Data Encryption Standard

Prashanth.S 19MID0020

In [1]:
```
def display_6(list1):
    for i in range(len(list1)):
        if (i%6==0 and i!=0):
            print("   ",end='')
        print(list1[i],end='')
```

In [2]:
```
def display_7(list1):
    for i in range(len(list1)):
        if (i%7==0 and i!=0):
            print("   ",end='')
        print(list1[i],end='')
```

In [3]:
```
def display_8(list1):
    for i in range(len(list1)):
        if (i%9==0):
            print("   ",end='')
        else:
            print(list1[i],end='')
```

In [4]:
```
def left_right_split(matrix, cnt):
    left_str  = matrix[:cnt]
    right_str = matrix[cnt:]

    return (left_str, right_str)
```

Plain Text - part

Initial Permutation --> For 64 bits plain text

In [5]:
```
def initial_permutation(elements):

    ## input  --> 64bits
    ## output --> 64bits

    ## initial_perm_matrix --> 1 to 64 bits
    ## 64bit plain text    --> 0 to 63 bits

    str_permutation_matrix = [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]

    permuted_matrix = [0 for i in range(64)]
    for i in range(0, len(str_permutation_matrix)):
        index              = str_permutation_matrix[i] - 1   ## so subtracting 1
        permuted_matrix[i] = elements[index]

    return permuted_matrix
```

Expansion Permutation --> For 32 bits-bit Right 64bit-plain text

In [6]:
```
def expansion_permutation(elements):

    ## input(right_str) --> 32 bits
    ## output           --> 48 bits

    ## initial_perm_matrix --> 1 to 64 bits
    ## 64bit key           --> 0 to 63 bits

    expansion_matrix = [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 ]

    expanded_matrix = [0 for i in range(48)]

    for i in range(0, len(expanded_matrix)):
        index              = expansion_matrix[i] - 1
        expanded_matrix[i] = elements[index]

    return expanded_matrix
```

XOR operation

In [9]:
```
def xor_operation(i,j):
    if i!=j: return 1
    else: return 0
```

In [10]:
```
def xor(expansion_permutation_matrix, key_permuted_matrix_2):

    ## input -->  48 bits
    ## output --> 48 bits

    list1 = []
    for i,j in zip(expansion_permutation_matrix,key_permuted_matrix_2):
        ans = xor_operation(i,j)
        list1.append(ans)
    return list1
```

In [11]:
```
def binaryToDecimal(binary):
    binary1 = binary
    decimal, i, n = 0, 0, 0

    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1

    return decimal
```

S-Box

In [12]:
```
def s_box(xor_output):

    ## input -->  48 bits
    ## output --> 32 bits

    sbox =  [
         [[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]]

          ]

    list1 = []

    start_index = 0
    end_index = 6

    box_index = 0
```

```
    for i in range(8): ## every 6 bits to each S-box (i.e 8 S-box's)
        temp_list = []
        s_bits = xor_output[start_index:end_index]

        row_binary = str(s_bits[0]) + str(s_bits[-1])  ## 1st and 6th bits (0th and 5th)
        row_number = binaryToDecimal(int(row_binary))  ## binary to decimal

        col_binary = s_bits[1:5]                       ## 2nd, 3rd, 4th and 5th bits
        col_binary = [str(int) for int in col_binary]  ## converting the int-list to string-list
        col_binary = ''.join(col_binary)               ## join the string-list
        col_number = binaryToDecimal(int(col_binary))  ## binary to decimal

        temp_sbox            = sbox[box_index]                      ## choosing the required box
        decimal__table_value = temp_sbox[row_number][col_number]    ## with row, column searching box
        temp_list.append("{0:b}".format(decimal__table_value).zfill(4)) ## binary to decimal

        list1 = list1 + temp_list

        start_index += 6
        end_index   += 6
        box_index   += 1

    return list1
```

Final Permutation

In [13]:
```
def final_permutation(s_box_output):

    ## input -->  32 bits
    ## output --> 32 bits

    final_perm = [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]

    permuted_matrix = [0 for i in range(32)]

    for i in range(0, len(final_perm)):
        index              = final_perm[i] - 1
        permuted_matrix[i] = s_box_output[index]

    return permuted_matrix
```

XOR operation

In [14]:
```
def xor(final_permutation_output, left_str):

    ## input -->  32 bits
    ## output --> 32 bits

    list1 = []
    for i,j in zip(final_permutation_output,left_str):
        ans = xor_operation(i,j)
        list1.append(ans)
    return list1
```

Inverse Initial Permutation

In [15]:
```
def pc1(table, key):
    parityDropped = []
    for i in table:
        parityDropped.append(key[i - 1])
    return parityDropped
```

Main Function

In [16]:
```
## input from the user

str1 = "0000000100100011010001010110011111000100110101011110011011101111"  ## 64 bits
key  = "0001001100110100010101110111100110011011101111001101111111110001"  ## 64 bits


print("Given String length : ",len(str1))
print("Given Key length    : ",len(key))

print("\nString : ",end='')
display_8(str1)

print("\nKey    : ",end='')
display_8(key)
```

```
Given String length :  64
Given Key length    :  64

String :    00000010   10001101   00101011   01111000   00110101   11110011   11110111
Key    :    00100110   11010001   10111011   10011001   01110111   00110111   11111000
```

In [17]:
```
print("############################################## STRING ##############################################")

print("\nInitial Permutation")
str_permuted_matrix = initial_permutation(str1)
print("\nLength : ",len(str_permuted_matrix))
display_8(str_permuted_matrix)
print("\n")

## Splitting into left and right string
left_str, right_str = left_right_split(str_permuted_matrix, 32)

print("\nLeft String --> ", end='')
display_8(left_str)

print("\nRight String --> ", end='')
display_8(right_str)

## expansion permutation matrix [input -> right_str(32 bits) || output -> 32bits]
expansion_permutation_matrix = expansion_permutation(right_str)
print("\n\nexpansion_permutation_matrix")
print("Length : ",len(expansion_permutation_matrix))
display_8(expansion_permutation_matrix)
```

```
############################################## STRING ##############################################

Initial Permutation

Length :  64
   10011000   00000011   01100111   11111111   00010101   10111100   01010101


Left String -->    10011000   00000011   01100111   1111
Right String -->    11100001   10101011   10000101   1010

expansion_permutation_matrix
Length :  48
   11110100   01010101   10101011   10100001   10101010   01
```

In [18]:
```
print("\n############################################ KEY ############################################")

#### Permuted Choice-1
key_permuted_matrix_1 = permuted_choice_1(key)
print("\nPermuted Choice-1 Matrix-Key")
print("Length : ",len(key_permuted_matrix_1))
display_7(key_permuted_matrix_1)
print("\n")

#### Splitting into left and right key
left_key, right_key = left_right_split(key_permuted_matrix_1,28)

print("\nLeft-Key --> ", end='')
display_7(left_key)

print("\nRight-Key --> ", end='')
display_7(right_key)
print("\n")

#### Permuted Choice-2
key_permuted_matrix_2 = permuted_choice_2(left_str + right_str)
print("\nPermuted Choice-2 Matrix-Key")
print("Length : ",len(key_permuted_matrix_2))
display_8(key_permuted_matrix_2)
print("\n")
```

```
############################################ KEY ############################################

Permuted Choice-1 Matrix-Key
Length :  56
1111000   0110011   0010101   0101111   0101010   1011001   1001111   0001111


Left-Key --> 1111000   0110011   0010101   0101111
Right-Key --> 0101010   1011001   1001111   0001111


Permuted Choice-2 Matrix-Key
Length :  48
   10011010   10000010   01001111   10101110   10010001   11
```

In [19]:
```
print("\n############################################ Joining ############################################")

## XOR (input -> expansion_permutation_matrix (48 bits) || output -> key_permuted_matrix_2 (48 bits) )
xor_output = xor(expansion_permutation_matrix, key_permuted_matrix_2)
print("\nX-OR Output")
print("Length : ",len(xor_output))
display_6(xor_output)
print("\n")

## S-BOX (input -> 48 bits || output -> 32 bits)
s_box_output = s_box(xor_output)
s_box_output = ''.join(s_box_output)
print("\nS-Box Output")
print("Length : ",len(s_box_output))
print(s_box_output)

## Final Permutation (input -> 32 bits || output -> 32 bits)
final_permutation_output = final_permutation(s_box_output)
print("\nFinal Permutation Matrix")
print("Length : ",len(final_permutation_output))
display_8(final_permutation_output)
print("\n")

## XOR Operation (input -> left_string -> (32 bits) || output -> final_permutation_output (32 bits) )
xor_output = xor(final_permutation_output, left_str)
print("\nXOR output")
print("Length : ",len(xor_output))
display_8(xor_output)
print("\n")
```

```
############################################ Joining ############################################

X-OR Output
Length :  48
001101   110111   010111   011100   100100   001111   000111   011010


S-Box Output
Length :  32
11011100111001000001010101110000

Final Permutation Matrix
Length :  32
   01000101   01100110   10101000   1110


XOR output
Length :  32
   11011101   01100101   11001111   0001
```

In [20]:
```
## Before swapping
print("\nBefore Swapping")
print("Left Text  : ")
display_8(left_str)

print("\nRight Text : ")
display_8(right_str)

## After swapping
print("\n\nAfter Swapping")
left_str  = right_str
right_str = xor_output

print("Left Text  : ")
display_8(left_str)

print("\nRight Text : ")
display_8(right_str)
```

```
Before Swapping
Left Text  :
   10011000   00000011   01100111   1111
Right Text :
   11100001   10101011   10000101   1010
```

```
After Swapping
Left Text  :
   11100001   10101011   10000101   1010
Right Text :
   11011101   01100101   11001111   0001
```

In [21]:
```
## Key-Generation for Round-1 to Round-16
```

In [22]:
```
def shift(pc1text, round):
    text = pc1text
    for _ in range(2):
        flow1 = text[0]
        flow2 = text[28]
        left = text[1:28]
        right = text[29:]
        left.append(flow1)
        right.append(flow2)
        left.extend(right)
        text = left
        if round in (1, 2, 9, 16):  ## 2
            break
    return text
```

In [23]:
```
for i in range(1, 17):
    permutatedc1 = shift(key_permuted_matrix_1, i)
    print("C{} --> ".format(i),end='')
    print(''.join(permutatedc1[:28]))

    print("D{} --> ".format(i),end='')
    print(''.join(permutatedc1[28:]))

    print("\n")
```

```
C1 --> 11100001100110010101010111111
D1 --> 10101010110011001111100011110


C2 --> 11100001100110010101010111111
D2 --> 10101010110011001111100011110


C3 --> 11000011001100101010101111111
D3 --> 01010101100110011110001111101


C4 --> 11000011001100101010101111111
D4 --> 01010101100110011110001111101


C5 --> 11000011001100101010101111111
D5 --> 01010101100110011110001111101
```

```
C6 --> 11000011001100101010101111111
D6 --> 01010101100110011111000111101


C7 --> 11000011001100101010101111111
D7 --> 01010101100110011111000111101


C8 --> 11000011001100101010101111111
D8 --> 01010101100110011111000111101


C9 --> 11100001100110010101011111
D9 --> 10101010110011001111100011110


C10 --> 11000011001100101010101111111
D10 --> 01010101100110011111000111101


C11 --> 11000011001100101010101111111
D11 --> 01010101100110011111000111101


C12 --> 11000011001100101010101111111
D12 --> 01010101100110011111000111101


C13 --> 11000011001100101010101111111
D13 --> 01010101100110011111000111101


C14 --> 11000011001100101010101111111
D14 --> 01010101100110011111000111101


C15 --> 11000011001100101010101111111
D15 --> 01010101100110011111000111101


C16 --> 11100001100110010101011111111
D16 --> 10101010110011001111100011110
```

My codes and assignment document
https://github.com/PrashanthSingaravelan/winter_semester-2022-assignments/blob/main/CSI3002%20Applied%20Cryptography%20and%20Network%20Security/lab%20assignment/assignment_2/DES%20part-2.ipynb