

Q1.

DES ENCRYPTION

PROGRAM CODE:

```
import numpy as np

#Initial permut matrix for the datas
PI = [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]

#Initial permut made on the key
CP_1 = [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]

#Permut applied on shifted key to get Ki+1
CP_2 = [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]

#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki
E = [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]

#SBOX
S_BOX = [

[[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],
]
]

#Permut made after each SBox substitution for each round

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]

#Final permut for datas after the 16 rounds

```
FI = [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]
```

#Matrix that determine the shift for each round of keys

```
SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]
```

def convert(char):

```
    return "{0:064b}".format(int(char, 16))
```

def xor(x,y):

```
    return (int(x,2) ^ int(y,2))
```

def sbox(text):

```
    new_text=""  
    for i in range(8):  
        x=int((text[0]+text[5]),2)  
        y=int((text[1:5]),2)  
        temp=S_BOX[i][x][y]  
        new_text+=('{0:04b}'.format(temp))  
        text=text[6:]  
    return new_text
```

def straight(text):

```
    inp=[0]*32  
    x=0  
    for i in P:  
        inp[x]=text[i-1]  
        x=x+1  
    inp="".join(inp)  
    return inp
```

def permu(text):

```
    inp=[0]*64  
    x=0  
    for i in PI:  
        inp[x]=text[i-1]  
        x=x+1  
    inp="".join(inp)  
    return inp
```

def final(text):

```
    inp=[0]*64  
    x=0  
    for i in FI:  
        inp[x]=text[i-1]
```

```

        x=x+1
    inp="".join(inp)
    return inp

```

```

def expand(text1):
    text=[0]*48
    x=0
    for i in E:
        text[x]=text1[i-1]
        x=x+1
    text="".join(text)
    return text

```

```

def drop(key):
    l_key=[0]*56
    x=0
    for i in CP_1:
        l_key[x]=key[i-1]
        x=x+1
    return l_key

```

```

def compress(key):
    r_key=[0]*48
    x=0
    for i in CP_2:
        r_key[x]=key[i-1]
        x=x+1
    r_key="".join(r_key)
    return r_key

```

```

def shift(key):
    return key[1:]+key[0]

```

```

def key_generator(key):
    s_key=convert(key)
    s_key=drop(s_key)
    left=s_key[:len(s_key)//2]
    right=s_key[len(s_key)//2:]
    left="".join(left)
    right="".join(right)
    key=[0]*16
    for i in range(16):
        for j in range(SHIFT[i]):
            left=shift(left)
            right=shift(right)
        temp=left+right
        key[i]=compress(temp)
        #temp = int(key[i], 2)
        #print(hex(temp).upper())
    return key

```

```

def funtion(right,key):

```

```

text=expand(right)
text=xor(text,key)
text=('{0:048b}'.format(text))
text=sbox(text)
text=straight(text)
return text

```

```

def swapper(left,right):
    return right,left

```

```

def des(inputt,key):
    inputt=convert(inputt)
    text=permu(inputt)
    print("Round\t Left\t\t Right\t\t Round Key\n")
    left=text[:len(text)//2]
    right=text[len(text)//2:]
    left=".join(left)
    right=".join(right)
    print("Initial\t",hex(int(left, 2)).upper(),"\t",hex(int(right, 2)).upper(),"\t (Initial
Permutation)\n")
    for i in range(16):
        out=funtion(right,key[i])
        left=xor(left,out)
        left=('{0:032b}'.format(left))
        if(i != 15):
            left,right=swapper(left,right)
        print(i+1,"\t",hex(int(left, 2)).upper(),"\t",hex(int(right, 2)).upper(),"\t",hex(int(key[i], 2)).upper())
        text=left+right
        output=final(text)
        print("\nCIPHER TEXT : ",hex(int(output, 2)).upper(),"\t (After Final Permutation)")
    return output

```

```

key="AABB09182736CCDD"
key=key_generator(key)

```

```

inputt="123456ABCD132536"
cipher=des(inputt,key)

```

OUTPUT:

```

saurav@saaurav-Lenovo-Ideapad-320-15IKB: ~/Desktop/6th Semester/Cryptography/As5$ python3 1_1.py
CIPHER TEXT : 0XC0B7A8D05F3A829C (After Final Permutation)
Round Left Right Round Key
Initial 0X14A7D678 0X18CA18AD (Initial Permutation)
1 0X18CA18AD 0X5A78E394 0X194CD072DE8C
2 0X5A78E394 0X4A1210F6 0X4568581ABCCE
3 0X4A1210F6 0XB8089591 0X6EDA4ACF5B5
4 0XB8089591 0X236779C2 0XDA2D032B6EE3
5 0X236779C2 0XA15A4B87 0X69A629FEC913
6 0XA15A4B87 0X2E8F9C65 0XC1948E87475E
7 0X2E8F9C65 0XA9FC20A3 0X708AD2DD83C0
8 0XA9FC20A3 0X308BEE97 0X34F822F0C66D
9 0X308BEE97 0X10AF9D37 0X84BB4473DCCC
10 0X10AF9D37 0X6CA6CB20 0X2765708B5BF
11 0X6CA6CB20 0XFF3C485F 0X6D5560AF7CA5
12 0XFF3C485F 0X22A5963B 0XC2C1E96A48F3
13 0X22A5963B 0X387CCDAA 0X99C31397C91F
14 0X387CCDAA 0XBD2DD2AB 0X25188BC717D0
15 0XBD2DD2AB 0XCF26B472 0X3330C5D9A36D
16 0X19BA9212 0XCF26B472 0X181C5D75C66D
CIPHER TEXT : 0XC0B7A8D05F3A829C (After Final Permutation)
saurav@saaurav-Lenovo-Ideapad-320-15IKB: ~/Desktop/6th Semester/Cryptography/As5$ python3 1_2.py
Round Left Right Round Key

```

DES DENCRIPTION

PROGRAM CODE:

```
import numpy as np

#Initial permut matrix for the datas
PI = [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]

#Initial permut made on the key
CP_1 = [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]

#Permut applied on shifted key to get Ki+1
CP_2 = [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]

#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki
E = [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]

#SBOX
S_BOX = [

[[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],
]
]

#Permut made after each SBox substitution for each round

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]

#Final permut for datas after the 16 rounds

```
FI = [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]
```

```
#Matrix that determine the shift for each round of keys
SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,1]
```

```
def convert(char):
    return "{0:064b}".format(int(char, 16))
```

```
def xor(x,y):
    return (int(x,2) ^ int(y,2))
```

```
def sbbox(text):
    new_text=""
    for i in range(8):
        x=int((text[0]+text[5]),2)
        y=int((text[1:5]),2)
        temp=S_BOX[i][x][y]
        new_text+=('{0:04b}'.format(temp))
        text=text[6:]
    return new_text
```

```
def straight(text):
    inp=[0]*32
    x=0
    for i in P:
        inp[x]=text[i-1]
        x=x+1
    inp="".join(inp)
    return inp
```

```
def permu(text):
    inp=[0]*64
    x=0
    for i in PI:
        inp[x]=text[i-1]
        x=x+1
    inp="".join(inp)
    return inp
```

```
def final(text):
    inp=[0]*64
    x=0
    for i in FI:
        inp[x]=text[i-1]
        x=x+1
```



```
inp=".join(inp)
return inp
```

```
def expand(text1):
    text=[0]*48
    x=0
    for i in E:
        text[x]=text1[i-1]
        x=x+1
    text=".join(text)
    return text
```

```
def drop(key):
    l_key=[0]*56
    x=0
    for i in CP_1:
        l_key[x]=key[i-1]
        x=x+1
    return l_key
```

```
def compress(key):
    r_key=[0]*48
    x=0
    for i in CP_2:
        r_key[x]=key[i-1]
        x=x+1
    r_key=".join(r_key)
    return r_key
```

```
def shift(key):
    return key[1:]+key[0]
```

```
def key_generator(key):
    s_key=convert(key)
    s_key=drop(s_key)
    left=s_key[:len(s_key)//2]
    right=s_key[len(s_key)//2:]
    left=".join(left)
    right=".join(right)
    key=[0]*16
    for i in range(16):
        for j in range(SHIFT[i]):
            left=shift(left)
            right=shift(right)
        temp=left+right
        key[i]=compress(temp)
        #temp = int(key[i], 2)
        #print(hex(temp).upper())
    return key
```

```
def funtion(right,key):
    text=expand(right)
```

```

text=xor(text,key)
text=('{0:048b}'.format(text))
text=sbox(text)
text=straight(text)
return text

def swapper(left,right):
    return right,left

def des(inputt,key):
    inputt=convert(inputt)
    text=permu(inputt)
    print("Round\t Left\t\t Right\t\t Round Key\n")
    left=text[:len(text)//2]
    right=text[len(text)//2:]
    left="".join(left)
    right="".join(right)
    print("Initial\t",hex(int(left, 2)).upper(),"\t",hex(int(right, 2)).upper(),"\t (Initial
Permutation)\n")
    for i in range(16):
        out=funtion(right,key[15-i])
        left=xor(left,out)
        left=('{0:032b}'.format(left))
        if(i != 15):
            left,right=swapper(left,right)
        print(i+1,"\t",hex(int(left, 2)).upper()),"\t",hex(int(right, 2)).upper(),"\
t",hex(int(key[15-i], 2)).upper())
        text=left+right
        output=final(text)
        print("\nPLAIN TEXT : ",hex(int(output, 2)).upper(),"\t (After Final Permutation)")
    return output

key="AABB09182736CCDD"
key=key_generator(key)

inputt="C0B7A8D05F3A829C"
ciper=des(inputt,key)

```

OUTPUT:

```

CIPHER TEXT : 0XC0B7A8D05F3A829C (After Final Permutation)
saurav@saurav-Lenovo-ideapad-320-15IKB:~/Desktop/6th Semester/Cryptography/Ass$ python3 1_2.py
Round Left Right Round Key
Initial 0X19BA9212 0XCF26B472 (Initial Permutation)
1 0XCF26B472 0XBD2DD2AB 0X181C5D75C66D
2 0XBD2DD2AB 0X387CCDAA 0X3330C5D9A36D
3 0X387CCDAA 0X22A5963B 0X251B8BC717D0
4 0X22A5963B 0XFF3C485F 0X99C31397C91F
5 0XFF3C485F 0X6CA6CB20 0XC2C1E96A4BF3
6 0X6CA6CB20 0X10AF9D37 0X6D5560AF7CA5
7 0X10AF9D37 0X308BEE97 0X2765708B5BF
8 0X308BEE97 0XA9FC20A3 0X84BB4473DCCC
9 0XA9FC20A3 0X2E8F9C65 0X34F822F0C66D
10 0X2E8F9C65 0XA15A4B87 0X708AD2DD3C0
11 0XA15A4B87 0X236779C2 0XC1948E87475E
12 0X236779C2 0XB8089591 0X69A629FEC913
13 0XB8089591 0X4A1210F6 0XDA2D032B6EE3
14 0X4A1210F6 0X5A78E394 0X6EDA4ACF5B5
15 0X5A78E394 0X18CA18AD 0X4568581ABCCE
16 0X14A7D678 0X18CA18AD 0X194CD072DE8C

PLAIN TEXT : 0X123456ABCD132536 (After Final Permutation)
saurav@saurav-Lenovo-ideapad-320-15IKB:~/Desktop/6th Semester/Cryptography/Ass$

```

Q2.

AES

(128 bit key and message)

PROGRAM CODE:

```
import numpy as np
s_box = [
    [0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7,
    0xAB, 0x76],
    [0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4,
    0x72, 0xC0],
    [0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31,
    0x15],
    [0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2,
    0x75],
    [0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3,
    0x2F, 0x84],
    [0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C,
    0x58, 0xCF],
    [0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F,
    0xA8],
    [0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3,
    0xD2],
    [0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D,
    0x19, 0x73],
    [0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E,
    0x0B, 0xDB],
    [0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95,
    0xE4, 0x79],
    [0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
    0xAE, 0x08],
    [0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD,
    0x8B, 0x8A],
    [0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D,
    0x9E],
    [0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28,
    0xDF],
    [0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB,
    0x16],
]
r_const=["01","02","04","08","10","20","40","80","1B","36"]
constant=['2','3','1','1','1','2','3','1','1','1','2','3','3','1','1','2']
constant=np.array(constant).reshape(4,4)

def multiply(p1,p2):
    p2=('{0:08b}'.format(p2))
    p1=('{0:08b}'.format(p1))
    mod="100011011"
```

```

result="0"
p2="0"+p2
p1= "".join(reversed(p1))
if(p1[0]=="1"):
    result=p2[1:]
for i in range(1,len(p1)):
    p2=p2[1:]+ "0"
    if(p2[0]=="1"):
        p2 = int(p2,2) ^ int(mod,2)
        p2=('{0:09b}'.format(p2))
    if(p1[i]=="1"):
        result=int(p2,2) ^ int(result,2)
        result=('{0:08b}'.format(result))
return result

```

```

def xor(x,y):
    state=[" "]*16
    state=np.array(state).reshape(4,4)
    for i in range(4):
        for j in range(4):
            temp=int(x[i][j],16) ^ int(y[i][j],16)
            state[i][j]=('{0:02x}'.format(temp)).upper()
    return state

```

```

def xor1(x,y):
    state=[" "]*4
    for i in range(4):
        temp=int(x[i],16) ^ int(y[i],16)
        state[i]=('{0:02x}'.format(temp)).upper()
    return state

```

```

def sub(text):
    for i in range(4):
        for j in range(4):
            x=int(text[i][j][0],16)
            y=int(text[i][j][1],16)
            temp=s_box[x][y]
            text[i][j]=('{0:02x}'.format(temp)).upper()
    return text

```

```

def shift(text):
    i=0
    for row in text:
        text[i]=np.roll(row,-i)
        i=i+1
    return text

```

```

def mix(text):
    result=["00"]*16
    result=np.array(result).reshape(4,4)
    for i in range(4):
        for j in range(4):

```

```

        for k in range(4):
            temp = int(result[i][j],16) ^ int(multiply(int(constant[i]
[k],16),int(text[k][j],16)),2)
            result[i][j]='{0:02x}'.format(temp)).upper()
    return result

```

```

def hex_gen(text):
    while(len(text) < 16):
        text=text+"z"
    if(len(text)>16):
        text=text[:16]
    new_text=[]
    for i in text:
        temp=(ord(i))
        temp=('{0:02x}'.format(temp)).upper()
        new_text.append(temp)
    text_array=np.array(new_text).reshape(4,4).transpose()
    return text_array

```

```

def state_gen(text,i):
    text=sub(text)
    text=shift(text)
    if(i != 9):
        text=mix(text)
    return text

```

```

def key_gen(key,index):
    key=np.array(key).transpose()
    temp=np.roll(key[3],-1)
    for i in range(4):
        x=int(temp[i][0],16)
        y=int(temp[i][1],16)
        temp1=s_box[x][y]
        temp[i]='{0:02x}'.format(temp1)).upper()
    temp2=int(temp[0],16) ^ int(r_const[index],16)
    temp[0]='{0:02x}'.format(temp2)).upper()
    for i in range(4):
        key[i]=xor1(key[i],temp)
        temp=key[i]
    key=np.array(key).transpose()
    return key

```

```

def aes(text,key):
    key=hex_gen(key)
    text=hex_gen(text)
    state=xor(text,key)
    print("Round 0 :")
    print(state)
    for i in range(10):
        state=state_gen(state,i)
        key=key_gen(key,i)
        #print(key)

```

```

        state=xor(state,key)
        print("\nRound",i+1,":")
        print(state)
    state=np.array(state).transpose()
    state=np.array(state).reshape(16)
    return state

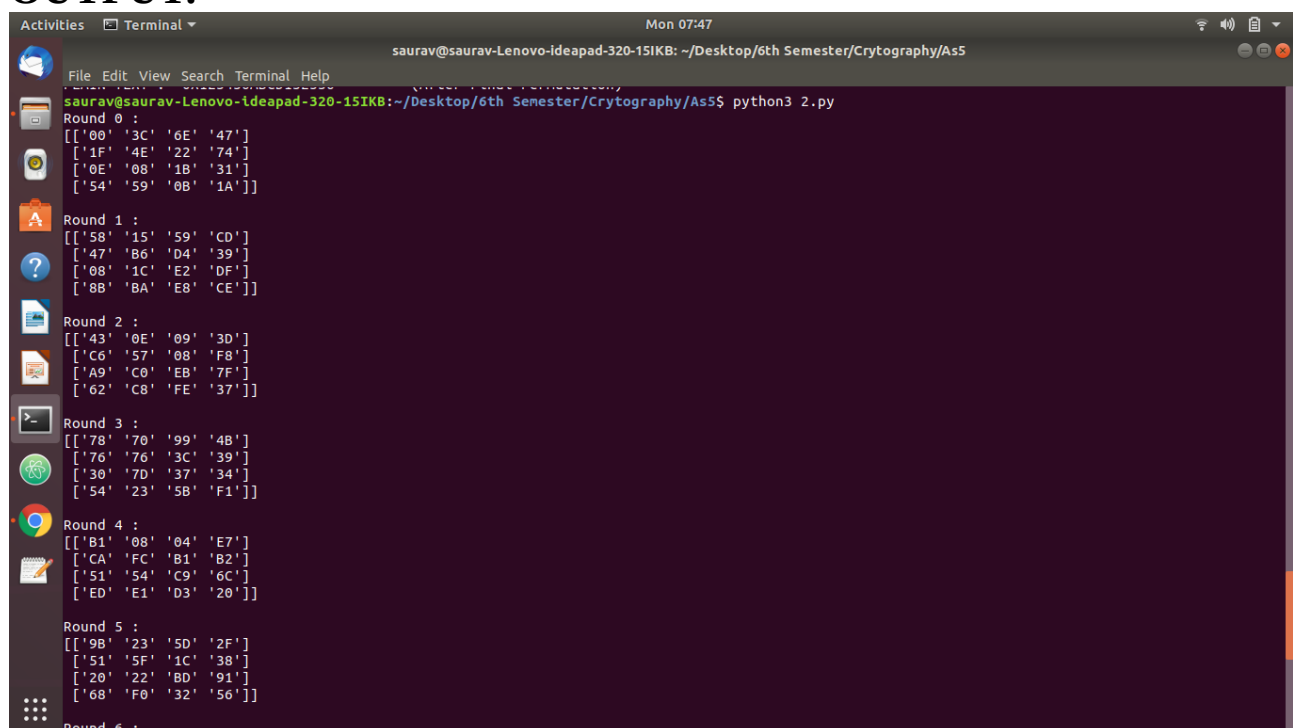
```

```

key="Thats my Kung Fu"
text="Two One Nine Two"
cipher=aes(text,key)
text=""
for i in cipher:
    text+=chr(int(i,16))
print("\nCIPHER TEXT : ",cipher)

```

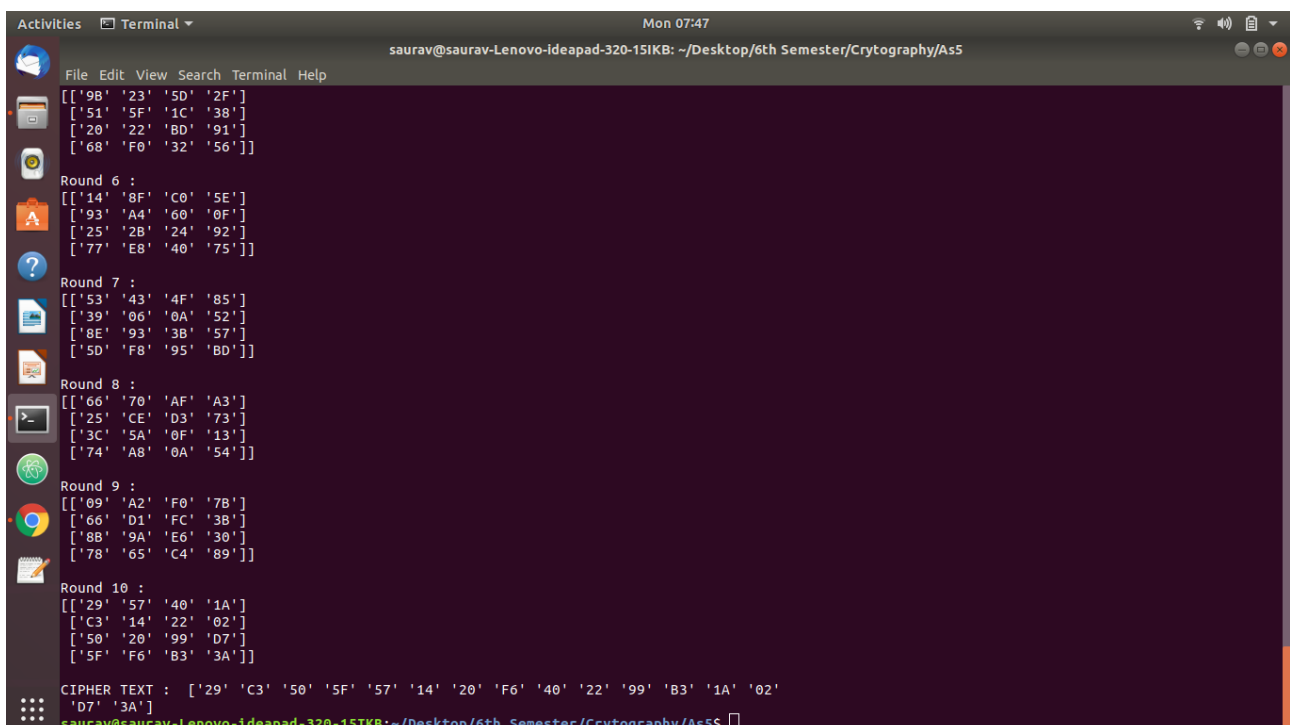
OUTPUT:



```

saurav@saurav-Lenovo-Ideapad-320-15IKB: ~/Desktop/6th Semester/Cryptography/As5
python3 2.py
Round 0 :
[['00' '3C' '6E' '47']
 ['1F' '4E' '22' '74']
 ['0E' '08' '1B' '31']
 ['54' '59' '0B' '1A']]
Round 1 :
[['58' '15' '59' 'CD']
 ['47' 'B6' 'D4' '39']
 ['08' '1C' 'E2' 'DF']
 ['8B' 'BA' 'E8' 'CE']]
Round 2 :
[['43' '0E' '09' '3D']
 ['C6' '57' '08' 'F8']
 ['A9' 'C0' 'EB' '7F']
 ['62' 'C8' 'FE' '37']]
Round 3 :
[['78' '70' '99' '4B']
 ['76' '76' '3C' '39']
 ['30' '7D' '37' '34']
 ['54' '23' '5B' 'F1']]
Round 4 :
[['B1' '08' '04' 'E7']
 ['CA' 'FC' 'B1' 'B2']
 ['51' '54' 'C9' '6C']
 ['ED' 'E1' 'D3' '20']]
Round 5 :
[['9B' '23' '5D' '2F']
 ['51' '5F' '1C' '38']
 ['20' '22' 'BD' '91']
 ['68' 'F0' '32' '56']]
Round 6 :

```



```

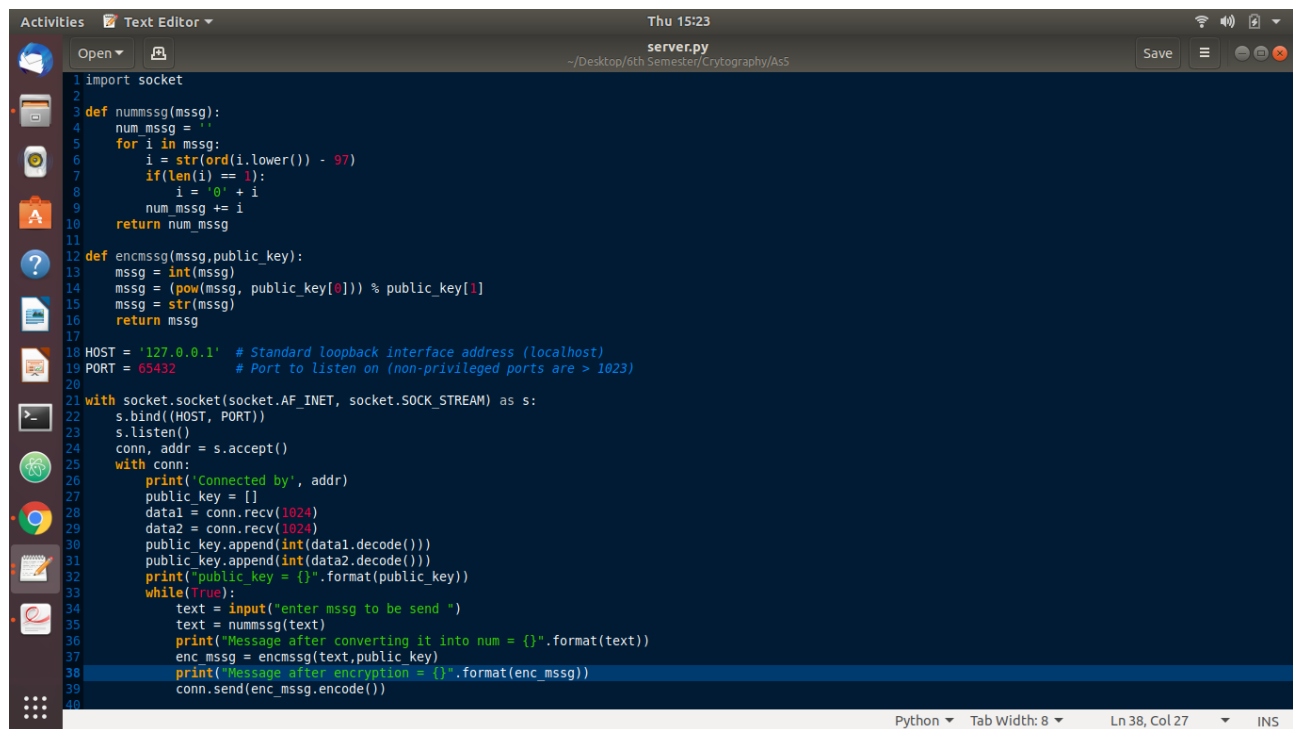
Round 6 :
[['14' '8F' 'C0' '5E']
 ['93' 'A4' '60' '0F']
 ['25' '2B' '24' '92']
 ['77' 'E8' '40' '75']]
Round 7 :
[['53' '43' '4F' '85']
 ['39' '06' '0A' '52']
 ['8E' '93' '3B' '57']
 ['5D' 'F8' '95' 'BD']]
Round 8 :
[['66' '70' 'AF' 'A3']
 ['25' 'CE' 'D3' '73']
 ['3C' '5A' '0F' '13']
 ['74' 'A8' '0A' '54']]
Round 9 :
[['09' 'A2' 'F0' '7B']
 ['66' 'D1' 'FC' '3B']
 ['8B' '9A' 'E6' '30']
 ['78' '65' 'C4' '89']]
Round 10 :
[['29' '57' '40' '1A']
 ['C3' '14' '22' '02']
 ['50' '20' '99' 'D7']
 ['5F' 'F6' 'B3' '3A']]
CIPHER TEXT : ['29' 'C3' '50' '5F' '57' '14' '20' 'F6' '40' '22' '99' 'B3' '1A' '02'
 'D7' '3A']
saurav@saurav-Lenovo-Ideapad-320-15IKB: ~/Desktop/6th Semester/Cryptography/As5

```

Q2.

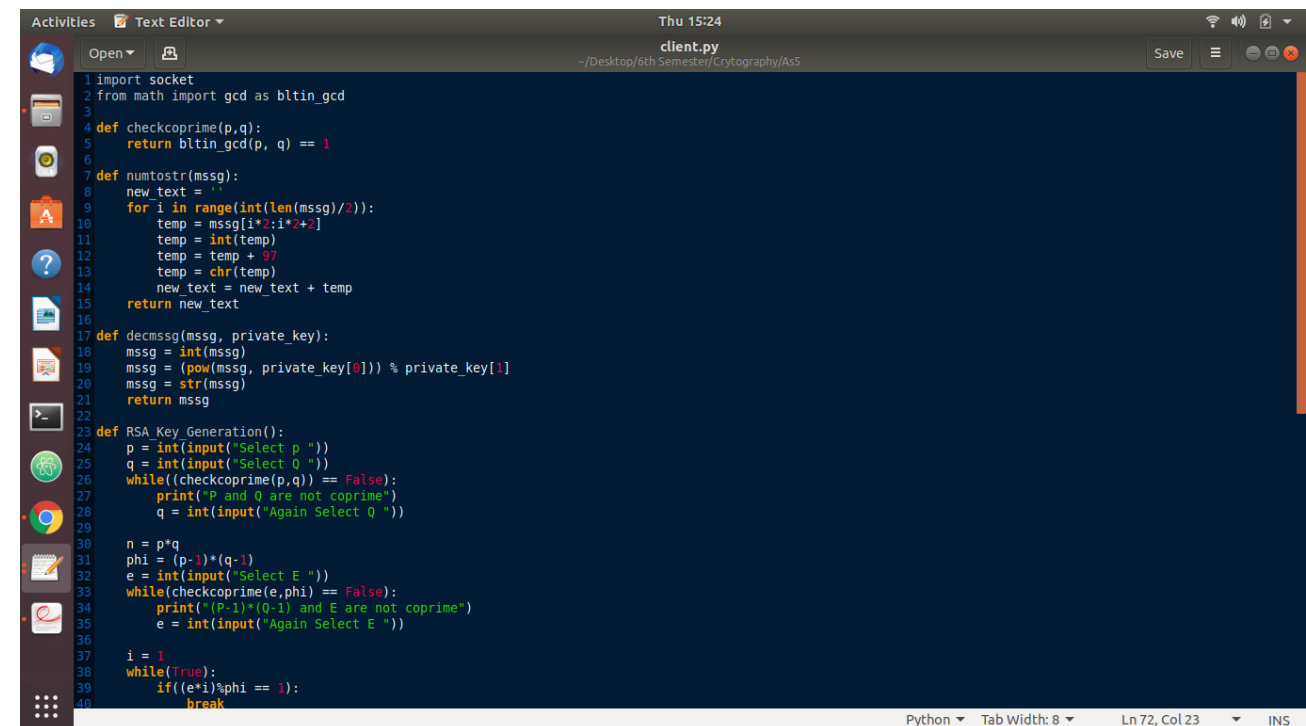
RSA

PROGRAM CODE (Server Side):

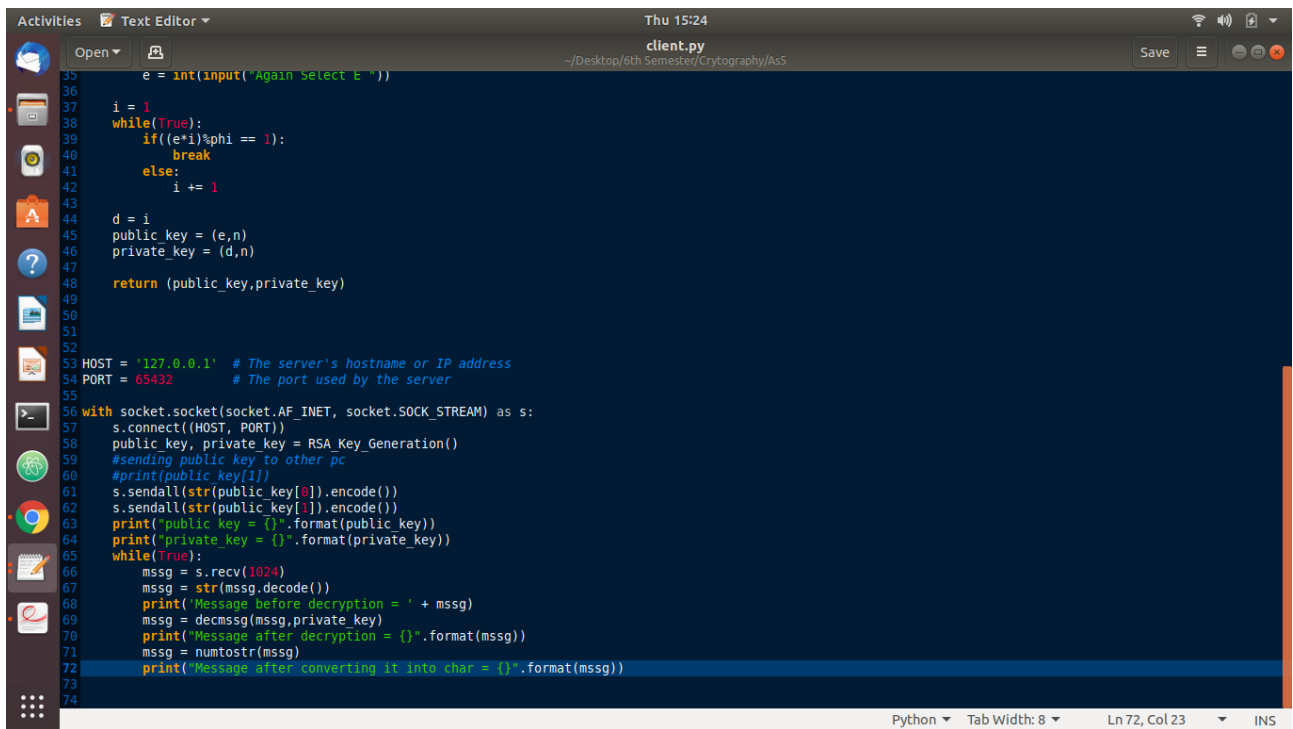


```
1 import socket
2
3 def nummssg(mssg):
4     num_mssg = ''
5     for i in mssg:
6         i = str(ord(i.lower()) - 97)
7         if(len(i) == 1):
8             i = '0' + i
9         num_mssg += i
10    return num_mssg
11
12 def encmssg(mssg, public_key):
13    mssg = int(mssg)
14    mssg = (pow(mssg, public_key[0])) % public_key[1]
15    mssg = str(mssg)
16    return mssg
17
18 HOST = '127.0.0.1' # Standard loopback interface address (localhost)
19 PORT = 65432 # Port to listen on (non-privileged ports are > 1023)
20
21 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
22     s.bind((HOST, PORT))
23     s.listen()
24     conn, addr = s.accept()
25     with conn:
26         print('Connected by', addr)
27         public_key = []
28         data1 = conn.recv(1024)
29         data2 = conn.recv(1024)
30         public_key.append(int(data1.decode()))
31         public_key.append(int(data2.decode()))
32         print("public_key = {}".format(public_key))
33         while(True):
34             text = input("enter mssg to be send ")
35             text = nummssg(text)
36             print("Message after converting it into num = {}".format(text))
37             enc_mssg = encmssg(text, public_key)
38             print("Message after encryption = {}".format(enc_mssg))
39             conn.send(enc_mssg.encode())
40
```

PROGRAM CODE (Client Side):



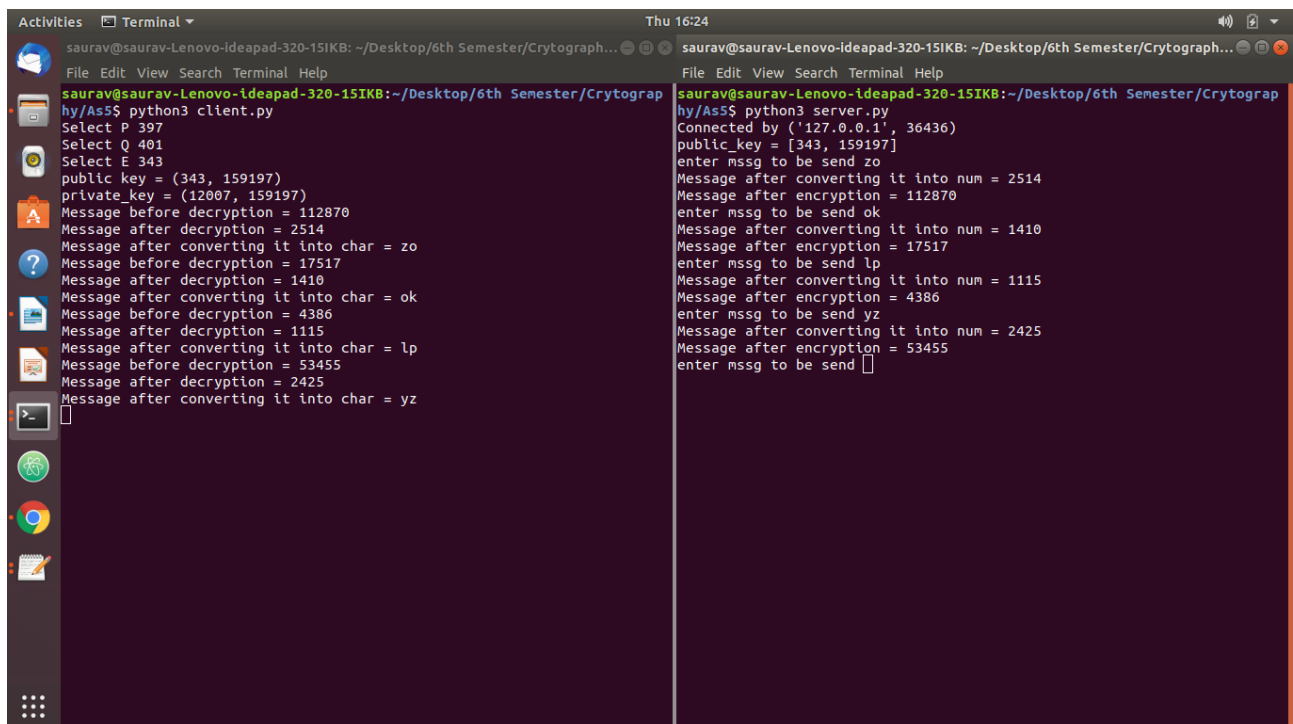
```
1 import socket
2 from math import gcd as bltin_gcd
3
4 def checkcoprime(p, q):
5     return bltin_gcd(p, q) == 1
6
7 def numtostr(mssg):
8     new_text = ''
9     for i in range(int(len(mssg)/2)):
10        temp = mssg[i*2:i*2+2]
11        temp = int(temp)
12        temp = temp + 97
13        temp = chr(temp)
14        new_text = new_text + temp
15    return new_text
16
17 def decmssg(mssg, private_key):
18    mssg = int(mssg)
19    mssg = (pow(mssg, private_key[0])) % private_key[1]
20    mssg = str(mssg)
21    return mssg
22
23 def RSA_Key_Generation():
24    p = int(input("Select p "))
25    q = int(input("Select Q "))
26    while((checkcoprime(p, q)) == False):
27        print("P and Q are not coprime")
28        q = int(input("Again Select Q "))
29
30    n = p*q
31    phi = (p-1)*(q-1)
32    e = int(input("Select E "))
33    while(checkcoprime(e, phi) == False):
34        print("(P-1)*(Q-1) and E are not coprime")
35        e = int(input("Again Select E "))
36
37    i = 1
38    while(True):
39        if((e*i)%phi == 1):
40            break
41
```



The screenshot shows a text editor window titled 'client.py' with the following Python code:

```
35 e = int(input("Again Select E "))
36
37 i = 1
38 while(True):
39     if((e+1)%phi == 1):
40         break
41     else:
42         i += 1
43
44 d = i
45 public_key = (e,n)
46 private_key = (d,n)
47
48 return (public_key,private_key)
49
50
51
52
53 HOST = '127.0.0.1' # The server's hostname or IP address
54 PORT = 65432      # The port used by the server
55
56 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
57     s.connect((HOST, PORT))
58     public_key, private_key = RSA_Key_Generation()
59     #sending public key to other pc
60     #print(public_key[1])
61     s.sendall(str(public_key[0]).encode())
62     s.sendall(str(public_key[1]).encode())
63     print("public key = {}".format(public_key))
64     print("private key = {}".format(private_key))
65     while(True):
66         mssg = s.recv(1024)
67         mssg = str(mssg.decode())
68         print("Message before decryption = " + mssg)
69         mssg = decmssg(mssg,private_key)
70         print("Message after decryption = {}".format(mssg))
71         mssg = numtostr(mssg)
72         print("Message after converting it into char = {}".format(mssg))
73
74
```

OUTPUT:



The screenshot shows a terminal window with two panes. The left pane shows the output of the client script, and the right pane shows the output of the server script.

Left Pane (client.py output):

```
saurav@saurav-Lenovo-Ideapad-320-15IKB: ~/Desktop/6th Semester/Cryptograph...
File Edit View Search Terminal Help
saurav@saurav-Lenovo-Ideapad-320-15IKB:~/Desktop/6th Semester/Cryptograph...
hy/As5$ python3 client.py
Select P 397
Select Q 401
Select E 343
public key = (343, 159197)
private key = (12007, 159197)
Message before decryption = 112870
Message after decryption = 2514
Message after converting it into char = zo
Message before decryption = 17517
Message after decryption = 1410
Message after converting it into char = ok
Message before decryption = 4386
Message after decryption = 1115
Message after converting it into char = lp
Message before decryption = 53455
Message after decryption = 2425
Message after converting it into char = yz
```

Right Pane (server.py output):

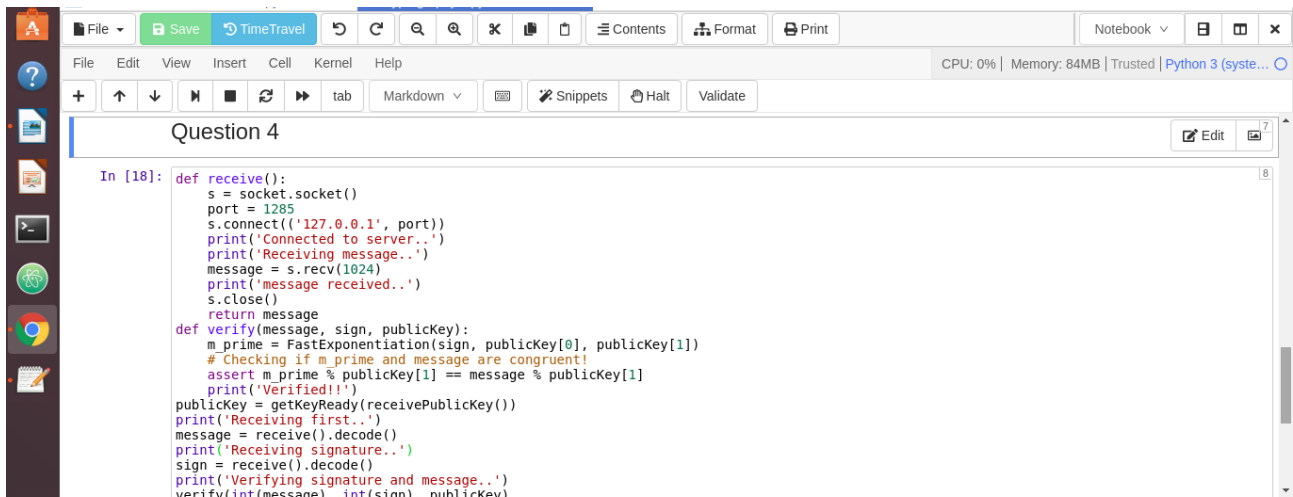
```
saurav@saurav-Lenovo-Ideapad-320-15IKB:~/Desktop/6th Semester/Cryptograph...
File Edit View Search Terminal Help
saurav@saurav-Lenovo-Ideapad-320-15IKB:~/Desktop/6th Semester/Cryptograph...
hy/As5$ python3 server.py
Connected by ('127.0.0.1', 36436)
public_key = [343, 159197]
enter mssg to be send zo
Message after converting it into num = 2514
Message after encryption = 112870
enter mssg to be send ok
Message after converting it into num = 1410
Message after encryption = 17517
enter mssg to be send lp
Message after converting it into num = 1115
Message after encryption = 4386
enter mssg to be send yz
Message after converting it into num = 2425
Message after encryption = 53455
enter mssg to be send
```


Q4.

RSA

(Digital Signature)

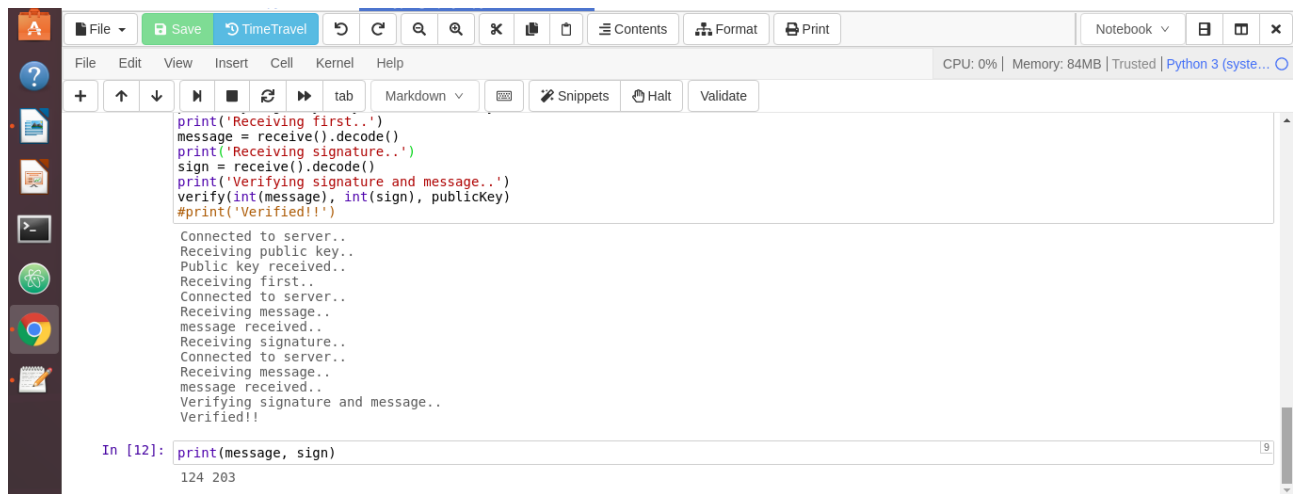
SCREENSHOT:



The screenshot shows a Jupyter Notebook interface with a dark sidebar on the left containing icons for home, search, and various file operations. The top toolbar includes buttons for File, Save, TimeTravel, undo, redo, search, and other standard notebook controls. The main area displays the code for 'Question 4'.

```
In [18]: def receive():
s = socket.socket()
port = 1285
s.connect(('127.0.0.1', port))
print('Connected to server..')
print('Receiving message..')
message = s.recv(1024)
print('message received..')
s.close()
return message

def verify(message, sign, publicKey):
m_prime = FastExponentiation(sign, publicKey[0], publicKey[1])
# Checking if m_prime and message are congruent!
assert m_prime % publicKey[1] == message % publicKey[1]
print('Verified!!')
publicKey = getKeyReady(receivePublicKey())
print('Receiving first..')
message = receive().decode()
print('Receiving signature..')
sign = receive().decode()
print('Verifying signature and message..')
verify(int(message), int(sign), publicKey)
```



The screenshot shows the same Jupyter Notebook interface as the first screenshot, but now displaying the output of the code. The code cell is followed by a large block of text output showing the execution flow: connecting to the server, receiving the public key, receiving the first message, receiving the signature, and finally verifying the signature and message, which results in 'Verified!!'.

```
In [12]: print(message, sign)
124 203
```