

# **1. Symmetric-Key Algorithms:**

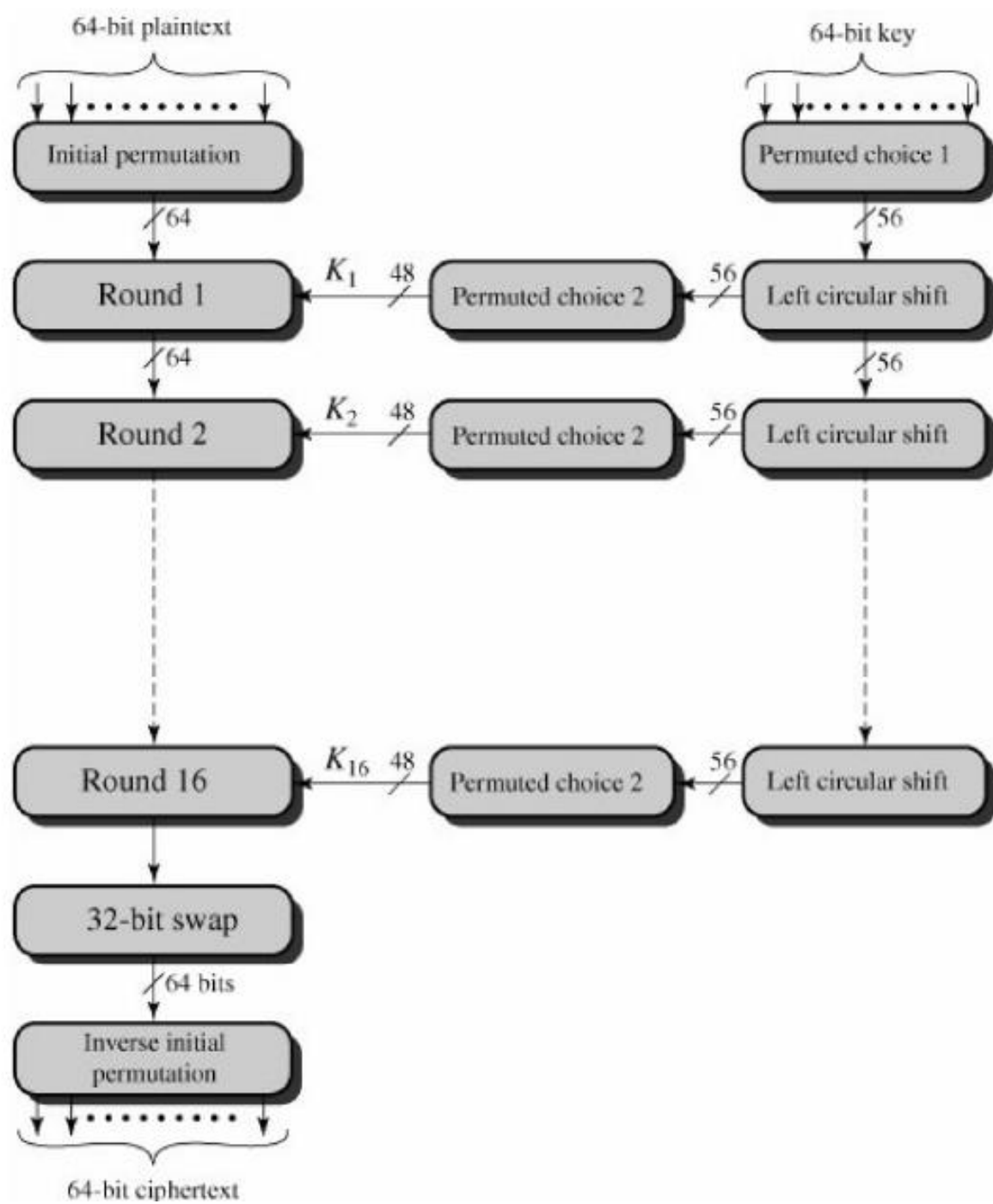
## **1.1 DES—The Data Encryption Standard, here**

### **1.2 16 round Feistel system**

### **3.1 DES—The Data Encryption Standard:**

The overall scheme for DES encryption is illustrated in Figure 3.4. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

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



The **left-hand side** of the figure, we can see that the processing of the plaintext proceeds in three phases.

- First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions.
- The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**.
- Finally, the preoutput is passed through a permutation (IP-1) that is the inverse of the initial permutation function, to produce the 64-bit cipher text. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

The **right-hand portion** of Figure shows the way in which the 56-bit key is used.

- Initially, the key is passed through a permutation function.
- Then, for each of the 16 rounds, a subkey ( $K_i$ ) is produced by the combination of a left circular shift and a permutation.
- The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

## **Initial Permutation**

The initial permutation and its inverse are defined by tables, as shown in Tables 3.2a and 3.2b, respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

**Table 3.2. Permutation Tables for DES**  
 (This item is displayed on page 76 in the print version)

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

(c) Expansion Permutation (E)

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

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10

(a) Initial Permutation (IP)

19	13	30	6	22	11	4	25
----	----	----	---	----	----	---	----

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

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



where  $M_j$  is a binary digit. Then the permutation  $X = IP(M)$  is as follows:

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

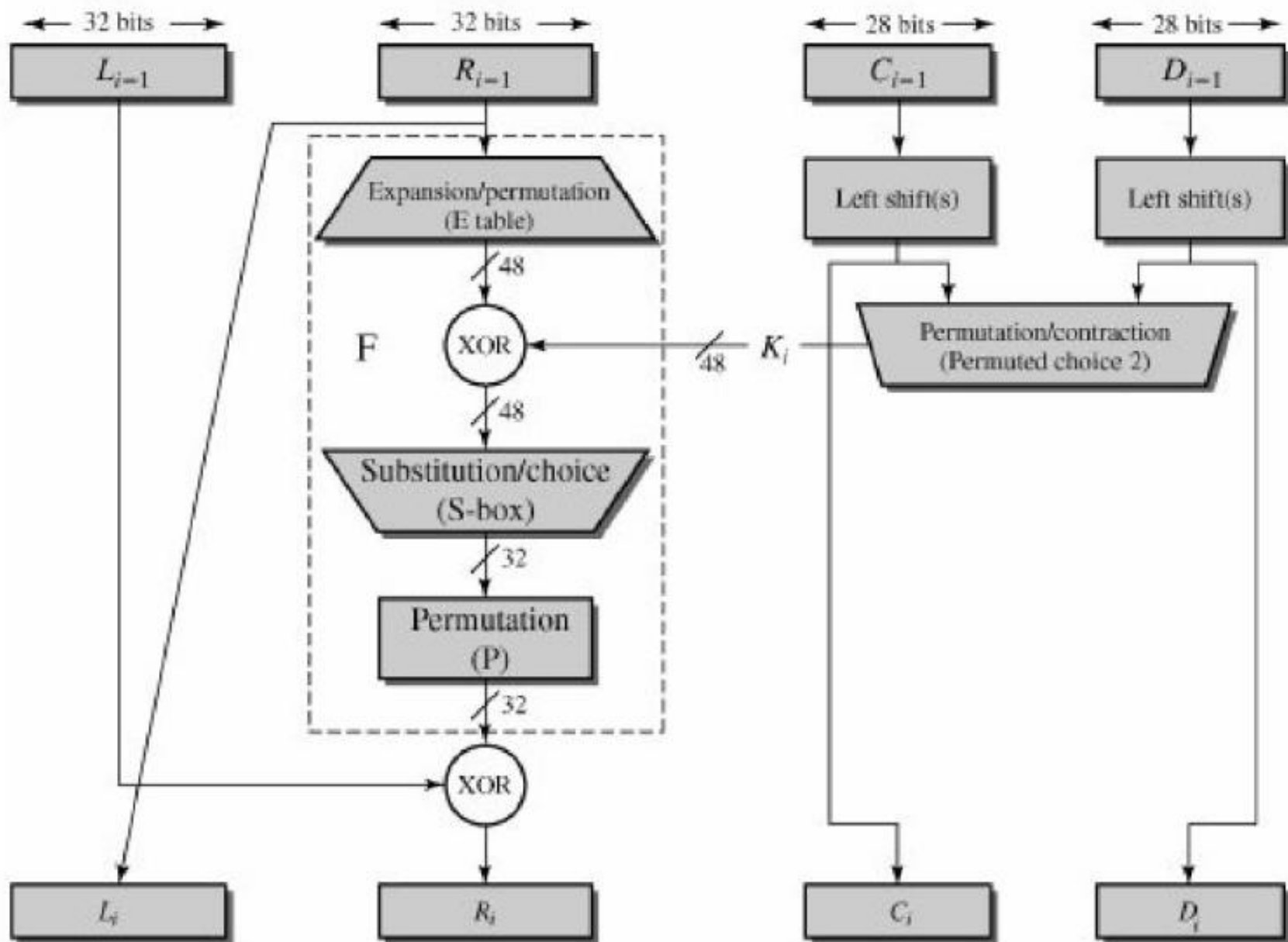
If we then take the inverse permutation  $Y = IP^{-1}(X) = IP^{-1}(IP(M))$ , it can be seen that the original ordering of the bits is restored.

## **Details of Single Round**

Figure 3.5 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$

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



The round key  $K_i$  is 48 bits. The  $R$  input is 32 bits. This  $R$  input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the  $R$  bits (Table 3.2c). The resulting 48 bits are XORed with  $K_i$ .

This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 3.2d.

The role of the S-boxes in the function  $F$  is illustrated in Figure 3.6. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box  $S_i$  form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for  $S_i$ . The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in  $S_1$  for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

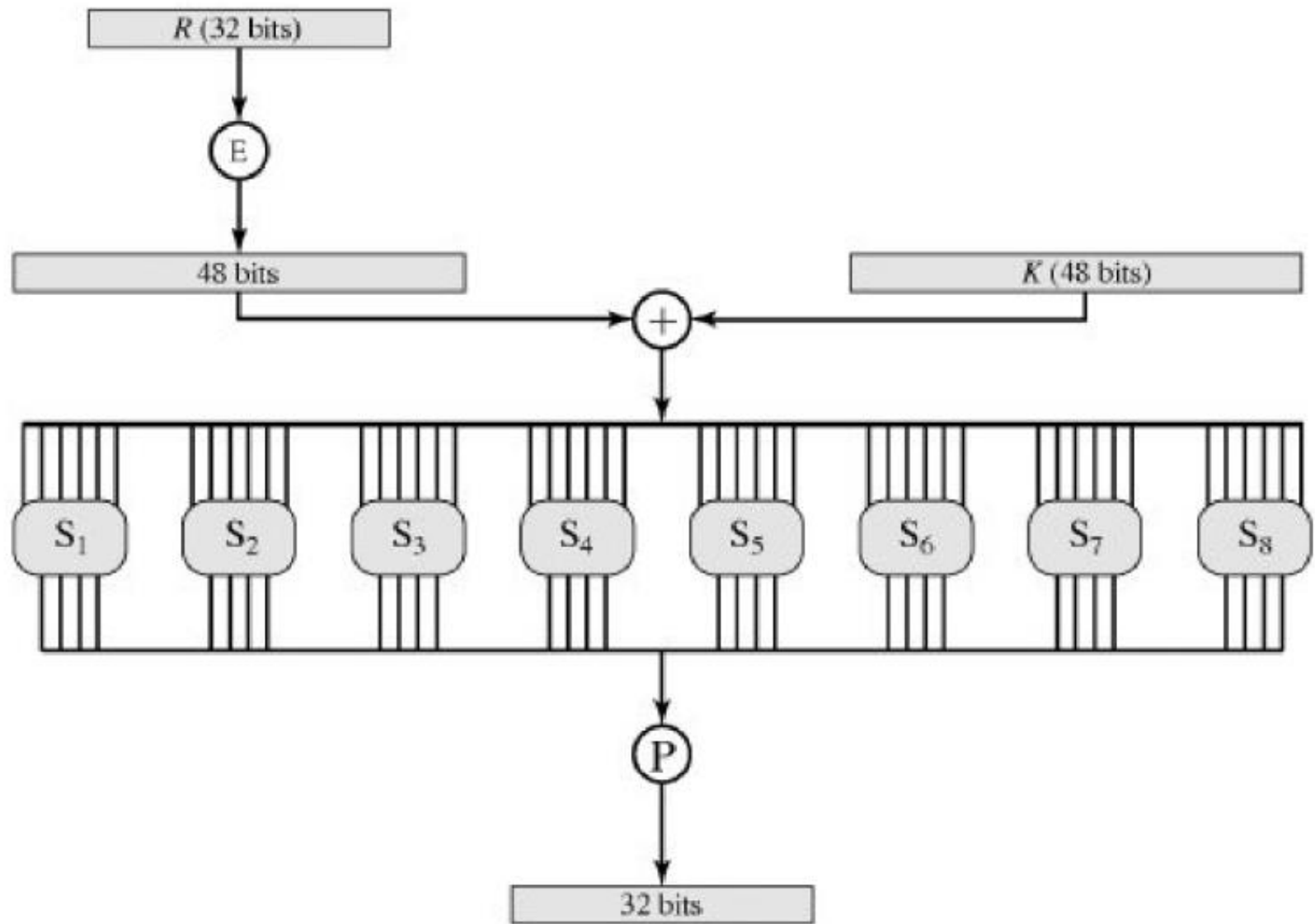


Table 3.3. Definition of DES S-Boxes.

$S_1$	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S_2$	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

$S_3$	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$S_4$	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

$S_5$ 

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

 $S_6$ 

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

 $S_7$ 

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

 $S_8$ 

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

this becomes

... defghi hijklm lmnopq ...

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



## Key Generation

Returning to Figures 3.4 and 3.5, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled  $C_0$  and  $D_0$ . At each round,  $C_{i-1}$  and  $D_{i-1}$  are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 3.4d.

These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the function  $F(R_{i-1}, K_i)$ .

**Table 3.4. DES Key Schedule Calculation****(a) Input Key**

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

**(b) Permuted Choice One (PC-1)**

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

**(c) Permuted Choice Two (PC-2)**

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

**(d) Schedule of Left Shifts**

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

This example is neat and orderly because our plaintext was exactly 64 bits long. The same would be true if the plaintext happened to be a multiple of 64 bits. But most messages will not fall into this category. They will not be an exact multiple of 64 bits (that is, an exact multiple of 16 hexadecimal numbers).

For example, take the message "Your lips are smoother than vaseline". This plaintext message is 38 bytes (76 hexadecimal digits) long. So this message must be padded with some extra bytes at the tail end for the encryption. Once the encrypted message has been decrypted, these extra bytes are thrown away. There are, of course, different padding schemes--different ways to add extra bytes. Here we will just add 0s at the end, so that the total message is a multiple of 8 bytes (or 16 hexadecimal digits, or 64 bits).

The plaintext message "Your lips are smoother than vaseline" is, in hexadecimal,

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365  
6C696E650D0A".

(Note here that the first 72 hexadecimal digits represent the English message, while "0D" is hexadecimal for Carriage Return, and "0A" is hexadecimal for Line Feed, showing that the message file has terminated.) We then pad this message with some 0s on the end, to get a total of 80 hexadecimal digits:

"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365  
6C696E650D0A0000".

If we then encrypt this plaintext message 64 bits (16 hexadecimal digits) at a time, using the same DES key "0E329232EA6D0D73" as before, we get the ciphertext:

"C0999FDDE378D7ED 727DA00BCA5A84EE 47F269A4D6438190  
9DD52F78F5358499 828AC9B453E0E653".

## How DES Works in Detail

DES is a *block cipher*--meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a *permutation* among the  $2^{64}$  (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block **L** and a right half **R**. (This division is only used in certain operations.)

**Example:** Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

**M** = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101  
1110 1111  
**L** = 0000 0001 0010 0011 0100 0101 0110 0111  
**R** = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered

8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create subkeys.

**Example:** Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

**K** = 00010011 00110100 01010111 01111001 10011011 10111100 11011111  
11110001

The DES algorithm uses the following steps:

## **Step 1: Create 16 subkeys, each of which is 48-bits long.**

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K+**. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

**PC-1**

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



**Example:** From the original 64-bit key

$\mathbf{K} = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001$

we get the 56-bit permutation

$\mathbf{K}^+ = 1111000\ 0110011\ 0010101\ 0101111\ 0101010\ 1011001\ 1001111\ 0001111$

Next, split this key into left and right halves,  $C_0$  and  $D_0$ , where each half has 28 bits.

**Example:** From the permuted key  $\mathbf{K}^+$ , we get

$C_0 = 1111000\ 0110011\ 0010101\ 0101111$

$D_0 = 0101010\ 1011001\ 1001111\ 0001111$

With  $C_0$  and  $D_0$  defined, we now create sixteen blocks  $C_n$  and  $D_n$ ,  $1 \leq n \leq 16$ . Each pair of blocks  $C_n$  and  $D_n$  is formed from the previous pair  $C_{n-1}$  and  $D_{n-1}$ , respectively, for  $n = 1, 2, \dots, 16$ , using the following schedule of "left shifts" of the previous block.

To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

This means, for example,  $C_3$  and  $D_3$  are obtained from  $C_2$  and  $D_2$ , respectively, by two left shifts, and  $C_{16}$  and  $D_{16}$  are obtained from  $C_{15}$  and  $D_{15}$ , respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3, ..., 28, 1.

**Example:** From original pair pair  $C_0$  and  $D_0$  we obtain:

$$C_1 = 1110000110011001010101011111$$

$$D_1 = 1010101011001100111100011110$$

$$C_2 = 1100001100110010101010111111$$

$$D_2 = 0101010110011001111000111101$$

$$C_3 = 0000110011001010101011111111$$

$$D_3 = 0101011001100111100011110101$$

$$C_4 = 0011001100101010101111111100$$

$$D_4 = 0101100110011110001111010101$$

$$C_5 = 1100110010101010111111110000$$

$$D_5 = 0110011001111000111101010101$$

$$C_6 = 0011001010101011111111000011$$

$$D_6 = 1001100111100011110101010101$$

$$C_7 = 1100101010101111111100001100$$

$$D_7 = 0110011110001111010101010110$$

$$C_8 = 0010101010111111110000110011$$

$$D_8 = 1001111000111101010101011001$$

$$C_9 = 0101010101111111100001100110$$

$$D_9 = 0011110001111010101010110011$$

$$C_{10} = 0101010111111110000110011001$$

$$D_{10} = 1111000111101010101011001100$$

$$C_{11} = 0101011111111000011001100101$$

$$D_{11} = 1100011110101010101100110011$$

$$C_{12} = 0101111111100001100110010101$$

$$D_{12} = 0001111010101010110011001111$$

$$C_{13} = 0111111110000110011001010101$$

$$D_{13} = 0111101010101011001100111100$$

$$C_{14} = 1111111000011001100101010101$$

$$D_{14} = 1110101010101100110011110001$$

$$C_{15} = 1111100001100110010101010111$$

$$D_{15} = 1010101010110011001111000111$$

$$C_{16} = 1111000011001100101010101111$$

$$D_{16} = 0101010101100110011110001111$$

We now form the keys  $K_n$ , for  $1 \leq n \leq 16$ , by applying the following permutation table to each of the concatenated pairs  $C_n D_n$ . Each pair has 56 bits, but PC-2 only uses 48 of these.

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

Therefore, the first bit of  $K_n$  is the 14th bit of  $C_n D_n$ , the second bit the 17th, and so on, ending with the 48th bit of  $K_n$  being the 32th bit of  $C_n D_n$ .

**Example:** For the first key we have  $C_1 D_1 = 1110000\ 1100110\ 0101010\ 1011111\ 1010101\ 0110011\ 0011110\ 0011110$

which, after we apply the permutation PC-2, becomes

$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$

For the other keys we have

$K_2 = 011110\ 011010\ 111011\ 011001\ 110110\ 111100\ 100111\ 100101$

$K_3 = 010101\ 011111\ 110010\ 001010\ 010000\ 101100\ 111110\ 011001$

$K_4 = 011100\ 101010\ 110111\ 010110\ 110110\ 110011\ 010100\ 011101$

$K_5 = 011111\ 001110\ 110000\ 000111\ 111010\ 110101\ 001110\ 101000$

$K_6 = 011000\ 111010\ 010100\ 111110\ 010100\ 000111\ 101100\ 101111$

$K_7 = 111011\ 001000\ 010010\ 110111\ 111101\ 100001\ 100010\ 111100$

$K_8 = 111101\ 111000\ 101000\ 111010\ 110000\ 010011\ 101111\ 111011$

$K_9 = 111000\ 001101\ 101111\ 101011\ 111011\ 011110\ 011110\ 000001$

$K_{10} = 101100\ 011111\ 001101\ 000111\ 101110\ 100100\ 011001\ 001111$

$K_{11} = 001000\ 010101\ 111111\ 010011\ 110111\ 101101\ 001110\ 000110$

$K_{12} = 011101\ 010111\ 000111\ 110101\ 100101\ 000110\ 011111\ 101001$

$K_{13} = 100101\ 111100\ 010111\ 010001\ 111110\ 101011\ 101001\ 000001$

$K_{14} = 010111\ 110100\ 001110\ 110111\ 111100\ 101110\ 011100\ 111010$

$K_{15} = 101111\ 111001\ 000110\ 001101\ 001111\ 010011\ 111100\ 001010$

$K_{16} = 110010\ 110011\ 110110\ 001011\ 000011\ 100001\ 011111\ 110101$

So much for the subkeys. Now we look at the message itself.

## Step 2: Encode each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

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



**Example:** Applying the initial permutation to the block of text **M**, given previously, we get

**M** = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101  
1110 1111

**IP** = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000  
1010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

Next divide the permuted block **IP** into a left half  $L_0$  of 32 bits, and a right half  $R_0$  of 32 bits.

**Example:** From **IP**, we get  $L_0$  and  $R_0$

$L_0$  = 1100 1100 0000 0000 1100 1100 1111 1111

$R_0$  = 1111 0000 1010 1010 1111 0000 1010 1010

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$



**Example:** For  $n = 1$ , we have

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

$$L_1 = R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

$$R_1 = L_0 + f(R_0, K_1)$$

It remains to explain how the function  $f$  works. To calculate  $f$ , we first expand each block  $R_{n-1}$  from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in  $R_{n-1}$ . We'll call the use of this selection table the function  $E$ .

Thus  $E(R_{n-1})$  has a 32 bit input block, and a 48 bit output block.

Let  $E$  be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

**E BIT-SELECTION TABLE**

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

Thus the first three bits of  $E(R_{n-1})$  are the bits in positions 32, 1 and 2 of  $R_{n-1}$  while the last 2 bits of  $E(R_{n-1})$  are the bits in positions 32 and 1.

---

**Example:** We calculate  $E(R_0)$  from  $R_0$  as follows:

$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$

$E(R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the  $f$  calculation, we XOR the output  $E(R_{n-1})$  with the key  $K_n$ :

$$K_n + E(R_{n-1}).$$

**Example:** For  $K_1$ ,  $E(R_0)$ , we have

$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$

$E(R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

$K_1 + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$

We have not yet finished calculating the function  $f$ . To this point we have expanded  $R_{n-1}$  from 32 bits to 48 bits, using the selection table, and XORed the result with the key  $K_n$ . We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different S box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the S boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each  $B_i$  is a group of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

**S1**

		Column Number															
Row No.		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If  $S_i$  is the function defined in this table and  $B$  is a block of 6 bits, then  $S_i(B)$  is determined as follows: The first and last bits of  $B$  represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be  $i$ . The middle 4 bits of  $B$  represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be  $j$ . Look up in the table the number in the  $i$ -th row and  $j$ -th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output  $S_i(B)$  of  $S_i$  for the input  $B$ . For example, for input block  $B = 011011$

011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence  $S_I(011011) = 0101$ .

The tables defining the functions  $S_1, \dots, S_8$  are the following:

**s1**

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

**s2**

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

**s3**

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

**s4**

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

**s5**

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

**s6**

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

**s7**

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

**s8**

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

**Example:** For the first round, we obtain as the output of the eight S boxes:

$$K_1 + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$$

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

The final stage in the calculation of  $f$  is to do a permutation  $\mathbf{P}$  of the S-box output to obtain the final value of  $f$ :

$$f = \mathbf{P}(S_1(B_1)S_2(B_2)\dots S_8(B_8))$$

The permutation  $\mathbf{P}$  is defined in the following table.  $\mathbf{P}$  yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

$\mathbf{P}$			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

**Example:** From the output of the eight S boxes:

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = \begin{array}{cccccccc} 0101 & 1100 & 1000 & 0010 & 1011 & 0101 & & \\ & & & & & & 1001 & 0111 \end{array}$$

we get

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$R_1 = L_0 + f(R_0, K_1)$$

$$\begin{aligned} &= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111 \\ &+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011 \\ &= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100 \end{aligned}$$

In the next round, we will have  $L_2 = R_1$ , which is the block we just calculated, and then we must calculate  $R_2 = L_1 + f(R_1, K_2)$ , and so on for 16 rounds. At the end of the sixteenth round we have the blocks  $L_{16}$  and  $R_{16}$ . We then *reverse* the order of the two blocks into the 64-bit block

$$R_{16}L_{16}$$

and apply a final permutation  $IP^{-1}$  as defined by the following table:



$IP^{-1}$ 

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

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.



**Example:** If we process all 16 blocks using the method defined previously, we get, on the 16th round,

$$L_{16} = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$$

$$R_{16} = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$$

We reverse the order of these two blocks and apply the final permutation to

$$R_{16}L_{16} = 00001010\ 01001100\ 11011001\ 10010101\ 01000011\ 01000010\ 00110010\ 00110100$$

$$IP^{-1} = 10000101\ 11101000\ 00010011\ 01010100\ 00001111\ 00001010\ 10110100\ 00000101$$

which in hexadecimal format is

$$85E813540F0AB405.$$

This is the encrypted form of  $M = 0123456789ABCDEF$ : namely,  $C = 85E813540F0AB405$ .

Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the subkeys are applied.