Q1 Team Name

0 Points

ANV

Q2 Commands

10 Points

List all the commands in sequence used from the start screen of this level to the end of the level. (Use -> to separate the commands)

Commands used:-

- **go**
- **dive**
- **dive**
- **back**
- **pull**
- **back**
- **back**
- **go**
- **wave**
- **back**
- **back**
- **thrnxxtzy**

- **read**
- **13472154209765902984527395 7**
- **c**
- **read**

After executing the above commands in sequence, we reach where w

To know "PASSWORD" encryption : –
- **password**

To complete the level, we typed the password "qivjgyimlb"

Q3 CryptoSystem

5 Points

What cryptosystem was used at this level? Please be precise.

- **DES with 6 rounds** was used in this level.
- Two characters for each byte was considered.
- Characters from **d** to **s** are in cipher text.
  We assumed **d** corresponds to **0000**, **e** corresponds to **0001**, ..., **s** corresponds to **1111**.

●We have used **ChosenPlaintextattack** to break the cryptosystem with a **4 ROUND CHARACTERISTIC** .

= >

$(\mathbf{405C0000}, \mathbf{04000000}, \mathbf{1/4}, \mathbf{04000000}, \mathbf{00540000}, \mathbf{5/128}, \mathbf{00540000}, \mathbf{0000}($

Q4 Analysis

80 Points

Knowing which cryptosystem has been used at this level, give a detailed description of the cryptanalysis used to figure out the password. (Use Latex wherever required. If your solution is not readable, you will lose marks. If necessary, the file upload option in this question must be used TO SHARE IMAGES ONLY.)

■ We have used **ChosenPlaintextattack** to break the cryptosystem with a **4 ROUND CHARACTERISTIC** .

We were told by the spirit that the characters were mapped such that 2 letters represent a single byte. We observed that the cipher text only had characters from 'd' to 's'. That means, each character should be represented by 4 bits. We assumed that **d** corresponds to **0000**, **e** corresponds to **0001**, ... **s** corresponds to **1111**.

■**Assumption**:-

●The spirit said that either 4 round or 6 round DES was used. We thought that 4 round DES is too easy to be given for the assignment. So, we assumed that it would be

6 round DES. **4 ROUND CHARACTERISTIC** is used in order to break the encryption.

= >
(405C0000, 04000000, 1/4, 04000000, 00540000, 5/128, 00540000, 0000(

●So, we will consider the plain text pairs with XOR values of **405C0000 04000000 (Hexadecimal)** after the initial permutation. The XOR before the initial permutation is **00009010 1000500**.

We then decided to generate the input text pairs with these XOR values. We generated around 100000 pairs with this XOR value (**00009010 1000500**).
We remembered that the spirit told us that it will give the cipher text to any plain text that we type.
●We used an **"expect"** script to automate cipher text generation from the server, for the given plain texts.
Using **"expect"** script , we captured all output actions of each ciphertext generation using **Logs**, and then using a Python script we retrieved Ciphertexts respectively.

●We wanted to use this cipher text in order to find the key in the 6th round. This was the hardest part. We have the left and right halves of the output of the 6th round (these are the cipher text). We know that for 4 round characteristic, after 4 rounds of encryption, the XOR of the outputs of the 4th round will be **00540000 04000000** with a probability of **0.00038**.

The right half of the XOR of output of the 4th round will be equal to the left half of the XOR of output of 5th round. The right half of the XOR of the output of the 5th round is equal to

the XOR of the left half of cipher text.

■**Analysis** :-

For each ciphertext pair, we will do the following:
We first took the ciphertext pair and converted each of them into binary based on our assumed mapping from above. Let each cipher text pair be **C1, C2**. Then, we applied **Inverse Final Permutation** to our cipher texts pairs.

C1_left be left half of the C1.
C1_right be right half of the C1.
Similarly for **C2_left** and **C2_right**.

We XORed the right halves of the cipher text pairs with each other and with 04000000.

$$=> XOR\_right = XOR(C1\_right, C2\_right, 04000000)$$

Then we applied inverse of round permutation to XOR_right. This will equal to output XOR of the sbox.

$$=> sbox\_out = inverse\_round\_permutation(XOR\_right)$$

The input of the sbox will be equal to the expansion of the XOR of the C1_left and C2_left.

$$=> sbox\_in = expansion(XOR(C1\_left, C2\_left))$$

We will expand C1_left

$$=> C1\_left = expansion(C1\_left)$$

We will use **sbox_in, sbox_out and C1_left** in order to guess the key.

For each sbox, we will do the following:
We will consider the corresponding 6 bits from sbox_in, corresponding 4 bits from sbox_out and corresponding 6 bits from C1_left for each particular sbox.
We generated some pair of values s_out1 and s_out2 such that
(XOR (s_out1, s_out2) = sbox_out) and (0<=s_out1, s_out2<=15).
We generated some pair of values s_in1 and s_in2 such that
(XOR (s_in1, s_in2) = sbox_in) and (sbox(s_in1) = s_out1) and (sbox(s_in2) = s_out2) and (0<=s_in1, sin_2<=63)
Then, we will perform XOR (s_in1, C1_left) to get the bits of the key for that particular sbox.
We will calculate the key frequencies from these.

For each sbox, we got the most frequent keys. We executed this procedure second time for **another 100000 pairs**. The results look like this:

•Result 1 (key,freq):

SBOX 1 : (45, 7064)
SBOX 2 : (51, 7151)
SBOX 3 : (21, 6453)
SBOX 4 : (56, 6413)
SBOX 5 : (15, 7243)
SBOX 6 : (16, 7081)
SBOX 7 : (9, 7105)
SBOX 8 : (61, 7102)

•Result 2 (key, freq):

SBOX 1 : (45, 6531)
SBOX 2 : (51, 8216)
SBOX 3 : (37, 6483)
SBOX 4 : (37, 6417)
SBOX 5 : (15, 6532)
SBOX 6 : (16, 6622)
SBOX 7 : (9, 6596)
SBOX 8 : (61, 6803)

From this, we can see that the most frequent keys of
SBOX 1, SBOX 2, SBOX 5, SBOX 6, SBOX 7, SBOX 8 in Result 1 and

Result 2 are same. So, we can say that we have found out the key values of all Sboxes except for SBOX 3 and SBOX 4. So, we decided to brute force the key values for these sboxes.

So, the 48 bit key looks like this:
101101110011XXXXXXXXXXXX001111010000001001111101

Here, 'X' means we do not know that particular bit.
We applied the reverse of the key scheduling to get the following 56 bit key:
XX1XX1XXX10X1X10XXX11XX10X0X1001001X10001100X11X1

From this, each 'X' can either take a value of '0' or '1'.
Unknown bits = > (20 bits = 12 +8)
So, we generated a total of 2^20 keys from the above key.

We took a plain text **jskenodqpqqdnigi** and got it's cipher text from server as **okrppopiopskesqs**.

So, we decided to use each of these 2^20 keys and decided to encode our plain text **jskenodqpqqdnigi**. If the encrypted text equals **okrppopiopskesqs**, that means we have found our correct key.

The correct 56bit key we found is:
01101110010111100111101100001001001110001100111111010001

The 64 bit key with odd parity is(in hexadecimal):
3BE3CDEC6BA8E90E

Now, when we type **"password"** in server, it gave us the following encrypted text:
sjfkhfkjsfpdlfjodegomrpkljqqkord

We divided it into 2 equal halves and decrypted each of them with the above key.
The binary value of the combination of the decryption is:
01110001011010010111011001101010011001110111001011010010111

We considered each byte (8 bits) in the the above thing as one ascii character and then we found the value to be:
qivjgyimlb000000

We removed the zeroes and the password we obtained is :

Decrypted password: **qivjgyimlb**

We entered this password to clear the level.

The codes uploaded are as follows:
0_generate_input.py is used for generating plain texts with the given XOR value.
1_get_ciphertext.exp is an expect script used to get cipher texts from the server for the corresponding plain texts.
2_extract_ciphertext.py is used to extract ciphertext from the logs file obtained from the above expect script.
3_findkeyfreq.py is used to get the frequencies of the most frequent keys for the sboxes.
4_generate_keys.py is used to generate 2^20 keys to be bruteforced.
5_find_key_password.py is used to find the correct key and decode the password.
8_64_bit_key.py is used to obtain the 64 bit key from 56 bit key.

No files uploaded

Q5 Password

5 Points

What was the password used to clear this level?

Password used to clear this level is :- **qivjgyimlb**

Q6 Codes

0 Points

Unlike previous assignments, this time it is MANDATORY that you upload the codes used in the cryptanalysis. If you fail to do so, you will be given 0 marks for the entire assignment.

0_generate_input.py Download

```
1  from random import randint
2  def convert(i):
3      s=''
4      for j in i:
5          j=int(j,16)
6          s+=chr(ord('d')+j)
7      return s
8  xor="0000901010005000"#xor for 4 round characteristics
9  file=open('plaintext.txt','w')
10 for _ in range(100000):
11     i=''
12     for __ in range(16):
13         i+=hex(randint(0,15))[2:]
```

```
14
15      j="".join(hex(int(i[k],16)^int(xor[k],16))[2:] for k in range(16))
16      i,j=convert(i),convert(j)
17      file.write(i+"\n")
18      file.write(j+"\n")
19      print(_,end="\r")
20  #generating 100k pairs of plain text with the xor value of 0000901010005000
```

1_get_ciphertext.exp Download

```
1   #!/usr/bin/expect -f
2   #!/usr/bin/expect -f
3
4   set timeout -1
5
6   spawn ssh students@172.27.26.188
7
8   expect "students@172.27.26.188's password:"
9
10  send -- "cs641a\n"
11
12  expect "*group name:"
13
14  send -- "ANV\n"
15
16  expect "*password:"
17
18  send --  "anv@1998\n"
19
20  expect "*Level you want to start at:"
21  send --  "3\r"
22
23  expect "*You decide to investigate*"
24  send -- "thrnxxtzy\r"
25
```

```
expect "*So does a glass panel next to it*"
send -- "read\r"

expect "*loudly to pass this level!*"
send -- "134721542097659029845273957\r"

expect "*continue>"
send -- "c\r"

expect "*your right and you shiver again."
send -- "read\r"

expect "*I am sure you can figure it out though*"


set f [open "/home/ar/Downloads/plaintext.txt" "r"]
set data [read $f]

set count 0

#linebyline
log_file -a "logs_cipher.txt"
foreach line $data {
        send -- "$line\r"
        send -- "\r"
        send -- "c\r"


}

close $f

interact


```

```
61
62  expect eof
63
```

2_extract_ciphertext.py Download

```
1   x=open("logs_cipher.txt")#extracting ciphertext from logs.
2   file=open("ciphertext.txt",'w')
3   count=0
4   for i in x:
5       if "\t" in i:
6           i=i.replace("\n","").replace("\t","").strip()
7           if len(i)==16:
8               file.write(i+"\n")
9               count+=1
10              print(count,end='\r')
11
12
```

3_find_keyfreq.py Download

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

```
11, 4, 1, 15, 2, 14, 12, 3, 5, 11, 10, 5, 14, 2, 7, 12],

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

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

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

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

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

```
inversefinalperm = [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,
          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]
inverperm = [9, 17, 23, 31,
       13, 28, 2, 18,
       24, 16, 30, 6,
       26, 20, 10, 1,
       8, 14, 25, 3,
       4, 29, 11, 19,
       32, 12, 22, 7,
       5, 27, 15, 21]
def inverse_permutation(a):
    res=""
    for i in range(len(a)):
        res+=a[inverperm[i]-1]
    return res
def inverse_finalpermutation(a):
    res=""

    for i in range(len(a)):
        res+=a[inversefinalperm[i]-1]
    return res

def expansion(a):
    res=''
    for i in range(len(expan)):
        res+=a[expan[i]-1]
    return res
def hexbin(a):
    res=''
```

```
    for i in a:
        i=bin(int(i,16))[2:]
        res+='0'*(4-len(i))+i
    return res
def strbin(a):
    a=a.replace("\n","")
    res=""
    for i in a:
        i=bin(ord(i)-ord('d'))[2:]
        res+='0'*(4-len(i))+i

    return res

l15=hexbin("04000000")
def xor(a,b):

    return "".join(str(int(a[i])^int(b[i])) for i in range(len(a)))
def left_right(j):
    i=x[j]
    i=i.replace("\n","")
    return expansion(i[:32]),i[32:]
def sbox(a,ind):
    res=[]
    a=int(a,2)
    for i in range(len(subbox)):
        if subbox[i]==a:
            res.append(i)
    row=int(a[0]+a[-1],2)
    col=int(a[1:-1],2)


    i=bin(subbox[ind][row][col])[2:]
    return '0'*(4-len(i))+i

def key(sbox_in,sbox_out,exp,ind):
```

```
    sbox_out=int(sbox_out,2)
    sbox_in=int(sbox_in,2)
    exp=int(exp,2)
    for sboxout1 in range(16):
        sboxout2=sbox_out^sboxout1
        for i in (kk for kk in range(len(subbox[ind])) if subbox[ind][kk]==sboxout1):
            for j in (ll for ll in range(len(subbox[ind])) if subbox[ind]
[ll]==sboxout2):
                if i^j==sbox_in:
                    zz=i^exp
                    if zz in ke[ind]:
                        ke[ind][zz]+=1
                    else:
                        ke[ind][zz]=1
ke=[dict() for i in range(8)]
x=list(open("ciphertext2.txt",encoding='utf-8'))
for j in range(0,len(x),2):
    print(j,end='\r')
    if j+1>=len(x):break
    x[j]=inverse_finalpermutation(strbin(x[j]))
    x[j+1]=inverse_finalpermutation(strbin(x[j+1]))
    left1,right1=x[j][:32],x[j][32:]
    left2,right2=x[j+1][:32],x[j+1][32:]
    right=xor(right1,right2)
    right=xor(right,l15)
    sbox_out=inverse_permutation(right)
    sbox_in=expansion(xor(left1,left2))
    left1=expansion(left1)
    ccc=0
    for i in range(0,48,6):
        key(sbox_in[i:i+6],sbox_out[ccc:ccc+4],left1[i:i+6],i//6)
        ccc+=4


for i in range(8):
```

```
    print(i,ke[i])
ff=open("keys.txt",'w')
for i in range(8):
    zz=sorted(ke[i].items(),key=lambda j:(j[1],j[0]),reverse=True)
    ff.write(str(zz)+"\n")
```

4_generate_keys.py Download

```
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
]


s="101101110011XXXXXXXXXXXX0011110100000010011111101"#hardcoded key from key frequencies. X means we do not know that particular bit.
key=['X']*56
for i in range(len(s)):
    key[pc2[i]-1]=s[i]

shifts=[1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
for i in range(6):
    for j in range(shifts[i]):
        le,ri=key[:28],key[28:]
        le=[le[-1]]+le[:-1]
        ri=[ri[-1]]+ri[:-1]
        key=le+ri
key="".join(key)#original 56bit key.
def f(i,s):
```

```
    if i==l:
        file.write(s+"\n")
        return
    if key[i]=='X':
        f(i+1,s+'0')
        f(i+1,s+'1')
    else:
        f(i+1,s+key[i])
file=open("allkeys.txt",'w')
l=len(key)
f(0,'')#generating all the 2^20 keys to be bruteforced.
```

5_find_key_password.py Download

```
#!/usr/bin/env python
# coding: utf-8

# In[18]:


def hex2bin(s):
    mp = {'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"}
    bin1 = ""
    for i in range(len(s)):
        bin1 = bin1 + mp[s[i]]
    return bin1

def bin2hex(s):
    mp = {"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'}
    hex = ""
    for i in range(0, len(s), 4):
```

```
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]

    return hex


# In[19]:


# Decimal to binary conversion
def decimal_to_binary(num):
    t = bin(num).replace("0b", "")

    if(len(t)%4 != 0):
        div = len(t) / 4
        div = int(div)
        counter =(4 * (div + 1)) - len(t)
        for i in range(0, counter):
            t = '0' + t
    return t
# binary to Decimal conversion
def bin2dec(b):
    res=0
    j=0
    for i in range(len(b)-1,-1,-1):
        res += int(b[i])*(2**(j))
        j+=1
    return res


# In[20]:
```

```
53
54
55  #XOR opereation
56  def XOR(e1,e2):
57      if e1==e2:
58          return '0'
59      else:
60          return '1'
61
62  def xor_bitwise(e1,e2):            #gives xor value of i/p pairs
63      op=''
64      for i in range(len(e1)):
65          op += XOR(e1[i],e2[i])
66      return op
67
68
69  # In[21]:
70
71
72  # shifting the bits towards left by nth shifts
73  def shift_left(b,num):
74     s=""
75     s = b[num:]
76     s = s + b[0:num]
77     return s
78
79
80  # In[22]:
81
82
83  # INITIAL PERMUTATION (IP) tables
84  IP = [57, 49, 41, 33, 25, 17,  9,  1,
85        59, 51, 43, 35, 27, 19, 11,  3,
86        61, 53, 45, 37, 29, 21, 13,  5,
87        63, 55, 47, 39, 31, 23, 15,  7,
```

```
      56, 48, 40, 32, 24, 16,  8,  0,
      58, 50, 42, 34, 26, 18, 10,  2,
      60, 52, 44,36, 28, 20, 12,  4,
      62, 54, 46, 38, 30, 22, 14,  6]

#  REVERSE PERMUTATION (RFP) tables
RFP = [ 7, 39, 15, 47, 23, 55, 31, 63,
       6, 38, 14, 46, 22, 54, 30, 62,
       5,37, 13, 45, 21, 53, 29, 61,
       4, 36, 12, 44, 20, 52, 28, 60,
       3, 35, 11, 43, 19, 51, 27, 59,
       2, 34, 10, 42, 18, 50, 26, 58,
       1, 33,  9, 41, 17, 49, 25, 57,
       0, 32,  8, 40, 16, 48, 24, 56]


# In[23]:


# Permute function to rearrange the bits
def initial_permute(k):
    permutation = ""
    for i in IP:
        permutation = permutation + k[i]
    return permutation

def final_permute(k):
    permutation = ""
    for i in RFP:
        permutation = permutation + k[i]
    return permutation


# In[32]:
```

```
123
124 #expansion table
125 exp_d = [31,  0,  1,  2,  3,  4,  3,  4,  5,  6,  7,  8,  7,  8,  9, 10, 11,
126        12, 11, 12, 13, 14, 15, 16, 15, 16, 17, 18, 19, 20, 19, 20, 21, 22,
127        23, 24, 23, 24, 25, 26, 27, 28, 27, 28, 29, 30, 31,  0]
128
129 #ROUND permutation table
130 per = [15,  6, 19, 20, 28, 11, 27, 16,  0, 14, 22, 25,  4, 17, 30,  9,  1,
131         7, 23, 13, 31, 26,  2,  8, 18, 12, 29,  5, 21, 10,  3, 24]
132 #ROUND Inverse-permutation
133 i_per = [ 8, 16, 22, 30, 12, 27,  1, 17, 23, 15, 29,  5, 25, 19,  9, 0, 7,
134        13, 24, 2, 3, 28, 10, 18, 31, 11, 21,  6,  4, 26, 14, 20]
135
136
137 #EXPANSION OPERATION
138 def expansion(b):
139     exp = ''
140     for ind in exp_d:
141         exp+=b[ind]
142     return exp
143
144 #PERMUTATION OPERATION
145 def permutation(b):
146     perm=''
147     for ind in per:
148         perm += b[ind]
149     return perm
150
151 #PERMUTATION OPERATION
152 def inv_perm(b):
153     perm=''
154     for ind in i_per:
155         perm += b[ind]
156     return perm
157
```

```
#S_BOX operation
def s_box(ip,num):  #6bits,whichsbox
    S=[[4, 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 ],

      [5, 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]
      ]

    b0b1 = ip[0]+ip[-1]

    val = S[num][int(b0b1,2)*16 + int(ip[1:5],2)]
    return decimal_to_binary(val)



# In[ ]:




# In[33]:


# Dropping parity bits
keyp = [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 ]

#  Number of bit shifts
shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
               1, 2, 2, 2,
               2, 2, 2, 1 ]

# Key- Compression Table : from 56 bits to 48 bits
key_comp = [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 ]

def key_56to48(k):
    permutation = ""
    for i in range(48):
        permutation = permutation + k[key_comp[i] - 1]
    return permutation



# In[ ]:




# In[37]:


```

```
def encryption(plaintext1,round_key_list):
    plaintext1 = hex2bin(plaintext1)

    # Initial Permutation
    plaintext1 = initial_permute(plaintext1)
    # Splitting
    left = plaintext1[0:32]
    right = plaintext1[32:64]

    #6ROUND DES
    for i in range(0, 6):
        #  Expansion
        e = expansion(right)

        # XOR RoundKey and expansion output
        xor_op = xor_bitwise(e, round_key_list[i])

        #S-BOX output
        s_op = ''
        s_i=0
        for i in range(0,48,6):
            s_op += s_box(xor_op[i:i+6],s_i)
            s_i += 1

        #permutation
        op_perm = permutation(s_op)

         # XOR left and permutation_output
        result = xor_bitwise(left, op_perm)

        #NEXT ROUND inputs
        left, right = right, result

    # Combination
    combine = left + right
```

```
    # Final permutation
    cipher_text = final_permute(combine)
    return cipher_text


# In[38]:


#doubt
#CONVERTING OUR PLAINTEXT INTO HEXADECIMAL FORMAT
c=0
plain_text = "jskenodqpqqdnigi"
plain_text = "".join(hex(ord(j)-ord('d'))[2:] for j in plain_text).upper()


# In[40]:

keyfound=False
for key in open("allkeys.txt"):            #change to allkeys.txt
    key=key.replace("\n","")

    # Divding into two equal-halfs
    left = key[0:28]    # rkb for RoundKeys in binary
    right = key[28:56]

    #Round Keys
    round_key_list = []

    # for des6
    for i in range(0, 6):
        # Shifting  bits of key

        left = shift_left(left, shift_table[i])
        right = shift_left(right, shift_table[i])
```

```
        combine_str = left + right

        # Compression of key from 56 to 48 bits
        round_key = key_56to48(combine_str)

        round_key_list.append(round_key)

    print(c,end='\r')
    c+=1

    cipher_text = encryption(plain_text, round_key_list)

    cipher_text=bin2hex(cipher_text)
    cipher_text="".join(chr(ord('d')+int(j,16)) for j in cipher_text)
    #print(cipher_text)
    if cipher_text=="okrppopiopskesqs":
        print("56 bit Key is : ",key)
        keyfound=True
        break



# In[41]:

if keyfound:

    res=""
    for ii in ("sjfkhfkjsfpdlfjo","degomrpkljqqkord"):
        ii="".join(hex(ord(j)-ord('d'))[2:] for j in ii).upper()
        res+= encryption(ii, round_key_list[::-1])
    print(res)
    s=res
    z=[]
    for i in range(0,len(s),8):
        z.append(chr(int(s[i:i+8],2)))
```

```
368     print("Decrypted ascii password is : ","".join([i for i in z if i!='0']))#getting
    final ascii password. we ignore the zeroes.
369
370
371
372 # In[ ]:
373
374
375
376
377
```

8_64_bit_key.py

Download

```
1  keyp = [57, 49, 41, 33, 25, 17, 9,
2          1, 58, 50, 42, 34, 26, 18,
3          10, 2, 59, 51, 43, 35, 27,
4          19, 11, 3, 60, 52, 44, 36,
5          63, 55, 47, 39, 31, 23, 15,
6          7, 62, 54, 46, 38, 30, 22,
7          14, 6, 61, 53, 45, 37, 29,
8          21, 13, 5, 28, 20, 12, 4 ]
9
10 k="0110111001011110011110110000100100111000110011111110100 01"
11 c=0
12 res=['X' for i in range(64)]
13 for i in range(64):
14     if i%8!=7:
15         res[i] = k[keyp.index(i+1)]
16
17
18 for i in range(64):
19     if res[i]=='X':
20         res[i]=str(1-c)
21         c=0
```

```
    else:
        c = (c+int(res[i]))%2
print("".join(res))

```

Assignment 4

GRADED

GROUP

Vikas
Dibbu Amar Raja
Idamakanti Venkata Nagarjun Reddy
View or edit group

TOTAL POINTS

100 / 100 pts

QUESTION 1

Team Name — **0** / 0 pts

QUESTION 2

Commands — **10** / 10 pts

QUESTION 3

CryptoSystem — **5** / 5 pts

QUESTION 4

Analysis **80** / 80 pts

QUESTION 5

Password **5** / 5 pts

QUESTION 6

Codes **0** / 0 pts