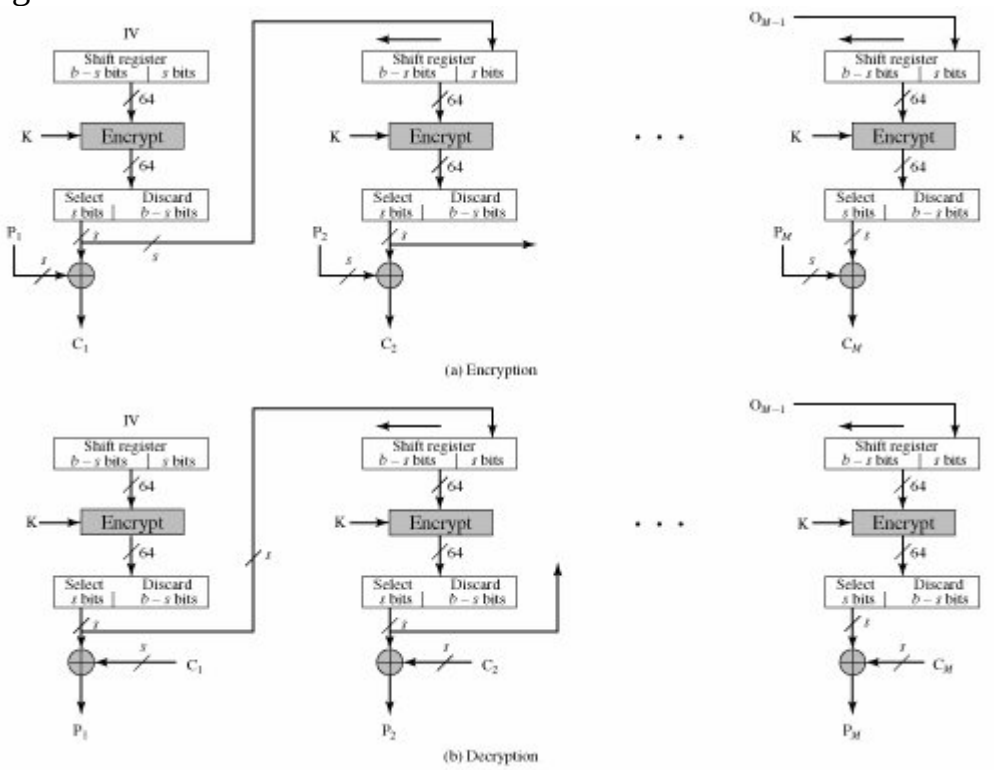

Figure 3.10. s-bit Output Feedback (OFB) Mode

One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C_1 only the recovered value of is P_1 affected;

subsequent plaintext units are not corrupted. With CFB, C_1 also serves as input to the shift register and therefore causes additional corruption downstream.

The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB. Consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintext. Thus, controlled changes to the recovered plaintext can be made. This may make it possible for an opponent, by making the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code. For a further discussion, see [VOYD83].

3.5.5. Counter Mode

Although interest in the counter mode (CTR) has increased recently, with applications to ATM (asynchronous transfer mode) network security and IPSec (IP security), this mode was proposed early on (e.g., [DIFF79]).

Figure 3.11 depicts the CTR mode. A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

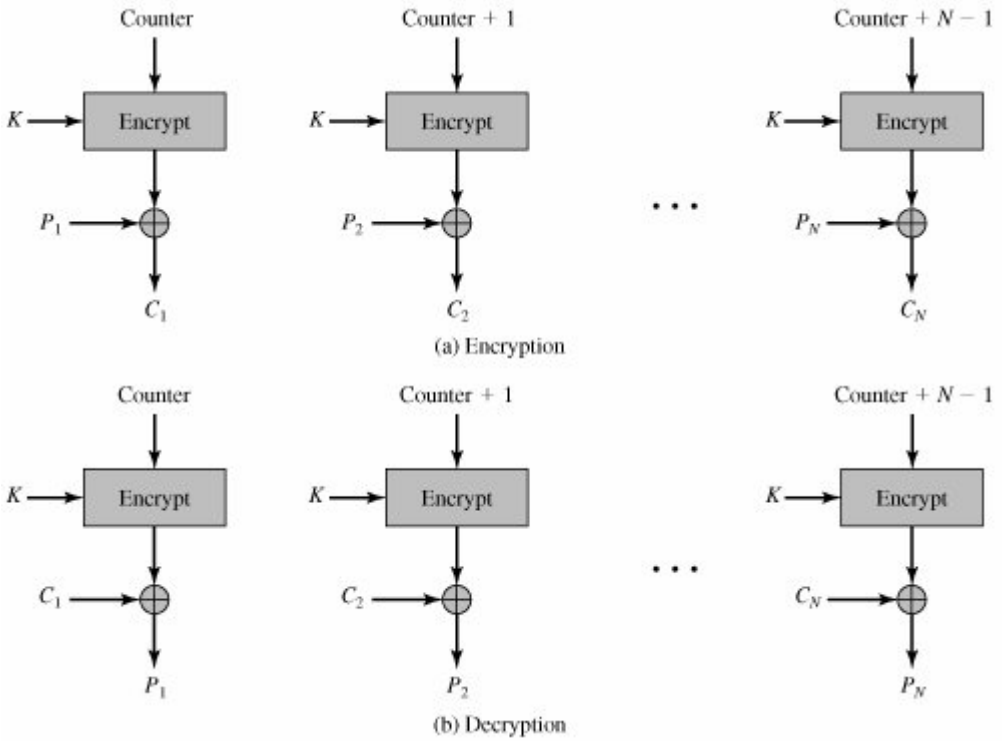

Figure 3.11. Counter (CTR) Mode

[LIPM00] lists the following advantages of CTR mode:

- Hardware efficiency: Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.
- Software efficiency: Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
- Preprocessing: The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in Figure 3.11. When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.
- Random access: The ith block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block C_i cannot be computed until the i - 1 prior block are computed. There may be applications in which a ciphertext is stored and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- Provable security: It can be shown that CTR is at least as secure as the other modes discussed in this section.
- Simplicity: Unlike ECB and CBC modes, CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.

3.6. Differential and Linear Cryptanalysis

For most of its life, the prime concern with DES has been its vulnerability to brute-force attack because of its relatively short (56 bits) key length. However, there has also been interest in finding cryptanalytic attacks on DES. With the increasing popularity of block ciphers with longer key lengths, including triple DES, brute-force attacks have become increasingly impractical. Thus, there has been increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers. In this section, we provide a brief overview of the two most powerful and promising approaches: differential cryptanalysis and linear cryptanalysis.

3.6.1. Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. In this section, we discuss the technique and its applicability to DES.

History

Differential cryptanalysis was not reported in the open literature until 1990. The first published effort appears to have been the cryptanalysis of a block cipher called FEAL by Murphy [MURP90]. This was followed by a number of papers by Biham and Shamir, who demonstrated this form of attack on a variety of encryption algorithms and hash functions; their results are summarized in [BIHA93].

The most publicized results for this approach have been those that have application to DES. Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 2^{55} complexity. The scheme, as reported in [BIHA93], can successfully cryptanalyze DES with an effort on the order of 2^{47} encryptions, requiring 2^{47} chosen plaintexts. Although 2^{47} is certainly significantly less than 2^{55} the need for the adversary to find 2^{47} chosen plaintexts makes this attack of only theoretical interest.

Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The reason, according to a member of the IBM team that designed DES [COPP94], is that differential cryptanalysis was known to the team as early as 1974. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P. As evidence of the impact of these changes, consider these comparable results reported in [BIHA93]. Differential cryptanalysis of an eight-round LUCIFER algorithm requires only 256 chosen plaintexts, whereas an attack on an eight-round version of DES requires 2^{14} chosen plaintexts.

Differential Cryptanalysis Attack

The differential cryptanalysis attack is complex; [BIHA93] provides a complete description. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block. Here, we provide a brief overview so that you can get the flavor of the attack.

We begin with a change in notation for DES. Consider the original plaintext block m to consist of two halves m_0,m_1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block $m_1(2 \leq i \leq 17)$, then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i), \quad i = 1, 2, ..., 16$$

In differential cryptanalysis, we start with two messages, m and m', with a known XOR difference $\Delta m = m \oplus m'$, and consider the difference between the intermediate message halves: $m_i = m_i \oplus m_i'$ Then we have:

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)]\end{aligned}$$

Now, suppose that many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used. To put this more precisely, let us say that X may cause Y with probability p, if for a fraction p of the pairs in which the input XOR is X, the output XOR equals Y. We want to suppose that there are a number of values of X that have high probability of causing a particular output difference. Therefore, if we know Δm_{i-1} and Δm_i with high probability, then we know Δm_{i+1} with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f.

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages m and m' with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext. Actually, there are two probable patterns of differences for the two 32-bit halves: $(\Delta m_{17} \| m_{16})$. Next, we submit m and m' for encryption to determine the actual difference under the unknown key and compare the result to the probable difference. If there is a match,

$E(K, m) \oplus E(K, m') = (\Delta m_{17} \| m_{16})$

then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

Figure 3.12, based on a figure in [BIHA93], illustrates the propagation of differences through three rounds of DES. The probabilities shown on the right refer to the probability that a given set of intermediate differences will appear as a function of the input differences. Overall, after three rounds the probability that the output difference is as shown is equal to 0.25 x 1 x 0.25 = 0.0625.

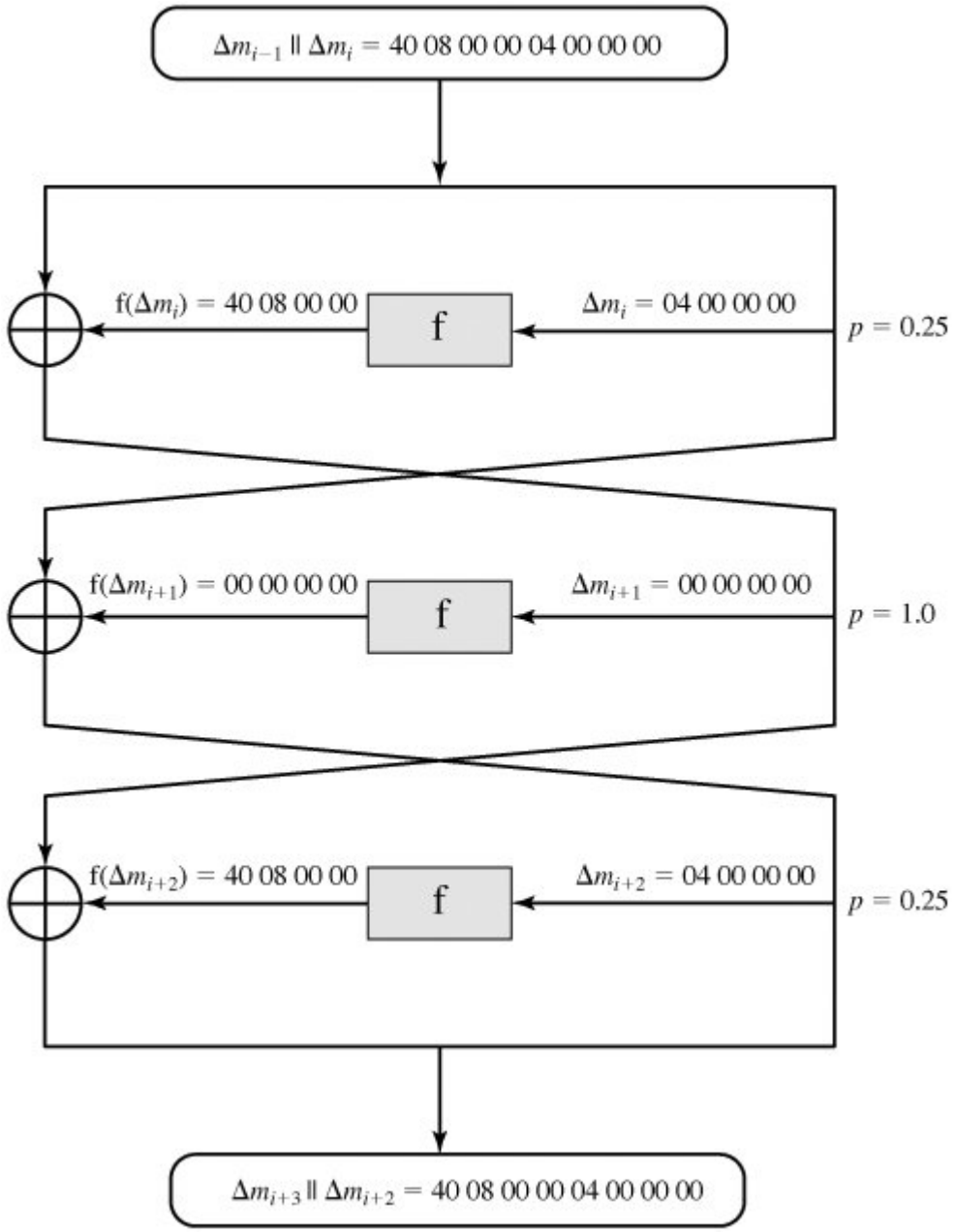

Figure 3.12. Differential Propagation through Three Round of DES (numbers in hexadecimal)

3.6.2. Linear Cryptanalysis

A more recent development is linear cryptanalysis, described in [MATS93]. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given 2^{43} known plaintexts, as compared to 2^{47} chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by other groups to validate the linear cryptanalytic approach.

We now give a brief summary of the principle on which linear cryptanalysis is based. For a cipher with n-bit plaintext and ciphertext blocks and an m-bit key, let the plaintext block be labeled P[1], ... P[n], the cipher text block C[1], ... C[n], and the key K[1], ... K[m]. Then define

$$A[i, j, ..., k] = A[i] \oplus A[j] \oplus ... \oplus A[k]$$

The objective of linear cryptanalysis is to find an effective linear equation of the form:

$$P[\alpha_1, \alpha_2, ..., \alpha_a] \oplus C[\beta_1, \beta_2, ..., \beta_b] = K[\gamma_1, \gamma_2, ..., \gamma_c]$$

(where x = 0 or 1; $1 \leq a, b \leq n$, $1 \leq c \leq m$, and where the α, β and γ terms represent fixed, unique bit locations) that holds with probability $p \neq 0.5$. The further p is from 0.5, the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is 0 more than half the time, assume $K[\gamma_1, \gamma_2, ..., \gamma_c] = 0$. If it is 1 most of the time, assume $K[\gamma_1, \gamma_2, ..., \gamma_c] = 1$. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.