Group – 13

Code

```
binary1 = {0:'0000', 1:'0001', 2:'0010', 3:'0011', 4:'0100', 5:'0101', 6:'0110', 7:'0111', 8:'1000',
9:'1001', 10:'1010', 11:'1011', 12:'1100', 13:'1101', 14:'1110', 15:'1111'}

binary = {'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'}

shift =  {1:1, 2:1, 3:2, 4:2, 5:2, 6:2, 7:2, 8:2, 9:1, 10:2, 11:2, 12:2, 13:2, 14:2, 15:2, 16:1}

q = {'00':0, '01':1, '10':2, '11':3}

q1 ={'0000':'0', '0001':'1', '0010':'2', '0011':'3', '0100':'4', '0101':'5', '0110':'6', '0111':'7', '1000':'8',
'1001':'9', '1010':'A', '1011':'B', '1100':'C', '1101':'D', '1110':'E', '1111':'F'}

q11 ={'0000':0, '0001':1, '0010':2, '0011':3, '0100':4, '0101':5, '0110':6, '0111':7, '1000':8, '1001':9,
'1010':10, '1011':11, '1100':12, '1101':13, '1110':14, '1111':15}

comp={'0':'F', '1':'E', '2':'D', '3':'C', '4':'B', '5':'A', '6':'9', '7':'8', '8':'7', '9':'6', 'A':'5', 'B':'4', 'C':'3', 'D':'2',
'E':'1', 'F':'0'}


PC1 =
[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]

PC2 =
[14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,4
8,44,49,39,56,34,53,46,42,50,36,29,32]

IP1 =
[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]

EBIT
=[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]

P =   [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]

IPP =
[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]
```

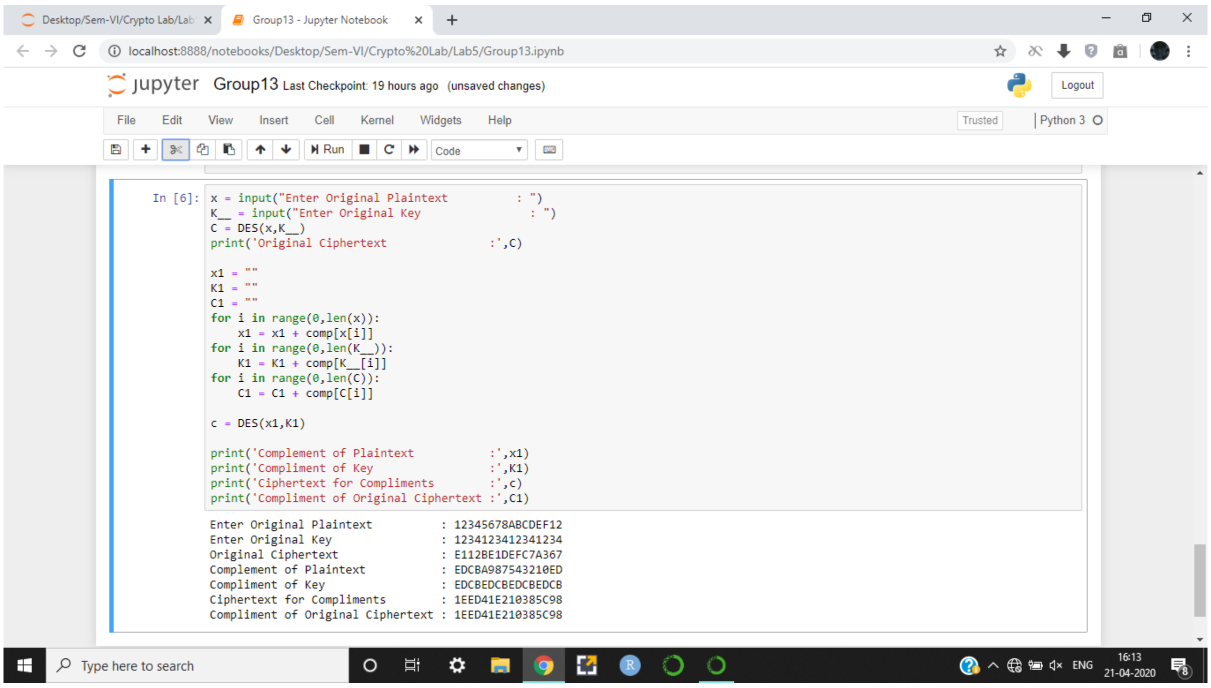
```
s1 =
[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]

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 =
[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,13,8,11,5,6,15,0,3,4,7,3,12,1,10,14,9,10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]

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]

s  = [s1] + [s2] + [s3] + [s4] + [s5] + [s6] + [s7] + [s8]


def EE(R):

   R_ = ""

   for i in range(0,48):

       R_ = R_ + str(R[EBIT[i]-1])

   return R_
```

```
def xor(a1,a2):
    empty = ""
    x = len(a1)
    for i in range(0,x):
        if(a1[i] == a2[i]):
            empty = empty + '0'
        else:
            empty = empty + '1'
    return empty

def spfunc(temp):
    B = [0] * 8
    S = [0] * 8
    S1 =[0] * 8
    SB =[0] * 8
    B[0] = temp[0:6]
    B[1] = temp[6:12]
    B[2] = temp[12:18]
    B[3] = temp[18:24]
    B[4] = temp[24:30]
    B[5] = temp[30:36]
    B[6] = temp[36:42]
    B[7] = temp[42:48]

    for i in range(0,8):
        S[i] = q[B[i][0] + B[i][len(B[i])-1]]
        S1[i]= q11[B[i][1:5]]

    for i in range(0,8):
        SB[i] = binary1[s[i][int(S[i]*16) + int(S1[i])]]
    return SB
def DES(x,K__):
    #x = input("Enter a string : ")
    L = ""
    R = ""

    y = len(x)
    for i in range(0,y//2):
        L = L + binary[x[i]]

    for i in range(y//2,y):
        R = R + binary[x[i]]

    K = ""
    for i in range(0,len(K__)):
        if(K__[i] in binary):
            K = K + binary[K__[i]]

    K_ = ""
    IP = ""
    IP_ = ""
    IP_ = L + R
    L0 = R0 = ""
    C0 = D0 = ""

    for i in range(0,len(PC1)):
        K_ = K_ + K[PC1[i]-1]
    for i in range(0,len(IP1)):
        IP = IP + IP_[IP1[i]-1]
    y = len(K_)
```

```
for i in range(0,y//2):
    C0 = C0 + K_[i]
for i in range(y//2,y):
    D0 = D0 + K_[i]
y1 = len(IP)
for i in range(0,y1//2):
    L0 = L0 + IP[i]
for i in range(y1//2,y1):
    R0 = R0 + IP[i]

C = [0] * 17
D = [0] * 17
C[0] = C0
D[0] = D0

for i in range(1,17):
    if shift[i] == 1:
        C[i] = C[i-1][1:] + C[i-1][0]
        D[i] = D[i-1][1:] + D[i-1][0]
    else:
        C[i] = C[i-1][2:] + C[i-1][0] + C[i-1][1]
        D[i] = D[i-1][2:] + D[i-1][0] + D[i-1][1]

K = [0] * 16
K_=""

for i in range(0,16):
    K[i] = C[i+1] + D[i+1]

for j in range(0,16):
    for i in range(0,len(PC2)):
        K_ = K_ + K[j][PC2[i]-1]
    K[j] = K_
    K_ = ""

L = [0] * 17
R = [0] * 17
L[0] = L0
R[0] = R0
for i in range(1,17):
    L[i] = R[i-1]
    temp = EE(R[i-1])
    temp1 = xor(temp,K[i-1])
    t1 = spfunc(temp1)
    t = ""
    t2 = ""
    for j in range(0,8):
        t2 = t2 + t1[j]
    for j in range(0,len(t2)):
        t = t + t2[P[j]-1]
    R[i] = xor(L[i-1],t)
R16L16 = R[16] + L[16]

l = ""
for i in range(0,len(R16L16)):
    l = l + R16L16[IPP[i]-1]
C = ""
for i in range(0,len(l)//4):
    C = C + q1[l[i*4:(i+1)*4]]
return C
```

```
x = input("Enter Original Plaintext          : ")

K__ = input("Enter Original Key               : ")

C = DES(x,K__)

print('Original Ciphertext               :',C)


x1 = ""

K1 = ""

C1 = ""

for i in range(0,len(x)):

    x1 = x1 + comp[x[i]]

for i in range(0,len(K__)):

    K1 = K1 + comp[K__[i]]

for i in range(0,len(C)):

    C1 = C1 + comp[C[i]]


c = DES(x1,K1)


print('Complement of Plaintext          :',x1)

print('Compliment of Key                :',K1)

print('Ciphertext for Compliments       :',c)

print('Compliment of Original Ciphertext :',C1)
```

Output



```
In [6]: x = input("Enter Original Plaintext          : ")
        K__ = input("Enter Original Key                : ")
        C = DES(x,K__)
        print('Original Ciphertext               :',C)

        x1 = ""
        K1 = ""
        C1 = ""
        for i in range(0,len(x)):
            x1 = x1 + comp[x[i]]
        for i in range(0,len(K__)):
            K1 = K1 + comp[K__[i]]
        for i in range(0,len(C)):
            C1 = C1 + comp[C[i]]

        c = DES(x1,K1)

        print('Complement of Plaintext           :',x1)
        print('Compliment of Key                 :',K1)
        print('Ciphertext for Compliments        :',c)
        print('Compliment of Original Ciphertext :',C1)
```

```
Enter Original Plaintext          : 12345678ABCDEF12
Enter Original Key                : 1234123412341234
Original Ciphertext               : E112BE1DEFC7A367
Complement of Plaintext           : EDCBA987543210ED
Compliment of Key                 : EDCBEDCBEDCBEDCB
Ciphertext for Compliments        : 1EED41E210385C98
Compliment of Original Ciphertext : 1EED41E210385C98
```