MT20062 Shivani Mishra
MT20066 Ravi Rathee

Network Security CSE 350/550
Project Report

DES BASICS

Encryption

Step 1 : **Initial Permutation**
Based on the matrix given below, our permute function will make a string out of the given plaintext.
For example :-
First value of the table suggests that value at 58th position will go to 1st position.
Second value of the table suggests that value at 50th position will go to 2st position and so on.

```
initial_perm = [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]
```

```
def permute(k, arr, n):
      permutation = ""
      for i in range(0, n):
            permutation = permutation + k[arr[i] - 1]
      return permutation
```

Step 2 : **Splitting of PlainText**
Plaintext will be splitted in 2 parts ;- LEFT PART and RIGHT PART, which will be used for each round.

```
# Splitting
      left = plaintext[0:32]
      right = plaintext[32:64]
```

Step 3 : **16 Rounds of DES**

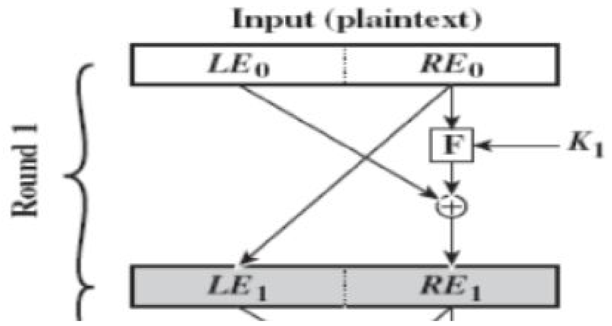
Now we will perform a set of tasks, 16 times. We call performing each set of tasks as 1 Round and in total there are 16 rounds, below we will explain what happens in each round 16 times.

Inside Each Round of DES

Now we will perform a set of tasks, 16 times. We call performing each set of tasks as 1 Round and in total there are 16 rounds, below we will explain what happens in each round 16 times.

In each round the
Left part will XOR with right part after computing inside F-BOX

LE1 = RE0 (directly no change)
RE1 = LE0 XOR RE0(resultant out of F-BOX)



Inside F-BOX
(4 steps - step a,b,c,d)

Step a) Making a 32 bit key to 48 bits using expansion P-box given below

Since we are sending 48 and expP as input we will get a 48 bits data as output.

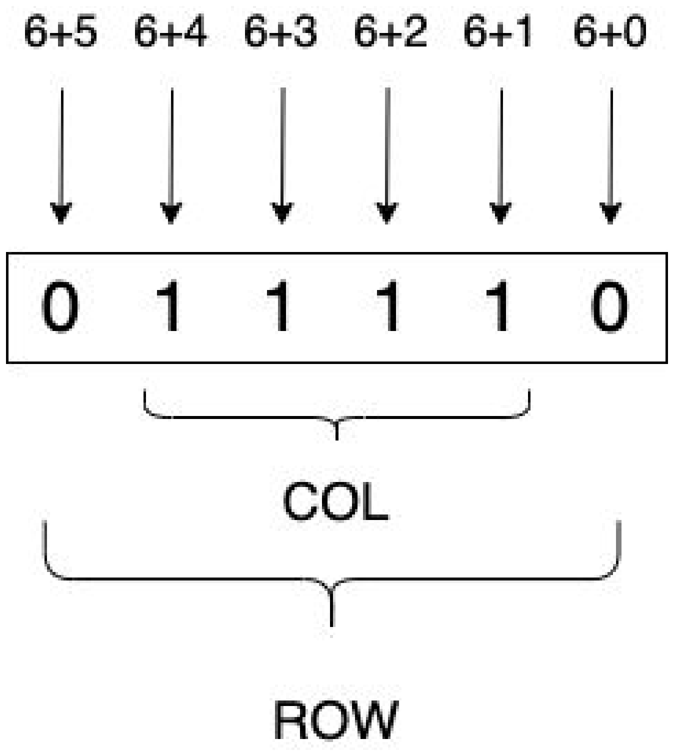
```
# Expansion P-box: Expanding the 32 bits data into 48 bits
right_expanded = permute(right, expP, 48)

expP = [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 ]
```

Step b) We will XOR RoundKey[i] and right_expanded

```
# XOR RoundKey[i] and right_expanded
xor_x = xor(right_expanded, roundkey_bin[i])
```

RoundKey[i]
For each round we will have our keys stored inside RoundKey as list of lists, we will talk about key generation at the last of this document.

Step c) S-Box Substitution
Here we are selecting row and col based on prefixes.



```
# S-boxex: substituting the value from s-box table by calculating row and
column
sbox_str = ""
for j in range(0, 8):
  row = bintodec(int(xor_x[j*6] + xor_x[j*6+5]))
  col = bintodec(int(     xor_x[j*6+1]
                        + xor_x[j*6+2]
                        + xor_x[j*6+3]
                        + xor_x[j*6+4]))

  val = sbox[j][row][col]
  sbox_str = sbox_str + dectobin(val)
```

d) Straight P-Box Substitution
We will send our output one last time in straight P-box which outputs 32 bits, its just substituting our values with new values according to the pBox straight table chosen below.

```
# Straight P-box: After substituting rearranging the bits
sbox_str = permute(sbox_str, per, 32)

# Straight Permutaion Table
pBox = [     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    ]
```

After coming out of F-Box
We will finally XOR our result like this :-
LE1 = RE0 (directly no change)
RE1 = LE0 XOR RE0(resultant out of F-BOX)

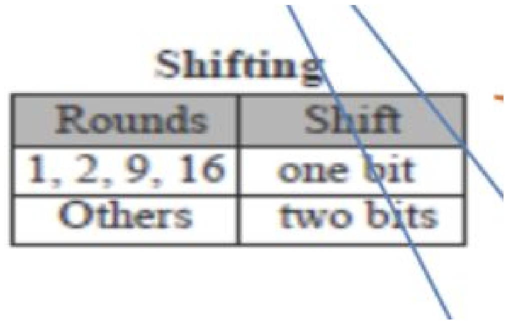
```
# XOR left and sbox_str
result = xor(left, sbox_str)
left = result
```

Decryption
For description code is written just to reverse the above steps in order to get the original output.

Key Generation

The following code is used for key generation.We will make our keys for each round and store all of them in one list, our i goes from 0 to 15 for each round.

We will divide our key in two halves equally and we will left shift each half.
We will left shift by how many bits, it depends on the round number given by the below table.



Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

```
# Number of bit shifts
shift_table     =      [1, 1, 2, 2,
                        2, 2, 2, 2,
                        1, 2, 2, 2,
                        2, 2, 2, 1 ]
```

Therefore, we will shift by one bit at round number 1,2,9,16 and for all other rounds we will shift by two bits.

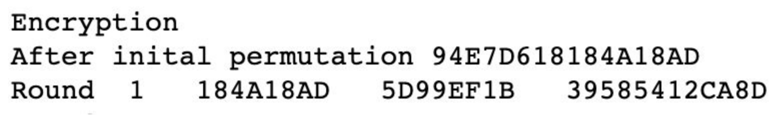
The above entire procedure is given by the below code :

```
roundkey_bin = []
roundkey_hex = []
for i in range(0, 16):
      # Shifting the bits by nth shifts by checking from shift table
      left = shift_left(left, shift_table[i])
      right = shift_left(right, shift_table[i])

      # Combination of left and right string
      combine_str = left + right

      # Compression of key from 56 to 48 bits
      roundkey = permute(combine_str, key_comp, 48)

      roundkey_bin.append(roundkey)
      roundkey_hex.append(bintohex(roundkey))
```

Conclusions

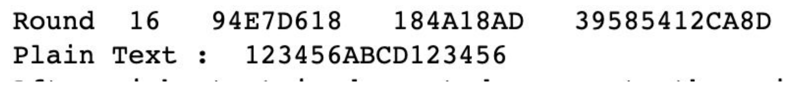
A. verify that indeed the output of the 1st encryption round is identical to the output of the 15th decryption round as illustrated below.

Output of 1st Encryption Round :



```
Encryption
After inital permutation 94E7D618184A18AD
Round  1   184A18AD   5D99EF1B   39585412CA8D
```

Output of Last Decryption Round :



```
Round  16   94E7D618   184A18AD   39585412CA8D
Plain Text :  123456ABCD123456
```

We observed that Both of them are indeed same

B. Also verify that once the ciphertext is decrypted one gets the original plaintext.

Input :

```
plaintext = "123456ABCD123456"
```

Output :

```
Plain Text :  123456ABCD123456
After ciphertext is decrypted, one gets the original plaintext.
```