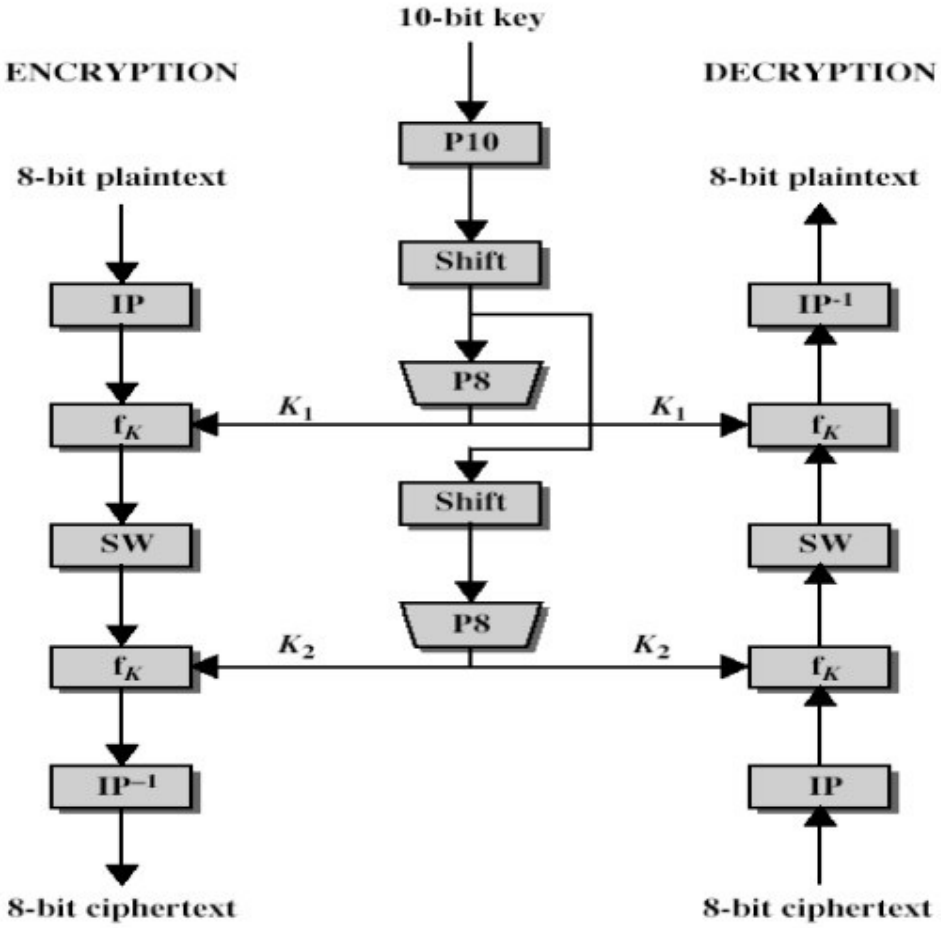
Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

NAME: DIVYESH KHUNT SAPID:60009210116 BATCH:D12

SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES)

The overall structure of the simplified DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.



The encryption algorithm involves five functions:

an initial permutation (IP)

a complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input

a simple permutation function that switches (SW) the two halves of the data

the function f_k again

a permutation function that is the inverse of the initial permutation

The function f_k takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition1 of functions: IP^{-1} o f_{K2} o SW o f_{k1} o IP

Which can also be written as

$$\text{Ciphertext} = IP^{-1}(f_{K2}(SW(f_{k1}(IP(\text{plaintext})))))$$

Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as

Plaintext = IP^{-1} (f_{K1} (SW (f_{k2} (IP (ciphertext)))))

1. S-DES key generation

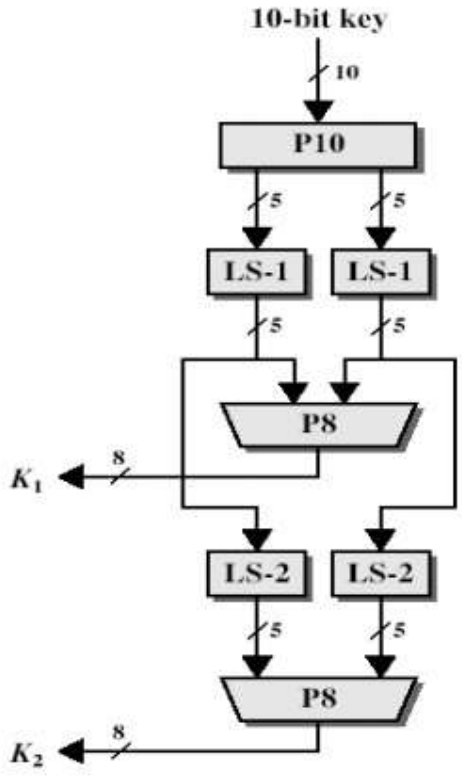


Figure: key generation for S-DES

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10 (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, K2, k7, k4, k10 10, k1, k9, k8, k6) P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

2 S-DES encryption

Encryption involves the sequential application of five functions.

Initial and Final Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up.

Consider the plaintext to be 11110011.

Permuted output = 10111101

At the end of the algorithm, the inverse permutation is used:

IP –1							
4	1	3	5	7	2	8	6

The Function f_k

The most complex component of S-DES is the function f_k, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f K, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

Where SK is a subkey and $\oplus$ is the bit-by-bit exclusive-OR function.

e.g., permuted output = 1011 1101 and suppose F (1101, SK) = (1110) for some key SK.

Then f K(10111101) = 1011$\oplus$1110, 1101

= 01011101

We now describe the mapping F. The input is a 4-bit number (n1 n2 n3 n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

e.g., R= 1101

E/P output = 11101011

It is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey K1 = (k11, k12 12, k13 13, k14 14, k15 15, k16 16, k17 17, k18) is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits: $\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if (p0,0 p0,3) =) (00) and (p0,1 p0,2) = (10), then the output is from row 0, column 2 of S0, which is 3, or (11) in) binary. Similarly, (p1,0 p1,3) and (p1,1 p1,2) are used to index into a row and column of S1 to produce an additional 2 bits. Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

3 The Switch Function

The function f K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext.

Example:
S-DES or Simplified Data Encryption Standard
The process of encrypting a plan text into an encrypted message with the use of S-DES has been divided into multi-steps which may help you to understand it as easily as possible.

Points should be remembered.

1. It is a block cipher.
2. It has 8-bits block size of plain text or cipher text.
3. It uses 10-bits key size for encryption.
4. It is a symmetric cipher.
5. It has Two Rounds.

Let's start the game!

Key Generation of S-DES or How to Generate the Key of Simplified DES
First and foremost, we need to generate a key. With the help of this key we will encrypt the message.
Now the interesting question is, how to generate the key, and where the key is to be used. Just follow the steps.

Step 1:

Just select a random key of 10-bits, which only should be shared between both parties which means sender and receiver.
As I selected below!

Select key:1010000010

Note:You can select any random number of 10-bits.

Step 2:

Put this key into P.10 Table and permute the bits.

P.10 Table:

Input	1	2	3	4	5	6	7	8	9	10
Output Should be	3	5	2	7	4	10	1	9	8	6

As I put key into P.10 Table.

Input	1	0	1	0	0	0	0	0	1	0

Output	1	0	0	0	0	0	1	1	0	0

Now the output will be:

Key: 1000001100

Step 3:

Divide the key into two halves, left half and right half;
{1 0 0 0 0} | {0 1 1 0 0}

Step 4:

Now apply the one bit Round shift on each half:
Before round shift: {10000} | {01100}
After round shift: {00001} | {11000}
The output will be:
{0 0 0 0 1} {1 1 0 0 0}

Step 5:

Now once again combine both halve of the bits, right and left. Put them into the P8 table. What you get, that will be the K1 or First key.
Combine: 0 0 0 0 1 1 1 0 0 0

Permute into 8bit table:

P8-Table

Input	1	2	3	4	5	6	7	8	9	10
Combine-bits	0	0	0	0	1	1	1	0	0	0
Output Should be	6	3	7	4	8	5	10	9		
Output bits	1	0	1	0	0	1	0	0		

See the table the 1 and 2 number of bits are removed and other are permuted, as 6 in place of one, 9 in place of 8 and so on.

The output and K1 or key One will be:

K1=1 0 1 0 0 1 0 0

Step6:

As we know S-DES has two round and for that **we also need two keys**, one key we

generate in the above steps (step 1 to step 5). Now we need to **generate a second** bit and after that we will move to encrypt the plain text or message.

It is simple to generate the second key. Simply, go in **step 4** copy both halves, each one consists of 5 bits. But be careful on the taking of bits. Select those halves which are output of first round shift, don't take the bits which are not used in the first round. In simple words, take the output of first round shift in above **step 4.**

Which are: {00001} | {11000}

Step 7:

Now just apply two round shift circulate on each half of the bits, which means to change the position of two bits of each halves.
left half: 00001
Right half: 11000

After the two rounds shift on each half out-put of each half will be.
Left half: 00100
Right half: 00011
Combine both together: As: 0 0 1 0 0 – 0 0 0 1 1

Step 8:

Now put the bits into 8-P Table, what you get, that will be your second key. Table is also given in **step 5**.

But here the combinations of bits are changed because of two left round shift from step 5. Check it in depth.
Combine bits: 0 0 1 0 0 0 0 0 1 1

P.8 Table

Input	1	2	3	4	5	6	7	8	9	10
Combine-bits	0	0	1	0	0	0	0	0	1	1
Output Should be	6	3	7	4	8	5	10	9		
Output bits	0	1	0	0	0	0	1	1		

The output of the bits are your Second key or K2:
K2: 0 1 0 0 0 0 1 1

Finally we create both keys successfully:

K1: 1 0 1 0 0 1 0 0 (see in step 5)
K2: 0 1 0 0 0 0 1 1 (see in step 8)

How To Encrypt the Plain Text into Cipher Text in S-DES After Generating Keys.

Now, let's start Encryption of plain text into cipher text.

Encryption of Plain text into Cipher text in S-DES:

Come on do it, **step by step**.

Note: the size of input text is 8 bit and output also will be 8-bit. Or the block size is 8-bit/one byte always.

Step 1:

Suppose this is our plain text in binary which is 8-bit.

Plain text: 01110010

Step 2:

Put the plain text into IP-8(initial permutation) table and permute the bits.

IP-8 table

Bits number	1	2	3	4	5	6	7	8
Bits to be permuted	0	1	1	1	0	0	1	0
Permute the bits	2	6	3	1	4	8	5	7
Permuted bits	1	0	1	0	1	0	0	1

Output is: 1 0 1 0 1 0 0 1

Step 3:

Now break the bits into two halves, each half will consist of 4 bits. The halves will be right and left.
Two Halves of the bits:

Left half {1 0 1 0} -right half {1 0 0 1}

Step 4:

Take the right 4 bits and put them into E.P (expand and per-mutate) Table.
Bits of right half: 1001

E.P Table

Right half bits	1	0	0	0				
Number	1	2	3	4	5	6	7	8
Expand bits	4	1	2	3	2	3	4	1
Output of bits	0	1	0	0	0	0	0	1

Output of right four bits will be 8 bits, after their expanding with the help of E.P table.

O-P: 0 1 0 0 0 0 0 1

Step 5: Now, just take the output and XOR it with First key Or K 1 (which we created in previous topic that is how to generate key.).

XOR (⊕) them:

O-p: 0 1 0 0 0 0 0 1
⊕
K1: 1 0 1 0 0 1 0 0
Output of XOR: **1 1 1 0 0101**

Step 6:

Once again split the output of XOR's bit into two halves and each half will consist of 4 bits.

Splitting them into two halves:

Left half :{ 1 1 1 0} right half :{ 0101}
Now put the each half into the **s-boxes**, there is only two s-boxes. **S-0 and S-1.**

S-0

Col	0	1	2	3
Rows				
0	01	00	11	10
1	11	10	01	00
2	00	10	01	11
3	11	01	11	10

S-1

Col	0	1	2	3
Rows				
0	00	01	10	11
1	10	00	01	11
2	11	00	01	00
3	10	01	00	11

Note: put the left half into S-0 box and put the right half into S-1 Box.

But how to put them in into S-Boxes?

Take any half, (but don't forget the above mentioned note..).
The most first and most last bit will be consider the row and other remaining, which are, 2 and 3, will be considered the columns.
See, here I'm taking the left half: which is 1 1 1 0.

Now I will take First and last bit which are: 1 and 0. These will be row.

And I also will take 2nd and 3rd bits which are: 11. These will be column number.
10 means=2rd **row**
11 means = 3rd **col**

See below how it will be the second row, and 3rd column. Please, remember the IP Addressing, such as 2^8 for 255.
$1^2\,0^1$ =2+0=2
$1^2\,1^1$ =2+1=3

Let's check the 2nd row and 3rd column.

For **left half** we check the in S-**0** . In which 2nd row and 3rd column.

The output is 00 for left half;

Now let's take the **right half** and find the output of the right of in **S-1 box**.
Right half: 0101
Row: 0 1= $0^2\,1^1$=0+1=1
Col: 1 0=$1^2\,0^1$=2+1=3

Which means the value will be in 1st row and 3rd column, let's in **check S-1.**
The output will be: 11 for left half.

Step 7:

Now combine these two halves together.

Left half: {00} and right half: {11}
It will be: 0 0 1 1

Step 8: Now take these 4 bits and put them in **P-4** (permutation 4) table and get the result.

P 4 Table

Numbers	1	2	3	4
Input	0	0	1	1
Output should be	2	4	3	1
Out-Put	0	1	1	0

Now the output is: 0 1 1 0

Step 9:

Now get XOR the output with left 4 bits of Initial Per-mutation. The left bits of initial per-mutation are in **step 3**, which are **1 0 1 0**.(please, in step 3).
Let them to be XOR.

0 1 1 0
⊕
1 0 1 0

Out-put will be: 1 1 0 0

Step 10:

Now get the right half of the initial permutation, which is **step 3,** and combine that with this out- put.

The out-put of XOR in **step 9:** 1 1 0 0
Right half of IP (initial permutation): 1 0 0 1
Let's combine both. 1 1 0 0 – 1 0 0 1= 1 1 0 0 1 0 0 1
Now the output is 8 bits.: **1 1 0 0 1 0 0 1**

Step 11: Now once again break the out-put into two halves, left and right;

Left: {1 1 0 0} right: {1 0 0 1}

Step 12:

Now swap both halves, which means put the left half in place of right and vice versa.

Result:
Left half: {1 0 0 1} right half: {1 1 0 0}

Step 13:

Now let's take these halves and once again start the same procedure from **step 2** or initial Permutation, **BUT** be careful on using key in this stage we use second key or K2 (not K1). And put that into IP^{-1} (IP inverse) Table. What you get will be your final cipher text.

Let me to do it in brief. You should check carefully what I did.
Ø 1 0 0 1 11 0 0

After initial permutation according to **IP-8 table (see step 2)**
Out-put will be: 0 1 0 1 1 0 1 0

Ø Now break it into two halves
Left {0 1 0 1} right {1 0 1 0}

Ø Now Take the right 4bits and put them into **EP table**, and get the result of 8 bits.
It will be: 0 1 0 1 0 1 0 1

Ø Let **XOR it with K2.**
K2: 0 1 0 0 0 0 1 1
⊕

Out-put of **EP**: 0 1 0 1 0 1 0 1
Out- Put: **0 0 0 1 0 1 1 0**

Ø Once again split the output of XOR's bit into two halves:
Left: {0 0 0 1} right: {0 1 1 0}

Ø Now put each half in S-Boxes, which are S-0 and S-1:

Note: put the left half into S-0 box and put the right half into S-1 Box.

Before that get Rows and column:

Left: 0 0 0 1

Row: 0 1= 1
Col: 0 0 = 0

Let find the row and column of left, in S-0, the value is row number one and col number Zero.

It will be. 1 1
Ø now check the right half: 0 1 1 0
Row: 0 0=0
Col: 1 1= 3

Let's find the row and column of right, in S-1, the value is row number zero and col number three.

It will be: 11
Ø Now combine both halves together.
It will be: 1 1 1 1

Ø Now take these 4 bits and put them in **P-4** (**Per-mutation** 4) table and get the result.

The out-put also will be: 1111 after permutation.

Ø Now get it XOR with left 4 bits of Initial Per-Mutation.
1 1 1 1
XOR
0 1 0 1

Out-put:1 0 1 0

Ø Now get the right half of the initial permutation and combine that with this out- put.
1 0 1 0 - 0 1 1 0

Ø Now once again break the it into two halves, left and right
Left: {1 0 1 0} right: {0 1 1 0}

Ø Now **swap both halves,** which means put the left half in place of right and vice versa.
0 1 1 0 1 0 1 0

Ø Now put it into IP^{-1} Table which is :

IP^{-1} Table

Numbers	1	2	3	4	5	6	7	8
input	0	1	1	0	1	0	1	0
Out-put to be	2	6	3	1	4	8	5	7
Out-Put	1	0	1	0	0	0	1	1

Out-Put is the cipher text. Which is: 1 0 1 0 0 0 1 1

Finally we Encrypted successfully our **plain text: 01110010** into **cipher text** which is:

1 0 1 0 0 0 1 1

Note: We have encrypted simply a single block.

```python
def blocksof4(text):
    blocks = []
    while(text!=''):
        blocks.append(text[:4])
        text = text[4:]
    return blocks

def expansionPermutation(RPT, KEY):
    blocksRPT = blocksof4(RPT)
    n = len(blocksRPT)

    expandedRPT = []
    for inx, word in enumerate(blocksRPT):
        if inx==0:
            temp = blocksRPT[-1][-1] + word + blocksRPT[1][0]
        elif inx==(n-1):
            temp = blocksRPT[-2][-1] + word + blocksRPT[0][0]
        else:
            temp = blocksRPT[inx-1][-1] + word + blocksRPT[inx+1][0]

        expandedRPT.append(temp)
    expandedRPT_str = ''.join(expandedRPT)

    XORed = ''
    for pt, k in zip(expandedRPT_str, KEY):
        XORed += str(int(pt)^int(k))

    blocksof6 = []
    while XORed!='':
        blocksof6.append(XORed[:6])
        XORed = XORed[6:]

    sbox = []
    for i in blocksof6:
        temp = i[1:5]
        sbox.append(temp)
    sbox_str = ''.join(sbox)
    return sbox_str
```

```
def encryption(PT, KEY1, KEY2):
    n = len(PT)
    n = n//2
    lpt = pt[:n]
    rpt = pt[n:]


    for i in range(2):
        print('\nLPT :', lpt)
        print('RPT :', rpt)
        if(i==0):
            sbox = expansionPermutation(rpt, KEY1)
        else:
            sbox = expansionPermutation(rpt, KEY2)
        xor = ''
        for l, k in zip(lpt, sbox):
            xor += str(int(l)^int(k))

        lpt = xor
        lpt, rpt = rpt, lpt
    return lpt + rpt

pt = '0100110010100001101101111110111000000110010111110100010001010010'
print("PLain text",pt)

key1 = '001111111010110111000011111100010110110011110100'
key2 = '110000000101001000111100000011101001001100001011'
print("key1:",key1)
print("key2:",key2)

print("\nPLAIN TEXT:", pt)
ct = encryption(pt, key1, key2)
print("\n\nCIPHERED TEXT:", ct)
```

```
PLain text 0100110010100001101101111110111000000110010111110100010001010010
key1: 001111111010110111000011111100010110110011110100
key2: 110000000101001000111100000011101001001100001011

PLAIN TEXT: 0100110010100001101101111110111000000110010111110100010001010010

LPT : 01001100101000011011011111101110
RPT : 00000110010111110100010001010010

LPT : 00000110010111110100010001010010
RPT : 00110111010011110001100000100110


CIPHERED TEXT: 00110111010011110001100000100110101100110101111001001000000010001
```

```python
def reverseExpansionPermutation(SBOX, KEY):
    blocksof6 = []
    while SBOX != '':
        blocksof6.append(SBOX[:6])
        SBOX = SBOX[6:]

    expandedRPT = ''.join(blocksof6)

    XORed = ''
    for pt, k in zip(expandedRPT, KEY):
        XORed += str(int(pt) ^ int(k))

    blocksRPT = []
    while XORed != '':
        blocksRPT.append(XORed[:4])
        XORed = XORed[4:]

    reversedRPT = []
    for inx, word in enumerate(blocksRPT):
        if inx == 0:
            temp = blocksRPT[1][-1] + word + blocksRPT[-1][0]
        elif inx == (len(blocksRPT) - 1):
            temp = blocksRPT[0][-1] + word + blocksRPT[-2][0]
        else:
            temp = blocksRPT[inx + 1][-1] + word + blocksRPT[inx - 1][0]

        reversedRPT.append(temp)
    reversedRPT_str = ''.join(reversedRPT)
    return reversedRPT_str
```

```python
def decryption(CT, KEY1, KEY2):
    n = len(CT)
    n = n // 2
    lct = CT[:n]
    rct = CT[n:]

    for i in range(2):
        print('\nLCT:', lct)
        print('RCT:', rct)
        if i == 0:
            sbox = reverseExpansionPermutation(rct, KEY2)
        else:
            sbox = reverseExpansionPermutation(rct, KEY1)

        xor = ''
        for l, k in zip(lct, sbox):
            xor += str(int(l) ^ int(k))

        lct = xor
        lct, rct = rct, lct

    return lct + rct

ct = '11100001101101111100111101111100001100101111000011011110001010000'
print("\nCIPHERED TEXT:", ct)

key1 = '00111111101011011100001111110001011011001111010O'
key2 = '11000000010100100011110000001110100100110000101l'
pt_decrypted = decryption(ct, key1, key2)
print("\n\nDECRYPTED TEXT:", pt_decrypted)
```

```
CIPHERED TEXT: 11100001101101111100111101111000011001011110000110111100010100

LCT: 11100001101101111100111101111100
RCT: 00110010111100001101111000101000

LCT: 00110010111100001101111000101000
RCT: 10011001111000101100101000001100


DECRYPTED TEXT: 10011001111000101100101000001100011001100010101011000000101011111
```