DATA ENCRYPTION STANDARD (DES)

A TERM PROJECT

Submitted by

ARNAB DAS (CRS2109)

Under the supervision of

Prof. Sabysachi Karati, ISI Kolkata

CRYPTOLOGY AND SECURITY
INDIAN STATISTICAL INSTITUTE
203 B.T Road, Kolkata 700108

JUNE, 2022

ACKNOWLEDGEMENT

We wish to express our sincerest gratitude to Dr Sabysachi Karati for his continuous guidance and mentorship that he provided us during the project. He showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. He was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Kolkata | Arnab Das

Date: 25.06.2022 | Sutirtha Ghosh

Contents

Chapter 1

INTRODUCTION

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

1.1 History of DES

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)

After the publication, the draft was criticized severely for two reasons. First, critics questioned the small key length (only 56 bits), which could make the cipher vulnerable to brute-force attack. Second, critics were concerned about some hidden design behind the internal structure of DES. They were suspicious that some part of the structure (the S-boxes) may have some hidden trapdoor that would allow the National Security Agency (NSA) to decrypt the messages without the need for the key. Later IBM designers mentioned that the internal structure was designed to prevent differential cryptanalysis

1.1.1 overview

At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

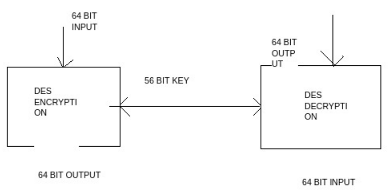

Figure 1.1: DES OVERVIEW

Chapter 2

DES ENCRYPTION AND DECRYPTION

2.0.1 DES STRUCTURE

INITIAL PERMUTATION

In the initial permutation, the 58th bit in the input becomes the first bit in the output. Similarly, in the final permutation, the first bit in the input becomes the 58th bit in the output. In other words, if the rounds between these two permutations do not exist, the 58th bit entering the initial permutation is the same as the 58th bit leaving the final permutation.

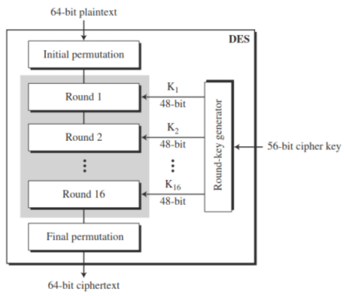

Figure 2.1: DES general structure

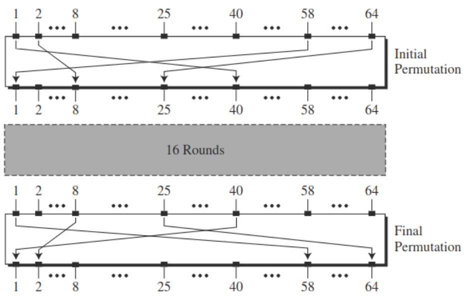

Figure 2.2: SHIFTING BITS

2.0.2 FIESTEL STRUCTURE

The round takes L_I1 and $R_(I1)$ from previous round (or the initial permutation box) and creates L_I and RI , which go to the next round (or final permutation box). As we discussed in Chapter 5, we can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All noninvertible elements are collected inside the function

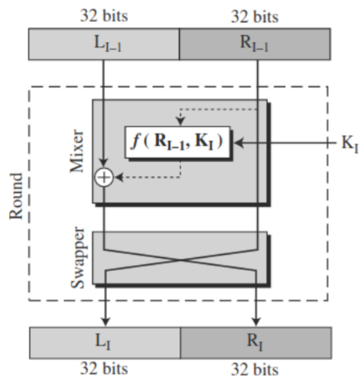

Figure 2.3: DES FIESTEL STRUCTURE

2.0.3 DES FUNCTION

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (RI1) to produce a 32-bit output. This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box as shown below.

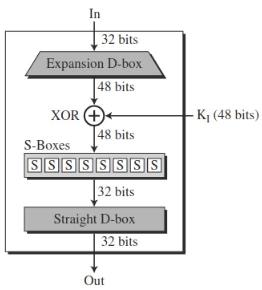

Figure 2.4: DES FIESTEL STRUCTURE

2.0.4 CIPHER AND REVERSE CIPHER

Using mixers and swappers, we can create the cipher and reverse cipher, each having 16 rounds. The cipher is used at the encryption site; the reverse cipher is used at the decryption site. The whole idea is to make the cipher and the reverse cipher algorithms similar.

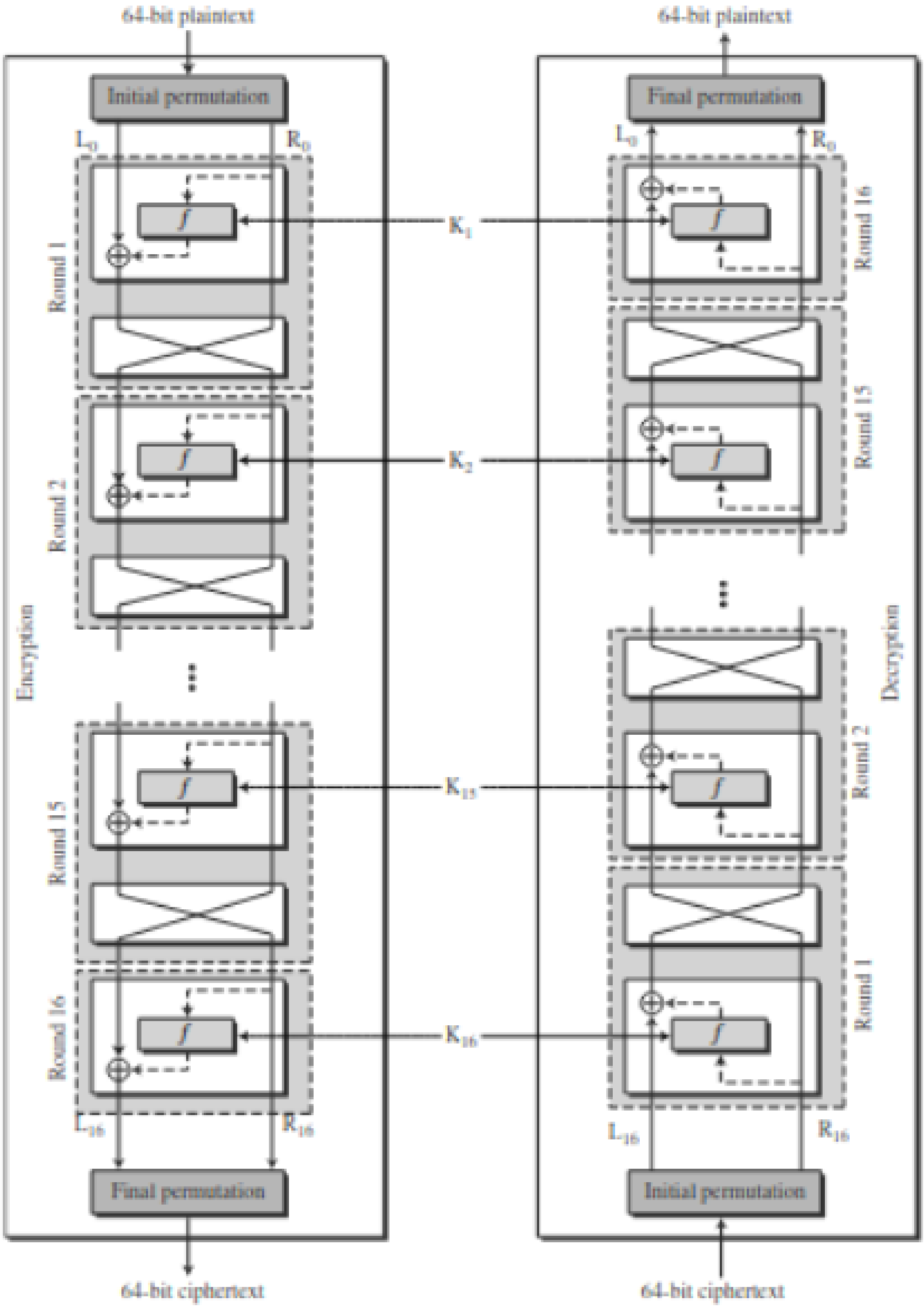

Figure 2.5: DES ENCRYPTION AND DECRYPTION

Chapter 3

DES IMPLEMENTATION(C-CODE)

3.1 DES ENCRYPTION

3.1.1 INITIAL PERMUTATION

```
int ip[8][8]=  {{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}},

```

This is the initial permutation , we have to shift 58th bit of initial message to the 0th position of our new array.

```
 for(i=0;i<8;i++)
  {
       for(j=0;j<8;j++)
       {
           getbit[i] = (a[(ip[i][j]-1)/8]>>(((ip[i][j]-1)%8)))&1;
           b[i] = b[i] | (getbit[i]<<j);
       }

  }
```

So , first we have to get the 58th bit of our message,i.e, a[].Since we have 8 bytes(64 bits), 58 th bit will be in 7th byte .

$$ip[0][0] = 58(ip[0][0] - 1)/8 = 7 \tag{3.1}$$

Exact position of the bit is (58-57)=1.

$$(ip[0][0] - 1)\%8 = 1 \tag{3.2}$$

So we will right shift that byte and applying operation 'AND' we wil get that position of bit. In our b[i] array we have stored 0 in every position and we will 'OR' with that bit after shifting j times. Thus in b[i][j] th position we have placed that bit.

3.1.2 FIESTEL STRUCTURE

DIVIDING 32 BITS

```
for(i=0;i<4;i++)
    {
       l0[i]= b[i+4];
    }
 for(i=0;i<4;i++)
    {
       r0[i]= b[i];
    }
```

I have taken from left to right, 4th bit of our array b[] is the 0th bit of l0, 5th bit is the 1st bit of l0 and 0th bit b[] is the 1st bit of r0 and so on.

EXPANSION FUNCTION

```
unsigned char ex[6][8]={{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 }};
void fiestel ( unsigned char l0[], unsigned char r0[] )
{
unsigned char l1[4], r1[4], expan[6] ={0} , c[8] , row, col,

s_value[8] , s_output[4]={0}, s[4] ,getbit[8];
int i,j;

//***** BEFORE ENCRYPTING L0 AND R0****  //

// EXPANSION FUNCTION//

for(i=0;i<6;i++)
for(j=0;j<8;j++)
 {

 getbit[i] = (r0[(ex[i][j]-1)/8]>>((ex[i][j]-1)%8))&1;
    expan[i] = expan[i] | (getbit[i]<<j);
   }
 for(i=0; i<6; i++)
             expan[i] = expan[i] ^ roundkey[round][i];

```

Here we have passed l0 and r0 in the fiestel structure and 32 bits of r0 expansioned to 48 bits using expansion matrix(similarly as initial permutation). We have to xor with 48 bits of key.

XOR WITH ROUND KEY

In this part round key will be XOR-ed with 48 bit output of key and the key scheduling part is described here.

6 BIT S_BOXINPUT

```
// APPLYING S-BOX //
 //PREPARING INPUTS FOR S-BOX//

```

```
c[0] = (expan[0] & 0x3F);
c[1] = (expan[0] >> 6);//getting last two bits and placing them to first
   two positions//
c[1] = c[1] |( (expan[1] & 0x0F) << 2);// getting first four bits and
   placing them//
c[2] = (expan[1] >> 4);//getting last four bits and placing them//
c[2] = c[2] | ((expan[2] & 0x03) << 4);//getting first two bits and
   placing them//
c[3] = (expan[2]  >> 2); //getting last six bits//

c[4] = (expan[3] & 0x3F);
c[5] = (expan[3] >> 6);//getting last two bits and placing them to first
   two positions//
c[5] = c[5] |( (expan[4] & 0x0F) << 2);// getting first four bits and
   placing them//
c[6] = (expan[4] >> 4);//getting last four bits and placing them//
c[6] = c[6] | ((expan[5] & 0x03) << 4);//getting first two bits and
   placing them//
c[7] = (expan[5]  >> 2); //getting last six bits//

```

for c[0] = (a8 a7 a6 a5 a4 a3 a2 a1) *and* (0 0 1 1 1 1 1 1 1 1) we store the first six bits in c[0]. For others c[], it is described in the code.

S_BOX-OUTPUT

```


// OUTPUT OF S-BOXES //
          for ( i=0;i<8;i++)
          {
            row = ((c[i] & 1) |( (c[i] >> 4) & 0x02));
            col = (( c[i] >> 1) & 0x0F) ;
         s_value[i] = s_box[i][row][col];
    }
```

picture(input) First and last bit of c[i] represts row of sbox and middle 4 bits represent the column of sbox.

- (c[i] 1) -gets the last bit of c[i]
- ((c[i] >> 4) 0x02)) - gets the first bit of c[i]

These represents row and

- col = ((c[i] >> 1) 0x0F) -this is the middle 4 bits

these represents column.

```
//ADDING TWO FOUR BITS TO MAKE 8 BITS (1CHAR = 8BITS)//
  for ( i=0; i<4; i++)
    s[i] = ((s_value[2*i] & 0x0F) | ((s_value[(2*i)+1] & 0x0F) << 4));

 //PERMUTING S-BOX OUTPUTS //
  int per[4][8] = {{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}};
```

```
  for(i=0;i<4;i++)
  for(j=0;j<8;j++)
   {
   getbit[i] = (s[(per[i][j]-1)/8]>>((per[i][j]-1)%8))&1;
      s_output[i] = s_output[i] | (getbit[i]<<j);
   }
  for( i=0;i< 4;i++)
    {
     r1[i] = s_output[i] ^ l0[i];
     l1[i] = r0[i];
     }
```

- s[i] = $((s_value[2*i]0x0F)|((s_value[(2*i)+1]x0F) << 4))$T;https://www.overleaf.com/project/62ba — -this is done two add 4 bytes sbox output and concatanate them to make it in a 8 byte charater.
- then apply s-box permutation to 4 bytes using per[][].
- $r1[i]$ = s output[i] $l0[i]$; The output of xbox will be XOR-ed with $l0$ and then swapped.

FIESTEL

```
//AFTER APPLYING 16 ROUND FIESTEL

printf("\n1st 32 bits l16\n");
for(i =0;i< 4;i++)
   {
     printf("%c ",l0[i]);
     }
   printf("\nlast 32 bits r16\n");
  for(i =0;i< 4;i++)
   {
     printf("%c ",r0[i]);
     }
   // *******SWAPPING L16 AND R16******///

   for( i=0; i< 4; i++)
    {
        swap[i] = l0[i];
        l0[i] = r0[i];
        r0[i] = swap[i];
    }

    //////****** APPLYING IP INVERSE ******////
    for( i = 0; i<4; i++)
    {
        b[i]= r0[i];
        b[i+4] =l0[i];
    }
    for(i=0;i<8;i++)
  {
      for(j=0;j<8;j++)
      {
          getbit[i] = (b[(ip1[i][j]-1)/8]>>(((ip1[i][j]-1)%8)))&1;
```

```
            cipher[i] = cipher[i] | (getbit[i]<<j);
        }

  }

```

- Same fiestel structure has been repeated for 16 rounds.
- We have swapped $l0$ and $r0$
- Then agai we have applied initial permutation inverse($IP^(- 1)$).

FINAL CIPHERTEXT

```
    //// ********—————————— FINAL CIPHER TEXT ——————————**********/////
printf("\n\n\n*******_____FINAL PLAINTEXT______******\n");
for(i =0;i< 8;i++)
  {
     printf("%c\t ",plaintext[i]);
     }

```

3.2 DES DECRYPTION

3.2.1 DECRPTION IN FIESTEL NETWORK

INIIAL PERMUTATION

```
     ///APPLYING IP ON CIPHER ////


for(i=0;i<8;i++)
    {
     for(j=0;j<8;j++)
    {
         getbit[i] = (cipher[(ip[i][j]-1)/8]>>(((ip[i][j]-1)%8)))&1;
         b[i] = b[i] | (getbit[i]<<j);
     }
  }

```

Here we have entered the cipher text in initial permutation.

$$l_0^d = r_16r_0^d = l_16 \tag{3.3}$$

Since we have already swapped l_16 and r_16 in our encryption , so we directly input our cipher.

```
   ///APPLYING IP ON CIPHER ////


for(i=0;i<8;i++)
    {
     for(j=0;j<8;j++)
    {
```

```
        getbit[i] = (cipher[(ip[i][j]-1)/8]>>(((ip[i][j]-1)%8)))&1;
        b[i] = b[i] | (getbit[i]<<j);
    }
  }
  //DIVIDING 32 BITS;
 for(i=0;i<4;i++)
    {
      l0[i]= b[i+4];
    }
 for(i=0;i<4;i++)
    {
      r0[i]= b[i];
    }
  // SENDING L0 AND R0 TO FIESTEL FUNCTION//


 for(i =0 ; i<16;i++)
   fiestel (l0,r0);
```

- apllied IP on cipher
- similarly as encryption we have also divided total 64 inputs into 32 bits.
- sending l_0 and r_0 to our feistel function and for 16 rounds.
- in fiestel structure key has been obtained from reverse key scheduling and this can be found here.

```

   // *******SWAPPING L16 AND R16******///

   for( i=0; i< 4; i++)
    {
        swap[i] = l0[i];
        l0[i] = r0[i];
        r0[i] = swap[i];
    }

    //////****** APPLYING IP INVERSE ******////
    for( i = 0; i<4; i++)
    {
        b[i]= r0[i];
        b[i+4] =l0[i];
    }
    for(i=0;i<8;i++)
  {
      for(j=0;j<8;j++)
      {
          getbit[i] = (b[(ip1[i][j]-1)/8]>>(((ip1[i][j]-1)%8)))&1;
          plaintext[i] = plaintext[i] | (getbit[i]<<j);
      }

  }


 //// ********------------- FINAL PLAINTEXT -------------**********/////
 printf("\n\n\n*******_____FINAL PLAINTEXT______******\n");
```

```
for(i =0;i< 8;i++)
  {
    printf("%c\t ",plaintext[i]);
    }


```

- after receiving l_16 and r_16 from fiestel we again swapped.
- applied $IP^(- 1)$ on.
- and the output is same as plaintext.

3.2.2 KEY-SCHEDULING FOR ENCRYPTION

```
int PC1[7][8] =  {
                { 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   }
                };
int PC2[6][8] =  {
                 { 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 }
                 };
int get_bit(unsigned char* arr, int i)
{
 int x = i/8;
 int y = i%8;
  return !!(arr[x] & (1<<(7 - y)));
}
void set_bit(unsigned char* arr, int i, int b)
{
 int x = i/8;
 int y = i%8;
 arr[x] = arr[x] | (b << (7 - y));
}
```

First we have used PC1 for transfering 64 bit key to 56 bit and after rotating 56 bit key to 48 bit. 'int get_bit' function x denotes poisition of byte and y be the poisiton of bit. return !!(arr[x] (1¡¡(7 - y))); This just check if that position bit is zero or non zero, if it is zero then it will zero and otherwise return 1. Similarly from set bit , we place the particular bit b in i th position.

KEY SCHEDULE FUNCTION

```
void key_schedule()
{
```

```
unsigned char tmp_key_1[7];
for(int i=0; i<7; i++) tmp_key_1[i] = 0;
unsigned char tmp_key_2[7];

// PC-1
for(int i=0; i<56; i++)
 set_bit(tmp_key_1, i, get_bit(masterkey, PC1[i/8][i%8] - 1));
 // tmp_key[i] <- key[PC_1[i] - 1]



    for(int round = 0; round < 16; round++)
    {

     int shift = 2;
     if(round == 0 || round == 1 || round == 8 || round == 15) shift = 1;

for(int i=0; i<7; i++) tmp_key_2[i] = 0;
     // rotate first half
     for(int i=0; i<28; i++)
      set_bit(tmp_key_2, i, get_bit(tmp_key_1, (i+shift) % 28));
      // tmp_key_2[i] <- tmp_key_1[ (i+shift) % 28]
```

I have declared the masterkey as global variable .

- get_bit(masterkey, PC1[i/8][i%8] - 1));
- using that , I am picking masterkey's PC1[i/8][i%8] th bit
- set_bit(tmp_key_1, i, get_bit(masterkey, PC1[i/8][i%8] - 1));
- using set bit I am storing that bit in temp_key_2's ith positon.

In the next phase I am shifting 1 or 2 bit according to the round. I am just picking (i+ shift)-th bit from temp_key_1 and using set bit function ,I am placing it to i-th bit for 1st 28 bits.

```
// rotate second half
     for(int i=28; i<56; i++)
     {
      int a = i + shift;
      if(a >= 56)
        a = 28 +( a - 56);
      set_bit(tmp_key_2, i, get_bit(tmp_key_1, a));
      // a = i + shift
      // if( a >= 56)
         a = 28 + a - 56;
      // tmp_key_2[i] <- tmp_key_1[ a ]
     }
```

For rotating second part we will get bit of the position 'a' and set it to temp_key_2 's ith position(i starts from 28). If a is greater than 56 then we have to take (a- 56)th bit after 28 th bit.temp_key_2 stores the value.

```
    // PC-2
for(int i=0; i<48; i++)
 set_bit(roundkey[round], i, get_bit(tmp_key_1, PC2[i/8][i%8] - 1));
 // roundkey[round][i] <- tmp_key[PC_2[i] - 1]
```

This is the last part of key scheduling as we will apply PC2, similarly as PC1 using getbit and setbit function and store all rond keys in a global variable.

3.2.3 KEY SCHEDULE FUNCTION FOR DECRYPTION

I have not used different function for key schedule decryption as for decryption she/he can use same function as from the reverse direction,i.e, (16-i) is used as round.

```
for(i =0 ; i<16;i++)
    fiestel (l0,r0, 16-i);
```

feistel function will take care of that as it will take key in reverse order.

Chapter 4

CIPHER FEEDBACK MODE(CFB)

4.0.1 CFB STRUCTURE

The Cipher Feedback (CFB) mode also uses a block cipher as a building block for a stream cipher. It is similar to the OFB mode but instead of feeding back the output of the block cipher, the ciphertext is fed back To generate the first key stream block s_1 , we encrypt an IV. For all subsequent key stream blocks$s_2, s_3, ...$,we encrypt the previous ciphertext. This scheme is shown in Fig. 4.1.

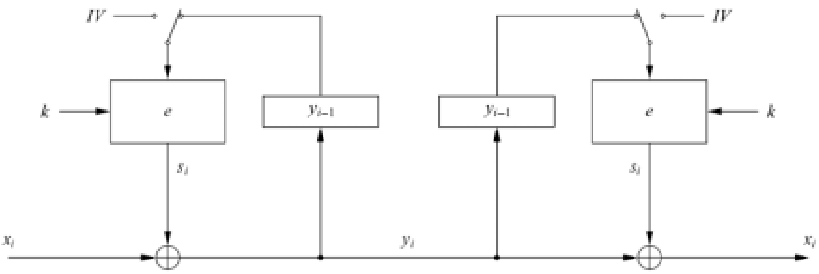

Figure 4.1: CFB general structure

4.0.2 CFB IMPLEMENTATION

```
#include<stdio.h>
#include<stdlib.h>
unsigned char masterkey[8] = "abcdefgh";
unsigned char roundkey[16][6];

int PC1[7][8] =  {
                  { 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    }
                  };
int PC2[6][8] =  {
                   { 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 }
                 };

  int   s_box[8][4][16] = {{{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}},
       {{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}},
       {{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}},
       {{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}},
       {{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}},
       {{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}},
       {{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}},
       {{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}}};


int ip[8][8]=  {{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}},


 ip1[8][8]={{ 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}};
```

```
unsigned char ex[6][8]={{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 }};



int get_bit(unsigned char* arr, int i)
{
 int x = i/8;
 int y = i%8;
 return !!(arr[x] & (1<<(7 - y)));
}
void set_bit(unsigned char* arr, int i, int b)
{
 int x = i/8;
 int y = i%8;
 arr[x] = arr[x] | (b << (7 - y));
}

void key_schedule()
{
 unsigned char tmp_key_1[7];
 for(int i=0; i<7; i++) tmp_key_1[i] = 0;
 unsigned char tmp_key_2[7];

 // PC-1
 for(int i=0; i<56; i++)
  set_bit(tmp_key_1, i, get_bit(masterkey, PC1[i/8][i%8] - 1));
  // tmp_key_1[i] <- key[PC_1[i] - 1]



    for(int round = 0; round < 16; round++)
    {

     int shift = 2;
     if(round == 0 || round == 1 || round == 8 || round == 15)
          shift = 1;

 for(int i=0; i<7; i++)
      tmp_key_2[i] = 0;
     // rotate first half
     for(int i=0; i<28; i++)
          set_bit(tmp_key_2, i, get_bit(tmp_key_1, (i+shift) % 28));
      // tmp_key_2[i] <- tmp_key_1[ (i+shift) % 28]

     // rotate second half
     for(int i=28; i<56; i++)
     {
      int a = i + shift;
      if(a >= 56) a = 28 + a - 56;
      set_bit(tmp_key_2, i, get_bit(tmp_key_1, a));
      // a = i + shift
      // if( a >= 56) a = 28 + a - 56;
      // tmp_key_2[i] <- tmp_key_1[ a ]
```

```
      }
 for(int i=0; i<7; i++) tmp_key_1[i] = tmp_key_2[i];

 // PC-2
 for(int i=0; i<48; i++)
  set_bit(roundkey[round], i, get_bit(tmp_key_1, PC2[i/8][i%8] - 1));
  // roundkey[round][i] <- tmp_key[PC_2[i] - 1]

    }
}

void fiestel ( unsigned char l0[], unsigned char r0[], int round )
{
 unsigned char l1[4], r1[4], expan[6] ={0} , c[8] , row, col, s_value[8] ,
    s_output[4]={0}, s[4] ,getbit[8];
 int i,j;

 //***** BEFORE ENCRYPTING L0 AND R0****  //

 // EXPANSION FUNCTION//

 for(i=0;i<6;i++)
  for(j=0;j<8;j++)
   {

   getbit[i] = (r0[(ex[i][j]-1)/8]>>((ex[i][j]-1)%8))&1;
          expan[i] = expan[i] | (getbit[i]<<j);
   }
        for(i=0; i<6; i++)
             expan[i] = expan[i] ^ roundkey[round][i];

 // APPLYING S-BOX //
 //PREPARING INPUTS FOR S-BOX//

 c[0] = (expan[0] & 0x3F);
 c[1] = (expan[0] >> 6);//getting last two bits and placing them to first
    two positions//
 c[1] = c[1] |( (expan[1] & 0x0F) << 2);// getting first four bits and
    placing them//
 c[2] = (expan[1] >> 4);//getting last four bits and placing them//
 c[2] = c[2] | ((expan[2] & 0x03) << 4);//getting first two bits and
    placing them//
 c[3] = (expan[2]  >> 2); //getting last six bits//

 c[4] = (expan[3] & 0x3F);
 c[5] = (expan[3] >> 6);//getting last two bits and placing them to first
    two positions//
 c[5] = c[5] |( (expan[4] & 0x0F) << 2);// getting first four bits and
    placing them//
 c[6] = (expan[4] >> 4);//getting last four bits and placing them//
 c[6] = c[6] | ((expan[5] & 0x03) << 4);//getting first two bits and
    placing them//
 c[7] = (expan[5]  >> 2); //getting last six bits//

 //PREPARING ROWS AND COLUMNS OF S-BOX//


          // OUTPUT OF S-BOXES //
```

```
        for ( i=0;i<8;i++)
        {
          row = ((c[i] & 1) |( (c[i] >> 4) & 0x02));
          col = (( c[i] >> 1) & 0x0F) ;
        s_value[i] = s_box[i][row][col];
    }

 //ADDING TWO FOUR BITS TO MAKE 8 BITS (1CHAR = 8BITS)//
  for ( i=0; i<4; i++)
    s[i] = ((s_value[2*i] & 0x0F) | ((s_value[(2*i)+1] & 0x0F) << 4));

 //PERMUTING S-BOX OUTPUTS //
  int per[4][8] = {{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}};

  for(i=0;i<4;i++)
  for(j=0;j<8;j++)
   {
   getbit[i] = (s[(per[i][j]-1)/8]>>((per[i][j]-1)%8))&1;
      s_output[i] = s_output[i] | (getbit[i]<<j);
   }
  for( i=0;i< 4;i++)
    {
     r1[i] = s_output[i] ^ l0[i];
     l1[i] = r0[i];
     }

 //***ATER 1ST ROUND ENCRYPTION , L1 AND R1***   //
 for(i =0;i< 4;i++)
   {
     l0[i]=l1[i];
     r0[i] = r1[i];
     }

}
```

This is DES part which is used as a block cipher mode encryption and this is already described in DES IMPLEMENTATION

Reading Input From file(main function)

```
void main()
{
 unsigned char iv[8]={'b','i','s','w','a','j','i','t'};
 int x,y,j,i, z = 0;
 FILE* fp = fopen("mess_text.txt", "r");
 fseek(fp, 0L, SEEK_END);
 int size = ftell(fp);
 int padded_size = size/8;
 if(size % 8) padded_size++;
 padded_size = padded_size * 8;
 rewind(fp);
```

This portion has beed done to take input from a file ,named "mess_text.txt" and it is allowed to read from starting to the end of the file. The I have done padding.

```
unsigned char* plaintext = malloc(padded_size * sizeof(unsigned char));
 unsigned char* ciphertext = malloc(padded_size * sizeof(unsigned char));
 unsigned char* outtext = malloc(padded_size * sizeof(unsigned char));

 for(i=0; i<size; i++)
  plaintext[i] = fgetc(fp);
 fclose(fp);
```

I have created spaces for plaintext and cipher text and store after reading from file into plaintext.

```
int p = padded_size - size;
        while(p--)
                plaintext[i++] = 0;

        printf("From file:\n");
        for(int i=0; i<padded_size; i++)
                printf("%c ", plaintext[i]);
        printf("\n\n");
```

I have printed 0 in padded bit.

```
int cfb_dec(unsigned char* ciphertext, int len, unsigned char* outtext,
    unsigned char* iv)
{
 unsigned char block_in[8];
 unsigned char *block_out;
 int cipher_idx = 0;

 for(int i=0; i<8; i++) block_in[i] = iv[i];

 for(int i=0; i<len/8; i++)
 {
  block_out = encryption(block_in);
  unsigned char c[8];
  int idx = 0;
  for(int j=i*8; j<i*8 + 8; j++)
  {
   c[idx] = ciphertext[j] ^ block_out[idx];
   idx++;
  }
  for(int j=0; j<8; j++)
   outtext[cipher_idx++] = c[j];

  for(int j=0; j<8; j++)
   block_in[j] = ciphertext[(i*8)+j];
 }
}
```

I have taken as input as plaintext and paded size , and IV and then applied DES on IV first and then xored with plaintext 1st block and the output is used as a feedback for next round. atlast we have output all the cipher.

Chapter 5

CONCLUSION

- DES is a old block cipher which supports key lengths of 56 bit. It does not provide any security against even brute-force attacks nowadays.
- DES is based on Feistel networks. Its basic operations use Galois field arithmetic and provide strong diffusion and confusion.
- DES was part of numerous open standards such as IPsec or TLS, in addition to being the mandatory encryption algorithm for US government applications.
- DES is efficient in software and hardware.

Bibliography

[1] Christof Paar, Jan Pelzl., "Understanding Cryptography A Textbook for Students and Practitioners", *Springer Berlin Heidelberg*

[2] *https : //en.wikipedia.org/wiki/AdvancedEncryptionStandard*