Q1 Team Name
0 Points

Group Name

d2ce09fd5842b342d5b3b66d10b6daef

Q2 Commands
5 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)

enter->jump->jump->back->pull->back->back->enter->wave->back->back->thrnxxtzy->read->the_magic_of_wand->c->read

Q3 Cryptosystem
10 Points

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

6 Round Data Encryption Standard (DES)

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.)

Detailed Description:

1.We used "read" command on the first screen because we could see a panel close, but there was nothing written there. Hence, we used “enter”.

2.We used "jump" because we were at the edge of the lake on the following screen. We emerged once more. So, we "jump" once more. We attempted to "pull" the magic wand there but failed.

3.So, we repeated the above commands but instead of “pull” we used “back” and after then entered “pull” which results to got our band successfully.

4. We then returned to the initial screen and entered "back" twice. And attempted "read," but it remained blank. After some time, we discovered that it has to do with the chapter's name, THE SPIRIT, and that level 3 contained a spirit.

5. So, we had gone once again to level 3 and used " enter" and reached the next (second) screen where we entered the command "wave" and freed the spirit. We then used "back" command twice and then after that we went further by writing "thrnxxtzy" in the command screen. Thereafter We typed "read" in the command screen. After that the flow of the commands we entered was "the_magic_of_wand", "c". Then from the first screen of 4th chapter we once again "read", and we got the question read by spirit.

6. After putting in the password on the level 4 screen, we got "ijnjpqjlnqupllihsqpirkhlnoijjmqu" as the ciphertext. In the question, it was said that this is 4 or 6 rounds of DES. As 4 rounds are easier to break, and the professor made it clear that it is not 10 round DES, and It would be easier to break 4 rounds. So, we thought we should start with 6 round thoughts and, if that doesn't work, switch to the same analysis for the 4 round DES.

Method we used for cryptanalysis is described in detail below:

1. Given that DES has a block size of 64 bits, or 8 bytes, it was assumed that two letters would equal one byte, means that 16 letters stood for one block size.

2. After attempting as many inputs as were feasible for our additional study, we discovered that the output contains 16 letters, ranging from the letter's "f" to "u." Additionally, since each alphabet was mapped to a number between 0 and 15, we were forced to assume that the input only comprised of these 16 letters. Letter 'f' was mapped to bit sequence f->0000. g->0001, h->0010, i->0011, j->0100, k->0101, l->0110, m->0111, n->1000. o->1001, p->1010, q->1011, r->1100, s->1101, t- >1110, u->1111, are the corresponding mappings of letters to the 4-bit binary numbers created out of plaintext and ciphertext bits.

3. To determine the key needed to decrypt data using DES, we performed the "Chosen Plaintext Attack". We used differential cryptanalysis to construct plaintext pairings, which we then fed into the DES algorithm to produce corresponding ciphertext pairs in order to develop this approach. The key was then discovered using this approach, which we subsequently used to achieve our objective of ciphertext decryption in order to find the required plaintext.

4. We ran the "plaingen.py" Python file to make about 5,000 different plaintexts for each characteristic. We used 2 and 3 round characteristics to get a chance(probability) of 0.0625%. Here is a list of what they are: [40 08 00 00 04 00 00 00] and [00 20 00 08 00 00 04 00]

To produce 5000 pairs which fulfil our demand of [40 08 00 00 04 00 00 00] characteristic we made it to happen that their XOR is [00 00 80 10 00 00 40 00] which is gathered by having the application of initial permutation on the before mentioned characteristic.

5. On the same basis, we made 5000 pairs of plaintexts that can satisfy the [00 20 00 08 00 00 04 00] characteristic. We made sure that their XOR is [00 00 08 01 00 10 00 00], which we got by applying initial permutation to the [00 20

00 08 00 00 04 00] characteristic. These plaintexts has been saved on plaintexts1.txt and plaintexts2.txt, respectively.

6. After finishing the plaintext generation task, which is necessary for the differential cryptanalysis method we are going to use now, we made 2 ciphertext files, named ciphertext1.txt and ciphertext2.txt, by:
a). Using a c++ program "script_generator.cpp" which generates a bash scripts.
b). These scripts use "execute" and "send" command of Linux terminal.
c). Using "execute" and "send" command helps us to automate interaction with ssh-server faster.
d). After running these scripts, we get logs, from which we get the ciphertext by using a python program "get_ciphertext.py" which extracts the corresponding cyphertext from these logs.

7. To identify the key used in the method, differential cryptanalysis is done. "cryptoanalysis.py" python script was used to do this. We started by reading ciphertext1.txt, and for each ciphertext, we turned each letter into binary using the mapping where f is 0000 and u is 1111, and letters in between are mapped to values between 0000 and 1111.

5. Next, we used inverse final permutation to acquire the values of (L6, R6) and (L'6, R'6). We already know that R5 = L6, so we use R5 and R'5 to get the output of the expansion box and the input XOR of s-boxes for the 6th round.

6. For the first characteristic, L5 = [04 00 00 00], and for the second characteristic, L5 = [00 00 04 00]. Then we do L5 xor (R6 xor R'6) and use inverse permutation to get XOR of s-boxes for 6th round.

Let,

$$E(R5) = \alpha_1\alpha_2...\alpha_8 \text{ and } E(R5') = \alpha_1'\alpha_2'...\alpha_8'$$

where,

$$|\alpha_i| = 6 = |\alpha_i^{'}|$$

and,

$$k_6 = k_{6,1}k_{6,2}...k_{6,8}$$

and,

$$\beta_i = \alpha_i \oplus k_{6,i} \text{ and } \beta_i^{'} = \alpha_i^{'} \oplus k_{6,i}$$

We know that,

$$\alpha_i, \alpha_i^{'}, \beta_i \oplus \beta_i^{'} \text{ and } \gamma_i \oplus \gamma_i^{'}$$

We created a 8 * 64 key matrix to store the number of times a key $k \in [1, 64]$ satisfies the possibility of being a key to S_i box, where $i \in [1, 8]$.

We find the set

X_i = {(β, $\beta^{'}$) | $\beta \oplus \beta^{'} = \beta_i \oplus \beta_i^{'}$ and $S_i(\beta) \oplus S_i(\beta^{'}) = \gamma$}

Then for each $k \in [1, 64]$, we check whether

$$\alpha_i \oplus k = \beta \quad \text{and} \quad (\beta, \beta^{'}) \in X_i \text{ for some } \beta^{'}$$

If above condition is satisfied for S_i box, then we incremented key[i][k] by 1.

Result of the above fruitful analysis for the characteristic [40 08 00 00 04 00 00 00] is that we get partial key using {S2, S5, S6, S7, S8} as {45, 40, 28, 2, 20} as input to these s-boxes is 0 in round4.

Our similar repetition of above application of procedures used for cryptanalysis is done again for ciphertexts in ciphertexts2.txt.

Result of the above same analysis for the characteristic [00 20 00 08 00 00 04 00] is that we get partial key using {S1, S2, S4, S5, S6} as {47, 12, 53, 30, 0} as input to these s-boxes is 0 in round4.

Above Characteristics have {S2, S5, S6} in common, and

the key bits deduced from both are of these characteristics are same for the before mentioned s-boxes. Therefore, we have successfully found 42 bits out of total 56 bits of the key.

The 48-bit key for s-box is:

[101111001100XXXXXX1101010111100000001010100110 1]

7. Since the input to S3 was never zero, a "X" is put in its place. After turning this into a 56-bit key and using the key schedule PC2, we get

[X10XX1XX00110X111XX01X01110X1011011X01001100X11 X0000X000]

8 We use brute force, which means that we went through all 2^14 potential keys to find the missing bits. We gave the system "fghijklmnopqrstu" as plaintext input. We receive "gmnhfrfpihuttosj" as the answer. Then, for each potential key, we encrypted the plaintext with that key to see if we received the mentioned cypher. The desired key is the one that, when used with encryption and the mentioned cipher, gives the same result as the real key.

The key is:
{0,1,1,0,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,0,1,1,0,0, 1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,1,0,1,0,0,1,1}

9. To get the password we divide the cyphertext result of "password" which is "ijnjpqjlnqupllihsqpirkhlnoijjmqu" into two halves after first converting to binary and then to decimal because DES can only process 8 bytes of plaintext at once, to get {129, 180, 89, 118, 127, 75, 177, 224} and {221, 163, 243, 154, 102 ,150 ,194 ,9} where each block is 8 bytes and this is passed one at time in des.cpp. To get the password we divide the cyphertext result of "password" which is "ijnjpqjlnqupllihsqpirkhlnoijjmqu" into two halves after first converting to binary and then to decimal because DES can only process 8 bytes of plaintext at once, to get {129, 180, 89, 118, 127, 75, 177, 224} and {221, 163, 243, 154, 102 ,150 ,194 ,9} where each block is 8 bytes and

this is passed one at time in des.cpp. After decryption, we obtained "qtrojhbnib000000." When we entered this value, we got nothing, so we believed the "000000" at the end could be padding. We tried "qtrojhbnib" as the password and were able to clear the level.

We have added all the important codes as plain files, and also a compressed file "all files.zip" which contains all the plaintexts, ciphertexts, log files, scripts, c++ codes, python codes, etc.

References:
1. https://link.springer.com/content/pdf/10.1007/3-540-38424-3_1.pdf
2. https://medium.com/lotus-fruit/breaking-des-using-differential-cryptanalysis-958e8118ff41#:~:text=Differential%20cryptanalysis%20is%20a%20method,locate%20the%20most%20probable%20key
3. http://desimplementation.blogspot.com/2015/09/data-encryption-standard-algorithm-data.html

Q5 Password
5 Points

What was the password used to clear this level?

qtrojhbnib

Q6 Code
0 Points

Please add your code here. It is MANDATORY.

constants.py Download

```
1  # Expansion E-box Table
2  E = [32, 1 , 2 , 3 , 4 , 5 , 4 , 5,
3           6 , 7 , 8 , 9 , 8 , 9 , 10, 11,
4           12, 13, 12, 13, 14, 15, 16, 17,
```

```
5           16, 17, 18, 19, 20, 21, 20, 21,
6           22, 23, 24, 25, 24, 25, 26, 27,
7           28, 29, 28, 29, 30, 31, 32, 1 ]
8
9   #Inverse of P
10  INVP = [9, 17, 23, 31,
11                13, 28,  2, 18,
12                24, 16, 30,  6,
13                26, 20, 10,  1,
14                 8, 14, 25,  3,
15                 4, 29, 11, 19,
16                32, 12, 22,  7,
17                 5, 27, 15, 21
18          ]
19
20  # S-box Table
21  sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3 , 10, 6, 12, 5,
    9, 0, 7],
22      [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3,
    8],
23      [4, 1 , 14, 8, 13, 6, 2, 11, 15, 12, 9, 7,3, 10, 5,
    0],
24      [15, 12, 8,2,4, 9, 1,7 , 5, 11, 3, 14, 10, 0, 6,
    13]],
25
26      [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0,5,
    10],
27      [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11,
    5],
28      [0, 14, 7, 11, 10, 4, 13, 1, 5, 8,12, 6, 9, 3, 2,
    15],
29      [13, 8, 10, 1, 3, 15, 4, 2,11,6, 7, 12, 0,5, 14,
    9]],
30
31      [[10, 0, 9,14,6,3,15,5, 1, 13, 12, 7, 11, 4,2,8],
32      [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15,
    1],
33      [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12,5, 10, 14,
    7],
34      [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2,
    12]],
35
36      [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4,
    15],
37      [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14,
    9],
38      [10, 6, 9, 0, 12, 11, 7, 13, 15, 1 , 3, 14, 5, 2,
    8, 4],
39      [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2,
    14]],
40
```

```
    [[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]]]



f2u_mapping={'f': '0000',
             'g': '0001',
             'h': '0010',
             'i': '0011',
             'j': '0100',
             'k': '0101',
             'l': '0110',
             'm': '0111',
             'n': '1000',
             'o': '1001',
             'p': '1010',
             'q': '1011',
             'r': '1100',
```

```
77                's': '1101',
78                't': '1110',
79                'u': '1111'}
80
81  ## Key Schedule
82  PC2 = [
83    14, 17, 11, 24,  1, 5,
84    3, 28 ,15,  6, 21, 10,
85    23, 19, 12,  4, 26, 8,
86    16,  7, 27, 20, 13, 2,
87    41, 52, 31, 37, 47, 55,
88    30, 40, 51, 45, 33, 48,
89    44, 49, 39, 56, 34, 53,
90    46, 42, 50, 36, 29, 32
91  ]
92
93  ## Reverse Final Permutation
94  RFP = [57, 49, 41, 33, 25, 17, 9,  1,
95         59, 51, 43, 35, 27, 19, 11, 3,
96         61, 53, 45, 37, 29, 21, 13, 5,
97         63, 55, 47, 39, 31, 23, 15, 7,
98         58, 50, 42, 34, 26, 18, 10, 2,
99         60, 52, 44, 36, 28, 20, 12, 4,
100        62, 54, 46, 38, 30, 22, 14, 6,
101        64, 56, 48, 40, 32, 24, 16, 8]
102
103 #shift table
104 shift_table = [1, 1, 2, 2,
105                2, 2, 2, 2,
106                1, 2, 2, 2,
107                2, 2, 2, 1 ]
108
109 # Key- Compression Table : Compression of key from 56
    bits to 48 bits
110 key_comp = [14, 17, 11, 24, 1, 5,
111             3, 28, 15, 6, 21, 10,
112             23, 19, 12, 4, 26, 8,
113             16, 7, 27, 20, 13, 2,
114             41, 52, 31, 37, 47, 55,
115             30, 40, 51, 45, 33, 48,
116             44, 49, 39, 56, 34, 53,
117             46, 42, 50, 36, 29, 32 ]
118
119
120 map_values = {
121             'f' : [0,0,0,0],
122             'g' : [0,0,0,1],
123             'h' : [0,0,1,0],
124             'i' : [0,0,1,1],
125             'j' : [0,1,0,0],
126             'k' : [0,1,0,1],
127             'l' : [0,1,1,0],
```

```
            'm' : [0,1,1,1],
            'n' : [1,0,0,0],
            'o' : [1,0,0,1],
            'p' : [1,0,1,0],
            'q' : [1,0,1,1],
            'r' : [1,1,0,0],
            's' : [1,1,0,1],
            't' : [1,1,1,0],
            'u' : [1,1,1,1]
          }

# Permutation at start of DES
initial_perm = [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]

## S Box Permutation Table
sboxper = [ 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 ]

## Permutation at the end of DES
final_perm = [ 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 ]
plain_text1='fghijklmnopqrstu'
cipher_text2='gmnhfrfpihuttosj'
```

cryptoanalysis.ipynb Download

In [2]:
```
import numpy as np
```

In [3]:
```
from constants import
E,INVP,sbox,f2u_mapping,PC2,RFP,shif
```

```
from functions import
hexCipher,ITFP,Xor_cipher,Expanded,X
```

In [4]:

```
#Computing XORs at IN and OUT of S-b
for characteristic 40 08 00 00 04 00


cipher_t =
open('ciphertext01.txt').read().spli

hexcipher = hexCipher(cipher_t,map_v


##Inverse the final permutation
invres = ITFP(hexcipher,RFP)

##Xoring the ciphertext pairs accord
differential
resxor = Xor_cipher(invres)

##Expanding Right side block of Roun
expanded = Expanded(invres,E)

##Xoring expanded to compute s box i
sin = Xor_e_sbox(expanded)

##Xoring L5 and R6
L5 = [0,0,0,0,0,1]+[0]*26
inxor = xor_l5_l6(resxor,L5)

##Xoring sbox outputs
sxor = Xor_sbox(inxor,INVP)

##Finding the Keys  corresponding to
6 for above charcateristic
keys = keys_6(sin,sxor,expanded,sbox

print(keys)
```

```
[[646. 558. 666. 612. 627. 645. 619.
  625. 627. 632. 600. 585. 627. 597.
  681. 635. 589. 634. 597. 605. 618.
  671. 625. 666. 615. 631. 642. 640.
  648. 648. 628. 640. 619. 649. 590.
 [639. 636. 659. 638. 672. 640. 630.
  597. 640. 648. 663. 622. 610. 618.
  606. 601. 635. 632. 629. 668. 655.
  637. 591. 615. 686. 643. 596. 639.
  631. 677. 636. 633. 660. 626. 635.
 [665. 641. 605. 602. 620. 599. 586.
```

```
 625. 592. 647. 615. 592. 625. 610.
 623. 690. 623. 640. 629. 629. 626.
 619. 619. 595. 591. 608. 601. 620.
 611. 596. 631. 632. 643. 634. 608.
[629. 625. 576. 632. 661. 614. 613.
 631. 653. 624. 618. 656. 636. 669.
 624. 638. 646. 635. 620. 657. 643.
 659. 598. 601. 628. 606. 670. 641.
 602. 594. 619. 606. 627. 668. 580.
[593. 605. 576. 660. 624. 579. 654.
 648. 620. 648. 623. 627. 627. 622.
 644. 598. 600. 615. 630. 678. 577.
 625. 598. 635. 669. 630. 600. 612.
 636. 633. 625. 644. 604. 647. 634.
[652. 606. 610. 635. 642. 624. 625.
 660. 637. 585. 596. 631. 591. 593.
 686. 645. 578. 661. 637. 650. 634.
 662. 656. 616. 568. 591. 647. 645.
 600. 657. 609. 627. 642. 622. 608.
[615. 604. 674. 635. 634. 588. 642.
 631. 636. 605. 613. 615. 600. 631.
 617. 606. 622. 598. 595. 636. 609.
 631. 673. 644. 609. 624. 615. 623.
 610. 630. 590. 622. 617. 602. 591.
[634. 597. 637. 642. 676. 679. 615.
 617. 637. 632. 631. 663. 629. 684.
 645. 638. 639. 605. 628. 656. 670.
 645. 619. 600. 593. 652. 644. 595.
 654. 614. 641. 633. 635. 626. 636.
```

In [5]:

```
max_v,mean_v,key_v =
max_mean_key_value(keys)

#[max,mean,key] value for each
block
for i in range(0,8):
  print("S"+ str(i+1) +"  ["+
str(max_v[i]) + "," +
str(mean_v[i]) + "] key = " +
str(key_v[i]))
```

```
S1  [681.0,622] key = 28
S2  [686.0,636] key = 45
S3  [690.0,618] key = 29
S4  [670.0,623] key = 47
S5  [700.0,624] key = 40
S6  [686.0,623] key = 28
S7  [674.0,619] key = 2
S8  [684.0,636] key = 20
```

In [6]:

```
cipher_t =
open('ciphertext02.txt').read().spli

hexcipher = hexCipher(cipher_t,map_v

##Inverse the final permutation
invres = ITFP(hexcipher,RFP)

##Xoring the ciphertext pairs accord
differential
resxor = Xor_cipher(invres)

##Expanding Right side block of Roun
expanded = Expanded(invres,E)

##Xoring expanded to compute s box i
sin = Xor_e_sbox(expanded)

##Xoring L5 and R6
L5 =
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
inxor = xor_l5_l6(resxor,L5)

##Xoring sbox outputs
sxor = Xor_sbox(inxor,INVP)

##Finding the Keys  corresponding to
for above charcateristic
keys = keys_6(sin,sxor,expanded,sbox

print(keys)
```

```
[[647. 670. 613. 665. 617. 623. 545.
  630. 644. 571. 661. 647. 626. 616.
  665. 611. 646. 641. 623. 635. 616.
  644. 601. 647. 624. 641. 678. 654.
  665. 632. 611. 631. 612. 644. 625.
 [664. 618. 615. 615. 613. 652. 645.
  638. 632. 643. 641. 631. 638. 617.
  615. 627. 571. 648. 660. 635. 633.
  590. 646. 628. 589. 632. 637. 639.
  655. 619. 602. 622. 641. 634. 619.
 [657. 679. 598. 589. 577. 586. 617.
  614. 598. 608. 638. 593. 621. 614.
  628. 589. 609. 627. 603. 628. 570.
  621. 618. 621. 583. 650. 622. 612.
  645. 650. 625. 625. 603. 588. 621.
 [660. 589. 613. 661. 615. 614. 589.
  631. 640. 611. 638. 636. 595. 642.
  639. 606. 657. 625. 661. 641. 651.
```

```
 638. 629. 620. 597. 615. 661. 618.
 633. 642. 573. 601. 633. 604. 627.
[642. 622. 634. 621. 605. 614. 639.
 648. 636. 596. 654. 622. 621. 637.
 612. 619. 673. 627. 634. 643. 594.
 657. 574. 626. 597. 628. 595. 581.
 640. 608. 594. 619. 623. 606. 652.
[678. 669. 615. 574. 669. 625. 633.
 591. 618. 642. 655. 656. 670. 619.
 659. 575. 665. 643. 659. 638. 648.
 626. 653. 614. 644. 623. 607. 598.
 629. 602. 644. 666. 648. 660. 611.
[647. 631. 649. 667. 628. 642. 672.
 638. 645. 598. 622. 619. 648. 657.
 681. 631. 596. 612. 603. 643. 595.
 635. 685. 641. 629. 631. 662. 584.
 610. 655. 624. 630. 643. 595. 673.
[636. 569. 643. 643. 607. 640. 633.
 604. 596. 620. 620. 644. 638. 563.
 616. 661. 630. 634. 606. 638. 655.
 582. 634. 646. 615. 621. 621. 639.
 604. 610. 627. 595. 644. 617. 629.
```

In [7]:

```
maxval,mean,keyval =
max_mean_key_value(keys)


#[max,mean] key =  value for each
block
for i in range(0,8):
  print("S"+ str(i+1) +"  ["+
str(max_v[i]) + "," +
str(mean_v[i]) + "] key = " +
str(key_v[i]))
```

```
S1  [681.0,622] key = 28
S2  [686.0,636] key = 45
S3  [690.0,618] key = 29
S4  [670.0,623] key = 47
S5  [700.0,624] key = 40
S6  [686.0,623] key = 28
S7  [674.0,619] key = 2
S8  [684.0,636] key = 20
```

In [8]:

```
## to find the Key

sbkey =
"011100101101XXXXXX10111110111101110
##Obtained by converting key value t
```

```
for each sbox 1 to 8 except 3 for wh
'XXXXXX' is appended as input to sbo
never 0

key = find_key(PC2,sbkey,shift_table

print(''.join(key))
miskey = ''.join(key)
```

```
X10XX1XX10011X010XX11X10111X1000100X
```

Using Brute Force Method to find mising 14 bits of key

In [9]:

```
poskey = []
binlist = []
for i in range(2**14):
  x = str(bin(i)[2:])
  binlist.append(('0')*(26-
len(x))+x)

for i in binlist:
  j = 0
  tempkey = list(miskey)
  for k in range(len(tempkey)):
    if tempkey[k] == 'X':
      tempkey[k] = i[j]
      j+=1
  poskey.append(''.join(tempkey))
```

In [11]:

```
## Bruteforce to find the key

mainkey = ''
binplain = ''
for i in 'fghijklmnopqrstu':
  binplain+= f2u_mapping[i]

bincipher = ''
for i in "gmnhfrfpihuttosj":
  bincipher+=f2u_mapping[i]

for k in poskey:
  key = roundkey(k,6,shift_table,key

if(encryption(binplain,key,6,initial
E, sbox, sboxper, final_perm)==
bincipher):
```

```
    mainkey = k
    print('The key is',k,'\n')
    for i in range(6):
      print('Round',i,'key is',key[i

    break
```

```
The key is 011011100101111001111011O
```

In [12]:

```
## seperate key into comma
seperated

for i in mainkey:
    print(i+',',end='')
```

```
0,1,1,0,1,1,1,0,0,1,0,1,1,1,1,0,0,1,
```

In [13]:

```
## Convert main cipher text to decim

password = 'ijnjpqjlnqupllihsqpirkhl

for i in range(0,len(password),2):
    a =
f2u_mapping[password[i]]+f2u_mapping
    b = int(a,2)
    print(password[i:i+2],b)
```

```
ij 52
nj 132
pq 171
jl 70
nq 139
up 250
ll 102
ih 50
sq 219
pi 163
rk 197
hl 38
no 137
ij 52
jm 71
qu 191
```

In [ ]:

des.cpp Download

```
#define BYTE unsigned char
#define INT unsigned int
#include<iostream>
#include<fstream>
#include <curses.h>
#include<string>
#include<math.h>
using namespace std;

/*********************************************************
 *UNPACK8()  Unpack 8 bytes at 8bits/byte into 64 bytes at 1 bit/byte

*********************************************************/

void unpack8(BYTE *packed,BYTE *binary)
    /* BYTE *packed;
     BYTE *binary;*/
{
  register INT i, j, k;

  for (i=0; i<8; i++) {
    k = *packed++;
    for (j=0; j<8;j++) *binary++ = (k>>(7-j)) &01 ;
  }
}


/*********************************************************
 *PACK8() Pack 64 bytes at 1 bits/byte into 8 bytes at 8 bit/byte

*********************************************************/

void pack8(BYTE *packed,BYTE *binary)
   /*  BYTE *packed;
     BYTE *binary;*/
{
  register INT i, j, k;

  for (i=0; i<8; i++) {
    k = 0;
    for (j=0; j<8;j++) k  = (k<<1)+ *binary++;
    *packed++ = k;
  }
}

INT S[8][64]=
```

```
{
  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
};



unsigned short shifts[] = {
1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
```

```
94  };
95
96  /* PERMUTED CHOICE 2 (PC@) */
97
98  INT PC2[] = {
99    14, 17, 11, 24,  1, 5,
100   3, 28 ,15,  6, 21, 10,
101   23, 19, 12,  4, 26, 8,
102   16,  7, 27, 20, 13, 2,
103   41, 52, 31, 37, 47, 55,
104   30, 40, 51, 45, 33, 48,
105   44, 49, 39, 56, 34, 53,
106   46, 42, 50, 36, 29, 32
107 };
108
109 /* Key Scedule of 16 498-bit subkey generated from 64-
    bit key */
110
111 BYTE KS[16][48];
112
113
114 void set_the_key(int sw1,BYTE *CD,int r)
115      /*INT sw1;   */  /* type of cryption 0=
    encryption, 1= decryption */
116    /*  BYTE *pkey;
117      INT r;*/
118 {
119   register INT i, j, k, t1, t2;
120   int z;
121   /* Rotate and permute C and D to generate 16 subkeys
    */
122   for ( i=0; i<r; i++) { /**--*/
123     /* Rotate C and D */
124     for (j =0; j <shifts[i]; j++) {
125       t1 = CD[0];
126       t2 = CD[28];
127       for ( k=0; k<27; k++) {
128         CD[k] = CD[k+1];
129         CD[k+28] = CD[k+29];
130       }
131       CD[27] = t1;
132       CD[55] = t2;
133     }
134     /* Set the order of subkeys for type of encryption
    */
135     j = sw1 ? r-1-i :i ;   /**--*/
136
137     /* Permute C and D with PC2 to generate KS[i] */
138     for (k=0; k< 48 ; k++)  KS[j][k] = CD[PC2[k] -1];
139   //  cout<<'\n';
140     //;
141   }
```

```
142 /*for(z=0;z<48;z++)
143         cout<<int(KS[5][z]);
144     getch();*/
145   return;
146 
147 }
148 
149 
150 
151 
152 /*********************************************************
153                         DES
154 *********************************************************
155 /* INITIAL PERMUTATION (IP) */
156 
157 INT IP[] = {
158   58,50,42, 34,26,18,10,2,
159   60,52,44,36,28,20,12,4,
160   62,54, 46, 38, 30, 22, 14,6,
161   64, 56, 48, 40,32,24, 16, 8,
162   57, 49, 41, 33,25,17, 9,1,
163   59, 51,43,35,27,19,11,3,
164   61,53,45,37,29,21,13, 5,
165   63,55, 47,39,31,23,15,7
166 };
167 int invip[]={
168 40,8,48,16,56,24,64,32,
169 39,7,47,15,55,23,63,31,
170 38,6,46,14,54,22,62,30,
171 37,5,45,13,53,21,61,29,
172 36,4,44,12,52,20,60,28,
173 35,3,43,11,51,19,59,27,
174 34,2,42,10,50,18,58,26,
175 33,1,41,9,49,17,57,25};
176 
177 /* REVERSE PERMUTATION (RFP) */
178 
179 INT RFP[] = {
180   8,40,16,48,24,56,32,64,
181   7, 39,15,47,23,55,31,63,
182   6,38,14,46,22,54,30,62,
183   5,37,13,45, 21,53,29,61,
184   4,36,12,44,20,52,28,60,
185   3, 35, 11,43,19,51,27,59,
186   2, 34, 10, 42,18, 50,26,58,
187   1,33,9,41, 17, 49, 25,57,
188 };
189 
190 /* E BIT_SELECTION TABLE */
191 
192 INT E[] = {
193   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
};


/* PERMUTATION FUNCTION P */
INT 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,
};


/* Inverse of P */
INT INV_P[] = {
        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,
};


int invrfp[]= {
 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
};


void des(BYTE *in,BYTE *out,INT r,char flag)
    /* BYTE *in;              /* packed 64 bit Input
block */
```

```
244     // BYTE *out;              /* packed 64 bit output
    block */
245     // INT r;                                  /*
    number of rounds */
246         // char flag;
247 {
248   register INT i, j, k, t;
249   static BYTE block[64]; /* unpacked 64-bit
    input/output block */
250   static BYTE LR[64], f[32], preS[48];
251
252   /*
253     in[9] ='\0';
254   */
255
256   /* Unpack the INPUT block */
257 int z;
258   unpack8(in, block);
259  /* for(z=0;z<64;z++)
260     cout<<int(block[z]);
261   getch();
262   /* Permute unpacked input block with IP to generate L
    and R */
263   /*
264     printf("\nThe block after the initial permutation
    IP \n");
265   */
266       for (j =0; j<64 ; j++)
267        {
268                LR[j] = block[invrfp[j] -1];
269                /*printf("%d", LR[j]);*/
270        }
271
272        /* Perform r rounds */
273        /*
274        printf(" In des 3rd round expandes block is
    \n");
275        */
276
277        for (i=0; i<r; i++) {  /**--*/
278          /* expand R to 48 bits with E and XOR  with
    ith subkey */
279          for( j=0; j<48; j++) {
280            preS[j] = LR[E[j]-1]^KS[i][j];
281            /*
282              if( i==2)
283              {
284              printf("%d", LR[E[j]+31]);
285              }
286            */
287          }
288
```

```
289
290
291
292           /* Map 8 6-bit blocks into 8 4-bit bolcks
    using S-boxes */
293           for (j=0; j<8; j++) {
294             /* Compute index t into jth s box */
295
296             k= 6*j;
297             t= preS[k];
298             t = (t<<1) | preS[k+5];
299             t = (t<<1)| preS[k+1];
300             t = (t<<1)| preS[k+2];
301             t = (t<<1)| preS[k+3];
302             t = (t<<1)| preS[k+4];
303             /* fetch t th entry fron jth sbox */
304             t = S[j][t];
305             /* generate 4-bit block from s-box entry */
306             k= 4*j;
307             f[k] = (t>>3)&1;
308             f[k+1] = (t >> 2) & 1;
309             f[k+2] = (t >> 1) & 1;
310             f[k+3] = t &1;
311
312           }
313
314
315           for (j=0; j <32; j++) {
316             /* Copy R */
317             t = LR[j];
318             /* Permute Permute f w/ P and XOR w/ L to
    generate new R*/
319                 if (flag == 'N')
320                 LR[j] = LR[j+32]^f[P[j] -1];
321                 else
322                 LR[j] = LR[j+32]^f[INV_P[j] -1];
323             /*LR[j+32] = LR[j]^f[j];*/
324             /* copy original R to new L */
325             LR[j+32] =t;
326           }
327
328         }
329         /*
330           printf( "\n\n I am in Des , block before
    final RFP permute\n");
331           for(i=0; i<64; i++)
332           printf( "%d", LR[i]);
333         */
334         /* Permute L and R with reverse IP-1 to
    generate output block*/
335         for (j=0; j < 64; j++) block [j] =
    LR[invip[j]-1];
```

```
336
337         /* Pack data into 8 bits per byte */
338
339
340     pack8(out, block);
341
342           out[9] = '\0';
343
344 }
345
346 int main()
347 {
348     BYTE password[9];
349     BYTE key[56]=
    {0,1,1,0,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,
350     set_the_key(1,key,6);
351     BYTE encrypted_password1[9]=
    {129,180,89,118,127,75,177,224,'\0'};
352     BYTE encrypted_password2[9]={221, 163, 243, 154,
    102 ,150 ,194 ,9,'\0'};
353     des(encrypted_password1,password,6,'N');
354     for(int i=0;i<8;i++)
355     cout<<password[i];
356     des(encrypted_password2,password,6,'N');
357     for(int i=0;i<8;i++)
358     cout<<password[i];
359     return 0;
360 }
361
```

functions.py Download

```
1  import numpy as np
2
3  def hexCipher(ciphertext,mapping):
4      hex_cipher=[]
5      for i in ciphertext:
6
7          hex=[]
8          if len(i) == 16:
9              for j in range(16):
10                 hex.extend(mapping[i[j]])
11             hex_cipher.extend([hex])
12     return hex_cipher
13
14 def ITFP(hexcipher, RFP):
15     invres = []
16     for i in hexcipher:
17         temp = [i[RFP[j]-1] for j in range(64)]
18         invres.append(temp)
```

```
19      return invres
20
21  def Xor_cipher(invres):
22      resxor = [list(np.bitwise_xor(invres[2*i],
    invres[2*i+1])) for i in range(len(invres)//2)]
23      return resxor
24
25
26  def Expanded(invres, E):
27      expanded = []
28
29      for i in invres:
30          temp = [i[E[j]-1] for j in range(48)]
31          expanded.append(temp)
32
33      return expanded
34
35  def Xor_sbox(inxor, INVP):
36      sxor = []
37      for i in inxor:
38          temp = [i[INVP[j]-1] for j in range(32)]
39          sxor.append(temp)
40      return sxor
41
42
43  def Xor_e_sbox(expanded):
44      sin = [list(np.bitwise_xor(expanded[2*i],
    expanded[2*i+1])) for i in range(len(expanded)//2)]
45      return sin
46
47
48  def keys_6(sin,sxor,expanded,sbox):
49      keys = np.zeros((8,64))
50
51      for i in range(len(sin)):
52          if sin[i]=="":
53              continue
54          for j in range(0,8):
55              inx = int(''.join([str(s) for s in sin[i]
    [j*6:j*6+6]]),2)
56              outx = int(''.join([str(s) for s in sxor[i]
    [j*4:j*4+4]]),2)
57              inp = int(''.join([str(s) for s in
    expanded[2*i+1][j*6:j*6+6]]),2)
58
59              for k in range(0,64):
60                  a = bin(k)[2:].zfill(6)
61                  b = bin(k^inx)[2:].zfill(6)
62                  if outx == sbox[j][int(a[0])*2 +
    int(a[5])][int(a[4]) + 2 *int(a[3]) + int(a[2]) * 4 +
    int(a[1])*8]^sbox[j][int(b[0])*2 + int(b[5])][int(b[4])
    + 2 *int(b[3]) + int(b[2]) * 4 + int(b[1])*8]:
```

```
                    keys[j][k^inp]+=1
    return keys

def xor_l5_l6(resxor, L5):
    inxor = [list(np.bitwise_xor(i[32:64], L5)) for i
in resxor]
    return inxor

def max_mean_key_value(keys):
    mean_v = [int(np.mean(k)) for k in keys]
    max_v = [max(k) for k in keys]
    key_v = [np.where(k == max(k))[0][0] for k in keys]
    return max_v, mean_v, key_v

def find_key(PC2,sbkey,shift_table):
    key = ['X']*56
    for i in range(48):
       key[PC2[i]-1] = sbkey[i]
    for i in range(0,6):
        for j in range(shift_table[i]):
            x = key[27]
            y = key[55]
            for k in range(27,0,-1):
                key[k] = key[k-1]
                key[k+28] = key[k+27]
            key[0] = x
            key[28] = y
    return key


def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation

def shift(k, n):
    return k[n:] + k[0:n]


def roundkey(k,rno,shift_table,key_comp):

    binkey = []

    for i in range(rno):

        key = shift(k[0:28],shift_table[i]) +
shift(k[28:56],shift_table[i])
        binkey.extend([permute(key,key_comp,48)])

    return binkey

```

```
113
114 def
    encryption(mess,key,rno,initial_perm,E,sbox,sboxper,fin
115
116   mess = permute(mess,initial_perm,64)
117
118   left = mess[:32]
119   right = mess[32:]
120
121   for i in range(rno):
122
123     expmess = permute(right,E,48)
124     inxor =
    str(bin(np.bitwise_xor(int(expmess,2),int(key[i],2)))
    [2:])
125     if(len(inxor)!=48):
126       inxor= ('0'*(48-len(inxor)))+inxor
127     sout = ''
128     for j in range(8):
129       temp = (bin(sbox[j]
    [int(inxor[j*6]+inxor[j*6+5],2)]
    [int(inxor[j*6+1:j*6+5],2)])[2:])
130       sout+= ('0'*(4-len(temp))+temp)
131     sout = permute(sout,sboxper,32)
132
133     roundxor =
    str(bin(np.bitwise_xor(int(left,2),int(sout,2)))[2:])
134     if(len(roundxor)!=32):
135       roundxor= ('0'*(32-len(roundxor)))+roundxor
136     left = roundxor
137
138     if(i!=5):
139       t = left
140       left = right
141       right = t
142
143
144   outmess = left + right
145   cipher = permute(outmess,final_perm,64)
146
147   return cipher
148
```

▾ get_ciphertext.py Download

```
1 import re
2
3 #mark whenever this line appears and the next line
  contains required output
4
```

```
pattern = re.compile("Slowly, a new text starts appearing on the screen. It reads ...")
flagged = False

f = open("ciphertext01.txt", "w")

for line in open("outputs_log1.log"):
    if flagged:
        flagged = False
        f.write("{}\n".format(line.strip()))
    else:
        for match in re.finditer(pattern, line):
            if match:
                flagged = True

f.close()

f = open("ciphertext02.txt", "w")

for line in open("outputs_log2.log"):
    if flagged:
        flagged = False
        f.write("{}\n".format(line.strip()))
    else:
        for match in re.finditer(pattern, line):
            if match:
                flagged = True

f.close()
```

script_generator.cpp Download

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    //auto-generate a script1 which will send 1 lakh inputs to the ssh server
    std::ofstream script1;
    script1.open("script1.sh");
    //log file stores output of the game screen
    script1 << "log_file -a game_outputs.log\n";
    script1 << "spawn ssh student@172.27.26.188\n";
    string sshpwd, gpname, gppwd, infile;
    cout << "Enter your SSH Password : ";
    cin >> sshpwd;
    cout << "\nEnter your team name : ";
    cin >> gpname;
    cout << "\nEnter your team pwd : ";
    cin >> gppwd;
```

```
    script1 << "expect  \"student@172.27.26.188's password:\"\n";
    script1 << "send -- \"";
    script1 << sshpwd;
    script1 << "\\r\"\n\n";
    script1 << "expect  \"group name:\"\n";
    script1 << "send -- \"";
    script1 << gpname;
    script1 << "\\r\"\n\n";
    script1 << "expect  \"password:\"\n";
    script1 << "send -- \"";
    script1 << gppwd;
    script1 << "\\r\"\n\n";
    script1 << "expect  \"at:\"\n";
    script1 << "send -- \"4\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"back\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"enter\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"wave\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"back\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"back\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"thrnxxtzy\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"read\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"the_magic_of_wand\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"c\\r\"\n\n";
    script1 << "expect  \"> \"\n";
    script1 << "send -- \"read\\r\"\n\n";
    std::ifstream input_random1("plaintext1.txt");
    std::string line1;
    if (input_random1.is_open())
    {
        while (std::getline(input_random1, line1))
        {
            script1 << "expect  \"> \"\n";
            script1 << "send -- \"";
            script1 << line1;
            script1 << "\\r\"\n\n";
            script1 << "expect  \"> \"\n";
            script1 << "send -- \"c\\r\"\n\n";
        }
        input_random1.close();
    }
    script1.close();

```

```
69      std::ofstream script2;
70      script2.open("script2.sh");
71      //log file stores output of the game screen
72      script2<< "log_file -a game_outputs.log\n";
73      script2<< "spawn ssh student@172.27.26.188\n";
74      cout << "Enter your SSH Password : ";
75      cin >> sshpwd;
76      cout << "\nEnter your team name : ";
77      cin >> gpname;
78      cout << "\nEnter your team pwd : ";
79      cin >> gppwd;
80      script2<< "expect  \"student@172.27.26.188's
    password:\"\n";
81      script2<< "send -- \"";
82      script2<< sshpwd;
83      script2<< "\\r\"\n\n";
84      script2<< "expect  \"group name:\"\n";
85      script2<< "send -- \"";
86      script2<< gpname;
87      script2<< "\\r\"\n\n";
88      script2<< "expect  \"password:\"\n";
89      script2<< "send -- \"";
90      script2<< gppwd;
91      script2<< "\\r\"\n\n";
92      script2<< "expect  \"at:\"\n";
93      script2<< "send -- \"4\\r\"\n\n";
94      script2<< "expect  \"> \"\n";
95      script2<< "send -- \"back\\r\"\n\n";
96      script2<< "expect  \"> \"\n";
97      script2<< "send -- \"enter\\r\"\n\n";
98      script2<< "expect  \"> \"\n";
99      script2<< "send -- \"wave\\r\"\n\n";
100     script2<< "expect  \"> \"\n";
101     script2<< "send -- \"back\\r\"\n\n";
102     script2<< "expect  \"> \"\n";
103     script2<< "send -- \"back\\r\"\n\n";
104     script2<< "expect  \"> \"\n";
105     script2<< "send -- \"thrnxxtzy\\r\"\n\n";
106     script2<< "expect  \"> \"\n";
107     script2<< "send -- \"read\\r\"\n\n";
108     script2<< "expect  \"> \"\n";
109     script2<< "send -- \"the_magic_of_wand\\r\"\n\n";
110     script2<< "expect  \"> \"\n";
111     script2<< "send -- \"c\\r\"\n\n";
112     script2<< "expect  \"> \"\n";
113     script2<< "send -- \"read\\r\"\n\n";
114     std::ifstream input_random2("plaintext2.txt");
115     std::string line2;
116     if (input_random2.is_open())
117     {
118         while (std::getline(input_random2, line2))
119         {
```

```
120             script2<< "expect  \"> \"\n";
121             script2<< "send -- \"";
122             script2<< line2;
123             script2<< "\\r\"\n\n";
124             script2<< "expect  \"> \"\n";
125             script2<< "send -- \"c\\r\"\n\n";
126         }
127         input_random2.close();
128     }
129     script2.close();
130 
131 
132 }
```

plaingen.ipynb Download

In [9]:

```
import numpy as np
import random

hex_to_char = {'0000': 'f','0001': 'g',
'h','0011': 'i','0100': 'j','0101': 'k'
'l','0111': 'm','1000': 'n','1001': 'o'
'p','1011': 'q','1100': 'r','1101': 's'
't','1111': 'u'}
characters=
['f','g','h','i','j','k','l','m','n','c

char_to_hex = {x:y for y,x in
hex_to_char.items()}

def create_binplaintexts(binary,filenam



    XOR_v = list((bin(binary))[2:].zfil
    XOR_v = [int(x) for x in XOR_v]


    b_plain_texts = []
    for i in range(2500):
        tmp = np.random.choice(characte
(1,16),replace = True)[0]
        bininput=[]
        for i in tmp:
            bininput.extend([int(a) for
char_to_hex[i]])

        b_plain_texts.append(bininput)

b_plain_texts.append(list(np.bitwise_xo
```

```
    plaintexts = []

    for i in range(len(b_plain_texts)):
        s = ""
        for j in range(0,64,4):

            s+=hex_to_char[''.join([str
a in b_plain_texts[i][j:j+4]])]

        plaintexts+=[s]

    file = open(filename,"w")
    for plaintext in plaintexts:
        file.write(plaintext+"\n")
    file.close()
```

In [10]:

```
# XOR value between pairs of plaintexts
20 00 08 00 00 04 00
create_binplaintexts(0x0020000800000400
# XOR value between pairs of plaintexts
08 00 00 04 00 00 00
create_binplaintexts(0x4008000004000000
```

▾ all files.zip Download

1	Large file hidden. You can download it using the button above.

Assignment 4

● Graded

Group

MADHAV MAHESHWARI
RAJ KUMAR

GUNJ MEHUL HUNDIWALA

View or edit group

Total Points

100 / 100 pts

Question 1

Team Name — **0** / 0 pts

Question 2

Commands — **5** / 5 pts

Question 3

Cryptosystem — **10** / 10 pts

Question 4

Analysis — R **80** / 80 pts

Question 5

Password — **5** / 5 pts

Question 6

Code — **0** / 0 pts