Lab Problem-12

In [1]:
```
# Below function is used to arrange the input keys according to the table pr
def table(input_key,tables):
    res=""
    for i in tables:
        res+=input_key[i-1]
    return res

# Below function is used for left shift
def left_shift(inp):
    return(inp[1:]+inp[0])

# Takes the input values
input_key=input("Enter the input key only in 0's and 1's:")
p10=[3,5,2,7,4,10,1,9,8,6]
p8=[6,3,7,4,8,5,10,9]

# Below part is used for dividing the key into left part and right part afte
temp=table(input_key,p10)
left_part=temp[:5]
right_part=temp[5:]
# Doing left shift
left_part=left_shift(left_part)
right_part=left_shift(right_part)

k1=table(left_part+right_part,p8)

# Doing left shift again
left_part=left_shift(left_part)
right_part=left_shift(right_part)
left_part=left_shift(left_part)
right_part=left_shift(right_part)

k2=table(left_part+right_part,p8)
print("k1 is:"+k1)
print("k2 is:"+k2)
```

```
Enter the input key only in 0's and 1's:1011000100
k1 is:10011100
k2 is:11000001
```

Lab Problem-13

In [2]:

```
#Creating table definition
def table_(inp, table):
    res = ""
    for i in table:
        res+=inp[i-1]
    return res
#Creating XOR definition
def XOR(a,b):
    ans=""
    for i in range(len(a)):
        if a[i]==b[i]:
            ans+="0"
        else:
            ans+="1"
    return ans

#Creating sbox definition
def sbox(s, data):
    row = int("0b"+data[0]+data[3],2)
    col = int("0b"+data[1:3],2)
    return bin(s[row][col])[2:]

#Creating process definition
def process(expansion,s0,s1,key,message):
    left=message[:4]
    right=message[4:]
    temp=table_(right,expansion)
    temp=XOR(temp,key)
    l=sbox(s0,temp[:4])
    r=sbox(s1,temp[4:])
    l_="0"*(2-len(l))+l
    r_="0"*(2-len(r))+r
    temp=table_(l_+r_,p4_table)
    temp=XOR(left,temp)
    return temp+right


#Input for the program
input_mes=input("Enter the plain text only in 0's and 1's:")
k1=input("Enter key k1:")
k2=input("Enter key k2:")

p4_table=[2, 4, 3, 1]
ip=[2, 6, 3, 1, 4, 8, 5, 7]
ip_inv=[4, 1 , 3, 5, 7, 2, 8, 6]
ep=[4, 1, 2, 3, 2, 3, 4, 1]
s0=[[1, 0, 3, 2],[3, 2, 1, 0],[0, 2, 1, 3],[3, 1, 3, 2]]
s1=[[0, 1, 2, 3],[2, 0, 1, 3],[3, 0, 1, 0],[2, 1, 0, 3]]

# encryption
temp=table_(input_mes,ip)
temp=process(ep,s0,s1,k1,temp)
temp=temp[4:]+temp[:4]
temp=process(ep,s0,s1,k2,temp)
Cipher_text=table_(temp,ip_inv)
print("Cipher text is:",Cipher_text)
```

```
57
58
```

```
Enter the plain text only in 0's and 1's:01101101
Enter key k1:10011100
Enter key k2:11000001
Cipher text is: 10111111
```

Lab Problem-14

In [3]:

```
#Creating table definition
def table_(inp, table):
    res = ""
    for i in table:
        res+=inp[i-1]
    return res
#Creating XOR definition
def XOR(a,b):
    ans=""
    for i in range(len(a)):
        if a[i]==b[i]:
            ans+="0"
        else:
            ans+="1"
    return ans

#Creating sbox definition
def sbox(s, data):
    row = int("0b"+data[0]+data[3],2)
    col = int("0b"+data[1:3],2)
    return bin(s[row][col])[2:]

#Creating process definition
def process(expansion,s0,s1,key,message):
    left=message[:4]
    right=message[4:]
    temp=table_(right,expansion)
    temp=XOR(temp,key)
    l=sbox(s0,temp[:4])
    r=sbox(s1,temp[4:])
    l_="0"*(2-len(l))+l
    r_="0"*(2-len(r))+r
    temp=table_(l_+r_,p4_table)
    temp=XOR(left,temp)
    return temp+right


#Input for the program
Cipher_text=input("Enter the Cipher text only in 0's and 1's:")
k1=input("Enter key k1:")
k2=input("Enter key k2:")

p4_table=[2, 4, 3, 1]
ip=[2, 6, 3, 1, 4, 8, 5, 7]
ip_inv=[4, 1 , 3, 5, 7, 2, 8, 6]
ep=[4, 1, 2, 3, 2, 3, 4, 1]
s0=[[1, 0, 3, 2],[3, 2, 1, 0],[0, 2, 1, 3],[3, 1, 3, 2]]
s1=[[0, 1, 2, 3],[2, 0, 1, 3],[3, 0, 1, 0],[2, 1, 0, 3]]

# decryption
temp=table_(Cipher_text,ip)
temp=process(ep,s0,s1,k2,temp)
temp=temp[4:] + temp[:4]
temp=process(ep,s0,s1,k1,temp)
Plain_text=table_(temp,ip_inv)
print("Plain text after decryption is:",Plain_text)
```

```
Enter the Cipher text only in 0's and 1's:10111111
Enter key k1:10011100
Enter key k2:11000001
Plain text after decryption is: 01101101
```

Lab Problem-15

In [4]:

```python
# In the below function r1==a and r2==b
def extended_eculedian_algo(r1,r2):
    a,b=r1,r2
    # Below two lines are used for declaring the values of s1,s2,t1,t2
    s1=t2=1
    s2=t1=0
    # The below while loop runs until r2!=0
    while r2!=0:
        # Finding quotient and remainder
        q=r1//r2
        r=r1%r2
        # Assigning the values of r1,r2,s1,s2,t1,t2
        # s=s1-(q*s2) and t=t1-(q*t2)
        r1,r2,s1,s2,t1,t2 = r2,r,s2,s1-(q*s2),t2,t1-(q*t2)
        # Calculation of gcd and returning the value
    final_ans=(s1*a)+(t1*b)
    return(final_ans)

# Below Two lines take the input from user
a=int(input("Enter the value of a:"))
b=int(input("Enter the value of b:"))
# Assigning the function to a variable called 'res'
res=extended_eculedian_algo(a,b)
#printing the final result
print("The GCD is:",res)
```

```
Enter the value of a:161
Enter the value of b:28
The GCD is: 7
```

Lab Problem-16

In [5]:

```
hexD={'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'a':10,'b'
roundList=[1,0,0,0]
sbox = [
    '63', '7c', '77', '7b', 'f2', '6b', '6f', 'c5', '30', '01', '67', '2b',
    'ca', '82', 'c9', '7d', 'fa', '59', '47', 'f0', 'ad', 'd4', 'a2', 'af',
    'b7', 'fd', '93', '26', '36', '3f', 'f7', 'cc', '34', 'a5', 'e5', 'f1',
    '04', 'c7', '23', 'c3', '18', '96', '05', '9a', '07', '12', '80', 'e2',
    '09', '83', '2c', '1a', '1b', '6e', '5a', 'a0', '52', '3b', 'd6', 'b3',
    '53', 'd1', '00', 'ed', '20', 'fc', 'b1', '5b', '6a', 'cb', 'be', '39',
    'd0', 'ef', 'aa', 'fb', '43', '4d', '33', '85', '45', 'f9', '02', '7f',
    '51', 'a3', '40', '8f', '92', '9d', '38', 'f5', 'bc', 'b6', 'da', '21',
    'cd', '0c', '13', 'ec', '5f', '97', '44', '17', 'c4', 'a7', '7e', '3d',
    '60', '81', '4f', 'dc', '22', '2a', '90', '88', '46', 'ee', 'b8', '14',
    'e0', '32', '3a', '0a', '49', '06', '24', '5c', 'c2', 'd3', 'ac', '62',
    'e7', 'c8', '37', '6d', '8d', 'd5', '4e', 'a9', '6c', '56', 'f4', 'ea',
    'ba', '78', '25', '2e', '1c', 'a6', 'b4', 'c6', 'e8', 'dd', '74', '1f',
    '70', '3e', 'b5', '66', '48', '03', 'f6', '0e', '61', '35', '57', 'b9',
    'e1', 'f8', '98', '11', '69', 'd9', '8e', '94', '9b', '1e', '87', 'e9',
    '8c', 'a1', '89', '0d', 'bf', 'e6', '42', '68', '41', '99', '2d', '0f',

def toHex(message): #converts message to hexadecimal
    inpmsg=[]
    for i in range(len(message)):
        inpmsg.append(hex(ord(message[i])))
    return inpmsg

def convertToCols(inpmsg): #converts hexadecimal to required columns
    #l=len(inpmsg)

    w0=list(inpmsg[:4])
    w1=list(inpmsg[4:8])
    w2=list(inpmsg[8:12])
    w3=list(inpmsg[12:16])
    return w0,w1,w2,w3

def leftShift(w3L): #performs left rotate once
# 1 is used to left rotate once (first element goes to last)
    g_1=w3L
    for j in range(1):
        temp1=g_1[0]
        for i in range(3):
            g_1[i]=g_1[i+1]
        g_1[3]=temp1
    return g_1

def byteSubs(g_1): #function to replace column values with sbox values
    g_2=[]
    for i in range(4):
        temp=0
        for j in range(2,4):
            if j==2:
                if g_1[i][j] in hexD.keys():
                    temp=hexD[g_1[i][j]]*16 #to find column in sbox and mult
            elif j==3:
                if g_1[i][j] in hexD.keys():
                    temp+=hexD[g_1[i][j]] #to find row in sbox and add it to
```

```
        g_2.append(str((sbox[temp])))
    return g_2

def roundConst(g_2): #adding round constant
    g_3=[]
    for i in range(len(g_2)): #the FOR statemnt block converts string to hex
        g_2[i]='0x'+g_2[i]
        temp=int(g_2[i],16)
        g_2[i]=temp
        #print("In roundConst function: ",g_2) #to print hexadecimal int
    for i in range(len(g_2)):
        g_3.append(hex(g_2[i] ^ roundList[i])) #performs XOR function of int
    return g_3

def nextCols(g_3,w0,w1,w2,w3):
    w4,w5,w6,w7=[],[],[],[] # initializing lists
    #converting string to hexadecimal int
    for i in range(len(g_3)):
        temp=int(g_3[i],16)
        g_3[i]=temp
    for i in range(len(w0)):
        temp=int(w0[i],16)
        w0[i]=temp
    for i in range(len(w1)):
        temp=int(w1[i],16)
        w1[i]=temp
    for i in range(len(w2)):
        temp=int(w2[i],16)
        w2[i]=temp
    for i in range(len(w3)):
        temp=int(w3[i],16)
        w3[i]=temp
    print("---",w3)

    for i in range(len(w0)): #performing the XOR functions
        w4.append(hex(w0[i] ^ g_3[i]))
    for i in range(len(w4)): #converting string to hexadecimal int
        temp=int(w4[i],16)
        w4[i]=temp
    for i in range(len(w1)): #performing the XOR functions
        w5.append(hex(w4[i] ^ w1[i]))
    for i in range(len(w5)): #converting string to hexadecimal int
        temp=int(w5[i],16)
        w5[i]=temp
    for i in range(len(w2)): #performing the XOR functions
        w6.append(hex(w5[i] ^ w2[i]))
    for i in range(len(w6)): #converting string to hexadecimal int
        temp=int(w6[i],16)
        w6[i]=temp
    print("---",w6)
    for i in range(len(w3)): #performing the XOR functions
        w7.append(hex(w6[i] ^ w3[i]))
    print("---",w7)
    for i in range(len(w7)): #converting string to hexadecimal int
        temp=int(w7[i],16)
        w7[i]=temp
    print("---",w7)
```

```
    return w4,w5,w6,w7


if __name__=='__main__':
    #inputs a message in english
    message=input("Enter a message: ")
    #call a function to convert message to hexadecimal
    inpmsg=toHex(message)
    print("Message in hexadecimal: ",inpmsg)
    #call a function to divide the hexadecimal to columns names w0 to w3
    w0,w1,w2,w3=convertToCols(inpmsg)
    print("\nColumn 0:",w0)
    print("Column 1:",w1)
    print("Column 2:",w2)
    print("Column 3:",w3)
    #call a function to perform g(w[3]) (w3=Column 4) part-1 = left shift
    w3L=list(w3) # make sures that changes made to w3L do not effect w3
    g_1=leftShift(w3L)
    print("\nAfter left shift in g(w[3]): ",g_1)
    #call a function to perform g(w[3]) (w3=Column 4) part-2 = byte substitu
    g_2=byteSubs(g_1)
    print("\nAfter byte substitution in g(w[3]): ",g_2)
    #call a function to perform g(w[3]) (w3=Column 4) part-3 = adding round
    g_3=roundConst(g_2)
    print("\nAfter adding round constant in g(w[3]): ",g_3)
    #call a function to find w4 to w7
    w4,w5,w6,w7=nextCols(g_3,w0,w1,w2,w3)
    #converting hexadecimal int to hexadecimal
    w4=[hex(i) for i in w4]
    w5=[hex(i) for i in w5]
    w6=[hex(i) for i in w6]
    w7=[hex(i) for i in w7]
    print("\nColumn 4:",w4)
    print("Column 5:",w5)
    print("Column 6:",w6)
    print("Column 7:",w7)
    print("\nFirst Round Key: ",w4+w5+w6+w7)
```

```
Enter a message: This is AES Key Expansion
Message in hexadecimal:  ['0x54', '0x68', '0x69', '0x73', '0x20', '0x69', '0x
73', '0x20', '0x41', '0x45', '0x53', '0x20', '0x4b', '0x65', '0x79', '0x20',
'0x45', '0x78', '0x70', '0x61', '0x6e', '0x73', '0x69', '0x6f', '0x6e']

Column 0: ['0x54', '0x68', '0x69', '0x73']
Column 1: ['0x20', '0x69', '0x73', '0x20']
Column 2: ['0x41', '0x45', '0x53', '0x20']
Column 3: ['0x4b', '0x65', '0x79', '0x20']

After left shift in g(w[3]):  ['0x65', '0x79', '0x20', '0x4b']

After byte substitution in g(w[3]):  ['4d', 'b6', 'b7', 'b3']

After adding round constant in g(w[3]):  ['0x4c', '0xb6', '0xb7', '0xb3']
--- [75, 101, 121, 32]
--- [121, 242, 254, 192]
--- ['0x32', '0x97', '0x87', '0xe0']
```

```
--- [50, 151, 135, 224]

Column 4: ['0x18', '0xde', '0xde', '0xc0']
Column 5: ['0x38', '0xb7', '0xad', '0xe0']
Column 6: ['0x79', '0xf2', '0xfe', '0xc0']
Column 7: ['0x32', '0x97', '0x87', '0xe0']

First Round Key:  ['0x18', '0xde', '0xde', '0xc0', '0x38', '0xb7', '0xad', '0
xe0', '0x79', '0xf2', '0xfe', '0xc0', '0x32', '0x97', '0x87', '0xe0']
```

Lab Problem-17

In [27]:

```
def keys(key):
    s = [x for x in range(256)]
    j=0
    key = [ord(x) for x in key]
    for i in range(256):
        j=(j+s[i] + int(key[i%len(key)]))%256
    return s

def encrypt(s, plaintext):
    i,j=0,0
    ciphertext = []
    for char in plaintext:
        i = (i+j)%256
        j = (j+s[i])%256
        s[i],s[j] = s[j],s[i]
        hexed = format(ord(chr(s[(s[i]+s[j])%256] ^ ord(char))), 'x')
        ciphertext.append(hexed)
    return ciphertext

key =  input("enter the key : ")
s = keys(key)
ciphertext = encrypt(s, input("enter the plaintext: "))

print("ciphertext: ")
for x in ciphertext:
    print(x, end="")
print("\n")
```

```
enter the key : SECRET
enter the plaintext: RC4 Implementation
ciphertext:
52433420496d706c656d656e746174696f6e
```

Lab Problem-18

In [28]:

```
def gcd(x,y):
    while(y!=0):
        x,y=y,x%y
    if x==1:
       return x
    else:
       print("Given e value is wrong")


def inv_mod(a,b):
    for i in range(1,b):
        if((a*i)%b==1):
            return i
    return 1

#RSA Encryption Algorithm

p=int(input("Enter the first prime number:"))
q=int(input("Enter the second prime number:"))
n=p*q
phi_of_n=(p-1)*(q-1)
e=int(input("Enter the value of e:"))
d=inv_mod(e,phi_of_n)
print("public key is:",e,n)
print("private key is:",d,n)
M=int(input("Enter the value for plain_text:"))
print("The Cipher text is:",pow(M,e)%n)
```

```
Enter the first prime number:17
Enter the second prime number:11
Enter the value of e:7
public key is: 7 187
private key is: 23 187
Enter the value for plain_text:88
The Cipher text is: 11
```