INFORMATION SECURITY [3 0 0 3]

ICT 3172:

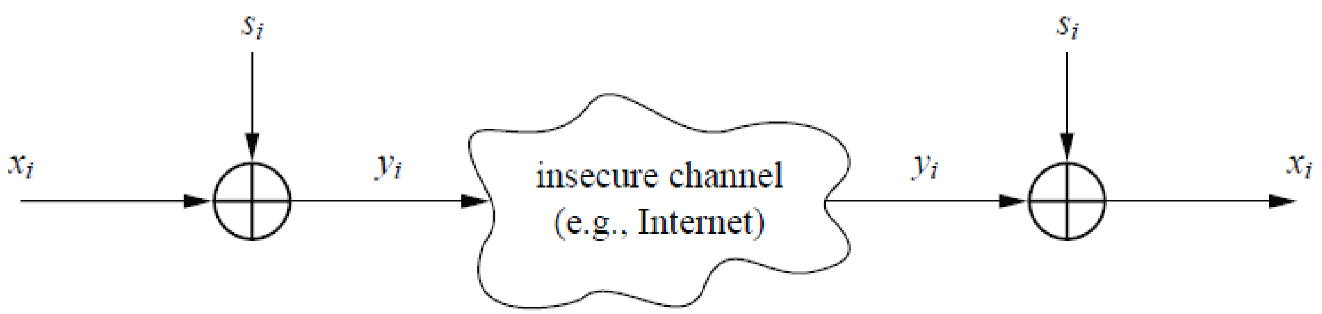
Data Encryption Standard (DES)

CONFUSION VS DIFFUSION

Confusion: is an encryption operation where the relationship between key and ciphertext is obscured. A common element for achieving confusion is substitution.

Diffusion: is an encryption operation where the influence of one plaintext symbol is spread over many ciphertext symbols with the goal of hiding statistical properties of the plaintext.

A simple diffusion element is the bit permutation, which is used frequently within DES.


s_i
x_i
y_i
insecure channel
(e.g., Internet)
y_i
s_i
x_i

Feistel structure

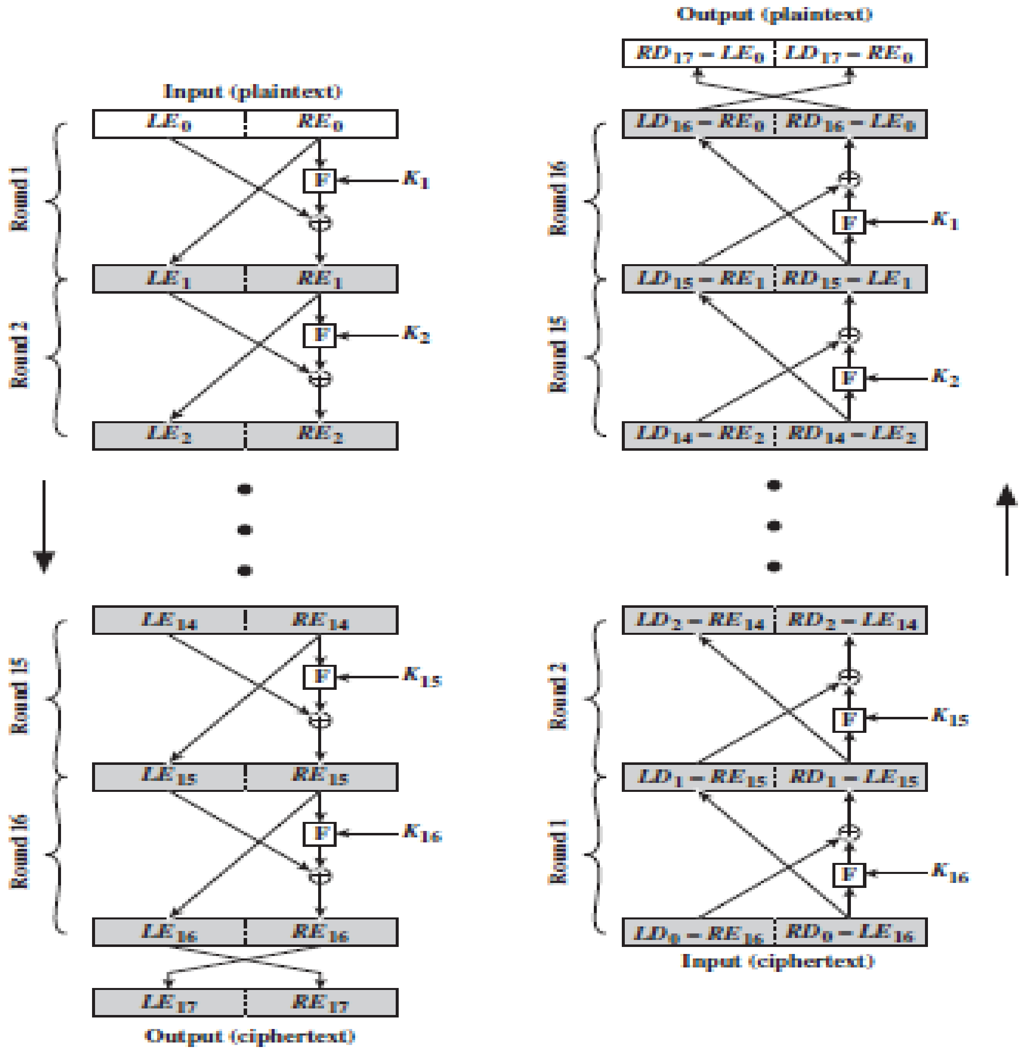
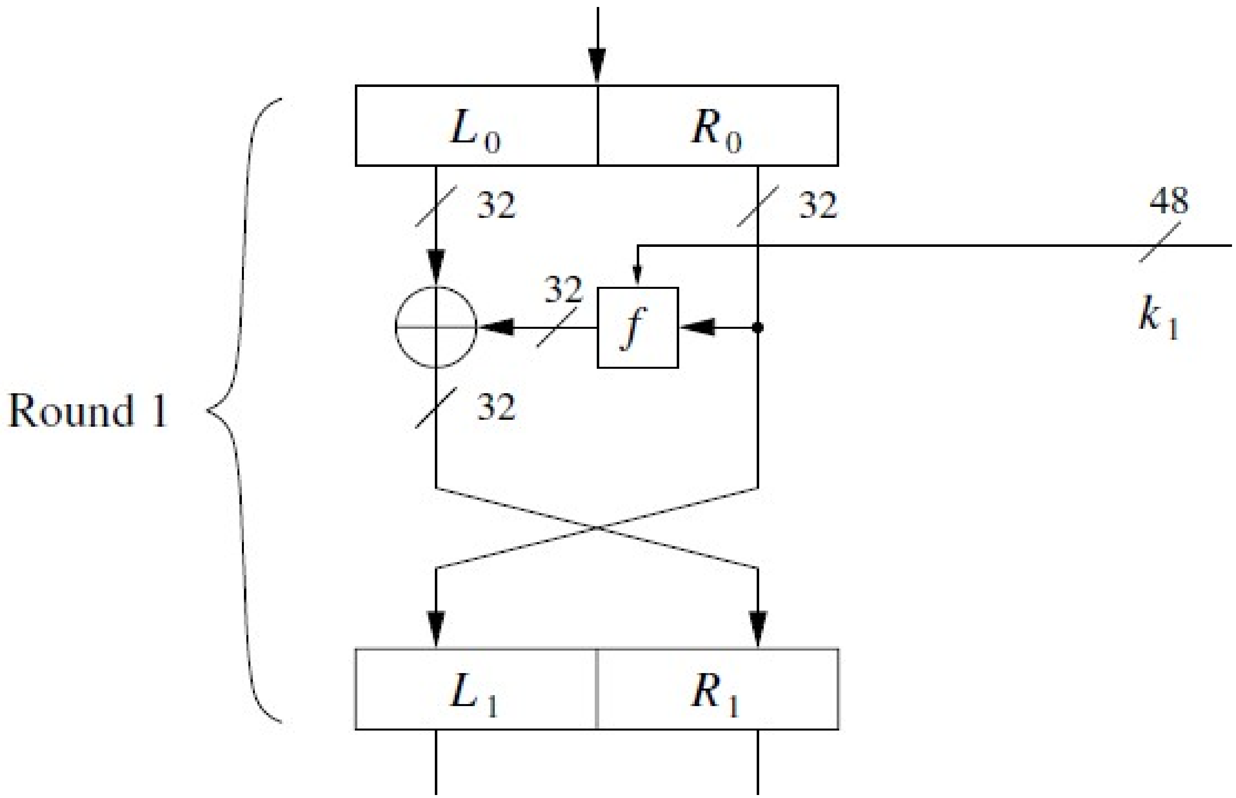


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

Feistel structure



Feistel structure

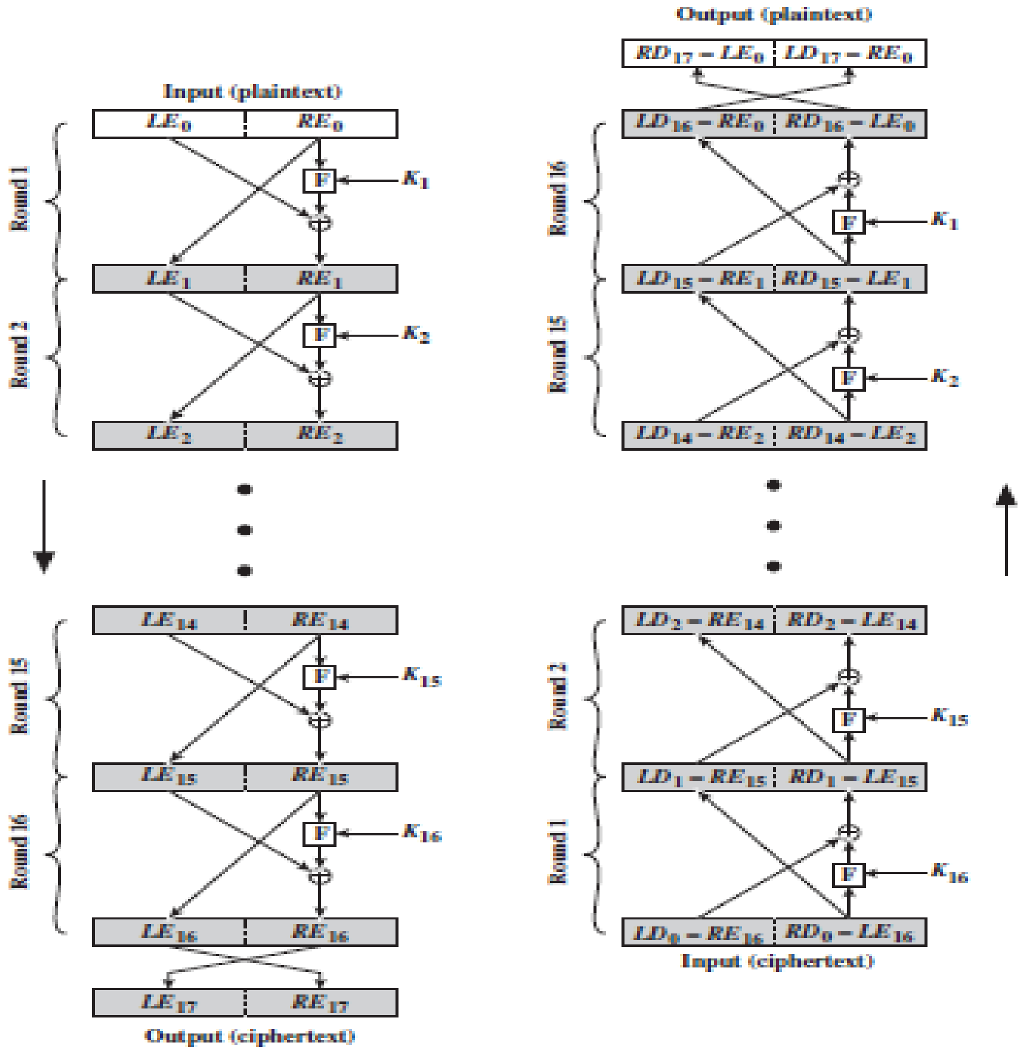


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

Feistel structure

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

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

On the decryption side,

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

The XOR has the following properties:

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

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

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

Rearranging terms:

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

Data Encryption Standard (DES)

Data DES is a symmetric-key block cipher.

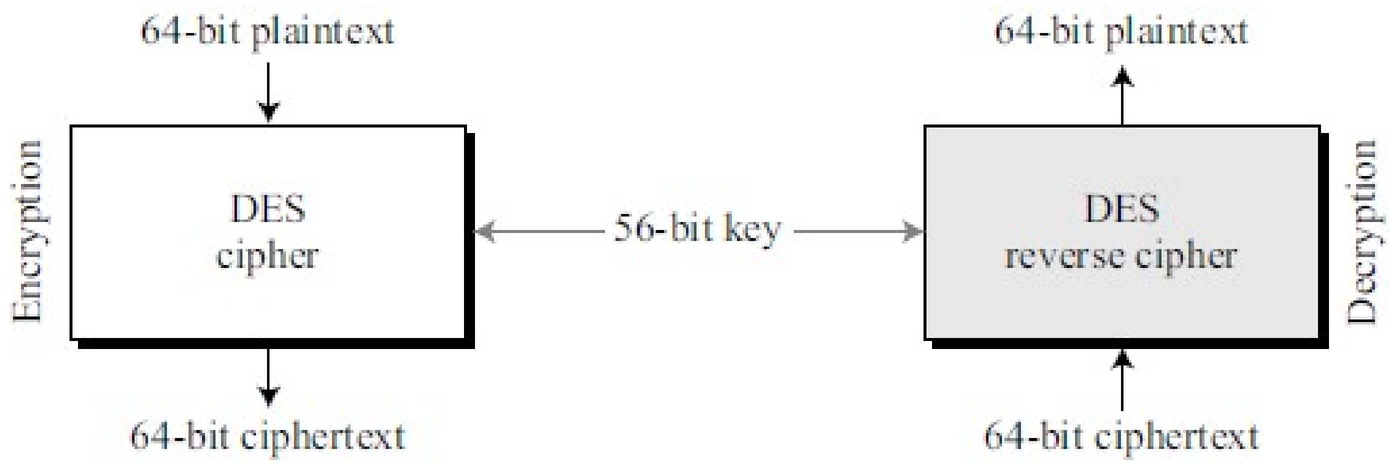
Published by the National Institute of Standards and Technology (NIST).

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).

Overview

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



- DES takes a 64-bit plaintext and creates a 64-bit ciphertext;
- DES takes a 64-bit ciphertext and creates a 64-bit plaintext.
- The same 56-bit cipher key is used for both encryption and decryption.

6.2 DES STRUCTURE

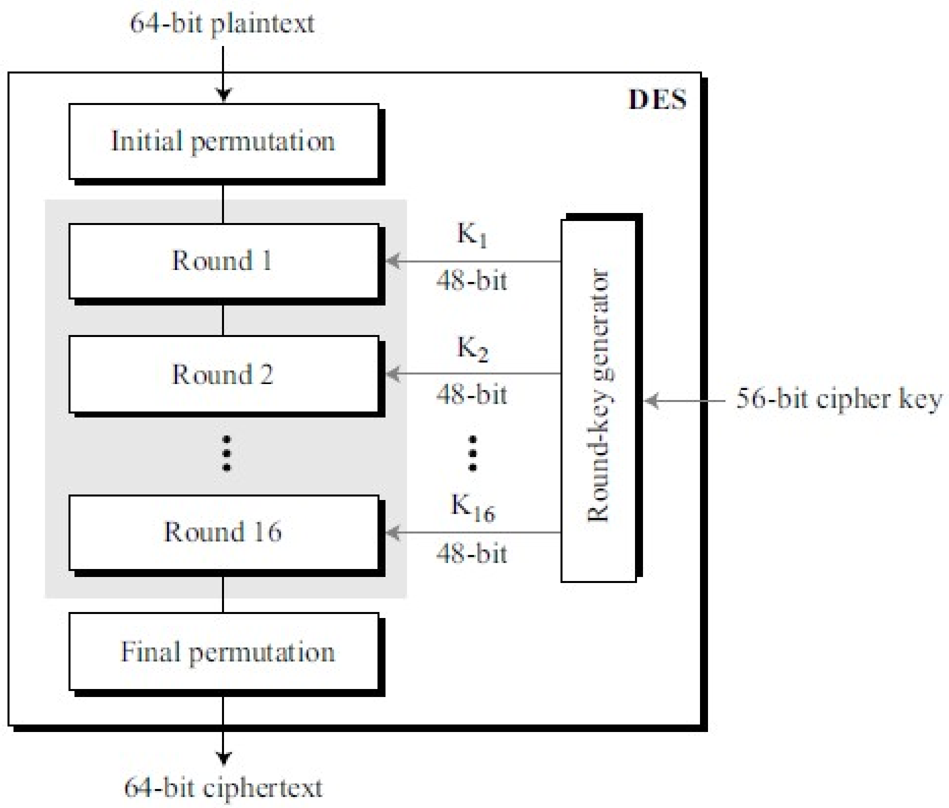


Figure 6.2 : elements of DES cipher at the encryption site.

- The encryption process is made of two permutations (P-boxes), which is initial and final permutations, and 16 Feistel rounds.
- Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm.

Initial and Final Permutations

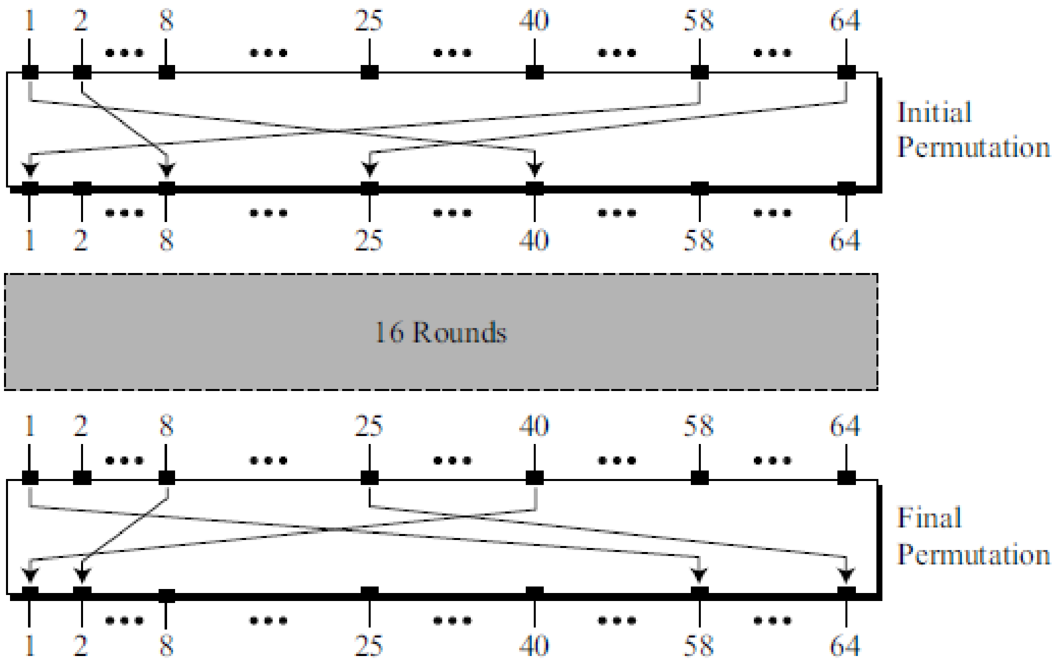


Figure 6.3 : initial and final permutations (P-boxes).

- Each of these permutations takes a 64-bit input and permutes them according to a predefined rule.
- 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.

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

Table 6.1 is the the permutation rules for these P-boxes.

Each side of the table can be thought of as a 64-element array.

Here the value of each element defines the input port number, and the order (index) of the element defines the output port number.

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.

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

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

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

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

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 is

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

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

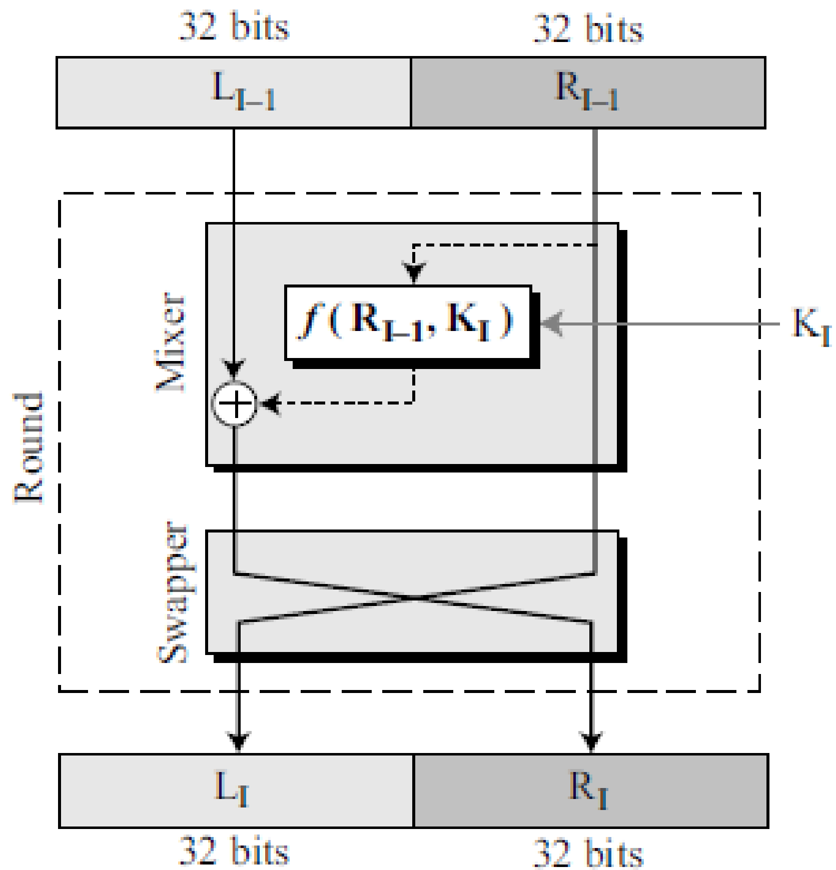
Only bit 25 and bit 64 are 1s; the other bits are 0s. In the final permutation, bit 25 becomes bit 64 and bit 63 becomes bit 15. The result is

0x0002 0000 0000 0001

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

Rounds

DES uses 16 rounds. Each round of DES is a Feistel cipher.



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).

Each round has two cipher elements (mixer and swapper).

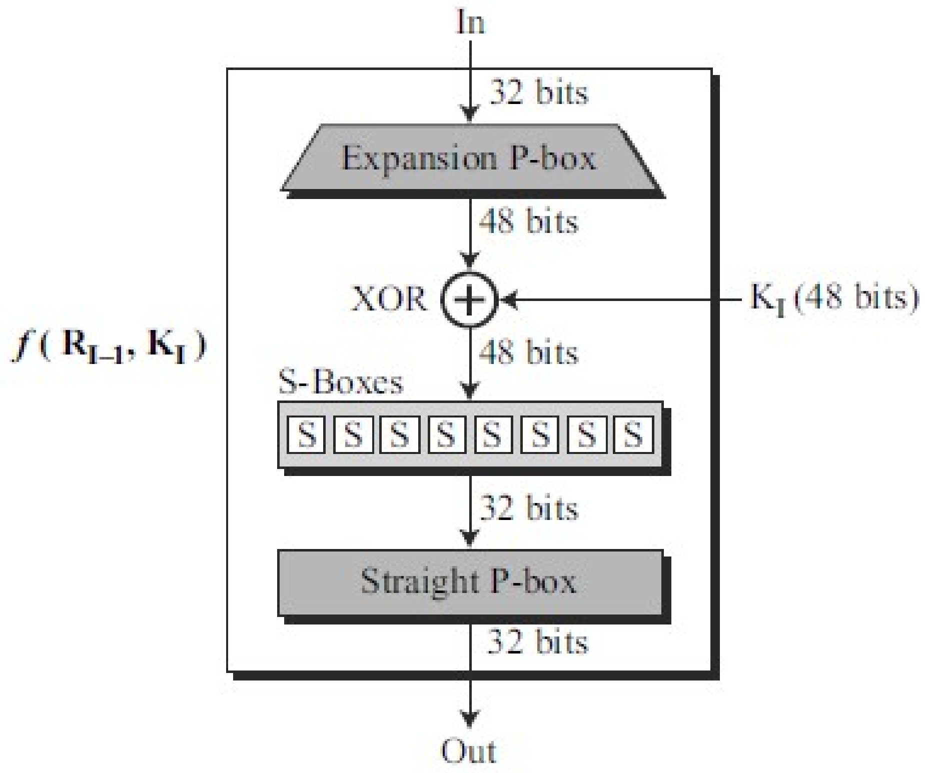
Each of these elements is invertible.

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).

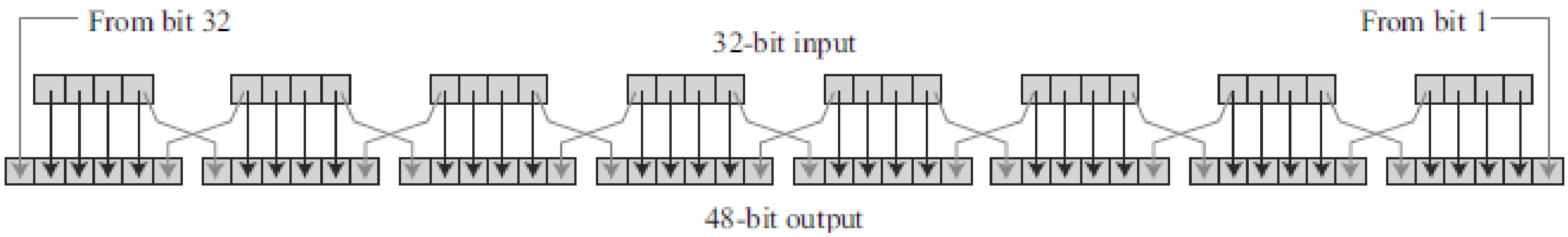
DES Function



- The heart of DES is the DES function.
- 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 4 sections:
  - ✓ an expansion P-box,
  - ✓ a whitener (that adds key),
  - ✓ a group of S-boxes, and
  - ✓ a straight P-box.

Expansion P-box

R_{I-1} is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits.



Expansion permutation

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.

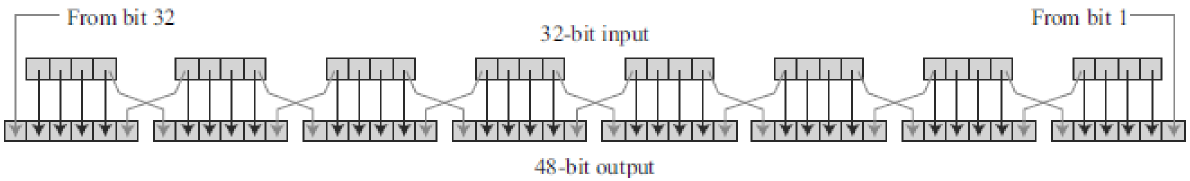


Table 6.2 to define this P-box.

Table 6.2 *Expansion P-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	30	31	32	01

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.

Whitener (XOR)

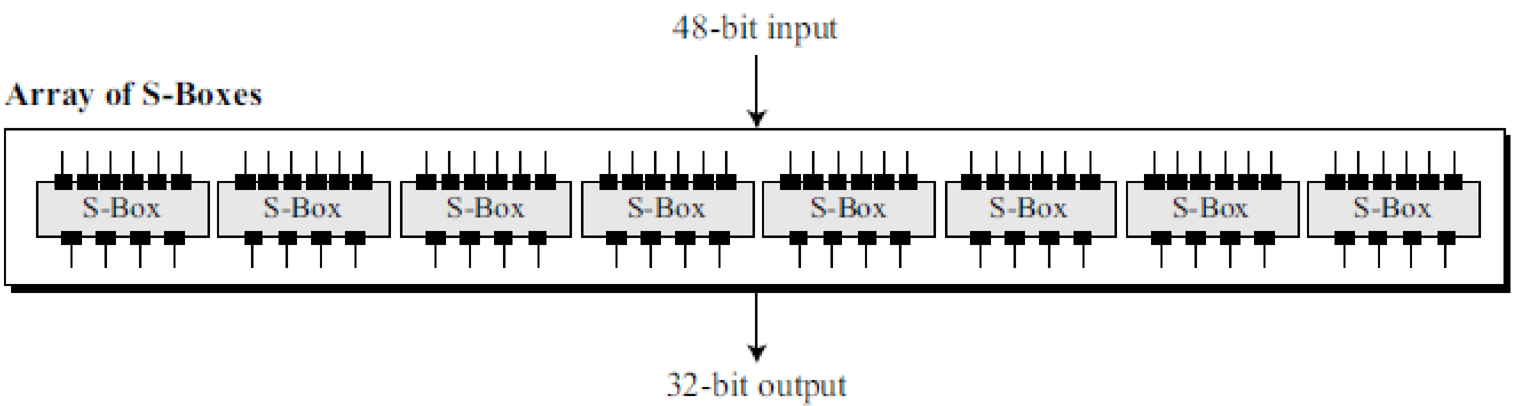
After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key.

Note that both the right section and the key are 48-bits in length.

S-Boxes

The S-boxes do the real mixing (confusion).

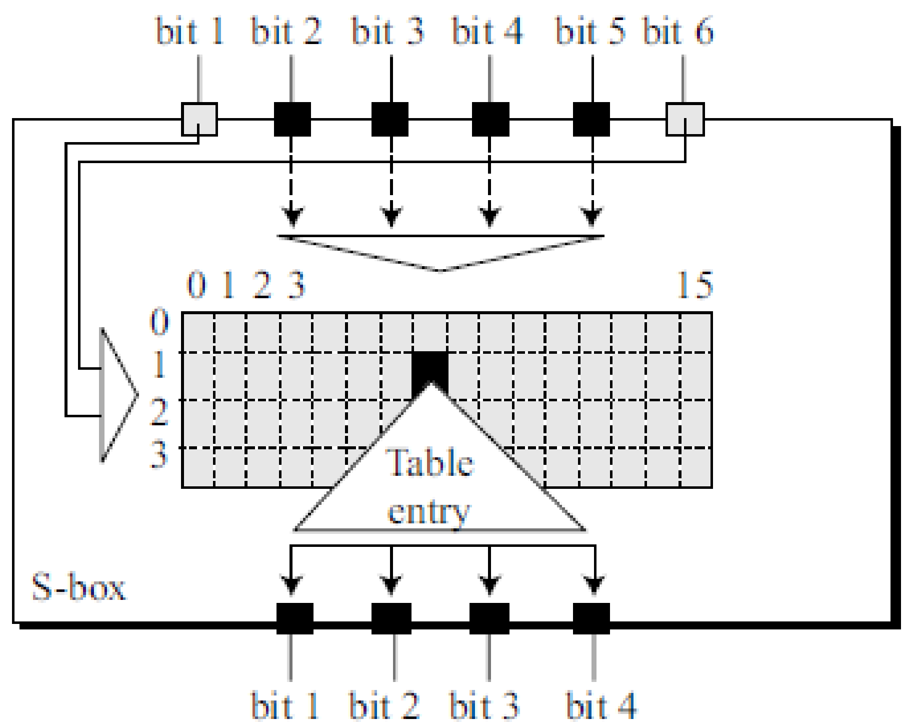
DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.



48-bit data from the second operation is divided into eight 6-bit chunks, and each chunk is fed into a box.

Result of each box is a 4-bit chunk; when these are combined the result is a 32-bit text.

Figure 6.8 *S-box rule*



- Substitution in each box follows a pre-determined rule based on a 4-row by 16-column table.
- Combination of bits 1 and 6 of the input defines one of four rows;
- Combination of bits 2 through 5 defines one of the sixteen columns

Because each S-box has its own table, **we need 8 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.

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 100011. What is the output?

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

Example 6.3 : The input to S-box 1 is 100011. What is the output?

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

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.

Example 6.4 : The input to S-box 8 is 000000. What is the output?

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 6.4 : The input to S-box 8 is 000000. What is the output?

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

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.

Straight Permutation

Last operation in the DES function is a straight permutation with a 32-bit input and a 32-bit 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

The input/output relationship for this operation is shown in and follows the same general rule as previous permutation tables.

For example, the 7^{th} bit of the input becomes the 2^{nd} bit of the output.

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;

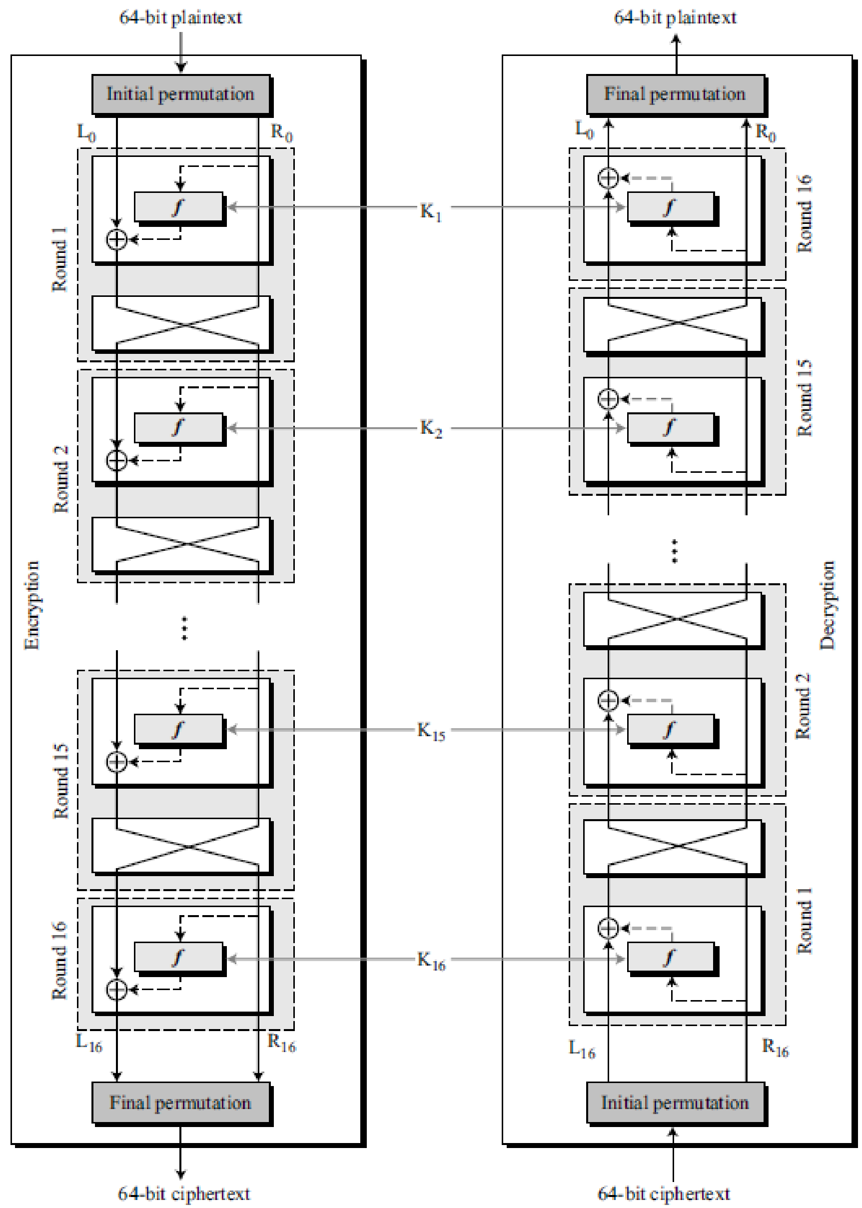
The reverse cipher is used at the decryption site.

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.

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}.

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.

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[32], rightBlock[32], 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).

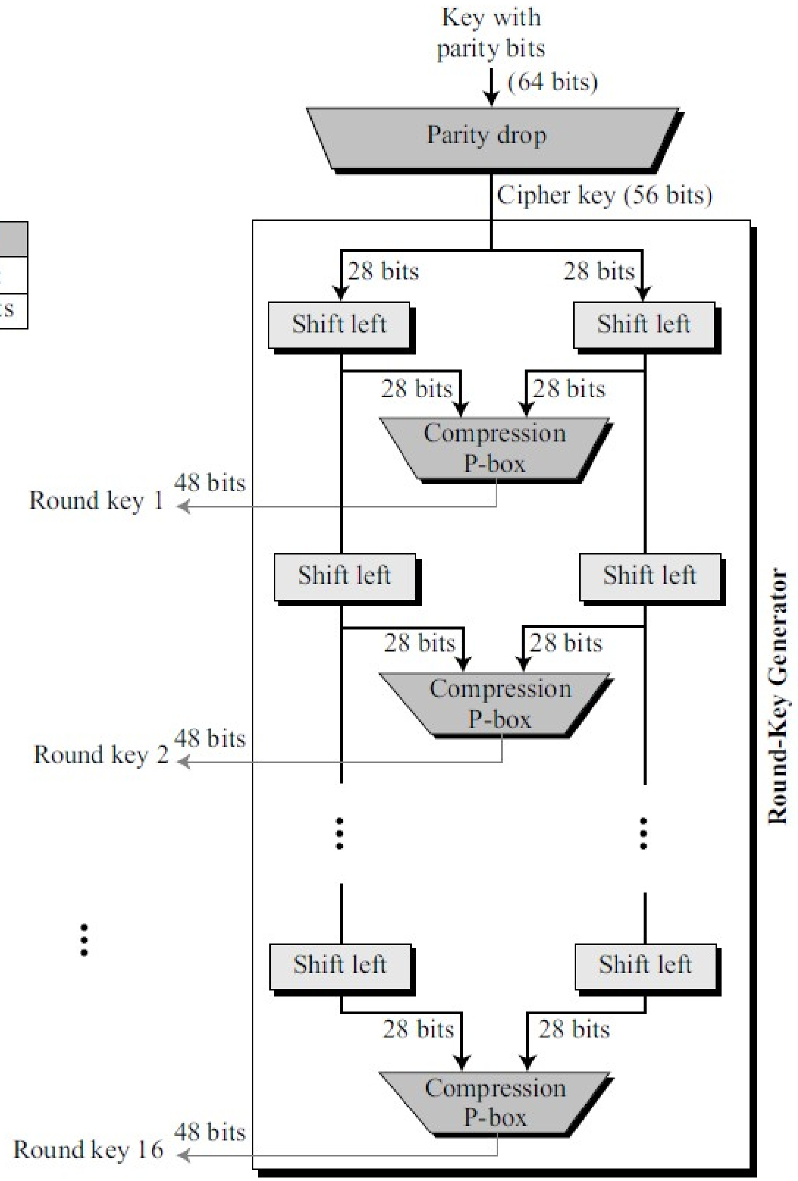
Key Generation

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 Figure 6.10.

Shifting

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



Parity 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 permutation (a compression P-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 Permutation

The compression permutation (P-box) changes the 58 bits to 48 bits, which are used as a key for a round.

The compression permutation 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 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])
        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
    }
}
```

Example 6.5

Plaintext: 123456ABCD132536 Key: AABB09182736CCDD

CipherText: C0B7A8D05F3A829C

Example 6.5

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
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)*

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

At the destination, Bob decipher the ciphertext received from Alice using the same key.

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

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.

6.3 DES ANALYSIS

Example : Encrypt 2 plaintext blocks (with the same key) that differ only in one bit.

Plaintext: 0000000000000000	Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1	
Plaintext: 000000000000000**1**	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.

6.3 DES ANALYSIS

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

6.3 DES ANALYSIS

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 P-boxes and S-boxes in DES, show a very strong completeness effect.

6.4 MULTIPLE DES

With available technology and the possibility of parallel processing, a brute-force attack on DES is feasible.

- ✓ One solution to improve the security of DES is to abandon DES and design a new cipher. (advent of AES.)

- ✓ The second solution is to use multiple (cascaded) instances of DES with multiple keys;

Double DES

The first approach is to use double DES (2DES).

This approach use two instances of DES ciphers for encryption and two instances of reverse ciphers for decryption.

Each instance uses a different key, which means that the size of the key is now doubled (112 bits).

Double DES is vulnerable to a known-plain text attack.

Triple DES

To improve the security of DES, triple DES (3DES) was proposed.

This uses three stages of DES for encryption and decryption.

Two versions of triple DES are in use today:

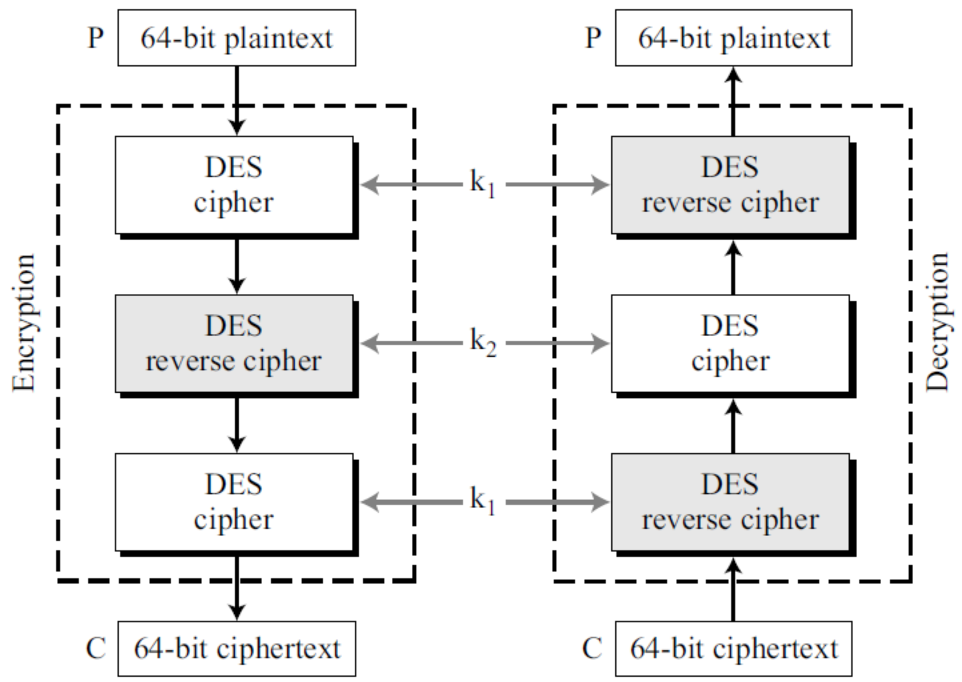
- ✓ triple DES with two keys and
- ✓ triple DES with three keys.

Triple DES with Two Keys

In triple DES with two keys, there are only two keys: k1 and k2.

The first and the third stages use k1; the second stage uses k2.

Figure 6.16 *Triple DES with two keys*



Triple DES with Three Keys

The possibility of known-plaintext attacks on triple DES with two keys has enticed some applications to use triple DES with three keys.