6

Data Encryption Standard (DES)

Objectives

In this chapter, we discuss the Data Encryption Standard (DES), the modern symmetric-key block cipher. The following are our main objectives for this chapter:

- ☞ To review a short history of DES
- ☞ To define the basic structure of DES
- ☞ To describe the details of building elements of DES
- ☞ To describe the round keys generation process
- ☞ To analyze DES

The emphasis is on how DES uses a Feistel cipher to achieve confusion and diffusion of bits from the plaintext to the ciphertext.

6.1 INTRODUCTION

The **Data Encryption Standard (DES)** is a symmetric-key block cipher published by the **National Institute of Standards and Technology (NIST).**

6.1.1 History

In 1973, NIST published a request for proposals for a national symmetric-key cryptosystem. A proposal from IBM, a modification of a project called Lucifer, was accepted as DES. DES was published in the *Federal Register* in March 1975 as a draft of the **Federal Information Processing Standard (FIPS).**

After the publication, the draft was criticized severely for two reasons. First, critics questioned the small key length (only 56 bits), which could make the cipher vulnerable to brute-force attack. Second, critics were concerned about some hidden design behind the internal structure of DES. They were suspicious that some part of the structure (the S-boxes) may have some hidden trapdoor that would allow the **National Security Agency (NSA)** to decrypt the messages without the need for the key. Later IBM designers mentioned that the internal structure was designed to prevent differential cryptanalysis.

DES was finally published as FIPS 46 in the *Federal Register* in January 1977. NIST, however, defines DES as the standard for use in unclassified applications. DES has been the most widely used symmetric-key block cipher since its publication. NIST later issued a new standard (FIPS 46-3) that recommends the use of triple DES (repeated DES cipher three times) for future applications. As we will see in Chapter 7, AES, the recent standard, is supposed to replace DES in the long run.

6.1.2 Overview

DES is a block cipher, as shown in Fig. 6.1.

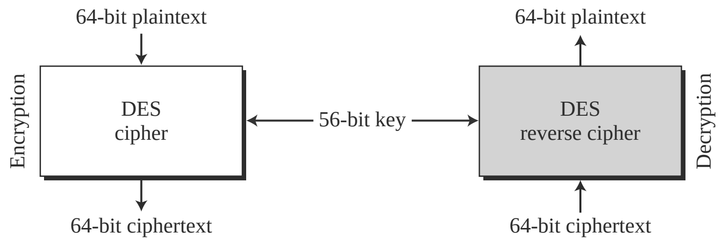


Fig. 6.1 *Encryption and decryption with DES*

At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

6.2 DES STRUCTURE

Let us concentrate on encryption; later we will discuss decryption. The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm described later in the chapter. Figure 6.2 shows the elements of DES cipher at the encryption site.

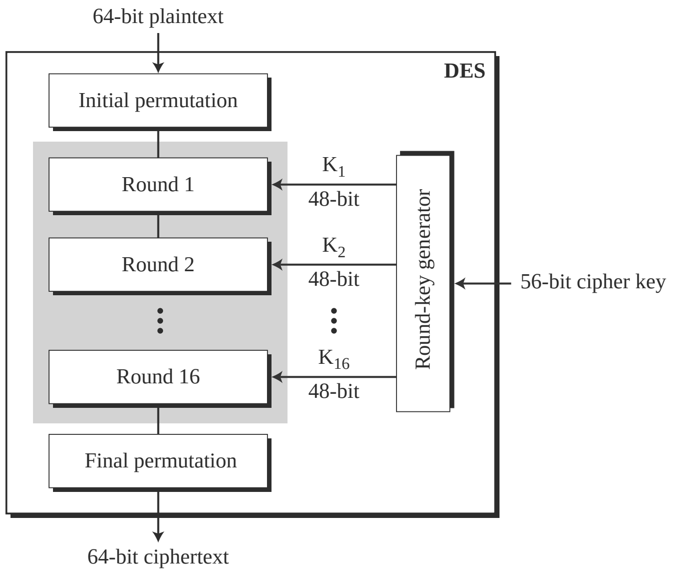


Fig. 6.2 *General structure of DES*

6.2.1 Initial and Final Permutations

Figure 6.3 shows the initial and final permutations (P-boxes). Each of these permutations takes a 64-bit input and permutes them according to a predefined rule. We have shown only a few input ports and the corresponding output ports. These permutations are keyless straight permutations that are the inverse of each other. For example, in the initial permutation, the 58th bit in the input becomes the first bit in the output. Similarly, in the final permutation, the first bit in the input becomes the 58th bit in the output. In other words, if the rounds between these two permutations do not exist, the 58th bit entering the initial permutation is the same as the 58th bit leaving the final permutation.

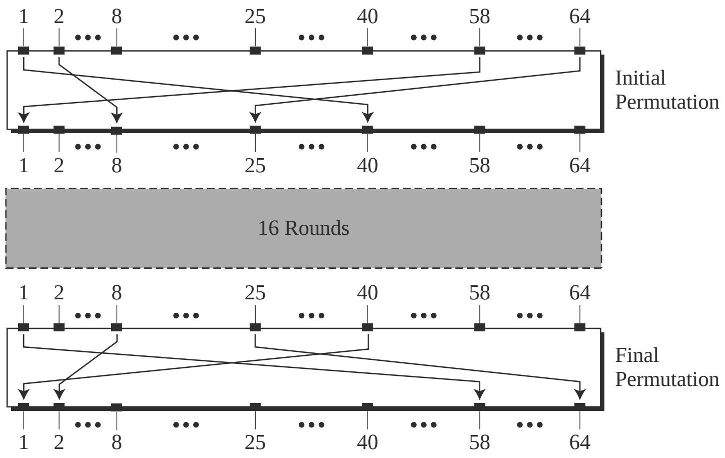


Fig. 6.3 *Initial and final permutation steps in DES*

The permutation rules for these P-boxes are shown in Table 6.1. Each side of the table can be thought of as a 64-element array. Note that, as with any permutation table we have discussed so far, the value of each element defines the input port number, and the order (index) of the element defines the output port number.

Table 6.1 *Initial and final permutation tables*

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

These two permutations have no cryptography significance in DES. Both permutations are keyless and predetermined. The reason they are included in DES is not clear and has not been revealed by the DES designers. The guess is that DES was designed to be implemented in hardware (on chips) and that these two complex permutations may thwart a software simulation of the mechanism.

Example 6.1 Find the output of the initial permutation box when the input is given in hexadecimal as:

0x0002 0000 0000 0001

Solution The input has only two 1s (bit 15 and bit 64); the output must also have only two 1s (the nature of straight permutation). Using Table 6.1, we can find the output related to these two bits. Bit 15 in the input becomes bit 63 in the output. Bit 64 in the input becomes bit 25 in the output. So the output has only two 1s, bit 25 and bit 63. The result in hexadecimal is

0x0000 0080 0000 0002

Example 6.2 Prove that the initial and final permutations are the inverse of each other by finding the output of the final permutation if the input is

0x0000 0080 0000 0002

Solution Only bit 25 and bit 64 are 1s; the other bits are 0s. In the final permutation, bit 25 becomes bit 64 and bit 63 becomes bit 15. The result

0x0002 0000 0000 0001

The initial and final permutations are straight D-boxes that are inverses of each other and hence are permutations. They have no cryptography significance in DES.

6.2.2 Rounds

DES uses 16 rounds. Each round of DES is a Feistel cipher, as shown in Fig. 6.4.

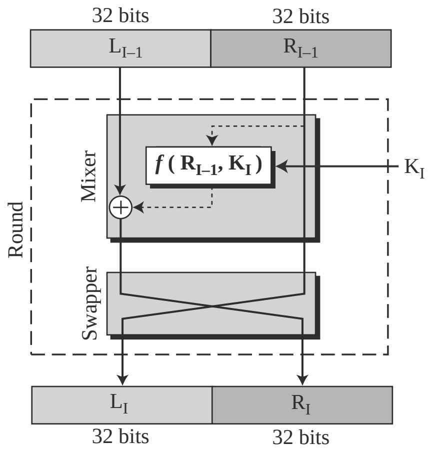


Fig. 6.4 *A round in DES (encryption site)*

The round takes L_{I-1} and R_{I-1} from previous round (or the initial permutation box) and creates L_I and R_I, which go to the next round (or final permutation box). As we discussed in Chapter 5, we can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All noninvertible elements are collected inside the function $f(R_{I-1}, K_I)$.

DES Function

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output. This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box as shown in Fig. 6.5.

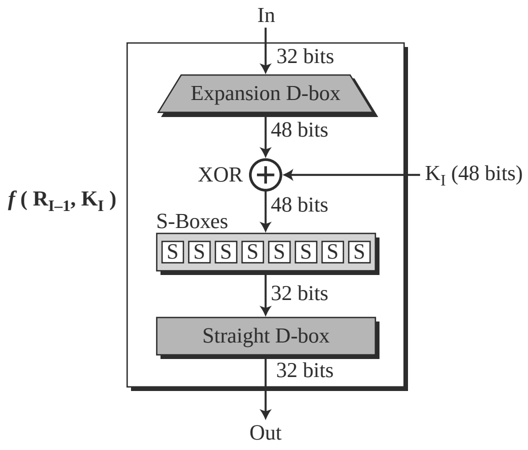


Fig. 6.5 *DES function*

Expansion D-box Since R_{I-1} is a 32-bit input and K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits. R_{I-1} is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits. This expansion permutation follows a predetermined rule. For each section, input bits 1, 2, 3, and 4 are copied to output bits 2, 3, 4, and 5, respectively. Output bit 1 comes from bit 4 of the previous section; output bit 6 comes from bit 1 of the next section. If sections 1 and 8 can be considered adjacent sections, the same rule applies to bits 1 and 32. Fig. 6.6 shows the input and output in the expansion permutation.

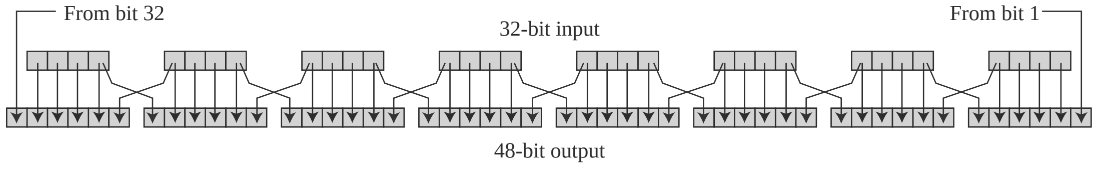


Fig. 6.6 *Expansion permutation*

Although the relationship between the input and output can be defined mathematically, DES uses Table 6.2 to define this D-box. Note that the number of output ports is 48, but the value range is only 1 to 32. Some of the inputs go to more than one output. For example, the value of input bit 5 becomes the value of output bits 6 and 8.

Table 6.2 *Expansion D-box table*

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

Whitener (XOR) After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. See Fig. 6.7.

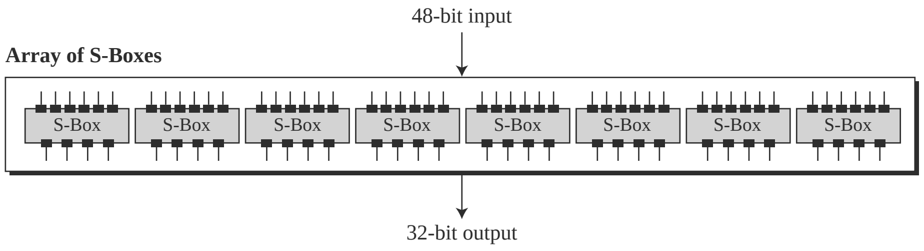


Fig. 6.7 *S-boxes*

The 48-bit data from the second operation is divided into eight 6-bit chunks, and each chunk is fed into a box. The result of each box is a 4-bit chunk; when these are combined the result is a 32-bit text. The substitution in each box follows a pre-determined rule based on a 4-row by 16-column table. The combination of bits 1 and 6 of the input defines one of four rows; the combination of bits 2 through 5 defines one of the sixteen columns as shown in Fig. 6.8. This will become clear in the examples.

Because each S-box has its own table, we need eight tables, as shown in Tables 6.3 to 6.10, to define the output of these boxes. The values of the inputs (row number and column number) and the values of the outputs are given as decimal numbers to save space. These need to be changed to binary.

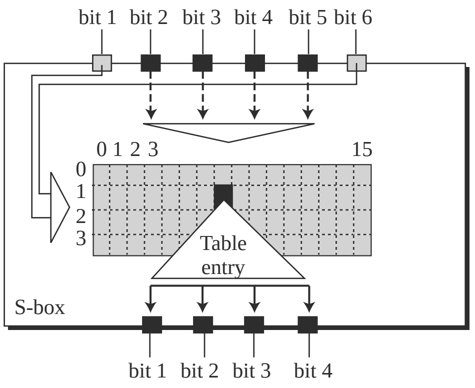


Fig. 6.8 *S-box rule*

Table 6.3 *S-box 1*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Table 6.4 *S-box 2*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Table 6.5 *S-box 3*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Table 6.6 *S-box 4*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	6	09	10	1	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Table 6.7 *S-box 5*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Table 6.8 *S-box 6*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

Table 6.9 *S-box 7*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Table 6.10 *S-box 8*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	8	13	15	12	09	09	03	05	06	11

Example 6.3 The input to S-box 1 is <u>1</u>0001<u>1</u>. What is the output?

Solution If we write the first and the sixth bits together, we get 11 in binary, which is 3 in decimal. The remaining bits are 0001 in binary, which is 1 in decimal. We look for the value in row 3, column 1, in Table 6.3 (S-box 1). The result is 12 in decimal, which in binary is 1100. So the input 100011 yields the output 1100.

Example 6.4 The input to S-box 8 is <u>0</u>0000<u>0</u>. What is the output?

Solution If we write the first and the sixth bits together, we get 00 in binary, which is 0 in decimal. The remaining bits are 0000 in binary, which is 0 in decimal. We look for the value in row 0, column 0, in Table 6.10 (S-box 8). The result is 13 in decimal, which is 1101 in binary. So the input 000000 yields the output 1101.

Final Permutation The last operation in the DES function is a permutation with a 32-bit input and a 32-bit output. The input/output relationship for this operation is shown in Table 6.11 and follows the same general rule as previous tables. For example, the seventh bit of the input becomes the second bit of the output.

Table 6.11 *Straight permutation table*

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

6.2.3 Cipher and Reverse Cipher

Using mixers and swappers, we can create the cipher and reverse cipher, each having 16 rounds. The cipher is used at the encryption site; the reverse cipher is used at the decryption site. The whole idea is to make the cipher and the reverse cipher algorithms similar.

First Approach To achieve this goal, one approach is to make the last round (round 16) different from the others; it has only a mixer and no swapper. This is done in Figure 6.9.

Although the rounds are not aligned, the elements (mixer or swapper) are aligned. We proved in Chapter 5 that a mixer is a self-inverse; so is a swapper. The final and initial permutations are also inverses of each other. The left section of the plaintext at the encryption site, L_0, is enciphered as L_{16} at the encryption site; L_{16} at the decryption is deciphered as L_0 at the decryption site. The situation is the same with R_0 and R_{16}.

A very important point we need to remember about the ciphers is that the round keys (K_1 to K_{16}) should be applied in the reverse order. At the encryption site, round 1 uses K_1 and round 16 uses K_{16}; at the decryption site, round 1 uses K_{16} and round 16 uses K_1.

In the first approach, there is no swapper in the last round.

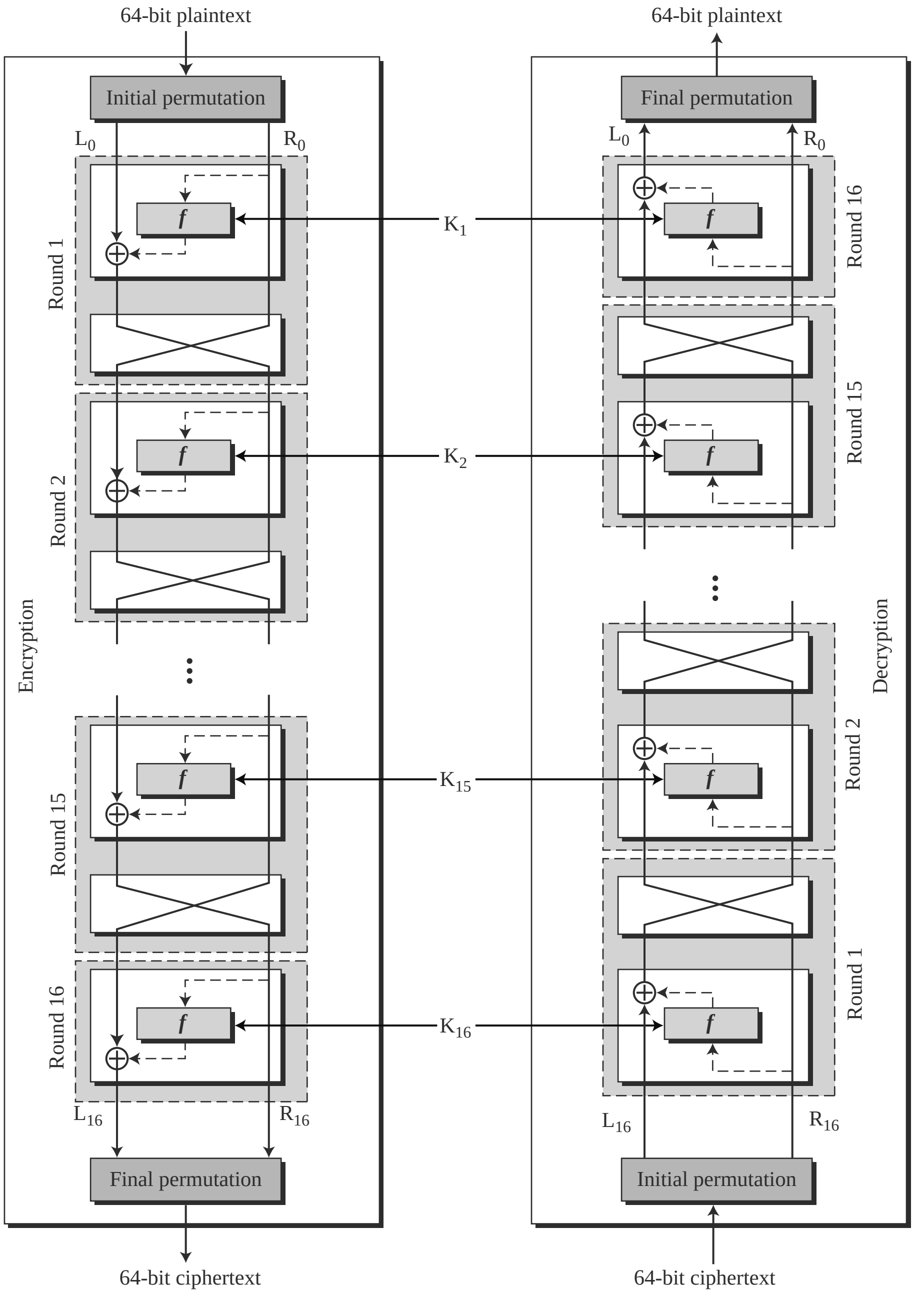


Fig. 6.9 *DES cipher and reverse cipher for the first approach*

Algorithm

Algorithm 6.1 gives the pseudocode for the cipher and four corresponding routines in the first approach. The codes for the rest of the routines are left as exercises.

Algorithm 6.1 Pseudocode for DES cipher

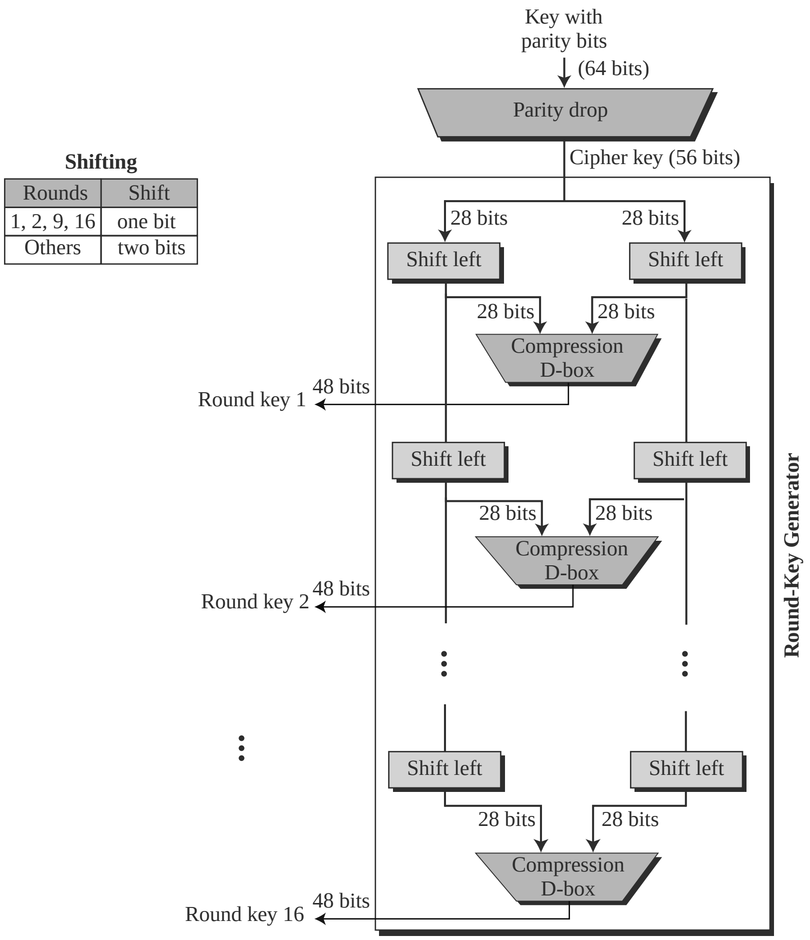
```
Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64])
{
    permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
    split (64, 32, inBlock, leftBlock, rightBlock)
    for (round = 1 to 16)
    {
        mixer (leftBlock, rightBlock, RoundKeys[round])
        if (round!=16) swapper (leftBlock, rightBlock)
    }
    combine (32, 64, leftBlock, rightBlock, outBlock)
    permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
mixer (leftBlock[48], rightBlock[48], RoundKey[48])
{
    copy (32, rightBlock, T1)
    function (T1, RoundKey, T2)
    exclusiveOr (32, leftBlock, T2, T3)
    copy (32, T3, rightBlock)
}
swapper (leftBlock[32], rigthBlock[32])
{
    copy (32, leftBlock, T)
    copy (32, rightBlock, leftBlock)
    copy (32, T, rightBlock)
}
function (inBlock[32], RoundKey[48], outBlock[32])
{
    permute (32, 48, inBlock, T1, ExpansionPermutationTable)
    exclusiveOr (48, T1, RoundKey, T2)
    substitute (T2, T3, SubstituteTables)
    permute (32, 32, T3, outBlock, StraightPermutationTable)
}
substitute (inBlock[32], outBlock[48], SubstitutionTables[8, 4, 16])
{
    for (i = 1 to 8)
    {
       row ← 2 × inBlock[i × 6 + 1] + inBlock [i × 6 + 6]
       col ← 8 × inBlock[i × 6 + 2] + 4 × inBlock[i × 6 + 3] +
              2 × inBlock[i × 6 + 4] + inBlock[i × 6 + 5]

       value = SubstitutionTables [i][row][col]

       outBlock[[i × 4 + 1] ← value / 8;   value ← value mod 8
       outBlock[[i × 4 + 2] ← value / 4;   value ← value mod 4
       outBlock[[i × 4 + 3] ← value / 2;   value ← value mod 2
       outBlock[[i × 4 + 4] ← value
    }
}
```

Alternative Approach In the first approach, round 16 is different from other rounds; there is no swapper in this round. This is needed to make the last mixer in the cipher and the first mixer in the reverse cipher aligned. We can make all 16 rounds the same by including one swapper to the 16th round and add an extra swapper after that (two swappers cancel the effect of each other). We leave the design for this approach as an exercise.

Key Generation The **round-key generator** creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in Fig. 6.10.


Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

Fig. 6.10 *Key generation*

Parity Drop The preprocess before key expansion is a compression transposition step that we call **parity bit drop.** It drops the parity bits (bits 8, 16, 24, 32, …, 64) from the 64-bit key and permutes the rest of the bits according to Table 6.12. The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop step (a compression D-box) is shown in Table 6.12.

Table 6.12 *Parity-bit drop table*

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Shift Left After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. Table 6.13 shows the number of shifts for each round.

Table 6.13 *Number of bit shifts*

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Compression D-box The compression D-box changes the 58 bits to 48 bits, which are used as a key for a round. The compression step is shown in Table 6.14.

Table 6.14 *Key-compression table*

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Algorithm Let us write a simple algorithm to create round keys from the key with parity bits. Algorithm 6.2 uses several routines from Algorithm 6.1. The new one is the shiftLeft routine, for which the code is given.

Algorithm 6.2 Algorithm for round-keys generation

```
Key_Generator (keyWithParities[64], RoundKeys[16, 48], ShiftTable[16])
{
   permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
   split (56, 28, cipherKey, leftKey, rightKey)
   for (round = 1 to 16)
   {
       shiftLeft (leftKey, ShiftTable[round])
       shiftLeft (rightKey, ShiftTable[round])
```

Algorithm 6.2 (Contd.)

```
            combine (28, 56, leftKey, rightKey, preRoundKey)
            permute (56, 48, preRoundKey, RoundKeys[round], KeyCompressionTable)
    }
}
shiftLeft (block[28], numOfShifts)
{
    for (i = 1 to numOfShifts)
    {
        T ← block[1]
        for (j = 2 to 28)
        {
            block [j−1] ← block [j]
        }
        block[28] ← T
    }
}
```

6.2.4 Examples

Before analyzing DES, let us look at some examples to see the how encryption and decryption change the value of bits in each round.

Example 6.5 We choose a random plaintext block and a random key, and determine what the ciphertext block would be (all in hexadecimal):

Plaintext: 123456ABCD132536 Key: AABB09182736CCDD
CipherText: C0B7A8D05F3A829C

Let us show the result of each round and the text created before and after the rounds. Table 6.15 first shows the result of steps before starting the round.

Table 6.15 *Trace of data for Example 6.5*

Plaintext: 123456ABCD132536

After initial permutation: 14A7D67818CA18AD
After splitting: L_0=14A7D678 R_0=18CA18AD

Round	*Left*	*Right*	*Round Key*
Round 1	18CA18AD	5A78E394	194CD072DE8C
Round 2	5A78E394	4A1210F6	4568581ABCCE
Round 3	4A1210F6	B8089591	06EDA4ACF5B5
Round 4	B8089591	236779C2	DA2D032B6EE3
Round 5	236779C2	A15A4B87	69A629FEC913
Round 6	A15A4B87	2E8F9C65	C1948E87475E
Round 7	2E8F9C65	A9FC20A3	708AD2DDB3C0
Round 8	A9FC20A3	308BEE97	34F822F0C66D
Round 9	308BEE97	10AF9D37	84BB4473DCCC

Table 6.15 (Contd.)

Round 10	10AF9D37	6CA6CB20	02765708B5BF
Round 11	6CA6CB20	FF3C485F	6D5560AF7CA5
Round 12	FF3C485F	22A5963B	C2C1E96A4BF3
Round 13	22A5963B	387CCDAA	99C31397C91F
Round 14	387CCDAA	BD2DD2AB	251B8BC717D0
Round 15	BD2DD2AB	CF26B472	3330C5D9A36D
Round 16	19BA9212	CF26B472	181C5D75C66D
After combination: 19BA9212CF26B472			
Ciphertext: C0B7A8D05F3A829C			*(after final permutation)*

The plaintext goes through the initial permutation to create completely different 64 bits (16 hexadecimal digit). After this step, the text is split into two halves, which we call L_0 and R_0. The table shows the result of 16 rounds that involve mixing and swapping (except for the last round). The results of the last rounds (L_{16} and R_{16}) are combined. Finally the text goes through final permutation to create the ciphertext.

Some points are worth mentioning here. First, the right section out of each round is the same as the left section out of the next round. The reason is that the right section goes through the mixer without change, but the swapper moves it to the left section. For example, R_1 passes through the mixer of the second round without change, but then it becomes L_2 because of the swapper. The interesting point is that we do not have a swapper at the last round. That is why R_{15} becomes R_{16} instead of becoming L_{16}.

Example 6.6 Let us see how Bob, at the destination, can decipher the ciphertext received from Alice using the same key. We have shown only a few rounds to save space. Table 6.16 shows some interesting points. First, the round keys should be used in the reverse order. Compare Table 6.15 and Table 6.16. The round key for round 1 is the same as the round key for round 16. The values of L_0 and R_0 during decryption are the same as the values of L_{16} and R_{16} during encryption. This is the same with other rounds. This proves not only that the cipher and the reverse cipher are inverses of each other in the whole, but also that each round in the cipher has a corresponding reverse round in the reverse cipher. The result proves that the initial and final permutation steps are also inverses of each other.

Table 6.16 *Trace of data for Example 6.6*

Ciphertext: C0B7A8D05F3A829C			
After initial permutation: 19BA9212CF26B472 After splitting: L_0=19BA9212 R_0=CF26B472			
Round	*Left*	*Right*	*Round Key*
Round 1	CF26B472	BD2DD2AB	181C5D75C66D
Round 2	BD2DD2AB	387CCDAA	3330C5D9A36D
...	...	...	...
Round 15	5A78E394	18CA18AD	4568581ABCCE
Round 16	14A7D678	18CA18AD	194CD072DE8C
After combination: 14A7D67818CA18AD			
Plaintext:123456ABCD132536			(after final permutation)

6.3 DES ANALYSIS

Critics have used a strong magnifier to analyze DES. Tests have been done to measure the strength of some desired properties in a block cipher. The elements of DES have gone through scrutinies to see if they have met the established criteria. We discuss some of these in this section.

6.3.1 Properties

Two desired properties of a block cipher are the avalanche effect and the completeness.

Avalanche Effect **Avalanche effect** means a small change in the plaintext (or key) should create a significant change in the ciphertext. DES has been proved to be strong with regard to this property.

Example 6.7 To check the avalanche effect in DES, let us encrypt two plaintext blocks (with the same key) that differ only in one bit and observe the differences in the number of bits in each round.

```
Plaintext: 0000000000000000          Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000001          Key: 22234512987ABB23
Ciphertext: 0A4ED5C15A63FEA3
```

Although the two plaintext blocks differ only in the rightmost bit, the ciphertext blocks differ in 29 bits. This means that changing approximately 1.5 percent of the plaintext creates a change of approximately 45 percent in the ciphertext. Table 6.17 shows the change in each round. It shows that significant changes occur as early as the third round.

Table 6.17 *Number of bit differences for Example 6.7*

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit differences	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

Completeness Effect **Completeness effect** means that each bit of the ciphertext needs to depend on many bits on the plaintext. The diffusion and confusion produced by D-boxes and S-boxes in DES, show a very strong completeness effect.

6.3.2 Design Criteria

The design of DES was revealed by IBM in 1994. Many tests on DES have proved that it satisfies some of the required criteria as claimed. We briefly discuss some of these design issues.

S-Boxes We have discussed the general design criteria for S-boxes in Chapter 5; we only discuss the criteria selected for DES here. The design provides confusion and diffusion of bits from each round to the next. According to this revelation and some research, we can mention several properties of S-boxes.

1. The entries of each row are permutations of values between 0 and 15.
2. S-boxes are nonlinear. In other words, the output is not an affine transformation of the input. See Chapter 5 for discussion on the linearity of S-boxes.
3. If we change a single bit in the input, two or more bits will be changed in the output.

4. If two inputs to an S-box differ only in two middle bits (bits 3 and 4), the output must differ in at least two bits. In other words, S(x) and S($x \oplus 001100$) must differ in at least two bits where x is the input and S(x) is the output.
5. If two inputs to an S-box differ in the first two bits (bits 1 and 2) and are the same in the last two bits (5 and 6), the two outputs must be different. In other words, we need to have the following relation $S(x) \neq S(x \oplus 11bc00)$, in which b and c are arbitrary bits.
6. There are only 32 6-bit input-word pairs (x_i and x_j), in which $x_i \oplus x_j \neq (000000)_2$. These 32 input pairs create 32 4-bit output-word pairs. If we create the difference between the 32 output pairs, $d = y_i \oplus y_j$, no more than 8 of these d's should be the same.
7. A criterion similar to # 6 is applied to three S-boxes.
8. In any S-box, if a single input bit is held constant (0 or 1) and the other bits are changed randomly, the differences between the number of 0s and 1s are minimized.

D-Boxes

Between two rows of S-boxes (in two subsequent rounds), there are one straight D-box (32 to 32) and one expansion D-box (32 to 48). These two D-boxes together provide diffusion of bits. We have discussed the general design principle of D-boxes in Chapter 5. Here we discuss only the ones applied to the D-boxes used inside the DES function. The following criteria were implemented in the design of D-boxes to achieve this goal:

1. Each S-box input comes from the output of a different S-box (in the previous round).
2. No input to a given S-box comes from the output from the same box (in the previous round).
3. The four outputs from each S-box go to six different S-boxes (in the next round).
4. No two output bits from an S-box go to the same S-box (in the next round).
5. If we number the eight S-boxes, $S_1, S_2, \ldots, S_8$,
   a. An output of S_{j-2} goes to one of the first two bits of S_j (in the next round).
   b. An output bit from S_{j-1} goes to one of the last two bits of S_j (in the next round).
   c. An output of S_{j+1} goes to one of the two middle bits of S_j (in the next round).
6. For each S-box, the two output bits go to the first or last two bits of an S-box in the next round. The other two output bits go to the middle bits of an S-box in the next round.
7. If an output bit from S_j goes to one of the middle bits in S_k (in the next round), then an output bit from S_k cannot go to the middle bit of S_j. If we let $j = k$, this implies that none of the middle bits of an S-box can go to one of the middle bits of the same S-box in the next round.

Number of Rounds DES uses sixteen rounds of Feistel ciphers. It has been proved that after eight rounds, each ciphertext is a function of every plaintext bit and every key bit; the ciphertext is thoroughly a random function of plaintext and ciphertext. Therefore, it looks like eight rounds should be enough. However, experiments have found that DES versions with less than sixteen rounds are even more vulnerable to known-plaintext attacks than brute-force attack, which justifies the use of sixteen rounds by the designers of DES.

6.3.3 DES Weaknesses

During the last few years critics have found some weaknesses in DES.

Weaknesses in Cipher Design

We will briefly mention some weaknesses that have been found in the design of the cipher.

S-boxes At least three weaknesses are mentioned in the literature for S-boxes.

1. In S-box 4, the last three output bits can be derived in the same way as the first output bit by complementing some of the input bits.
2. Two specifically chosen inputs to an S-box array can create the same output.
3. It is possible to obtain the same output in a single round by changing bits in only three neighboring S-boxes.

D-boxes One mystery and one weakness were found in the design of D-boxes:

1. It is not clear why the designers of DES used the initial and final permutations; these have no security benefits.
2. In the expansion permutation (inside the function), the first and fourth bits of every 4-bit series are repeated.

Weakness in the Cipher Key

Several weaknesses have been found in the cipher key.

Key Size Critics believe that the most serious weakness of DES is in its key size (56 bits). To do a brute-force attack on a given ciphertext block, the adversary needs to check 2^{56} keys.

a. With available technology, it is possible to check one million keys per second. This means that we need more than two thousand years to do brute-force attacks on DES using only a computer with one processor.
b. If we can make a computer with one million chips (parallel processing), then we can test the whole key domain in approximately 20 hours. When DES was introduced, the cost of such a computer was over several million dollars, but the cost has dropped rapidly. A special computer was built in 1998 that found the key in 112 hours.
c. Computer networks can simulate parallel processing. In 1977 a team of researchers used 3500 computers attached to the Internet to find a key challenged by RSA Laboratories in 120 days. The key domain was divided among all of these computers, and each computer was responsible to check the part of the domain.
d. If 3500 networked computers can find the key in 120 days, a secret society with 42,000 members can find the key in 10 days.

The above discussion shows that DES with a cipher key of 56 bits is not safe enough to be used comfortably. We will see later in the chapter that one solution is to use triple DES (3DES) with two keys (112 bits) or triple DES with three keys (168 bits).

Weak Keys Four out of 2^{56} possible keys are called **weak keys.** A weak key is the one that, after parity drop operation (using Table 6.12), consists either of all 0s, all 1s, or half 0s and half 1s. These keys are shown in Table 6.18.

Table 6.18 *Weak keys*

Keys before parities drop (64 bits)	*Actual key (56 bits)*
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

The round keys created from any of these weak keys are the same and have the same pattern as the cipher key. For example, the sixteen round keys created from the first key is all made of 0s; the one from the second is made of half 0s and half 1s. The reason is that the key-generation algorithm first divides the cipher key into two halves. Shifting or permutation of a block does not change the block if it is made of all 0s or all 1s.

What is the disadvantage of using a weak key? If we encrypt a block with a weak key and subsequently encrypt the result with the same weak key, we get the original block. The process creates the same original block if we decrypt the block twice. In other words, each weak key is the inverse of itself $E_k(E_k(P)) = P$, as shown in Fig. 6.11.

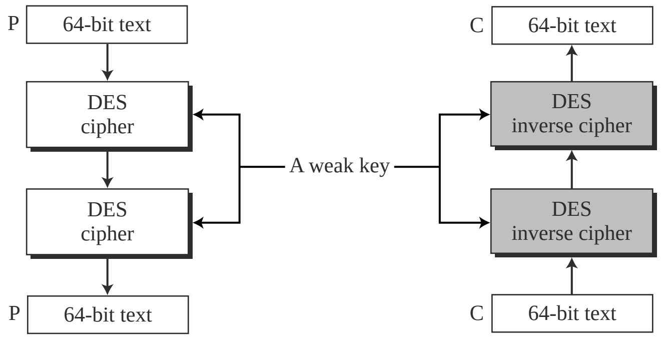


Fig. 6.11 *Double encryption and decryption with a weak key*

Weak keys should be avoided because the adversary can easily try them on the intercepted ciphertext. If after two decryptions the result is the same, the adversary has found the key.

Example 6.8 Let us try the first weak key in Table 6.18 to encrypt a block two times. After two encryptions with the same key the original plaintext block is created. Note that we have used the encryption algorithm two times, not one encryption followed by another decryption.

Key: 0x0101010101010101	
Plaintext: *0x1234567887654321*	Ciphertext: 0x814FE938589154F7
Key: 0x0101010101010101	
Plaintext: 0x814FE938589154F7	Ciphertext: *0x1234567887654321*

Semi-weak Keys There are six key pairs that are called **semi-weak keys.** These six pairs are shown in Table 6.19 (64-bit format before dropping the parity bits).

Table 6.19 *Semi-weak keys*

First key in the pair	*Second key in the pair*
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

A semi-weak key creates only two different round keys and each of them is repeated eight times. In addition, the round keys created from each pair are the same with different orders. To show the idea, we have created the round keys from the first pairs as shown below:

Round key 1	9153E54319BD	6EAC1ABCE642
Round key 2	6EAC1ABCE642	9153E54319BD
Round key 3	6EAC1ABCE642	9153E54319BD
Round key 4	6EAC1ABCE642	9153E54319BD
Round key 5	6EAC1ABCE642	9153E54319BD
Round key 6	6EAC1ABCE642	9153E54319BD
Round key 7	6EAC1ABCE642	9153E54319BD
Round key 8	6EAC1ABCE642	9153E54319BD
Round key 9	9153E54319BD	6EAC1ABCE642
Round key 10	9153E54319BD	6EAC1ABCE642
Round key 11	9153E54319BD	6EAC1ABCE642
Round key 12	9153E54319BD	6EAC1ABCE642
Round key 13	9153E54319BD	6EAC1ABCE642
Round key 14	9153E54319BD	6EAC1ABCE642
Round key 15	9153E54319BD	6EAC1ABCE642
Round key 16	6EAC1ABCE642	9153E54319BD

As the list shows, there are eight equal round keys in each semi-weak key. In addition, round key 1 in the first set is the same as round key 16 in the second; round key 2 in the first is the same as round key 15 in the second; and so on. This means that the keys are inverses of each other $E_{k_2}(E_{k_1}(P)) = P$, as shown in Fig. 6.12.

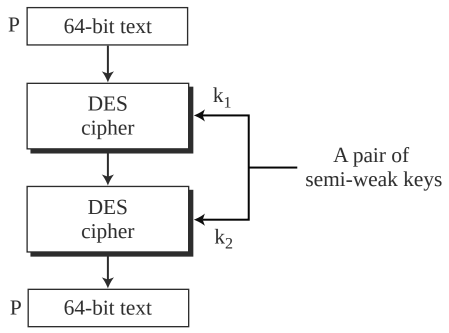


Fig. 6.12 *A pair of semi-weak keys in encryption and decryption*

Possible Weak Keys There are also 48 keys that are called **possible weak keys.** A possible weak key is a key that creates only four distinct round keys; in other words, the sixteen round keys are divided into four groups and each group is made of four equal round keys.

Example 6.9 What is the probability of randomly selecting a weak, a semi-weak, or a possible weak key?

Solution DES has a key domain of 2^{56}. The total number of the above keys are 64 (4 + 12 + 48). The probability of choosing one of these keys is 8.8×10^{-16}, almost impossible.

Key Complement In the key domain (2^{56}), definitely half of the keys are *complement* of the other half. A **key complement** can be made by inverting (changing 0 to 1 or 1 to 0) each bit in the key. Does a key complement simplify the job of the cryptanalysis? It happens that it does. Eve can use only half of the possible keys (2^{55}) to perform brute-force attack. This is because

$$C = E\,(K, P) \rightarrow \overline{C} = E\,(\overline{K}, \overline{P})$$

In other words, if we encrypt the complement of plaintext with the complement of the key, we get the complement of the ciphertext. Eve does not have to test all 2^{56} possible keys, she can test only half of them and then complement the result.

Example 6.10 Let us test the claim about the complement keys. We have used an arbitrary key and plaintext to find the corresponding ciphertext. If we have the key complement and the plaintext, we can obtain the complement of the previous ciphertext (Table 6.20).

Table 6.20 *Results for Example 6.10*

	Original	*Complement*
Key	1234123412341234	EDCBEDCBEDCBEDCB
Plaintext	12345678ABCDEF12	EDCBA987543210ED
Ciphertext	E112BE1DEFC7A367	1EED41E210385C98

Key Clustering Key clustering refers to the situation in which two or more different keys can create the same ciphertext from the same plaintext. Obviously, each pair of the semi-weak keys is a key cluster. However, no more clusters have been found for the DES. Future research may reveal some more.

6.4 SECURITY OF DES

DES, as the first important block cipher, has gone through much scrutiny. Among the attempted attacks, three are of interest: brute-force, differential cryptanalysis, and linear cryptanalysis.

6.4.1 Brute-Force Attack

We have discussed the weakness of short cipher key in DES. Combining this weakness with the key complement weakness, it is clear that DES can be broken using 2^{55} encryptions. However, today most applications use either 3DES with two keys (key size of 112) or 3DES with three keys (key size of 168). These two multiple-DES versions make DES resistant to brute-force attacks.

6.4.2 Differential Cryptanalysis

We discussed the technique of differential cryptanalysis on modern block ciphers in Chapter 5. DES is not immune to that kind of attack. However, it has been revealed that the designers of DES already knew about this type of attack and designed S-boxes and chose 16 as the number of rounds to make DES specifically resistant to this type of attack. Today, it has been shown that DES can be broken using differential cryptanalysis if we have 2^{47} chosen plaintexts or 2^{55} known plaintexts. Although this looks more efficient than a brute-force attack, finding 2^{47} chosen plaintexts or 2^{55} know plaintexts is impractical. Therefore, we can say that DES is resistant to differential cryptanalysis. It has also been shown that increasing the number of rounds to 20 require more than 2^{64} chosen plaintexts for this attack, which is impossible because the possible number of plaintext blocks in DES is only 2^{64}.

We show an example of DES differential cryptanalysis in Appendix N.

6.4.3 Linear Cryptanalysis

We discussed the technique of linear cryptanalysis on modern block ciphers in Chapter 5. Linear cryptanalysis is newer than differential cryptanalysis. DES is more vulnerable to linear cryptanalysis than to differential cryptanalysis, probably because this type of attack was not known to the designers of DES. S-boxes are not very resistant to linear cryptanalysis. It has been shown that DES can be broken using 2^{43} pairs of known plaintexts. However, from the practical point of view, finding so many pairs is very unlikely.

We show an example of DES linear cryptanalysis in Appendix N.

6.5 MULTIPLE DES—CONVENTIONAL ENCRYPTION ALGORITHMS

If a block cipher has a key size, which is small in context to the present day computation power, then a natural way out may be to perform multiple encryptions by the block cipher. As an example, consider the DES algorithm which has a key size of 56 bits, which is short in context to the modern computation capability. The threat is that such a key value can be evaluated by brute force key search. Hence two DES applications give what is known as 2-DES.

6.5.1 2-DES and Meet in the Middle Attack

Consider a message m, which is to be encrypted. The corresponding block cipher for one application of the DES applications is represented by E_k, where k is the corresponding DES key. The output of 2-DES is $c = E_{k_2}(E_{k_1}(m))$. To decrypt similarly, $m = D_{k_1}(D_{k_2}(\mathrm{c}))$. This cipher, 2-DES should offer additional security, equivalent to both k_1 and k_2. The cipher 2-DES obtained by the repeated application of DES is called, $2 - DES = DES \times DES$. This is called a product cipher obtained by the composition of two ciphers. Such an idea can similarly be extended to multiple ciphers.

It may be noted that such a product on the DES cipher is expected to provide additional security, because DES does not form a group under the composition operation. That is the composition (application) of two ciphers with two different keys cannot be obtained by a single application of DES with a key. Thus 2-DES is expected to provide security equivalent to 56×2=112 bits. However it can be shown that such a cipher can be attacked by an attack method which is called Meet-in-the-Middle attack.

6.5.2 Meet-in-the-Middle (MIM) Attack and 3-DES

Consider the cipher 2-DES as defined above. The plaintext and the ciphertext of the cipher is $P=\{0,1\}^m$. The key space of DES is $K=\{0,1\}^n$, the key size of the product cipher is expected to be $K_1 \times K_2$, where the key is represented as the ordered pair (k_1,k_2), where k_1 belongs to K_1 and k_2 belongs to K_2.

The attacker obtains l pairs of plaintexts and ciphertexts: $(p_1,c_1),\ldots,(p_1,c_1)$. The key is say (K_1,K_2) but unknown to the attacker (obviously, else why will he/she be an attacker).

It is easy to prove that for all $1 \le i \le l$, $DES_{K_1}(p_i) = \text{DES}^{-1}_{K_2}(c)$. There are in total 2^{2n} keys. The probability of a key satisfying this equation for a particular value of i is 2^{-m}, as that is the block size of the cipher. Since all the i values of the plaintext, ciphertext pairs are independent, the probability of a key satisfying the above equation for all the l values of i, is 2^{-ml}.

Thus the reduced key space which satisfies the above test is expected to be $2^{2n}.2^{-ml} = 2^{2n-ml}$.

Suppose $l \ge 2n/m$, hence the number of keys passing the above test is ≤ 1. Thus if for a key (K_1,K_2), for all $1 \le i \le l$, $DES_{K_i}(p_i) = \text{DES}^{-1}_{K_i}(c)$ is satisfied, there is a high probability that the key is the correct key.

The attacker maintains two lists L_1 and L_2 as follows:

L_1 contains 2^n rows, where each row stores one round DES encryptions of the l plaintexts, $p_1,\ldots,p_l$. L_2 contains also 2^n rows where each row stores one round DES decryptions of the l ciphertexts, $c_1, \ldots, c_l$. The lists are sorted in lexicographical order with respect to the plaintexts and ciphertexts. The lists look like as shown in the Fig. 6.13.

L1:Plain Texts	Key
$DES_{K_1^1}(p_1)DES_{K_1^1}(p_2)...DES_{K_1^1}(p_l)$	K_1^1
$DES_{K_1^{2^n}}(p_1)DES_{K_1^{2^n}}(p_2)...DES_{K_1^{2^n}}(p_l)$	$K_1^{2^n}$

(a)

L2: Cipher Texts	Key
$DES^{-1}_{K_1^1}(c_1)DES^{-1}_{K_1^1}(c_2)...DES^{-1}_{K_1^1}(c_l)$	K_2^1
$DES^{-1}_{K_1^{2^n}}(c_1)DES^{-1}_{K_1^{2^n}}(c_2)...DES^{-1}_{K_1^{2^n}}(c_l)$	$K_2^{2^n}$

(b)

Fig. 6.13

The attacker now searches the lists L_1 and L_2 and looks for a row i in L_1 which matches with a row j in L_2. Then by the above discussion, if $l \ge 2n/m$ there is a high probability that the key is (K_1^i, K_2^j). What is the complexity of the attack? Each table has 2^n rows. Each row has l blocks of size m bits each plus an additional n bits for the key. Hence each row of the table has ml+n bits. Thus the memory required by the attacker per table is $2^n(ml + n)$, and for the two tables it is equal to $2^{n+1}(ml + n)$. The time complexity of the attack is proportional to the number of encryptions or decryptions required. This works out to $2.l.2^n=l.2^{n+1}$.

This is an example of known plaintext attack, because the plaintext is known but not chosen.

Thus we see that for typical values of DES, where $n = m = 56$, the security provided by DES against a meet-in-the-middle attack is that of 57 bits, as opposed to the expected security of 112 bits. Also it may be noted that the attack works with a high probability of success if $l \ge 2$, which means that only two plaintexts needs to be known for the attack.

Since double DES or 2-DES has a problem of this meet-in-the-middle attack, Triple-DES or 3-DES was developed. The expected security of 3-DES is 112 bits (why?).

There are in general two flavors of 3-DES. There are at least two flavors of implementation of 3-DES. The first implementation uses three keys, namely K_1, K_2, K_3. The ciphertext of m is thus obtained by $C = DES_{k_1}[DES_{k_2}(DES_{k_3}(m))]$. The second way to implement 3-DES is using two keys, thus $C = DES_{k_1}[DES_{k_2}^{-1}(DES_{k_1}(m))]$. Thus if the keys K_1 and K_2 are the same then we obtain a single DES. This backward compatibility of the two key version of 3-DES is the reason why the middle layer is a decryption. It has otherwise no security implications.

6.6 EXAMPLES OF BLOCK CIPHERS INFLUENCED BY DES

6.6.1 The CAST Block Cipher

The CAST Block Cipher is an improvement of the DES block cipher, invented in Canada by Carlisle Adams and Stafford Tavares. The name of the cipher seems to be after the initials of the inventors. The CAST algorithm has 64 bit block size and has a key of size 64 bits.

CAST is based on the Feistel structure to implement the substitution permutation network. The authors state that they use the Feistel structure, as it is well studied and free of basic structural weaknesses.

S-Boxes of CAST CAST uses S-Boxes of dimension $m \times n$ ($m < n$). The typical dimension of the S-Boxes of CAST is 8×32. The principle behind the construction is as follows: choose n distinct binary bent functions of length 2^m, such that the linear combinations of these functions sum to highly non-linear, Boolen functions. Bent function are Boolen functions with even input variables having the highest possible non-linearity. The resultant functions also satisfy Strict Avalanche Criteria (SAC). SAC states that S-Box output bit j should change with probability ½ when any single input bit is changed, for all i, j. Note that the probability is computed over the set of all pairs of input vectors which differ only in bit i. Half of the bent functions have a weight of $(2^{m-1}+2^{(m/2)-1})$ and the other have a weight of $(2^{m-1} - 2^{(m/2)-1})$.

Encryption Function The plaintext block is divided into a left half and a right half. The algorithm has 8 rounds. Each round is essentially a Feistel structure. In each round the right half is combined with the round key using a function f and then XOR-ed with the left half. The new left half after the round is the same as the right half before the round. After 8 iterations of the rounds, the left and the right half are concatenated to form the ciphertext.

The Round Function f The round function in CAST can be realized as follows. The 32 bit input can be combined with 32 bits of the round key through a function, denoted by "a" (refer Fig. 6.14).

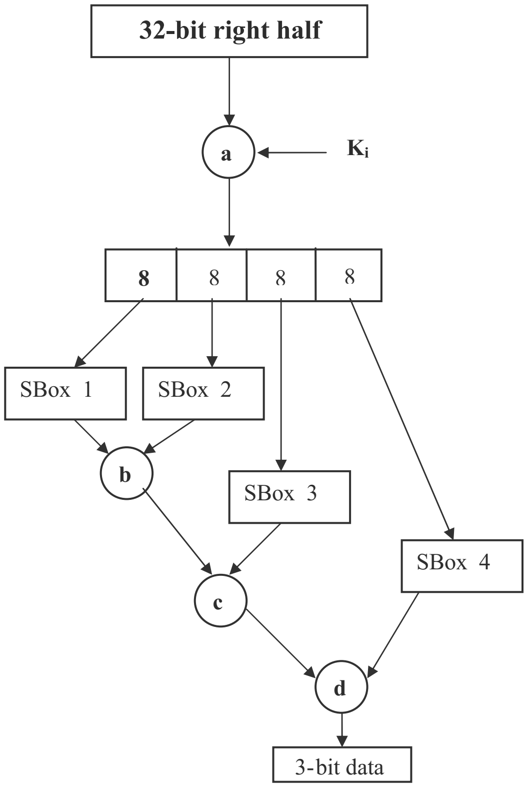


Fig. 6.14

The 32-bit data half is combined using operation "*a*" and the 32-bit result is split into 8 bit pieces. Each piece is input into a 8 × 32 S-Box. The output of S-Box 1 and 2 are combined using the operation "*b*"; the 32 bit output is combined with the output of S-Box 3, the output is combined in turn with the output of S-Box 4. The combining functions are denoted in the figure by "*c*" and "*d*". A simple way would be where all the combining functions are XOR functions, however more complex operations may also be used.

Key Scheduling of CAST The key scheduling in CAST has three main components:

1. A key transformation step which converts the primary key (input key) to an intermediate key.
2. A relatively simple bit-selection algorithm mapping the primary key and the intermediate key to a form, referred as partial key bits.
3. A set of key-schedule S-Boxes which are used to create subkeys from the partial key bits.

Let, the input key be denoted by KEY = $k_1k_2k_3k_4k_5k_6k_7k_8$, where k_i is the i^{th} byte of the primary key. The key transformation step generates the intermediate key, KEY' = $k'_1k'_2k'_3k'_4k'_5k'_6k'_7k'_8$ as follows:

$$k_1'k_2'k_3'k_4' = k_1k_2k_3k_4 \oplus S_1[k_5] \oplus S_2[k_7]$$
$$k_5'k_6'k_7'k_8' = k_5k_6k_7k_8 \oplus S_1[k_2'] \oplus S_2[k_4']$$

Here, S_1 and S_2 are key-schedule *S*-Boxes of dimension 8 × 32.

Subsequently, there is a bit-selection step which operates as shown below:

$$K_1' = k_1k_2$$
$$K_2' = k_3k_4$$
$$K_3' = k_5k_6$$
$$K_4' = k_7k_8$$
$$K_5' = k_4'k_3'$$
$$K_6' = k_2'k_1'$$
$$K_7' = k_8'k_7'$$
$$K_8' = k_6'k_5'$$

The partial key bits are used to obtain the subkeys, K_i. The subkeys are 32 bits, and are obtained as follows:

$$K_i = S_1(K_{i,1}') \oplus S_2(K_{i,2}')$$

Here, $K'_{i,j}$ is the j^{th} byte of K'_i. Thus the 8 round subkeys are obtained.

The CAST block cipher can also be implemented with 128 bits, and is referred to as CAST-128. The essential structure of the cipher is still the same as discussed above.

6.6.2 Blowfish

Blowfish is a 64-bit block cipher invented by Bruce Schneier. Blowfish was designed for fast ciphering on 32-bit microprocessors. Blowfish is also compact and has a variable key length which can be increased to 448 bits.

Blowfish is suitable for applications where the key does not change frequently like communication links or file encryptors. However for applications like packet switching or as an one-way hash function, it is unsuitable. Blowfish is not ideal for smart cards, which requires even more compact ciphers. Blowfish is faster than DES when implemented on 32-bit microprocessors. Next we discuss on the round structure of Blowfish.

Round Structure The algorithm is based on the Feistel structure and has two important parts: the round structure and the key expansion function.

There are 16 rounds, and each round are made of simple transformations which are iterated. Each round consists of a key-dependent permutation, and a key and data-dependent substitution. All the operations are additions and XORs on 32 bit words, and lookups in 4 32-bit S-Boxes. Blowfish has a P-array, $P_0, P_1, \ldots, P_{18}$ each of which are 32 bit subkeys. There are 4 S-Boxes, each of which maps an 8-bit input to 32-bits. The round structure of Blowfish is illustrated in Fig. 6.15.

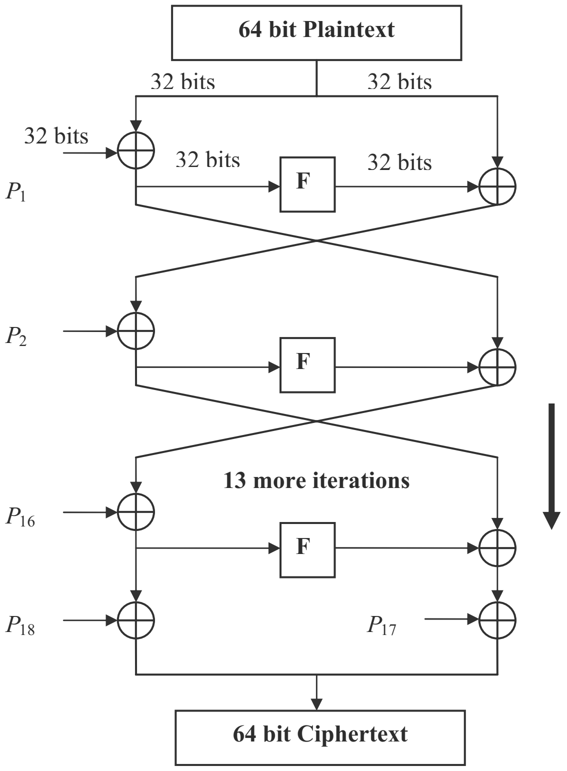


Fig. 6.15

The round function is also explained underneath with a pseudo-code.

```
Divide x into two 32-bit halves: $x_L, x_R$
For $i = 1$ to 16:
    $x_L = x_L \oplus P_i$
    $x_R = F[x_L] \oplus x_R$
    Swap $x_L$ and $x_R$
(undo the last swap)
$x_R = x_R \oplus P_{17}$
$x_L = x_L \oplus P_{18}$
Ciphertext = Concatenation of $x_L$ and $x_R$
```

The function F is central to the security of the block cipher and is defined as below:

Divide x_L into four 8-bit parts: a, b, c, d

$$F[x_L] = ((S_1[a] + S_2[b] \bmod 2^{32}) \oplus S_3[c]) + S_4[d] \bmod 2^{32}$$

Key Scheduling Algorithm The subkeys are computed using the following method:

1. The P-array and then the four *S*-Boxes are initialized with a fixed string. The string is the hexadecimal digits of π.
2. P_1 is XOR-ed with 32 bits of the key, P_2 is XOR-ed with the next 32 bits of the key, and so on for all the bits of the key. If needed the key bits are cycled to ensure that all the P-array elements are XOR-ed.

3. An all-zero string is encrypted with the Blowfish algorithm, with the subkeys P_1 to P_{18} obtained so far in steps 1 and 2.
4. P_1 and P_2 are replaced by the 64 bit output of step 3.
5. The output of step 3 is now encrypted with the updated subkeys to replace P_3 and P_4 with the ciphertext of step 4.
6. This process is continued to replace all the *P*-arrays and the *S*-Boxes in order.

This complex key-scheduling implies that for faster operations the subkeys should be precomputed and stored in the cache for faster access.

Security analysis by Serge Vaudenay shows that for a Blowfish algorithm implemented with known S-Boxes (note that in the original cipher the S-Boxes are generated during the encryption process) and with r-rounds, a differential attack can recover the P-array with 2^{8r+1} chosen plaintexts.

6.6.3 IDEA

IDEA is another block cipher. It operates on 64 bit data blocks and the key is 128 bit long. It was invented by Xuejia Lai and James Massey, and named IDEA (International Data Encryption Algorithm) in 1990, after modifying and improving the initial proposal of the cipher based on the seminal work on Differential cryptanalysis by Biham and Shamir.

The design principle behind IDEA is the "mixing of arithmetical operations from different algebraic groups". These arithmetical operations are easily implemented both in hardware and software.

The underlying operations are XOR, addition modulo 2^{16}, multiplication modulo $2^{10}+1$.

The cipher obtains the much needed non-linearity from the later two arithmetical operations and does not use an explicit S-Box.

Round Transformation of IDEA The 64-bit data is divided into four 16 bit blocks: X_1, X_2, X_3, X_4. These four blocks are processed through eight rounds and transformed by the above arithmetical operations among each other and with six 16 bit subkeys. In each round the sequence of operations is as follows:

1. Multiply X_1 and the first subkey.
2. Add X_2 and the second subkey.
3. Add X_3 and the third subkey.
4. Multiply X_4 and the fourth subkey.
5. XOR the results of step 1 and 3.
6. XOR the results of step 2 and 4.
7. Multiply the results of steps 5 with the fifth subkey.
8. Add the results of steps 6 and 7.
9. Multiply the results of steps 8 with the sixth subkey.
10. Add the results of steps 7 and 9.
11. XOR the results of steps 1 and 9.
12. XOR the results of steps 3 and 9.
13. XOR the results of steps 2 and 10.
14. XOR the results of steps 4 and 10.

The outputs of steps 11, 12, 13 and 14 are stored in four words of 16 bits each, namely Y_1, Y_2, Y_3 and Y_4. The blocks Y_2 and Y_3 are swapped, and the resultant four blocks are the output of a round of IDEA. It may be noted that the last round of IDEA does not have the swap step.

Instead the last round has the following additional transformations:

1. Multiply Y_1 and the first subkey.

2. Add Y_2 and the second subkey.
3. Add Y_3 and the third subkey.
4. Multiply Y_4 and the fourth subkey.

Finally, the ciphertext is the concatenation of the blocks Y_1, Y_2, Y_3 and Y_4.

Key Scheduling of IDEA IDEA has a very simple key scheduling. It takes the 128 bit key and divides it into eight 16 bit blocks. The first six blocks are used for the first round, while the remaining two are to be used for the second round. Then the entire 128 bit key is given a rotation for 25 steps to the left and again divided into eight blocks. The first four blocks are used as the remaining subkeys for the second round, while the last four blocks are to be used for the third round. The key is then again given a left shift by 25 bits, and the other subkeys are obtained. The process is continued till the end of the algorithm.

For decryption, the subkeys are reversed and are either the multiplicative or additive inverse of the encryption subkeys. The all zero subkey is considered to represent $2^{16}=-1$ for the modular multiplication operation, mod $2^{16}+1$. Thus the multiplicative inverse of 0 is itself, as –1 multiplied with –1 gives 1, the multiplicative identity in the group. Computing these keys may have its overhead, but it is a one time operation, at the beginning of the decryption process.

IDEA has resisted several cryptanalytic efforts. The designers gave argument to justify that only 4 rounds of the cipher makes it immune to differential cryptanalysis.

Joan Daemen, Rene Govaerts and Joos Vandewalle showed that the cipher had certain keys which can be easily discovered in a chosen plaintext attack.

They used the fact that the use of multiplicative subkeys with the value of 1 or -1 gives rise to linear factors in the round function. A linear factor is a linear equation in the key, input and output bits that hold for all possible input bits. The linear factors can be revealed by expressing the modulo 2 sum of LSBs of the output subblocks of an IDEA round in terms of inputs and key bits.

From the round structure of IDEA, the XOR of the LSBs of the first and second output subblock of a round are represented by y_1 and y_2.

$$y_1 \oplus y_2 = (X_1.Z_1)|_0 \oplus 1 \oplus x_3 \oplus z_3$$

If Z_1=(-)1=0…01 (i,e if the 15 MSB bits of the Z_1 are 0), we have the following linear equation:

$$y_1 \oplus y_2 = x_1 \oplus x_3 \oplus z_1 \oplus z_3 \oplus 1$$

If the key bits are considered as constants, this linear factor can be interpreted as the propagation of knowledge from to $x_1 \oplus x_3$ to $y_1 \oplus y_2$. This is indicated by $(1,0,1,0) \rightarrow (1,1,0,0)$.

Similar factors and their corresponding conditions on subkey blocks can be found for all 15 combinations of LSB output bits and are listed in the following table:

Table 6.21 *Linear Factors in the round function with conditions on the subkeys*

Linear Factor	Z_1	Z_4	Z_5	Z_6
(0,0,0,1)→(0,0,1,0)	–	(–)1	–	(–)1
(0,0,1,0)→(1,0,1,1)	–	–	(–)1	(–)1
(0,0,1,1)→(1,0,0,1)	–	(–)1	(–)1	–
(0,1,0,0)→(0,0,0,1)	–	–	–	(–)1
(0,1,0,1)→(0,0,1,1)	–	(–)1	–	–

(0,1,1,0)→(1,0,1,0)	–	–	(–)1	–
(0,1,1,1)→(1,0,0,0)	–	(–)1	(–)1	(–)1
(1,0,0,0)→(0,1,1,1)	(–)1	–	(–)1	(–)1
(1,0,0,1)→(0,1,0,1)	(–)1	(–)1	(–)1	–
(1,0,1,0)→(1,1,0,0)	(–)1	–	–	–
(1,0,1,1)→(1,1,1,0)	(–)1	(–)1	–	(–)1
(1,1,0,0)→(0,1,1,0)	(–)1	–	(–)1	–
(1,1,0,1)→(0,1,0,0)	(–)1	(–)1	(–)1	(–)1
(1,1,1,0)→(1,1,0,1)	(–)1	–	–	(–)1
(1,1,1,1)→(1,1,1,1)	(–)1	(–)1	–	–

The linear factors in the rounds can be combined to obtain multiple round linear factors, by combining linear factors such that the intermediate terms cancel out. For every round they impose conditions on subkeys that can be converted into conditions on global keys, using the following table (which follows from the key scheduling algorithm of IDEA):

Table 6.22 *Derivation of encryption subkeys from the global key of size 128 bits*

r	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6
1	0–15	16–31	32–47	48–63	64–79	80–95
2	96–111	112–127	25–40	41–56	57–72	73–88
3	89–104	105–120	121–8	9–24	50–65	66–81
4	82–97	98–113	114–1	2–17	18–33	34–49
5	75–90	91–106	107–122	123–10	11–26	27–42
6	43–58	59–74	100–115	116–3	4–19	20–35
7	36–51	52–67	68–83	84–99	125–12	13–28
8	29–44	45–60	61–76	77–92	93–108	109–124
9	22–37	38–53	54–69	70–85	–	–

A possible combination for a multiple round linear factor for IDEA is shown in the underlying table. The conditions on the global key bits are also mentioned. The global key bits whose indices are there in the table should be zero. Since key bits with indices 26-28, 72-74 or 111-127 do not appear, there are 2^{23} global keys that can have this linear factor. This is called a class of weak keys as they can be detected by checking the satisfaction of linear factors by some plaintext-ciphertext combinations.

Table 6.23 *Conditions on key bits for linear factor (1,0,1,0)->(0,1,1,0)*

Round	*Input Term*	Z_1	Z_5
1	(1,0,1,0)	0–14	–
2	(1,1,0,0)	96–110	57–71
3	(0,1,1,0)	–	50–64
4	(1,0,1,0)	82–96	–

5	(1,1,0,0)	75–89	11–25
6	(0,1,1,0)	–	4–18
7	(1,0,1,0)	36–50	–
8	(1,1,0,0)	29–44	93–107
9	(0,1,1,1)	–	–

6.7 RECOMMENDED READING

The following books and websites provide more details about subjects discussed in this chapter. The items enclosed in brackets […] refer to the reference list at the end of the book.

Books

[Sta06], [Sti06], [Rhe03], [Sal03], [Mao04], and [TW06] discuss DES.

WebSites

The following websites give more information about topics discussed in this chapter.

http://www.itl.nist.gov/fipspubs/fip46-2.htm
www.nist.gov/director/prog-ofc/report01-2.pdf
www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.ps
islab.oregonstate.edu/koc/ece575/notes/dc1.pdf
homes.esat.kuleuven.be/~abiryuko/Cryptan/matsui_des
http://nsfsecurity.pr.erau.edu/crypto/lincrypt.html

Key Terms

avalanche effect
completeness effect
Data Encryption Standard (DES)
double DES (2DES)
Federal Information Processing (FIPS)
key complement
meet-in-the-middle attack
National Institute of Standards and Technology (NIST)
National Security Agency (NSA)
parity bit drop
possible weak keys
round-key generator
Standard semi-weak keys
triple DES (3DES)
triple DES with three keys
triple DES with two keys
weak keys

Summary

★ The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST) as FIPS 46 in the *Federal Register*.

★ At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext. At the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

★ The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round of DES is a Feistel cipher with two elements (mixer and swapper). Each of these elements is invertible.

★ The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output. This function is made up of four operations: an expansion permutation, a whitener (that adds key), a group of S-boxes, and a straight permutation.

★ The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally presented as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process.

★ DES has shown a good performance with respect to avalanche and completeness effects. Areas of weaknesses in DES include cipher design (S-boxes and P-boxes) and cipher key (length, weak keys, semi-weak keys, possible weak keys, and key complements).

★ Since DES is not a group, one solution to improve the security of DES is to use multiple DES (double and triple DES). Double DES is vulnerable to meet-in-the-middle attack, so triple DES with two keys or three keys is common in applications.

★ The design of S-boxes and number of rounds makes DES almost immune from the differential cryptanalysis. However, DES is vulnerable to linear cryptanalysis if the adversary can collect enough known plaintexts.

Practice Set

Review Questions

6.1 What is the block size in DES? What is the cipher key size in DES? What is the round-key size in DES?

6.2 What is the number of rounds in DES?

6.3 How many mixers and swappers are used in the first approach of making encryption and decryption inverses of each other? How many are used in the second approach?

6.4 How many permutations are used in a DES cipher algorithm? How many permutations are used in the round-key generator?

6.5 How many exclusive-or operations are used in the DES cipher?

6.6 Why does the DES function need an expansion permutation?

6.7 Why does the round-key generator need a parity drop permutation?

6.8 What is the difference between a weak key, a semi-weak key, and a possible weak key?

6.9 What is double DES? What kind of attack on double DES makes it useless?

6.10 What is triple DES? What is triple DES with two keys? What is triple DES with three keys?

Exercises

6.11 Answer the following questions about S-boxes in DES:

a. Show the result of passing 110111 through S-box 3.

b. Show the result of passing 001100 through S-box 4.

c. Show the result of passing 000000 through S-box 7.

d. Show the result of passing 111111 through S-box 2.

6.12 Draw the table to show the result of passing 000000 through all 8 S-boxes. Do you see a pattern in the outputs?

6.13 Draw the table to show the result of passing 111111 through all 8 S-boxes. Do you see a pattern in the outputs?

6.14 Check the third criterion for S-box 3 using the following pairs of inputs.

a. 000000 and 000001

b. 111111 and 111011

6.15 Check the fourth design criterion for S-box 2 using the following pairs of inputs.

a. 001100 and 110000

b. 110011 and 001111

6.16 Check the fifth design criterion for S-box 4 using the following pairs of inputs.

a. 001100 and 110000

b. 110011 and 001111

6.17 Create 32 6-bit input pairs to check the sixth design criterion for S-box 5.

6.18 Show how the eight design criteria for S-box 7 are fulfilled.

6.19 Prove the first design criterion for P-boxes by checking the input to S-box 2 in round 2.

6.20 Prove the second design criterion for P-boxes by checking inputs to S-box 3 in round 4.

6.21 Prove the third design criterion for P-boxes by checking the output of S-box 4 in round 3.

6.22 Prove the fourth design criterion for P-boxes by checking the output of S-box 6 in round 12.

6.23 Prove the fifth design criteria for P-boxes by checking the relationship between S-boxes 3, 4, and 5 in rounds 10 and 11.

6.24 Prove the sixth design criteria for P-boxes by checking the destination of an arbitrary S-box.

6.25 Prove the seventh design criterion for P-boxes by checking the relationship between S-box 5 in round 4 and S-box 7 in round 5.

6.26 Redraw Fig. 6.9 using the alternate approach.

6.27 Prove that the reverse cipher in Fig. 6.9 is in fact the inverse of the cipher for a three-round DES. Start with a plaintext at the beginning of the cipher and prove that you can get the same plaintext at the end of the reverse cipher.

6.28 Carefully study the key compression permutation of Table 6.14.

a. Which input ports are missing in the output?

b. Do all left 24 output bits come from all left 28 input bits?

c. Do all right 24 output bits come from all right 28 input bits?

6.29 Show the results of the following hexadecimal data

0110 1023 4110 1023

after passing it through the initial permutation box.

6.30 Show the results of the following hexadecimal data

AAAA BBBB CCCC DDDD

after passing it through the final permutation box.

6.31 If the key with parity bit (64 bits) is 0123 ABCD 2562 1456, find the first round key.

6.32 Using a plaintext block of all 0s and a 56-bit key of all 0s, prove the key-complement weakness assuming that DES is made only of one round.

6.33 Can you devise a meet-in-the- middle attack for a triple DES?

6.34 Write pseudocode for the *permute* routine used in Algorithm 6.1

permute (n, m, inBlock[n], outBlock[m], *permutationTable[m]*)

6.35 Write pseudocode for the *split* routine used in Algorithm 6.1

split (n, m, inBlock[n], leftBlock[m], rightBlock[m])

6.36 Write pseudocode for the *combine* routine used in Algorithm 6.1

combine (n, m, leftBlock[n], rightBlock[n], outBlock[m])

6.37 Write pseudocode for the *exclusiveOr* routine used in Algorithm 6.1

exclusiveOr (n, firstInBlock[n], secondInBlock[n], outBlock[n])

6.38 Change Algorithm 6.1 to represent the alternative approach.

6.39 Augment Algorithm 6.1 to be used for both encryption and decryption.