3.1.
As stated in Sect. 3.5.2, one important property which makes DES secure is that the S-boxes are nonlinear. In this problem we verify this property by computing the output of S1 for several pairs of inputs.
Show that S1(x1)⊕S1(x2) ≠ S1(x1 ⊕x2), where "⊕" denotes bitwise XOR, for:
1. x1 = 000000, x2 = 000001
2. x1 = 111111, x2 = 100000
3. x1 = 101010, x2 = 010101

Answer:

Table 3.2 S-box S_1

S_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	01	10	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

1.x1 = 000000, x2 = 000001
S(x1):
row=0, col=0
So, result is 14 = 1110 (in binary)
S(x2)
row=1, col=0 val is 00 = 0000 (in binary)

So, S1(x1)⊕S1(x2) = 1110 ⊕ 0000 = 1110

S1(x1 ⊕x2)

x1 ⊕x2 = 000000 ⊕ 000001 = 000001
S1(x1 ⊕x2) =
For 000001,
row=1, col=0 val = 00 = 0000 (in binary)

So, S1(x1)⊕S1(x2) which is 1110 is not equal to S1(x1 ⊕x2) (=0000)

2. x1 = 111111, x2 = 100000

S1(x1):
row=3, col=15 (=1111 in binary)
So, result is 13 = 1101 (in binary)
S(x2)
row=2, col=0 val is 04 = 0100 (in binary)

So, S1(x1)⊕S1(x2) = 1101 ⊕ 0100 = 1001

S1(x1 ⊕x2)

x1 ⊕x2 = 111111 ⊕ 100000 = 011111
S1(x1 ⊕x2) =
For 011111,
row=1, col=15 val = 08 = 1000 (in binary)

So, S1(x1)⊕S1(x2) which is 1001 is not equal to S1(x1 ⊕x2) (=1000)

3. x1 = 101010, x2 = 010101

S1(x1):
row=2, col=5 (=0101 in binary)
So, result is 06 = 0110 (in binary)
S(x2)
row=1, col=10 val is 12 = 1100 (in binary)

So, S1(x1)⊕S1(x2) = 0110 ⊕ 1100 = 1010

S1(x1 ⊕x2)

x1 ⊕x2 = 101010 ⊕ 010101 = 111111
S1(x1 ⊕x2) =
For 111111,
row=3, col=15 val = 13 = 1101 (in binary)

So, S1(x1)⊕S1(x2) which is 1010 is not equal to S1(x1 ⊕x2) (=1101)

3.2.
We want to verify that IP(·) and IP−1(·) are truly inverse operations. We consider a vector x = (x1,x2,...,x64) of 64 bit. Show that IP−1(IP(x)) = x for the first five bits of x, i.e. for xi, i = 1,2,3,4,5.

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

Fig. 3.8 Initial permutation *IP*

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

Fig. 3.9 Final permutation IP^{-1}

For i=1
IP(x) will move to 40th position and with IP-1, 40th position bit moves to 1st where this bit was originally present

For i=2, IP(x) move to 8th position with IP-1, 8th position bit moves to 1st where this bit was originally present

For i=3,IP(x) move to 48th position with IP-1, 48th position bit moves to 3rd where this bit was originally present

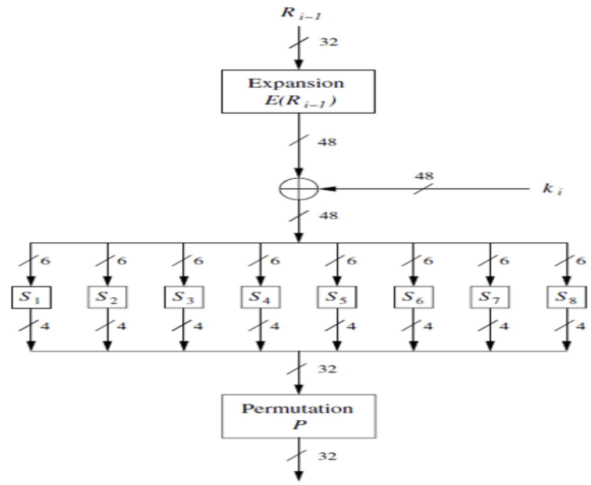
For i=4, IP(x) move to 16th position with IP-1, 16th position bit moves to 4th where this bit was originally present

For i=5, IP(x) move to 56th position with IP-1, 56th position bit moves to 5th where this bit was originally present

3.3.

What is the output of the first round of the DES algorithm when the plaintext and the key are both all zeros?

Computing result of f-function



ɔck diagram of the f-function

We start with 0's as input to f function. Expansion step will produce 48 bit 0 number. XOR of this number with 48 bit key which is also 0, will produce 48 bit 0 as result. This will be fed to the S boxes.

Table 3.2 S-box S_1

S_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	01	10	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

Each S-box contains 2^6=64 entries, which are typically represented by a table with 16 columns and 4 rows. Each entry is a 4-bit value. All S-boxes are listed in Tables 3.2 to 3.9. Note that all S-boxes are different.

Table 3.3 S-box S_2

S_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 3.4 S-box S_3

S_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 3.5 S-box S_4

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	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 3.6 S-box S_5

S_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 3.7 S-box S_6

S_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	06	00	08	13

Table 3.8 S-box S_7

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	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 3.9 S-box S_8

S_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	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Using these S boxes, we will get 1st entry in each S box as our transformed result as following.

S1 -> 14
S2 -> 15
S3 -> 10
S4 -> 07
S5 -> 02
S6 -> 12
S7 -> 04
S8 -> 13

Writing this in binary

1110 1111 1010 0111 0010 1100 0100 1101

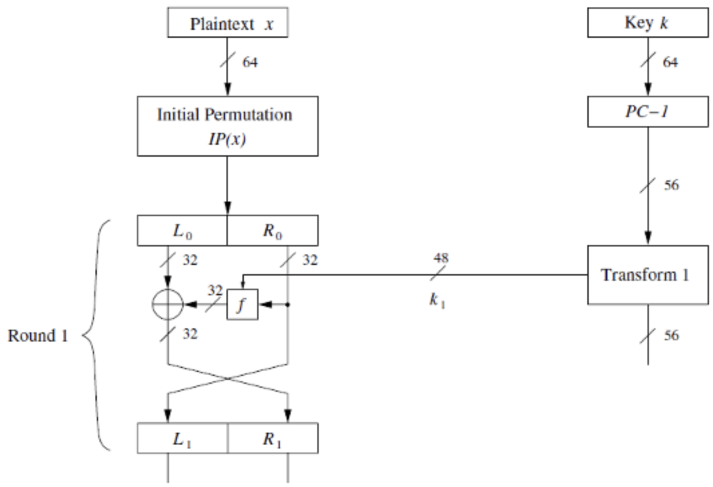
Finally applying permutation

Table 3.10 The permutation P within the f-function

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

We get the following:
1101 1000 1101 1000 1101 1011 1011 1100

We will XOR the above result of f-function with the 32 left bits which are all 0.



2/21/23 Professor Iona

So, we will get the answer as

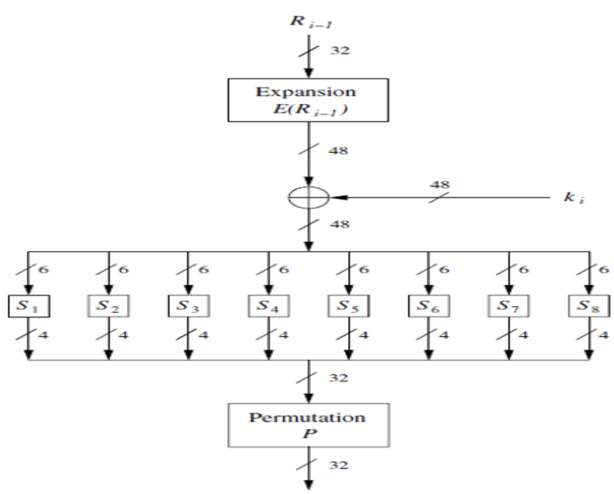
1101 1000 1101 1000 1101 1011 1011 1100

As XOR with 0 does not change the result.

For the next round,
R1 will be 1101 1000 1101 1000 1101 1011 1011 1100
L1 will be 0000 0000 0000 0000 0000 0000 0000 0000

3.4.
What is the output of the first round of the DES algorithm when the plaintext and the key are both all ones?

Computing result of f-function



ɔck diagram of the f-function

We start with 1’s as input to f function. Expansion step will produce 48 bit 1’s number. XOR of this number with 48 bit key which is also 1, will produce 48 bit 0 as result. This will be fed to the S boxes.

Table 3.2 S-box S_1

S_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	01	10	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

Each S-box contains 2^6=64 entries, which are typically represented by a table with 16 columns and 4 rows. Each entry is a 4-bit value. All S-boxes are listed in Tables 3.2 to 3.9. Note that all S-boxes are different.

Table 3.3 S-box S_2

S_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 3.4 S-box S_3

S_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 3.5 S-box S_4

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	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 3.6 S-box S_5

S_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 3.7 S-box S_6

S_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	06	00	08	13

Table 3.8 S-box S_7

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	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 3.9 S-box S_8

S_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	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Using these S boxes, we will get 1st entry in each S box as our transformed result as following.

S1 -> 14
S2 -> 15

S3 -> 10
S4 -> 07
S5 -> 02
S6 -> 12
S7 -> 04
S8 -> 13

Writing this in binary

1110 1111 1010 0111 0010 1100 0100 1101

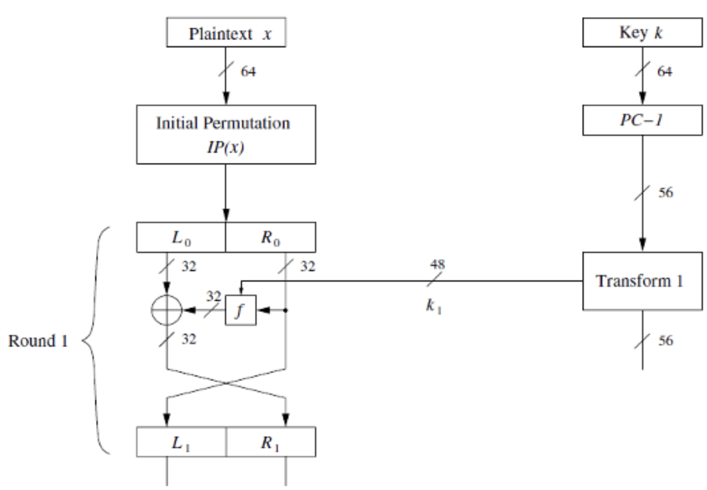
Finally applying permutation

Table 3.10 The permutation P within the f-function

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

We get the following:
1101 1000 1101 1000 1101 1011 1011 1100

We will XOR the above result of f-function with the 32 left bits which are all 1.



1101 1000 1101 1000 1101 1011 1011 1100 XOR
1111 1111 1111 1111 1111 1111 1111 1111
result= 0010 0111 0010 0111 0010 0100 0100 0011

For the next round,
R1 will be 0010 0111 0010 0111 0010 0100 0100 0011

L1 will be 1111 1111 1111 1111 1111 1111 1111 1111

3.5.
Remember that it is desirable for good block ciphers that a change in one input bit affects many output bits, a property that is called diffusion or the avalanche effect. We try now to get a feeling for the avalanche property of DES. We apply an input word that has a "1" at bit position 57 and all other bits as well as the key are zero. (Note that the input word has to run through the initial permutation.)
1. How many S-boxes get different inputs compared to the case when an all-zero plaintext is provided?
2. What is the minimum number of output bits of the S-boxes that will change according to the S-box design criteria?
3. What is the output after the first round?
4. How many output bit after the first round have actually changed compared to
the case when the plaintext is all zero? (Observe that we only consider a single round here. There will be more and more output differences after every new round. Hence the term avalanche effect.)

Answer:
Applying initial transformation to the input

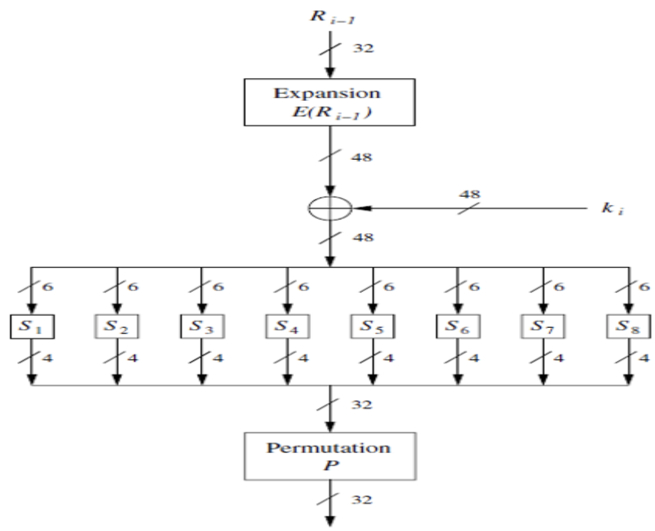
IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Fig. 3.8 Initial permutation *IP*

After applying IP, the 57th bit will move to the 33rd position.
So, our input will be transformed to

0000 0000 0000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000
So,
L0 is 0000 0000 0000 0000 0000 0000 0000 0000
R0 is 1000 0000 0000 0000 0000 0000 0000 0000

Computing result of f-function for R0



ɔck diagram of the f-function

In the Expansion step, 1st bit which is the only set bit, will be copied to 2nd and 48th positions.

So, our input is transformed from R0 to

0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001.

Since the key is 0, the XOR with the key will not change the above result.

Now, this result is fed to S boxes.

Bits going to S1 = 010000 -> row=0, col=8, val= 03
Bits going to S2 = 000000 -> 15
Bits going to S3 = 000000 -> 10
Bits going to S4 = 000000 -> 07
Bits going to S5 = 000000 -> 02
Bits going to S6 = 000000 -> 12
Bits going to S7 = 000000 -> 04
Bits going to S8 = 000001 -> row=1, col=0 val=01

Writing this in binary

0011 1111 1010 0111 0010 1100 0100 0001

1. In this case, S2 to S7 still take 0 as input but S1 and S8 take different inputs. S1 takes 010000 as input and S8 takes 000001 as input
2. In 0 as input, we got following as output of S boxes

1110 1111 1010 0111 0010 1100 0100 1101

3 bits are different in output for S1.
2 bits are different in output for S8.

This matches the S-box design criteria i..e if 1 bit is changed in input, 2 or more bits will be changed in output.

3. To compute final output after first round, first we apply the permutation to:

0011 1111 1010 0111 0010 1100 0100 0001

Applying permutation,

Table 3.10 The permutation P within the f-function

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

We get the following
1101 0000 0101 1000 0101 1011 1001 1110

Then we need to XOR this result with L0 but because L0 is 0, this will not change the above result.
So, our answer after 1st round is

L1=1000 0000 0000 0000 0000 0000 0000 0000
R1=1101 0000 0101 1000 0101 1011 1001 1110

4. When we used all 0s, our result was
L1 will be 0000 0000 0000 0000 0000 0000 0000 0000
R1 will be 1101 1000 1101 1000 1101 1011 1011 1100

Now, our result is

L1=1000 0000 0000 0000 0000 0000 0000 0000
R1=1101 0000 0101 1000 0101 1011 1001 1110

So, there is 1 bit difference is L1 and 5 bits difference in R1.

3.6.
An avalanche effect is also desirable for the key: A one-bit change in a key should result in a dramatically different ciphertext if the plaintext is unchanged.
1. Assume an encryption with a given key. Now assume the key bit at position 1 (prior to PC − 1) is being flipped. Which S-boxes in which rounds are affected by the bit flip during DES encryption?
2. Which S-boxes in which DES rounds are affected by this bit flip during DES decryption?

Answer:

1.
THe bit at position 1, which is flipped, will be moved to position 8 by PC-1 step.

- **Parity bits are removed** in a first **permuted choice** *PC-1*:
  (note that the bits 8, 16, 24, 32, 40, 48, 56 and 64 are not used at all)

$PC-1$							
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

Round 1
8th position bit will be in C0.
After applying 1 left rotation in first round, 8th position bit will move to 7th position.
When we apply, PC-2 transformation, 7th position bit will move to 20th in k1

- *In each round i permuted choice* ***PC-2***
  selects a permuted subset of 48 bits of C_i and D_i as round key k_i, i.e. **each k_i is a permutation of k**!

$PC-2$							
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

Round 2
In round 2, 7th position bit will be moved to 6th bit
When we apply, PC-2 transformation, 6th position bit will move to 10th
Continuing this way, we get following table

Round	Position in C	Position in ki	S-box affected
1	7	20	4
2	6	10	2
3	4	16	3
4	2	24	4
5	28	8	2
6	26	17	3
7	24	4	1
8	22	Not present in PC-2	-
9	21	11	2
10	19	14	3
11	17	2	1
12	15	9	2
13	13	23	4
14	11	3	1
15	9	Not present in PC-2	-
16	8	18	3

2. During decryption, we have C0=16. Our bit is at position 8th. It will be in position 8th in k16.

Continuing this way, we get following table

Round	Position in C	Position in ki	S-box affected
1	8	18	3
2	9	Not present in PC-2	-
3	11	3	1
4	13	23	4
5	15	9	2
6	17	2	1
7	19	14	3
8	21	11	2
9	22	Not present in PC-2	-
10	24	4	1
11	26	17	3
12	28	8	2
13	2	24	4
14	4	16	3
15	6	10	2
16	7	20	4

Note that, above table is the same as the first part but in reverse.

3.9.
Assume we perform a known-plaintext attack against DES with one pair of plaintext and ciphertext. How many keys do we have to test in a worst-case scenario if we apply an exhaustive key search in a straightforward way? How many on average?

Answer:

We know that key bit length in DES encryption is 56 bits. So, the number of possible keys are 2^56. We can brute force through all of these keys and the plaintext. If we get the given ciphertext as our output for one of the keys, most likely(assuming no 2 keys produce the same given ciphertext from plaintext) that will be the correct key.

On average, we will have to go through half the total number of keys i..e 2^56/2 = 2^55.
