PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

"Implementation of DES Algorithm"

Submitted by

Akshay Kumar Kanive M (PES1201700399)

Sahil F (PES1201701653)

Shambu V B (PES1201701753)

Introduction to Cryptography - UE17EC341

Jan - May 2020

under the guidance of

Ms. Santha S Meena

Assistant Professor
Department of Electronics & Communications
PES University
Bengaluru -560085

Contents

Project Aim

The DES algorithm is the most popular security algorithm. It's a symmetric algorithm, which means that the same keys are used to encrypt/decrypt sensitive data. Hence, the aim of this project is to implement DES Algorithm for encryption and decryption of data

History & Theory

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NSIT).In 1973, NIST published a request for proposals for a national symmetric-key cryptosystem. A proposal from IBM, a modification of a project called Lucifer, was accepted as DES. DES was published in the Federal Register in March 1975 as a draft of the Federal Information Processing Standard (FIPS).

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher. This was a block cipher developed by the IBM cryptography researcher Horst Feistel in the early 70’s. It consists of a number of rounds where each round contains bit-shuffling, non-linear substitutions (S-boxes), and exclusive OR operations. Most symmetric encryption schemes today are based on this structure (known as a Feistel network).

Implementation

As with most encryption schemes, DES expects two inputs - the plaintext to be encrypted and the secret security key. The manner in which the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is, therefore, a symmetric, 64-bit block cipher as it uses the same key for both encryption and decryption and only operates on 64-bit blocks of data at a time (be they plaintext or ciphertext). The key size used is 56 bits, however, a 64 bit (or eight-byte)The key is actually input. The least significant bit of each byte is either used for parity (odd for DES) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight-bit of each byte the parity bit.

Once a plain-text message is received to be encrypted, it is arranged into 64-bit blocks required for input. If the number of bits in the message is not evenly divisible by 64, then the last block will be padded. DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each of the 16 rounds is identical and performs the steps of substitution and transposition. In the beginning, DES performs an initial permutation on the entire 64-bit block of data. It is then split into 2, 32-bit sub-blocks, L_i and R_i which are then passed onto the first round. At the end of the 16th round, the 32 bit Li and Ri output quantities are swapped to create what is known as the pre-output. This [R_{16} L_{16}] concatenation is permuted using a function which is the exact inverse of the initial permutation. The output of this final permutation is the 64-bit ciphertext.

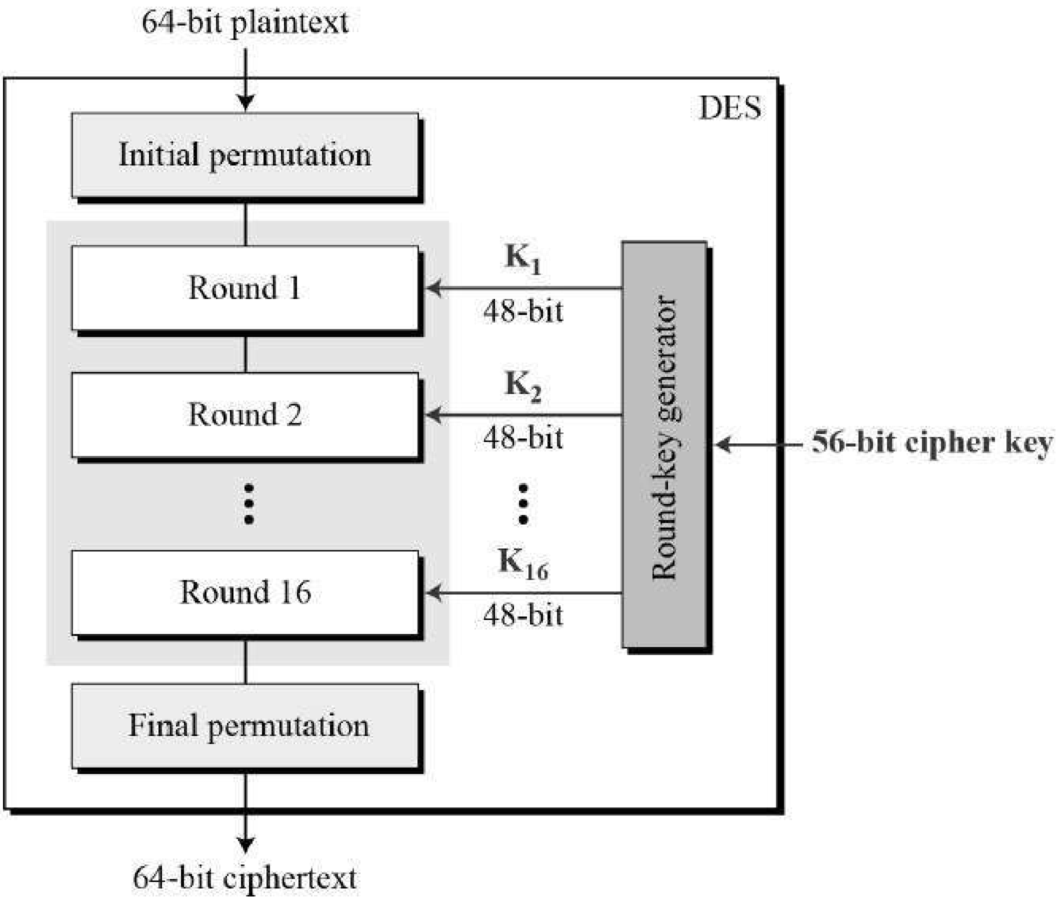


Fig: DES Cipher Flow Chart

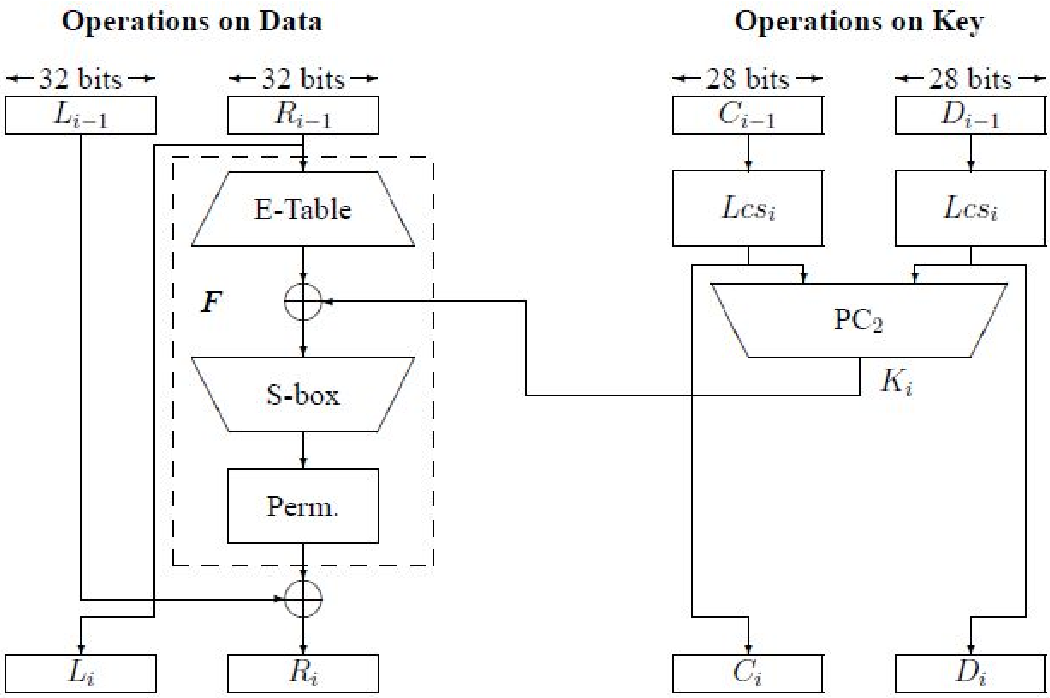
Initial and Final Permutation steps in DES transposes the bit positions of the plain text. Both Initial and Final permutation boxes are straight permutation in which the number of bits in the input and the output is the same.

Initial Permutation	Final Permutation
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

Fig: Initial & Final Permutation after the implementation of DES Algorithm

Workflow

Encryption



Details of an individual round can be seen in the above figure. The main operations on the data are encompassed into what is referred to as the cipher function and are labeled F .This function accepts two different length inputs of 32 bits and 48 bits and outputs a single 32-bit number. Both the data and key are operated on in parallel, however, the operations are quite different. The 56-bit key is split into two 28 bit halves C_i and D_i.

The value of the key used in any round is simply a left cyclic shift and a permuted contraction of that used in the previous round. Mathematically, this can be written as

$$\mathbf{C_i = Lcs_i(\ C_{i-1})\ \ and\ \ D_i = Lcs_i(\ D_{i-1})}$$

$$\mathbf{K_i = Lcs_i(C_i\ , D_i\ )}$$

where Lcs_i is the left cyclic shift for round i, C_i and D_i are the outputs after the shifts, PC2(.) is a function which permutes and compresses a 56-bit number into a 48 bit number and Ki is the actual key used in round i. The number of shifts is either one or two and is determined by the round number i. For i = {1, 2, 9, 16} the number of shifts is one and for every other round, it is two. The common formulas used to describe the relationships between the input to one round and its output (or the input to the next round) are:

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

$$\mathbf{R_i = L_i \oplus F(\ R_{i-1}, K_i)}$$

where F(.) is the cipher function. This function F is the main part of every round and consists of four separate stages:

1. The E-box expansion permutation - the 32-bit input data from R_{i-1} is expanded and permuted to give the 48 bits necessary for combination with the 48-bit key. The E-box expansion permutation delivers a larger output by splitting its input into 8, 4-bit blocks and copying every first and fourth bit in each block into the output in a defined manner.
2. The bit by bit XOR of the E-box output and bit subkey K_i.
3. The S-box substitution - which accepts a 48-bit input and outputs a 32-bit number. The S-boxes are the only non-linear operation in DES and are the most important part of its security. The input to the S-boxes is 48 bits long arranged into 8, 6-bit blocks (b_1, b_2,..,b_6). There are 8 S-boxes (S1, S2,..., S8) each of which accepts one of the 6-bit blocks. The output of each S-box is a four-bit number. Each of the S-boxes can be thought of as a 4 × 16 matrix. Each cell of the matrix is identified by a coordinate pair (i, j), where $0 \leq i \leq 3$ and $0 \leq j \leq 15$. The value of i is taken as the decimal representation of the first and last bits of the input to each S-box,i.e. Decimal(b_1b_6)=i, and the value of j is taken from the decimal representation of the inner four bits that remain, i.e. Decimal($b_2b_3b_4b_5$)= j. Each cell within the S-box matrices contains a 4-bit number. The output of the S-box is one of these numbers which will be selected by the input.
4. The P-box permutation - permutes the output of the S-box without changing the size of the data. It has a one to one mapping of its input to its output giving a 32-bit output from a 32-bit input.

Decryption

The decryption process with DES is essentially the same as the encryption process and is as follows:

- Use the ciphertext as the input to the DES algorithm but use the keys K_i
  reverse order. That is, use K_{16} on the first iteration, K_{15} on the second until K_1 which is used on the 16th and last iteration.

Python Code

Input Code

```
def conversion_text(p,var):
    #conversion of hexadecimal to binary if var=1  and vice-versa if var=0

d={'0':'0000','1':'0001','2':'0010','3':'0011','4':'0100','5':'0101','6':'0110','7
':'0111','8':'1000','9':'1001','A':'1010',
'B':'1011','C':'1100','D':'1101','E':'1110','F':'1111'}
    res=""
    if var==1:
        for i in p:
            res+=d[i]
    else:
        for i in range(0,len(p),4):
            for key,value in d.items():
                if p[i:i+4]==value:
                    res+=key
    return  res
def initialpermutaion(b):
    #initial_permutation
    mat=[58,60,62,64,57,59,61,63]
    ip_result=""
    for i in mat:
        temp=i
        while temp>0:
            ip_result+=b[temp-1]
            temp-=8
    return  ip_result

def parity_drop(b):
    #parity_drop for key(64bits to 56bits)
    res=""

mat=[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
]
    for i in mat:
        res+=b[i-1]
    return res

def shifting_operation_enc(key,rounds):
    #left circular shift of key for encryption
    res=""
    if rounds in [1,2,9,16]:
        shifts=1
    else:
        shifts=2
    res=key[shifts:]+key[0:shifts]
```

```
    return res

def shifting_operation_dec(key,rounds):
    #left circular shift of key for decryption
    res=""
    d={1:28, 2:27, 3:25, 4:23, 5:21, 6:19, 7:17, 8:15, 9:14, 10:12, 11:10, 12:8,
13:6, 14:4, 15:2, 16:1}
    shifts=d[rounds]
    res=key[shifts:]+key[0:shifts]
    return res

def key_generation(key):
    #generate key for each round(48bits)
    res=""

mat=[14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,4
7,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32]
    for i in mat:
        res+=key[i-1]
    return res

def expansion_box(p):
    #convert the right half of permuted text to 48 bits (initially it was 32 bits)
    res=""

mat=[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]
    for i in mat:
        res+=p[i-1]
    return res

def whitener(p,k):
    #to perform xor operation
    res=""
    for i in range(0,len(p)):
        res+=str(int(p[i]) ^ int(k[i]))
    return res

def sbox(p):
    #All the eight s-boxes
    res=""
    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]]

sbox_num=1
for i in range(0,48,6):
    t=p[i:i+6]
    row=int(t[0]+t[5],2)
    col=int(t[1:5],2)
    if sbox_num==1:
        match=S1[row][col]
    elif sbox_num==2:
        match=S2[row][col]
    elif sbox_num==3:
        match=S3[row][col]
    elif sbox_num==4:
        match=S4[row][col]
    elif sbox_num==5:
        match=S5[row][col]
    elif sbox_num==6:
        match=S6[row][col]
    elif sbox_num==7:
        match=S7[row][col]
    else:
        match=S8[row][col]
    sbox_num+=1
```

```
        temp1=bin(match).replace('0b','')
        temp2='0'*(4-len(temp1))+temp1
        res+=temp2
    return  res

def perm_box(p):
    #straight permutation box
    res=""

mat=[16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,2,8,24,14,32,27,3,9,19,13,30,6,2
2,11,4,25]
    for i in mat:
        res+=p[i-1]
    return res

def inverse_permutation(p):
    #final permutation box
    res=""

mat=[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,
13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,34,2,42,10,50,18,58,2
6,33,1,41,9,49,17,57,25]
    for i in mat:
        res+=p[i-1]
    return res


def algorithm_des(plain_text,org_key,enc):
    #Here if enc=1 encryption happens  and if enc=0 decryption happens

    bin_plain_text=conversion_text(plain_text,1)     #convert to bin
    bin_org_key=conversion_text(org_key,1)           #convert to bin
    ip_text=initialpermutaion(bin_plain_text)

    print("After Initial Permutation:",conversion_text(ip_text,0))


    left_half_text=ip_text[0:32]#dividing initially permuted text into two halves
    right_half_text=ip_text[32:64]

    parity_key=parity_drop(bin_org_key)#convert 64 bit to 56bit
    left_key=parity_key[0:28]
    right_key=parity_key[28:56]
    for rounds in range(1,17):
        if enc==1:
            shifted_left_key=shifting_operation_enc(left_key,rounds) #leftshift
            shifted_right_key=shifting_operation_enc(right_key,rounds) #rightshift
            left_key = shifted_left_key
            right_key = shifted_right_key
        else:
            shifted_left_key = shifting_operation_dec(left_key,rounds)  #
leftshift
```

```
            shifted_right_key = shifting_operation_dec(right_key,rounds)  #
rightshift

        round_key=key_generation(shifted_left_key+shifted_right_key) #generate key
of 48 bit length

        print("Round"+str(rounds)+" key :",conversion_text(round_key,0))

        expanded_right_plain_text=expansion_box(right_half_text) #convert
right_half of plain text to 48 bits (original 32 bts)

        xor_result=whitener(expanded_right_plain_text,round_key) #perform the xor

        mixer_result=sbox(xor_result)

        straight_perm_result=perm_box(mixer_result)

        temp=right_half_text

        right_half_text=whitener(straight_perm_result,left_half_text)

        left_half_text=temp
        if rounds==16:
            temp2=left_half_text
            left_half_text=right_half_text
            right_half_text=temp2
        print("After Round"+str(rounds)+" ",conversion_text(left_half_text,0),
conversion_text(right_half_text,0))

    final_result=inverse_permutation(left_half_text+right_half_text)
    return conversion_text(final_result,0)

plain_text='123456ABCD132536'
org_key='AABB09182736CCDD'

print("Encryption Part :")
cipher_text=algorithm_des(plain_text,org_key,1)
print("Cipher text: ",cipher_text)

print()

print("Decryption Part :")
plain_text=algorithm_des(cipher_text,org_key,0)
print("Plain text:",plain_text)
```

Output

Encryption

After Initial Permutation: 14A7D67818CA18AD
Round1 key : 194CD072DE8C

After Round1 18CA18AD 5A78E394
Round2 key : 4568581ABCCE
After Round2 5A78E394 4A1210F6
Round3 key : 06EDA4ACF5B5
After Round3 4A1210F6 B8089591
Round4 key : DA2D032B6EE3
After Round4 B8089591 236779C2
Round5 key : 69A629FEC913
After Round5 236779C2 A15A4B87
Round6 key : C1948E87475E
After Round6 A15A4B87 2E8F9C65
Round7 key : 708AD2DDB3C0
After Round7 2E8F9C65 A9FC20A3
Round8 key : 34F822F0C66D
After Round8 A9FC20A3 308BEE97
Round9 key : 84BB4473DCCC
After Round9 308BEE97 10AF9D37
Round10 key : 02765708B5BF
After Round10 10AF9D37 6CA6CB20
Round11 key : 6D5560AF7CA5
After Round11 6CA6CB20 FF3C485F
Round12 key : C2C1E96A4BF3
After Round12 FF3C485F 22A5963B
Round13 key : 99C31397C91F
After Round13 22A5963B 387CCDAA
Round14 key : 251B8BC717D0
After Round14 387CCDAA BD2DD2AB
Round15 key : 3330C5D9A36D
After Round15 BD2DD2AB CF26B472
Round16 key : 181C5D75C66D
After Round16 19BA9212 CF26B472

Cipher text: C0B7A8D05F3A829C

Decryption

After Initial Permutation: 19BA9212CF26B472
Round1 key : 181C5D75C66D
After Round1 CF26B472 BD2DD2AB
Round2 key : 3330C5D9A36D
After Round2 BD2DD2AB 387CCDAA
Round3 key : 251B8BC717D0
After Round3 387CCDAA 22A5963B
Round4 key : 99C31397C91F
After Round4 22A5963B FF3C485F
Round5 key : C2C1E96A4BF3
After Round5 FF3C485F 6CA6CB20
Round6 key : 6D5560AF7CA5
After Round6 6CA6CB20 10AF9D37
Round7 key : 02765708B5BF
After Round7 10AF9D37 308BEE97
Round8 key : 84BB4473DCCC
After Round8 308BEE97 A9FC20A3
Round9 key : 34F822F0C66D

After Round9 A9FC20A3 2E8F9C65
Round10 key : 708AD2DDB3C0
After Round10 2E8F9C65 A15A4B87
Round11 key : C1948E87475E
After Round11 A15A4B87 236779C2
Round12 key : 69A629FEC913
After Round12 236779C2 B8089591
Round13 key : DA2D032B6EE3
After Round13 B8089591 4A1210F6
Round14 key : 06EDA4ACF5B5
After Round14 4A1210F6 5A78E394
Round15 key : 4568581ABCCE
After Round15 5A78E394 18CA18AD
Round16 key : 194CD072DE8C
After Round16 14A7D678 18CA18AD

Plain text: 123456ABCD132536

Result & Conclusions

Encryption and Decryption of plain text using DES Algorithm is achieved.

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- Avalanche effect − A small change in plaintext results in the very great change in the ciphertext.
- Completeness − Each bit of ciphertext depends on many bits of plaintext.

Appendix

Expansion P-box

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Straight Permutation Box

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Parity bit drop table

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Key compression table(Permutation Choice 2)

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

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

Bibliography

[1]Data Encryption Standard (DES) Cryptography & Network Security | Cleveland State University
https://academic.csuohio.edu/yuc/security/Chapter_06_Data_Encription_Standard.pdf

[2]The DES Algorithm Illustrated by J. Orlin Grabbe

http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

[3]DES- Data Encryption Standard By Indumathi Saikumar | Volume: 04 Issue: 03 | Mar -2017 | International Research Journal of Engineering and Technology (IRJET) https://www.irjet.net/archives/V4/i3/IRJET-V4I3489.pdf

[4]The improved data encryption standard (DES) algorithm | Seung-Jo Han; Heang-Soo Oh; Jongan Park | IEEE https://ieeexplore.ieee.org/document/563518/references#reference

*****