Applied Cryptography and Network Security (CSI3002)

LAB ASSESSMENT – 2

Name	**: RITHIV.R**
Reg No	**: 19MIC0113**
Slot	**: L27+L28**

1) DES ENCRYPTION:

Code:

```
print('\nDES ENCRYPTION-RITHIV.R(19MIC0113)\n')

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,
    30,40,51,45,33,48,
    44,49,39,56,34,53,
    46,42,50,36,29,32,
    ]

iterations = [
  1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

list_keys = []
```

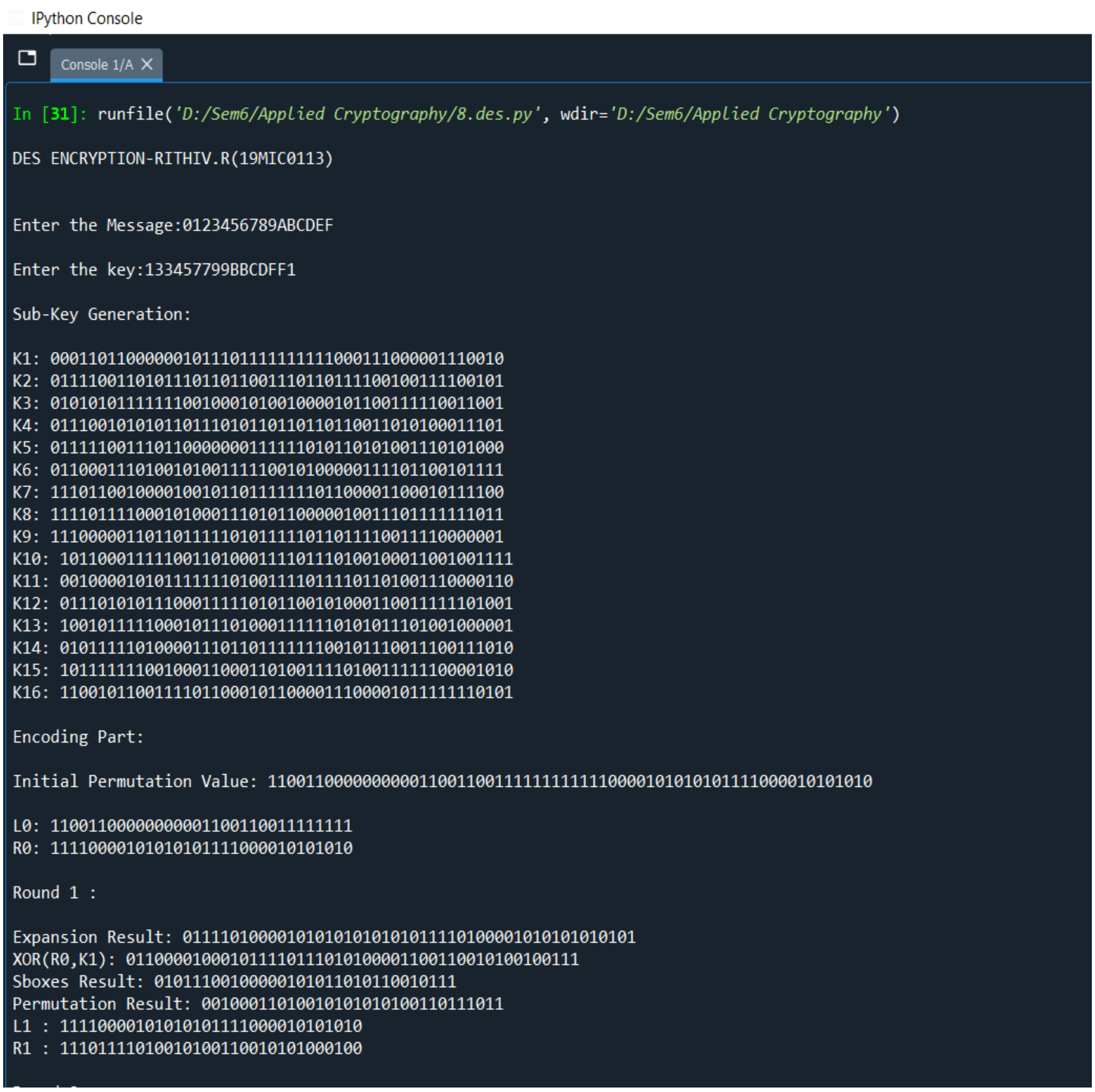
```
IP = [
     58,50,42,34,26,18,10,2,
     60,52,44,36,28,20,12,4,
     62,54,46,38,30,22,14,6,
     64,56,48,40,32,24,16,8,
     57,49,41,33,25,17,9,1,
     59,51,43,35,27,19,11,3,
     61,53,45,37,29,21,13,5,
     63,55,47,39,31,23,15,7
    ]

Expansion = [
   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
    ]

sb = [
[
  [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]
]

]
```

```
IP_inv = [
   40,8,48,1,56,24,64,32,
   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
   ]



def XOR(i,j):
   if(i==j):
      return '0'
   else:
      return '1'

def hextodec(trial):
   value = ''
   for i in trial:
      result = "{0:08b}".format(int(i, 16))[-4:]
      value = value+result
   return value

def key_PC_1(key):
   global PC1
   value = hextodec(key)
   temp = ''
   temp1 = ''
   for j,i in enumerate(value):
       if((j+1)%8==0):
           pass
       else:
           temp = temp + i
   for j,i in enumerate(temp):
      temp1 = temp1 + value[PC1[j]-1]
   return(temp1)

def shift(key):
   getter = key_PC_1(key)
   val1 = getter[:len(getter)//2]
   val2 = getter[len(getter)//2:]
   myarray = []
   for i in iterations:
      temp1 = val1[i:]+val1[:i]
      temp2 = val2[i:]+val2[:i]
      myarray.append(temp1+temp2)
      val1 = temp1
      val2 = temp2
   return myarray

def key_PC_2(key):
   global list_keys
   myarr = shift(key)
   print('\nSub-Key Generation:\n')
   for j,i in enumerate(myarr):
      rem1 = ''
      rem2 = ''
      for c2,c1 in enumerate(i):
           if((c2+1)%7==0):
               pass
           else:
               rem1 = rem1 + c1
      for c2,c1 in enumerate(rem1):
           rem2 = rem2 + i[PC2[c2]-1]
      list_keys.append(rem2)
      print('K'+str(j+1)+':',rem2)

def Initial_Perm(message):
   temporary = ''
   for i in IP:
      temporary = temporary + message[i-1]
   return temporary

def Exp_Perm(R):
   temp = ''
   for i in Expansion:
      temp = temp+R[i-1]
   return temp

def sbox(Box,message):
   b16 = message[0]+message[-1]
   b2345 = message[1:-1]
   val = Box[int(b16,2)][int(b2345,2)]
   res = bin(val)[2:].zfill(4)
   return res

def Permutation(msg):
   val = ''
   for i in P:
      val = val + msg[i-1]
   return val
```

```
def func(R,K,counter):
   global sb
   val = Exp_Perm(R)
   print('Expansion Result:',val)
   modified = ''
   for i,j in zip(val,K):
      modified=modified+XOR(i,j)
   print('XOR('+'R'+str(counter)+',K'+str(counter+1)+'):',modified)
   sixthocc = [modified[i:i+6] for i in range(0, len(modified), 6)]
   sbval = ''
   for Box,message in zip(sb,sixthocc):
      sbval = sbval+sbox(Box, message)
   print('Sboxes Result:',sbval)
   final = Permutation(sbval)
   print('Permutation Result:',final)
   return final

def swap(a,b):
   return b+a

def inverse_IP(encoded):
   temp = ''
   for i in IP_inv:
      temp = temp+encoded[i-1]
   return temp

def bintohex(val):
   number = int(val,2)
   hexa_number = format(number, 'x')
   return hexa_number
```

```
def encode(message):
   print('\nEncoding Part:')
   hexa = hextodec(message)
   ipermutted = Initial_Perm(hexa)
   print('\nInitial Permutation Value:',ipermutted)
   l = ipermutted[:len(ipermutted)//2]
   r = ipermutted[len(ipermutted)//2:]
   print('\nL0:',l)
   print('R0:',r)
   for i in range(16):
      print('\nRound',i+1,':\n')
      k1 = list_keys[i]
      f = func(r,k1,i)
      temp1 = r
      value1 = ''
      for c1,c2 in zip(l,f):
         value1 = value1 + XOR(c1,c2)
      r = value1
      l = temp1
      print('L'+str(i+1),':',l)
      print('R'+str(i+1),':',r)
   swapped = swap(l,r)
   print('\nSwapping Result:')
   print('\nL16:',r)
   print('R16:',l)
   final = inverse_IP(swapped)
   print('\nInverse Pernutation:',final)
   fourocc = [final[i:i+4] for i in range(0, len(final), 4)]
   result = ''
   for i in fourocc:
        result = result + bintohex(i)
    return result.upper()

message = input('Enter the Message:').lower()
key = input('Enter the key:').lower()
key_PC_2(key)
output=encode(message)
print('\nCipher Text Generated:',output)
```

Output:

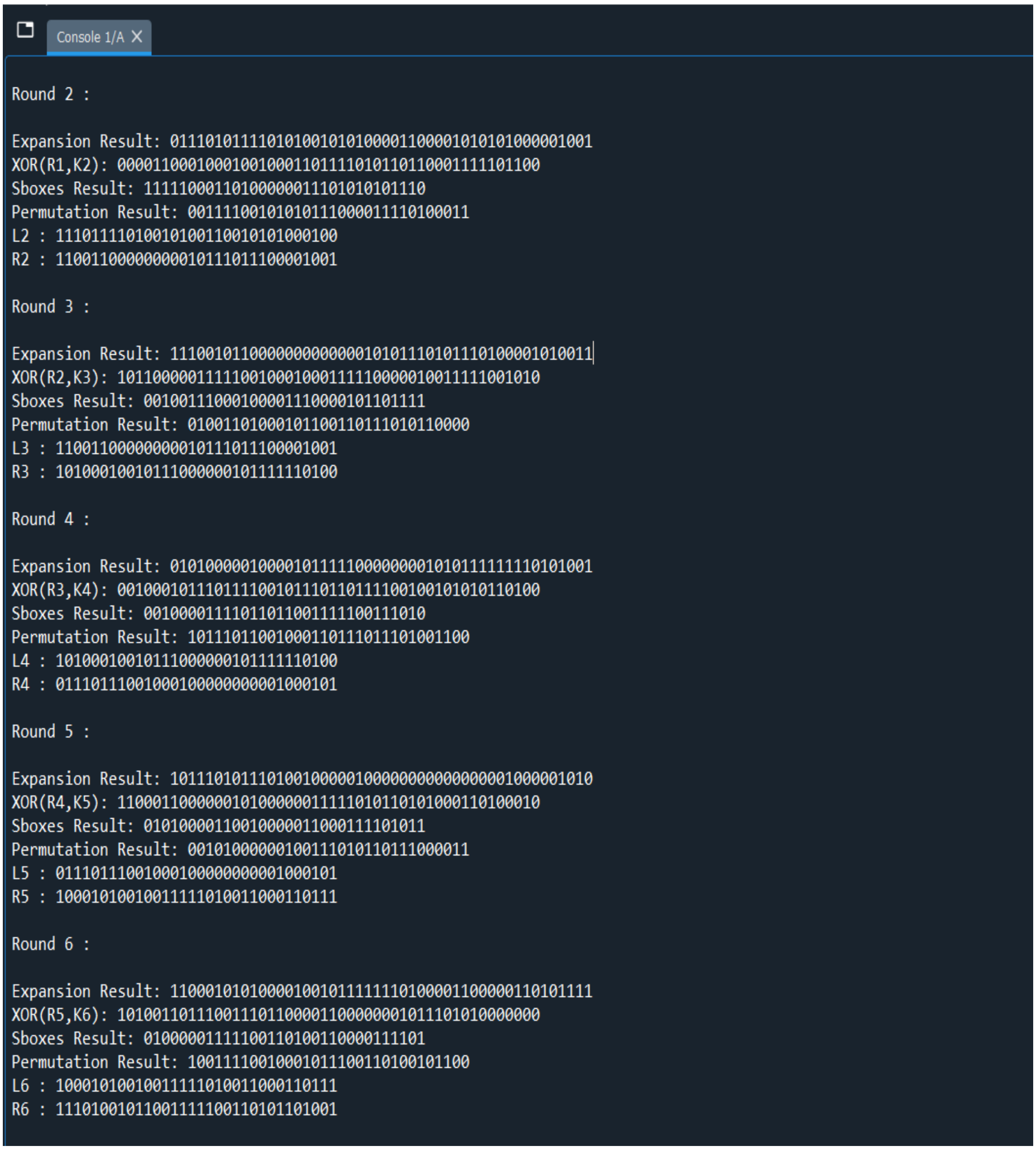


IPython Console

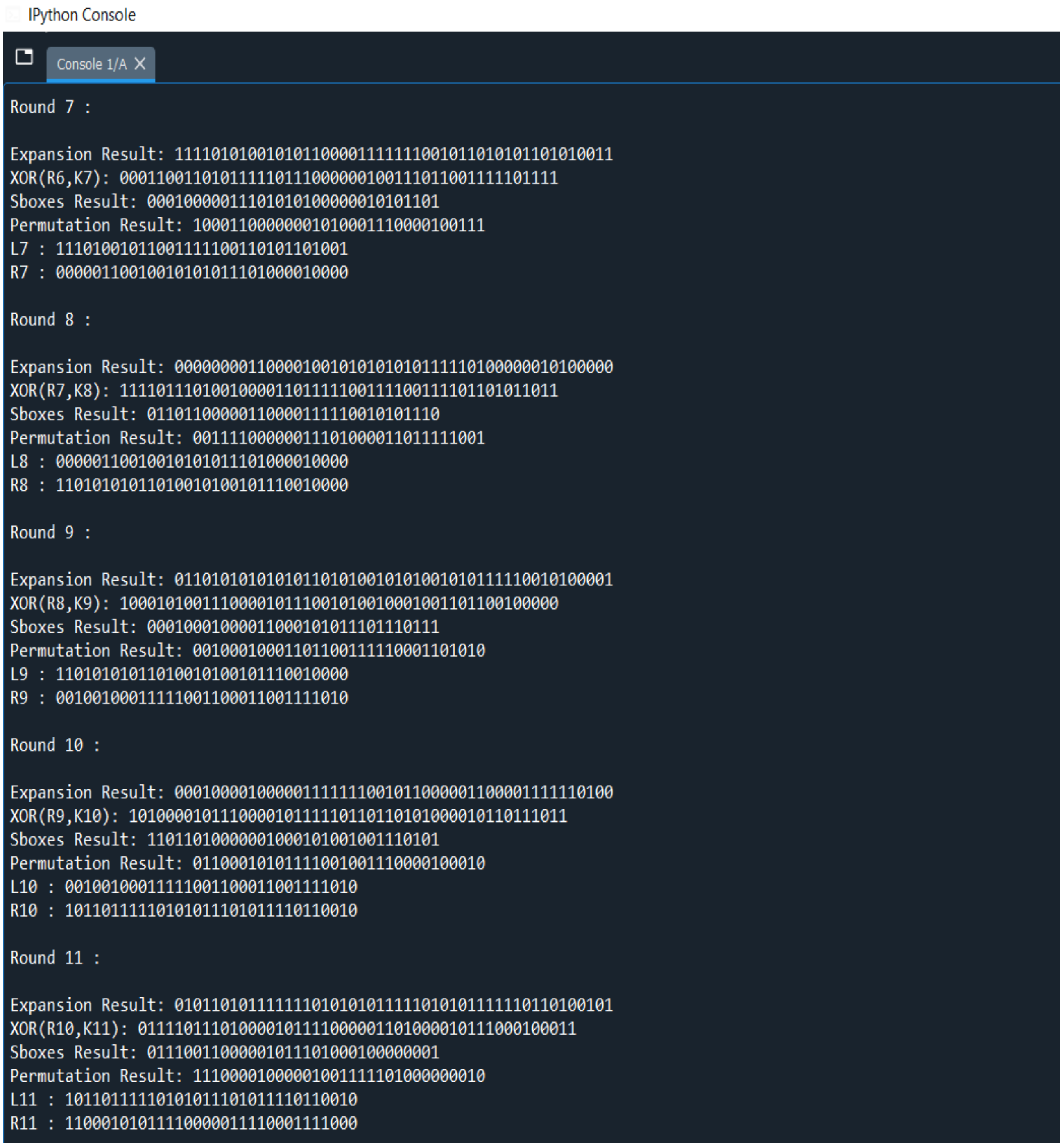
Console 1/A

```
In [31]: runfile('D:/Sem6/Applied Cryptography/8.des.py', wdir='D:/Sem6/Applied Cryptography')

DES ENCRYPTION-RITHIV.R(19MIC0113)


Enter the Message:0123456789ABCDEF

Enter the key:133457799BBCDFF1

Sub-Key Generation:

K1: 000110110000001011101111111111000111000001110010
K2: 011110011010111011011001110110111100100111100101
K3: 010101011111110010001010010000101100111110011001
K4: 011100101010110111010110110110110011010100011101
K5: 011111001110110000000111111010110101001110101000
K6: 011000111010010100111110010100000111101100101111
K7: 111011001000010010110111111101100001100010111100
K8: 111101111000101000111010110000010011101111111011
K9: 111000001101101111101011111011011110011110000001
K10: 101100011111001101000111101110100100011001001111
K11: 001000010101111111010011110111101101001110000110
K12: 011101010111000111110101100101000110011111101001
K13: 100101111100010111010001111110101011101001000001
K14: 010111110100001110110111111100101110011100111010
K15: 101111111001000110001101001111010011111100001010
K16: 110010110011110110001011000011100001011111110101

Encoding Part:

Initial Permutation Value: 1100110000000000110011001111111111110000101010101111000010101010

L0: 11001100000000001100110011111111
R0: 11110000101010101111000010101010

Round 1 :

Expansion Result: 011110100001010101010101011110100001010101010101
XOR(R0,K1): 011000010001011110111010100001100110010100100111
Sboxes Result: 01011100100000101011010110010111
Permutation Result: 00100011010010101010100110111011
L1 : 11110000101010101111000010101010
R1 : 11101111010010100110010101000100
```

IPython Console

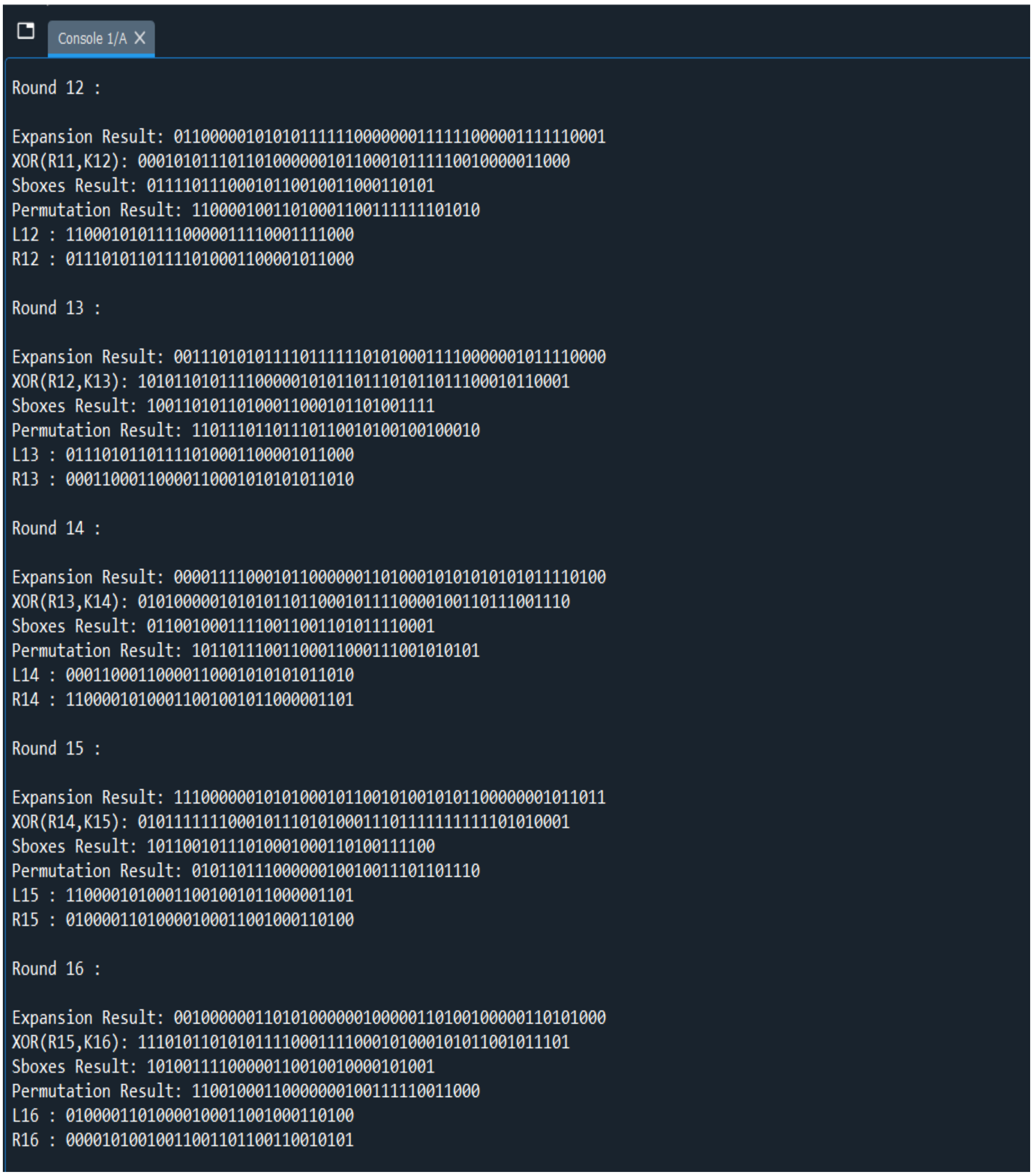


```
Round 2 :

Expansion Result: 011101011110101001010100001100001010101000001001
XOR(R1,K2): 000011000100010010001101111010110110001111101100
Sboxes Result: 11111000110100000011101010101110
Permutation Result: 00111100101010111000011110100011
L2 : 11101111010010100110010101000100
R2 : 11001100000000010111011100001001

Round 3 :

Expansion Result: 111001011000000000000001010111010111010000101001 1
XOR(R2,K3): 101100000111110010001000111110000010011111001010
Sboxes Result: 00100111000100000111000010110111 1
Permutation Result: 01001101000101100110111010110000
L3 : 11001100000000010111011100001001
R3 : 10100010010111000000101111110100

Round 4 :

Expansion Result: 010100000100001011111000000001010111111110101001
XOR(R3,K4): 001000101110111100101110110111100100101010110100
Sboxes Result: 00100001111011011001111100111010
Permutation Result: 10111011001000110111011101001100
L4 : 10100010010111000000101111110100
R4 : 01110111001000100000000001000101

Round 5 :

Expansion Result: 101110101110100100000100000000000000000100001010
XOR(R4,K5): 110001100000010100000011111010110101000110100010
Sboxes Result: 01010000110010000001100011101011
Permutation Result: 00101000000100111010110111000011
L5 : 01110111001000100000000001000101
R5 : 10001010010011111010011000110111

Round 6 :

Expansion Result: 110001010100001001011111110100001100000110101111
XOR(R5,K6): 101001101110011101100001100000001011101010000000
Sboxes Result: 01000001111100110100110000111101
Permutation Result: 10011110010001011100110100101100
L6 : 10001010010011111010011000110111
R6 : 11101001011001111100110101101001
```

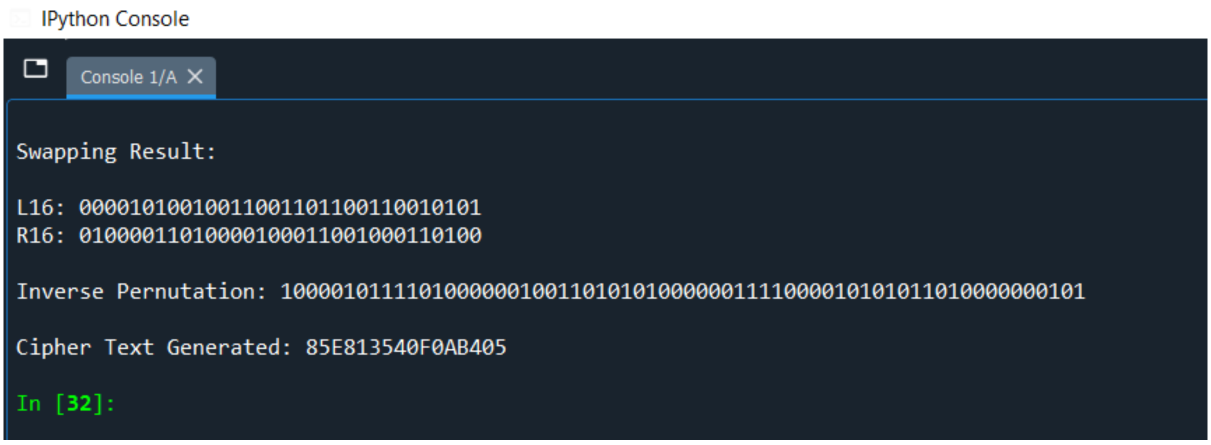


```
Round 7 :

Expansion Result: 111101010010101100001111111001011010101101010011
XOR(R6,K7): 000110011010111110111000000100111011001111101111
Sboxes Result: 00010000011101010100000010101101
Permutation Result: 10001100000001010001110000100111
L7 : 11101001011001111100110101101001
R7 : 00000110010010101011101000010000

Round 8 :

Expansion Result: 000000001100001001010101010111110100000010100000
XOR(R7,K8): 111101110100100001101111100111100111101101011011
Sboxes Result: 01101100000110000111110010101110
Permutation Result: 00111100000011101000011011111001
L8 : 00000110010010101011101000010000
R8 : 11010101011010010100101110010000

Round 9 :

Expansion Result: 011010101010101101010010101001010111110010100001
XOR(R8,K9): 100010100111000010111001010010001001101100100000
Sboxes Result: 00010001000011000101011101110111
Permutation Result: 00100010001101100111110001101010
L9 : 11010101011010010100101110010000
R9 : 00100100011111001100011001111010

Round 10 :

Expansion Result: 000100001000001111111001011000001100001111110100
XOR(R9,K10): 101000010111000010111110110101000010110111011
Sboxes Result: 11011010000001000101001001110101
Permutation Result: 01100010101111001001110000100010
L10 : 00100100011111001100011001111010
R10 : 10110111110101011101011110110010

Round 11 :

Expansion Result: 010110101111111010101011111010101111110110100101
XOR(R10,K11): 011110111010000101111000001101000010111000100011
Sboxes Result: 01110011000001011101000100000001
Permutation Result: 11100001000001001111101000000010
L11 : 10110111110101011101011110110010
R11 : 11000101011110000011110001111000
```

IPython Console



```
Round 12 :

Expansion Result: 011000001010101111111000000011111100000111111110001
XOR(R11,K12): 000101011101101000000101100010111110010000011000
Sboxes Result: 01111011100010110010011000110101
Permutation Result: 11000010011010001100111111101010
L12 : 11000101011110000011110001111000
R12 : 01110101101111010001100001011000

Round 13 :

Expansion Result: 001110101011110111111010100011110000001011110000
XOR(R12,K13): 101011010111100000101011011101011011100010110001
Sboxes Result: 10011010110100011000101101001111
Permutation Result: 11011101101110110010100100100010
L13 : 01110101101111010001100001011000
R13 : 00011000110000110001010101011010

Round 14 :

Expansion Result: 000011110001011000000110100010101010101011110100
XOR(R13,K14): 010100000101010110110001011110000100110111001110
Sboxes Result: 01100100011110011001101011110001
Permutation Result: 10110111001100011000111001010101
L14 : 00011000110000110001010101011010
R14 : 11000010100011001001011000001101

Round 15 :

Expansion Result: 111000000101010001011001010010101100000001011011
XOR(R14,K15): 010111111100010111010100011101111111111101010001
Sboxes Result: 10110010111010001000110100111100
Permutation Result: 01011011100000010010011101101110
L15 : 11000010100011001001011000001101
R15 : 01000011010000100011001000110100

Round 16 :

Expansion Result: 001000000110101000000010000011010010000001101010000
XOR(R15,K16): 111010110101011110001111000101000101011001011101
Sboxes Result: 10100111100000110010010000101001
Permutation Result: 11001000110000000100111110011000
L16 : 01000011010000100011001000110100
R16 : 00001010010011001101100110010101
```


IPython Console

Console 1/A

```
Swapping Result:

L16: 00001010010011001101100110010101
R16: 01000011010000100011001000110100

Inverse Pernutation: 1000010111101000000100110101010000001111000010101011010000000101

Cipher Text Generated: 85E813540F0AB405

In [32]:
```

CODE UPLOADED IN GOOGLE DRIVE LINK:

https://drive.google.com/file/d/1ltzP9_XlPw6R5mgV1rEwM1V3JDSI_dwO/view?usp=sharing