How DES Works in Detail

DES is a ***block cipher***--meaning it operates on plaintext blocks of a given size (64-bits) and returns cipher text 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** = NEVRQUIT,

Plain text in ASCII: 78 69 86 82 81 85 73 84

Plain text in Hex=4e 45 56 52 51 55 49 54

M = 01001110 01000101 01010110 01010010 01010001 01010101 01001001 01010100
L = 01001110 01000101 01010110 01010010
R = 01010001 01010101 01001001 01010100

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 sub keys.

Example: Let **K** be the key **K** = KASHISAB.

Key in ASCII= 75 65 83 72 73 83 65 66

Key in Hex=4b 41 53 48 49 53 41 42

In binary the key is

K=01001011 01000001 01010011 01001000 01001001 01010011 01000001 01000010

The DES algorithm uses the following steps:

Create 16 sub keys, 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

K=01001011 01000001 01010011 01001000 01001001 01010011 01000001 01000010

K + =0000000 0111111 1100000 0000010 1010010 1000000 0000011 0010100

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 **K**+, we get

C0=0000000 0111111 1100000 0000010

D0=1010010 1000000 0000011 0010100

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, 1<=n<=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... 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 C_0 and D_0 we obtain:

C_1 = 0000000 1111111 1000000 0000100
D_1 = 0100101 0000000 0000110 0101001

C_2 = 0000001 1111111 0000000 0001000
D_2 = 1001010 0000000 0001100 1010010

C_3 = 0000111 1111100 0000000 0100000
D_3 = 0101000 0000000 0110010 1001010

C_4 = 0011111 1110000 0000001 0000000
D_4 = 0100000 0000001 1001010 0101001

C_5 = 1111111 1000000 0000100 0000000
D_5 = 0000000 0000110 0101001 0100101

C_6 = 1111110 0000000 0010000 0000011
D_6 = 0000000 0011001 0100101 0010100

C_7 = 1111000 0000000 1000000 0001111
D_7 = 0000000 1100101 0010100 1010000

C_8 = 1100000 0000010 0000000 0111111
D_8 = 0000011 0010100 1010010 1000000

C_9 = 1000000 0000100 0000000 1111111
D_9 = 0000110 0101001 0100101 0000000

C_{10} = 0000000 0010000 0000011 1111110
D_{10} =0011001 0100101 0010100 0000000

C_{11} = 0000000 1000000 0001111 1111000
D_{11} =1100101 0010100 1010000 0000000

C_{12} = 0000010 0000000 0111111 1100000
D_{12} =0010100 1010010 1000000 0000011

C_{13} = 0001000 0000001 1111111 0000000
D_{13} = 1010010 1001010 0000000 0001100

C_{14} = 0100000 0000111 1111100 0000000
D_{14} = 1001010 0101000 0000000 0110010

C_{15} = 0000000 0011111 1110000 0000001
D_{15} =0101001 0100000 0000001 1001010

C_{16} = 0000000 0111111 1100000 0000010
D_{16} =1010010 1000000 0000011 0010100

We now form the keys K_n, for 1<=n<=16, by applying the following permutation table to each of the concatenated pairs C_nD_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_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of C_nD_n.

Example: For the first key we have

C_1 = 0000000 1111111 1000000 0000100
D_1 = 0100101 0000000 0000110 0101001

Which after we apply the permutation **PC-2**, becomes

K_1 = 101000 001001 001011 000010 000010 101011 000101 000000

For the other keys we have

K_2 = 101000 000001 001001 010010 010011 000000 000010 101011
K_3 = 001001 000101 101001 010000 000001 100101 100001 001001
K_4 = 000001 100111 000101 010000 000000 101001 000101 110000
K_5 = 000011 100100 010101 010001 100000 011000 110100 100000

K_6 = 010011 110100 000100 001001 010010 000000 111000 010000
K_7 = 000010 111000 000110 001001 010110 010100 000000 011100
K_8 = 000110 010000 100010 001011 000000 010101 000010 001000
K_9 = 000110 010000 101010 001000 000110 000010 111010 010000
K_{10} =000100 000011 100010 001100 001110 010100 000000 010001

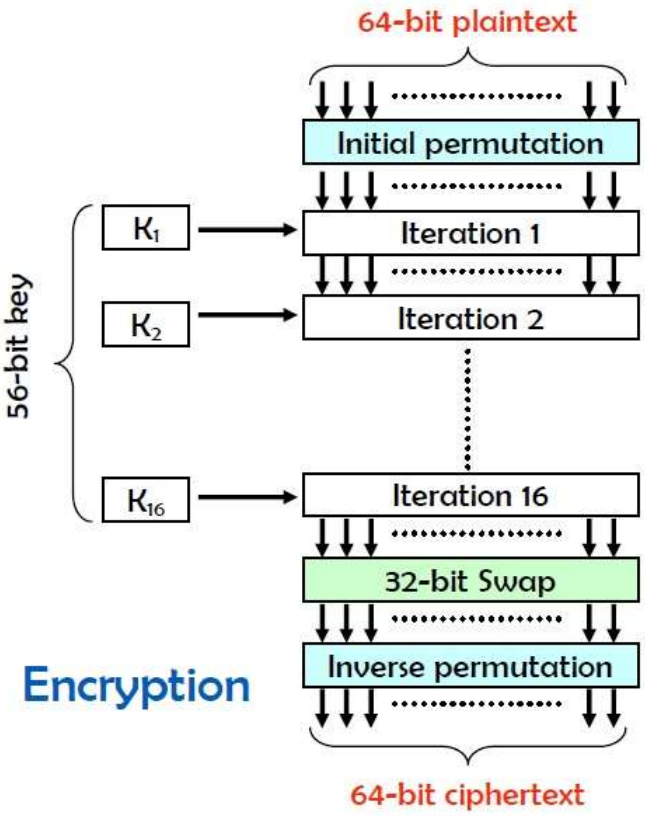
K_{11} =000100 000010 110001 000100 000000 110110 000000 000010

K_{12} =010000 000110 110000 100100 101001 000010 000100 000100

K_{13} =110000 001010 010100 100100 101000 000000 001011 000110
K_{14} =110000 001000 011000 100011 010101 001000 001010 000011
K_{15} =111000 011001 001000 100010 000101 100000 010001 001001

And at last
K_{16} =101000 001001 001000 101010 011000 000001 010010 000110

Encryption



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**.

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

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

M = 01001110 01000101 01010110 01010010 01010001 01010101 01001001 01010100

IP = 11111111 10111100 10100111 01110010 00000000 00000000 01000001 00001101

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

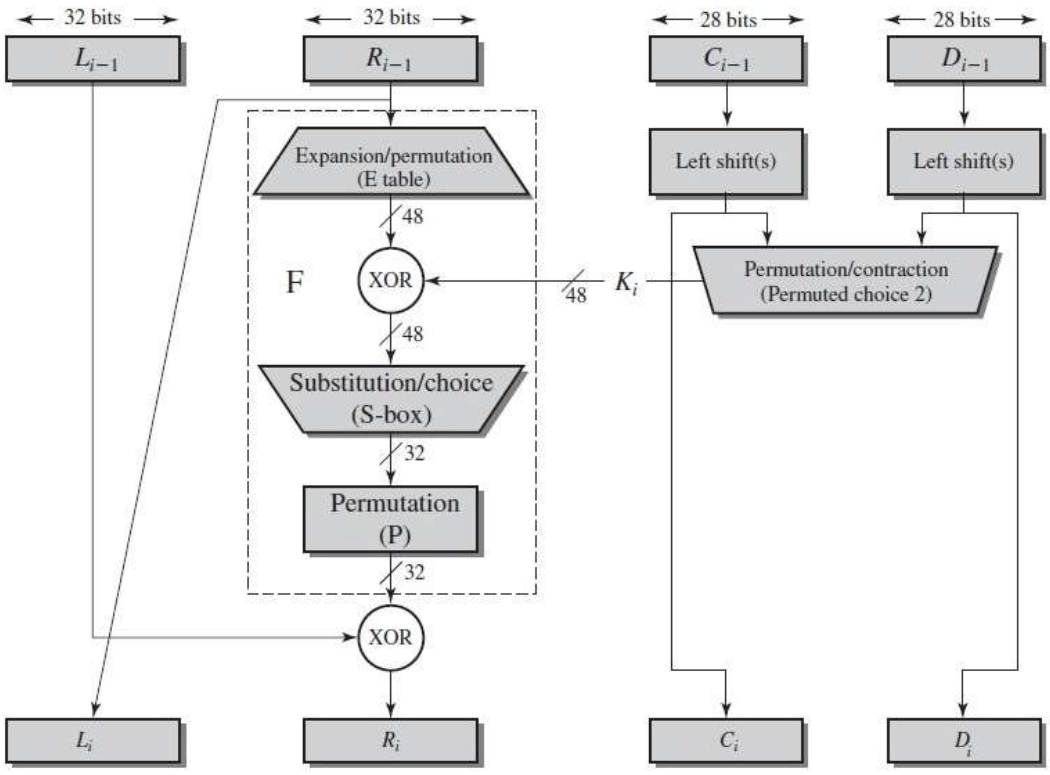
Example: From **IP**, we get L_0 and R_0

L_0 = 11111111 10111100 10100111 01110010
R_0 = 00000000 00000000 01000001 00001101

We now proceed through 16 iterations, for $1 <= n <= 16$, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2)**. Then for **n** going from 1 to 16 we calculate

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

This results in a final block, for $n = 16$, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f.



Single Round of DES Algorithm

Example: For $n = 1$, we have

K1=101000 001001 0010110 00010 000010 101011 000101 000000

$L_1 = R_0$ = 00000000 00000000 01000001 00001101
$R_1 = L_0 + f(R_0,K_1)$

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 = 00000000 00000000 01000001 00001101

E(R_0) = 100000 000000 000000 000000 001000 000010 100001 011010

(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:

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

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

K_1 = 101000 001001 001011 000010 000010 101011 000101 000000
E(R_0) = 100000 000000 000000 000000 001000 000010 100001 011010
K_1+**E**(R_0) = 001000 001001 001011 000010 001010 101001 100100 011010

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:

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

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

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

where $S_i(B_i)$ referres to the output of the i-th **S** box.

To repeat, each of the functions ***S1, S2,..., S8***, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained below:

S1

Row No.	Column Number															
	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_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ 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 $S_1(B)$ of S_1 for the input B. For example, for input block B = 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 $S_1(011011) = 0101$.

The tables defining the functions $S_1,...,S_8$ 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:

$K_1 + \mathbf{E}(R_0)$ = 001000 001001 001011 000010 001010 101001 100100 011010.

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

0010 1111 0100 1101 1010 1001 1011 0000

The final stage in the calculation of f is to do a permutation **P** of the **S**-box output to obtain the final value of f:

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

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

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:

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

0010 1111 0110 1101 1010 1001 1011 0000

f* = *1101 0011 0000 1001 0111 0110 1101 0001

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

= 1111 1111 1011 1100 1010 0111 0111 0010

+ 1101 0011 0000 1001 0111 0110 1101 0001

So now

L1=0000 0000 0000 0000 0100 0001 0000 1101
R1= 0010 1100 1011 0101 1101 0001 1010 0011

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

Step2……

L2=0010 1100 1011 0101 1101 0001 1010 0011

R2=L1 +f (R1, K2)

f (R1,K2)= 0010 1100 1010 1000 1011 0000 0101 0011

R2=0000 0000 0000 0000 0100 0001 0000 1101
+

0010 1100 1010 1000 1011 0000 0101 0011=

0010 1100 1010 1000 1111 0001 0101 1110

L2=0010 1100 1011 0101 1101 0001 1010 0011

R2=0010 1100 1010 1000 1111 0001 0101 1110

Step 3……

L3=0010 1100 1010 1000 1111 0001 0101 1110

R3=L2+f(R2, K3)

f(R2, K3)= 10001000101001010101101111010110

R3= 0010 1100 1011 0101 1101 0001 1010 0011

+

1000 1000 1010 0101 0101 1011 1101 0110

= 1010 0100 0001 0000 1000 1010 0111 0101

L3=0010 1100 1010 1000 1111 0001 0101 1110

R3=1010 0100 0001 0000 1000 1010 0111 0101

Step4……..

L4=1010 0100 0001 0000 1000 1010 0111 0101

R4=L3+f(R3,K4)

So R4=1101 0010 1010 1110 1111 0010 0011 1101

Step5……..

L5=1101 0010 1010 1110 1111 0010 0011 1101

R5=L4+f(R4,K5)

So R5= 0111 1111 1010 0110 1110 1001 1001 0001

Step6…….

L6=0111 1111 1010 0110 1110 1001 1001 0001

R6=L5+f(R5,K6)

So R6=1011 0110 1110 1100 0100 0011 0101 1011

Step7……..

L7=1011 0110 1110 1100 0100 0011 0101 1011

R7=L6+f(R6,K7)

So R7=0100 0010 0011 0100 1001 1000 1001 1111

Step8……..

L8=0100 0010 0011 0100 1001 1000 1001 1111

R8=L7+f(R7,K8)

So R8=1110 1110 0001 1101 0111 1101 0111 0000

Step9……

L9=1110 1110 0001 1101 0111 1101 0111 0000

R9=L8+f(R8,K9)

So R9=1010 1000 0011 0111 0111 0000 1011 1000

Step10……

L10=1010 1000 0011 0111 0111 0000 1011 1000

R10=L9+f(R9,K10)

So R10=1110 1110 1100 1010 0100 0111 0101 0001

Step11……..

L11=1110 1110 1100 1010 0100 0111 0101 0001

R11=L10+f(R10,K11)

So R11=0000 1010 1011 0001 0001 0111 0010 1000

Step12……

L12=0000 1010 1011 0001 0001 0111 0010 1000

R12=L11+f(R11,K12)

So R12=0101 1010 0001 1100 0100 1000 1101 1001

Step13……

L13=0101 1010 0001 1100 0100 1000 1101 1001

R13=L12+f(R12,K13)

So R13=0110 0001 1001 1010 1100 0010 1011 0110

Step14……..

L14=0110 0001 1001 1010 1100 0010 1011 0110

R14=L13+f(R13,K14)

So R14=1010 0111 0000 1000 1111 0101 1111 1011

Step15……..

L15=1010 0111 0000 1000 1111 0101 1111 1011

R15=L14+f(R14,K15)

So R15=0001 1000 0000 0011 1101 0100 1011 1001

Step16……

L16=0001 1000 0000 0011 1101 0100 1011 1001

R16=L15+f(R15,K16)

S0 R16=0010 1101 0010 1011 1110 0011 0011 1110

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

L_{16} = 0001 1000 0000 0011 1101 0100 1011 1001
R_{16} = 0010 1101 0010 1011 1110 0011 0011 1110

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

$R_{16}L_{16}$ = 0010 1101 0010 1011 1110 0011 0011 1110 0001 1000 0000 0011 1101 0100 1011 1001

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

$\mathbf{IP^{-1}}$

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

IP^{-1} = 0111011000110101010010011101001110001011010101110000110000001110

Which in hexadecimal format is

76 35 49 d3 8b 57 0c 0e

This is the encrypted form of

Plain Text = NEVRQUIT:

Key=KASHISAB

Cipher Text = 76 35 49 d3 8b 57 0c 0e.

Decryption

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

So now we have

Cipher text=76 35 49 d3 8b 57 0c 0e

And our key is= KASHISAB

Cipher text in binary is=01110110001101010100100111010011100010110101011100001100000001110

After Ip table the cipher text is=

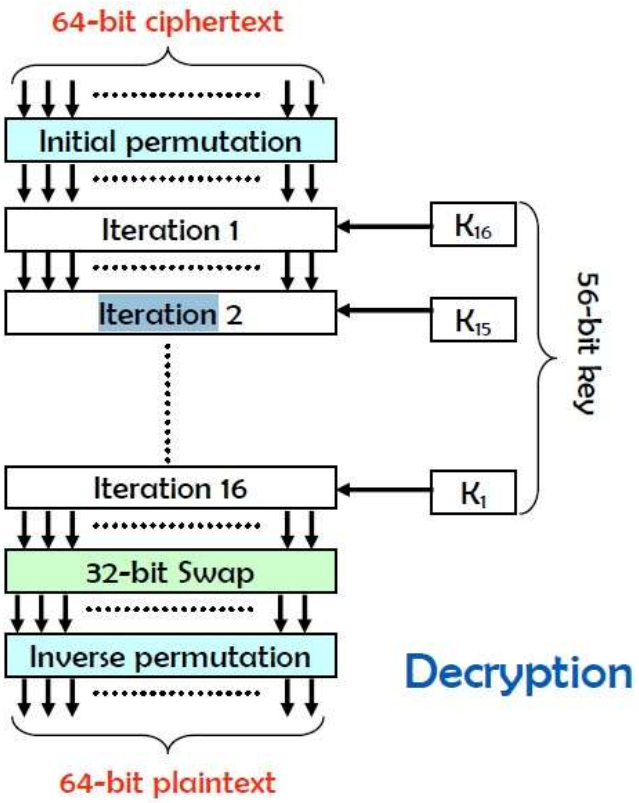
0010 1101 0010 1011 1110 0011 0011 1110 0001 1000 0000 0011 1101 0100 1011 1001

So now from above text

L16=0010 1101 0010 1011 1110 0011 0011 1110

R16=0001 1000 0000 0011 1101 0100 1011 1001

From now on the process is same but we only use our keys in reverse order.



So the simple formula we use for decryption is

$L_{n-1} = R_n$
$R_{n-1} = L_n + f(R_n, Kn)$
And here n goes from 16 to 1

Step 1…….

From the above function we can write this

L15=0001 1000 0000 0011 1101 0100 1011 1001

R15=0010 1101 0010 1011 1110 0011 0011 1110 + f(0001 1000 0000 0011 1101 0100 1011 1001, 101000 001001 001000 101010 011000 000001 010010 000110)

Now we calculate the R15……

- By first we expand R16 with 'E' table that will expand our 32 bits to 48 bits.
- Next we XOR K16 with the output of 'E' table.
- Then after XOR our output goes to the 8 S BOXES that will convert 48 bits to 32 bits.
- Then our output goes to the 'P' table.

Then we had the answer of our function.

The function answer is=10001010001000110001011011000101

Then at last when we XOR with L16 we finally get

R15= 1010 0111 0000 1000 1111 0101 1111 1011

//All the steps given below undergo with same procedure.

Step 2…….

L14=1010 0111 0000 1000 1111 0101 1111 1011

R14=0001 1000 0000 0011 1101 0100 1011 1001+f(1010 0111 0000 1000 1111 0101 1111 1011, 111000 011001 001000 100010 000101 100000 010001 001001)

The function answer is=01111001100110010001011000001111

So after XOR with L15

R14=0110 0001 1001 1010 1100 0010 1011 0110

Step 3……

L13=0110 0001 1001 1010 1100 0010 1011 0110

R13=1010 0111 0000 1000 1111 0101 1111 1011 + f(0110 0001 1001 1010 1100 0010 1011 0110

, 110000 001000 011000 100011 010101 001000 001010 000011)

So R13=0101 1010 0001 1100 0100 1000 1101 1001

Step 4……..

L12=0101 1010 0001 1100 0100 1000 1101 1001

R12=0110 0001 1001 1010 1100 0010 1011 0110 + f(0101 1010 0001 1100 0100 1000 1101 1001, 110000 001010 010100 100100 101000 000000 001011 000110)

So R12=0000 1010 1011 0001 0001 0111 0010 1000

Step 5……

L11=0000 1010 1011 0001 0001 0111 0010 1000

R11=0101 1010 0001 1100 0100 1000 1101 1001 + f(0000 1010 1011 0001 0001 0111 0010 1000, 010000 000110 110000 100100 101001 000010 000100 000100)

So R11= 1110 1110 1100 1010 0100 0111 0101 0001

Step 6……..

L10=1110 1110 1100 1010 0100 0111 0101 0001

R10=0000 1010 1011 0001 0001 0111 0010 1000 + f(1110 1110 1100 1010 0100 0111 0101 0001, 000100 000010 110001 000100 000000 110110 000000 000010)

So R10=1010 1000 0011 0111 0111 0000 1011 1000

Step 7……

L9=1010 1000 0011 0111 0111 0000 1011 1000

R9=1110 1110 1100 1010 0100 0111 0101 0001 + f(1010 1000 0011 0111 0111 0000 1011 1000, 000100 000011 100010 001100 001110 010100 000000 010001)

So R9= 1110 1110 0001 1101 0111 1101 0111 0000

Step 8……

L8=1110 1110 0001 1101 0111 1101 0111 0000

R8=1010 1000 0011 0111 0111 0000 1011 1000 + f(1110 1110 0001 1101 0111 1101 0111 0000, 000110 010000 101010 001000 000110 000010 111010 010000)

So R8=0100 0010 0011 0100 1001 1000 1001 1111

Step 9……

L7=0100 0010 0011 0100 1001 1000 1001 1111

R7=1110 1110 0001 1101 0111 1101 0111 0000 + f(0100 0010 0011 0100 1001 1000 1001 1111, 000110 010000 100010 001011 000000 010101 000010 001000)

So R7= 1011 0110 1110 1100 0100 0011 0101 1011

Step 10……

L6=1011 0110 1110 1100 0100 0011 0101 1011

R6=0100 0010 0011 0100 1001 1000 1001 1111 + f(1011 0110 1110 1100 0100 0011 0101 1011, 000010 111000 000110 001001 010110 010100 000000 011100)

So R6=0111 1111 1010 0110 1110 1001 1001 0001

Step 11……..

L5=0111 1111 1010 0110 1110 1001 1001 0001

R5=1011 0110 1110 1100 0100 0011 0101 1011 + f(0111 1111 1010 0110 1110 1001 1001 0001, 010011 110100 000100 001001 010010 000000 111000 010000)

So R5=1101 0010 1010 1110 1111 0010 0011 1101

Step 12……

L4=1101 0010 1010 1110 1111 0010 0011 1101

R4=0111 1111 1010 0110 1110 1001 1001 0001 + f(1101 0010 1010 1110 1111 0010 0011 1101, 000011 100100 010101 010001 100000 011000 110100 100000)

So R4= 1010 0100 0001 0000 1000 1010 0111 0101

Step 13……..

L3=1010 0100 0001 0000 1000 1010 0111 0101

R3=1101 0010 1010 1110 1111 0010 0011 1101 + f(1010 0100 0001 0000 1000 1010 0111 0101, 000001 100111 000101 010000 000000 101001 000101 110000)

So R3=0010 1100 1010 1000 1111 0001 0101 1110

Step 14……..

L2=0010 1100 1010 1000 1111 0001 0101 1110

R2=1010 0100 0001 0000 1000 1010 0111 0101 + f(0010 1100 1010 1000 1111 0001 0101 1110, 001001 000101 101001 010000 000001 100101 100001 001001)

So R2=0010 1100 1011 0101 1101 0001 1010 0011

Step 15……

L1=0010 1100 1011 0101 1101 0001 1010 0011

R1=0010 1100 1010 1000 1111 0001 0101 1110 + f(0010 1100 1011 0101 1101 0001 1010 0011, 101000 000001 001001 010010 010011 000000 000010 101011)

So R1=0000 0000 0000 0000 0100 0001 0000 1101

Step 16……

L0=0000 0000 0000 0000 0100 0001 0000 1101

R0=0010 1100 1011 0101 1101 0001 1010 0011 + f(0000 0000 0000 0000 0100 0001 0000 1101, 101000 001001 001011 000010 000010 101011 000101 000000)

So R0=1111 1111 1011 1100 1010 0111 0111 0010

So

L0=0000 0000 0000 0000 0100 0001 0000 1101

R0=1111 1111 1011 1100 1010 0111 0111 0010

After 32 bit swap

R0L0=1111 1111 1011 1100 1010 0111 0111 0010 0000 0000 0000 0000 0100 0001 0000 1101

So after 1p inverse

Ip Inv=0100 1110 0100 0101 0101 0110 0101 0010 0101 0001 0101 0101 0100 1001 0101 0100

In Binary

L=0100 1110 0100 0101 0101 0110 0101 0010

R=0101 0001 0101 0101 0100 1001 0101 0100

In Hex

L=4e 45 56 52

R=51 55 49 54

And in Text form

NEVRQUIT.

That's our plain text.

Done ☺