Q1 Commands

0 Points

List the commands used in the game to reach the first ciphertext.

Various command, but after clearing all level , the command used were "Read"-> Final decrypted command

Q2 Cryptosystem

10 Points

What cryptosystem was used in this level?

6-round DES

Q3 Analysis

80 Points

What (mathematical and programming) tools, observations and cryptanalysis were used to figure out the cryptosystem and the password? (Explain in less than 1500 lines)

Despite all the difficulties, I was able to complete level 4 of the game by checking several websites and reading a few papers about breaking 6 round DES.

Please take note that I have attached all scripts and their output with my analysis.

Following are few articles/links which i referred during this game,
1. Code shared by TA-Gargi, this helped me to cross check my decryption code.
2. https://medium.com/@jnaman806/breaking-des-using-differential-cryptanalysis-958e8118ff41
3. https://tragoedia.weebly.com/6-round-des.html
4. https://github.com/ksevta/Break-DES-6-round-using-chosen-plain-text-attack
5. https://github.com/supriya363/break-des-6round/blob/master/README.md
6. http://koclab.cs.ucsb.edu/teaching/ccs130h/2016/des/dc1.pdf
7. https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/

Script and its purpose
1. generate_inputs.ipynb - To generate set of chosen text
2. GameServer-Fetchciphertext-Set-1.py/GameServer-Fetchciphertext-Set-2.py - Fetch cyphertext from game server corresponding to chosen plaintext.
3. Char-to-binaries-Decimal.py - convert Characters to Binaries and Decimal with provided mapping
4. CryptAnalysis.ipynb - Avrious Encryption Analysis such as Expansion,S-Box Keys , Master Key,Brute-force,Finding Actual Key and Keys for all round
5. decrypt.cpp - to decrypt final code.

I was able to see the Level 4 console message after using

```
the correct command to clear the previous all screens,
"The rumbling sound is very loud here. It is coming from
your right side. A cold blast of air hits you sending
shivers up your spine. You look in that direction.
There is a large opening on the right from where the
sound and the air is coming from. There is a fair amount
of light also coming from that direction (you realize that
you have not lighted a matchstick and still you can see).
There is another door, with a panel nearby, to your left
which is closed. The chamber is rocky and cold. Another
blast of air hits you from your right and you shiver again."

>> Read

"You come up to the closed door and look at the glass panel ...
        ... there is nothing written on it!!

        As you wonder what is happening here, you hear the spirit whispering
        in your ears ...

        "This is a magical screen. You can whisper something close to the
        screen and the corresponding coded text would appear on it after a while.
        So go ahead and try to break the code! The code used for this is
        a 4-round DES, so it should be easy for you!! Er wait ... maybe it is
        a 6-round DES ... sorry, my memory has blurred after so many years.
        But I am sure you can break even 6-round DES easily. A 10-round DES
        is a different matter, but this one surely is not 10-round ...
        (long pause) ... at least that is what I remember. One thing that I
        surely remember is that you can see the coded password by whispering 'password'.
        There was something funny about how the text appears, two letters for one
        byte or something like that. I do not recall more than
```

that.

I am sure you can figure it out though ..."

I typed in the password and it displayed the ciphertext, "rjugkhhullgusorklsklholnruogslhn," which needs to be deciphered in order to pass this level..

The text above strongly suggests that I need to use 6-round DES to clear this level., Even after double checking with the previous batch assignment, which only contained three round DES, the wording of the preceding paragraph strongly suggests that I need to use 6-round DES to break this function.

I have made the decision to use the chosen plain text attack to crack the 6-round DES (thinking like an ELA, thanks to Prof. Mahindra Sir's lecture videos).

In order to perform cryptanalysis, I created a Python script called GameServer-Fetchciphertext-Set-1.py. This script connects to our game server and retrieves the cyphertext that matches the selected plaintext (which is produced by the script generate_inputs.ipynb). I then use these pairs of ciphertexts and plaintexts to determine the encryption key used in level 6 game.

IP(M): The initial permutation.
IP_INV (M) - This is applied after all 6 rounds of DES are done on message M.
E (M) - Expand 32-bits of text M to 48-bits.
P (M) - This step permutes the 32-bit input M.
S - There are 8 S-boxes. Each S-box has 6-bit input and a 4-bit output.
PC1 - Key permutation that maps 64 bits of key to 56 bits and removes the parity bits
Shift - Shift that is performed on the key obtained as output of PC1
PC2 - Key permutation that maps 56 bits of Shift’s output to 48 bits

TTP(Tactics/Tools, Techniques & Procedures):
-> I utilized chosen-plaintext attack to cryptanalyze 6-round DES and performed differential cryptanalysis utilizing two

3-round features.
The 40080000 04000000 and 00200008 00000400 characteristics are employed.

-> The game's displayed message made it very clear that one byte contains two characters; so, one character is represented by four bits. Since I can only represent 16 characters with 4 bits, I attempted a number of plaintexts and compared the ciphertexts to see which 16 characters were actually utilized in the game. I discovered after dissecting the ciphertexts (ciphertexts1-MD--20240606.txt) that were retrieved from the game server , moreover, i found that: The game uses the alphabets f through u.

I did mapping of letters f-u to 0-15 respectively:

```
  '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'
```

- One DES block has an input and output size of 64 bits, or 8 bytes (the block size), or 16 letters. I therefore made the decision to work on size 16 letter plaintexts.

Step 1: Generation of Plaintext Pairs
I created about 1000 pairs of plaintext-cipher text pairs using the differential iterative characteristic of the 6-round DES 40 08 00 00 04 00 00 00 with probability 1/16 and 00 20 00 08 00 00 04 00 with probability 1/16.

Using inverse initial permutation on the characteristic 40

08 00 00 04 00 00 00, the first plaintext pairs (plaintexts1-MD-20240606.txt) are generated with an XOR of 00 00 80 10 00 00 40 00. The second 1000 pairs of plaintext (plaintexts2-MD-20240606.txt) have an XOR of 00 00 08 01 00 10 00 00, which is obtained by applying inverse initial permutation on the characteristic 00 20 00 08 00 00 04 00. Plaintexts1-MD-20240606.txt and plaintexts2-MD-20240606.txt, respectively, are the files containing these inputs.

I then used plaintexts1-MD-20240606.txt and plaintexts2-MD-20240606.txt to run two different Python scripts (GameServer-Fetchciphertext-Set-1.py & GameServer-Fetchciphertext-Set-2) to obtain the matching cyphertext for each of the 2000 selected plaintext pairs.

Step 2: Finding the key bits of round key K6
Steps 2.1 to 2.4 were carried out for the ciphertexts obtained corresponding to each of the two characteristics.

2.1 : I converted the obtained ciphertext to binary using the above-mentioned character mapping and then used CryptAnalysis.ipynb to apply reverse final permutation on these binary ciphertexts to get (L6R6) and (L'6R'6), which are the outputs of the 6th round of DES.
We know that R5=L6, thus I calculated the output of the Expansion box (exp_out1-MD-20240606.txt & exp_out2-MD-20240606.txt) and the input XOR of the S-boxes (sbox_in1-MD-20240606.txt & sbox_in2-MD-20240606.txt) for the sixth round using the values R5 and R'5.

2.2 : For the first characteristic mentioned above, L5 =04000000 and for the second characteristic L5 =00000400. We found output of permutation box by performing L5 ⊕ (R6 ⊕ R'6), then i applied inverse permutation on this value to obtain output XOR of S-boxes(sbox_out1-MD-20240606.txt & sbox_out2-MD-20240606.txt) for 6th round.

2.3 : Let E(R5)= α1α2....α8 and E(R'5)=α'1α'2....α'8 and βi =αi ⊕ k6,i and β'i =α'i ⊕k6,i

Where |αi| = 6= |α'i | and k6 =k6,1 k6,2 ···k6,8.

At this point, we know αi,α'i, βi ⊕β'i and γi ⊕γ'i. therefore, i created a 8 * 648∗64 key matrix to store the number of times a key $k \in[1,64]$ satisfies the possibility of being a key to Si box, where $i\in[1,8]$.

2.4 : I calculated the sets $S(\beta)\oplus S(\beta')=\gamma i \oplus \gamma i'$ and $Xi =(\beta,\beta')|\beta\oplus\beta' =\beta i \oplus\beta i'$.
Next, we discovered the key k, such that for each β′, $(\beta,\beta')\in Xi$ and $\alpha i \oplus k=\beta$. I also increased the count of each key k that met this requirement for the Si box in the key matrix, key_matrix[i][k].

- After performing the above analysis to find the keys, I obtained the below results for characteristic 40 08 00 00 04 00 00 004008000004000000:

S-box	Max	Mean	Key
S1	144	70	45
S2	316	84	51
S3	126	66	37
S4	110	65	7
S5	153	73	59
S6	312	78	16
S7	195	73	11
S8	188	74	46

=>Key

S1	S2	S3	S4	S5	S6	S7	S8
45	51	37	7	59	16	11	46

- and the following results for characteristic 0020000800000400:

S-box	Max	Mean	Key
S1	187	70	45
S2	178	77	51
S3	128	65	37
S4	310	82	7
S5	178	68	59
S6	308	74	16
S7	109	66	11
S8	105	64	46

The computed key values corresponding to S1, S2, S4, S5, S6, S7 and S8 as 45, 51, 7 , 59, 16, 11 and 46 are the same using both the characteristics as well as with Max

Frequency & Mean Frequency values for round key K6.

Now I have 42 bits(101101110011000111111011010000001011101110) of the 56-bit key.
S1: 101101
S2: 110011
S4: 000111
S5: 111011
S6: 010000
S7: 001011
S8: 101110

Step 3: Find the Actual Key from 42 known bits
Next, I applied key scheduling algorithm to obtain the actual positions of these known 42 bits in the 56 bit key and the outcome was as follows::

Key guessed(The Master Key) after analyzing using 2 characteristics(CryptAnalysis.ipynb):
X11XX1XX01011X100XX11X11000X1001011X10101001X10X1110X001

A key scheduling algorithm can be used to determine their placements. By employing brute force, the last 14 bits of the key can be located.

For that, I utilized CryptAnalysis.upynb and brute force to extract the final 14 bits of the key, I verified if an input is encrypted in the same way by running it through 2^14 potential key permutations. That's the key, if it is.

Given input : ffffffffffffffff {0,0,0,0,0,0,0,0}
Given output : ojopnpgqhiljlffl

Key :
01101110010111100111101100001001011110101001010111100001

Now I have actual key, After obtaining the 56 bit key, I found the 48 bit round key for each round.

Round 1
111011000100111100000111001011010111010110001100
Round 2
011011110011011101100010000111100011111010110001
Round 3
111010101101010011101101011110110110100101110001
Round 4
110110011100001101011010001000111110100100011010
Round 5
001001001101101110111011111001010011010100010110
Round 6
101101110011100101000111111011010000001011101110

```
Binary Pair  |  Decimal Value  |  Corresponding Characters
------------------------------------------------------
11000100  |  196  |  rj
11110001  |  241  |  ug
01010010  |  82  |  kh
00101111  |  47  |  hu
01100110  |  102  |  ll
00011111  |  31  |  gu
11011001  |  217  |  so
11000101  |  197  |  rk
01101101  |  109  |  ls
01010110  |  86  |  kl
00101001  |  41  |  ho
01101000  |  104  |  ln
11001111  |  207  |  ru
10010001  |  145  |  og
11010110  |  214  |  sl
00101000  |  40  |  hn
```

Step 5: Decryption of Passcode to clear level 4
-The ciphertext corresponding to our password is "rjugkhhullgusorklsklholnruogslhn" and its corresponding Binary Pair and Decimal value(use script Char-to-binaries-Decimal.py)

```
Binary Pair  |  Decimal Value  |  Corresponding Characters
------------------------------------------------------
11000100  |  196  |  rj
11110001  |  241  |  ug
01010010  |  82  |  kh
```

00101111	47	hu
01100110	102	ll
00011111	31	gu
11011001	217	so
11000101	197	rk
01101101	109	ls
01010110	86	kl
00101001	41	ho
01101000	104	ln
11001111	207	ru
10010001	145	og
11010110	214	sl
00101000	40	hn

Consequently, I used decryption (decrypt.cpp) on this ciphertext to get the password. There are 32 characters in this ciphertext. Since there are four bits in each character, this is a 128-bit string, or two blocks of DES ciphertext.
This is {196,241,82,47,102,31,217,197, 109, 86, 41, 104, 207, 145,214, 40} according to our mapping.

- With our actual key now in hand, I decrypted the ciphertext using the DES implementation's six-round decryption method, taking into account 16 characters (=64 bits).
Then i rann decrypt.cpp with appropriate input as mentioned below and decrypted the password using the key.
Actual Key=
[01101110010111100111101100001001011110101001010111100001]
Encrypted Password : rjugkhhullgusorklsklholnruogslhn
{196,241,82,47,102,31,217,197, 109, 86, 41, 104, 207 ,145 ,214 ,40}

The decrypt.cpp output the decrypted text as (I ran all code on Kali Linux):

```
┌──(root㉿kali)-[/media/.../CS961/HW/4/4.DES-6]
└─# ./decrypt
m q t j g u h e i t 0 0 0 0 0 0
```

I removed the zeroes as they must have been used for padding.

When I finally entered the decoded word "mqtjguheit" into the game, I was taken to a screen that stated I had to take a breather in order to advance to the next level:) " Take a break. It is too early to go to the next level.".

Hurrah. I just broke the 6-round DES, and i can see only my name on the wall : mjawed23 :)

Q4 Password

10 Points

What was the final command used to clear this level?

mqtjguheit

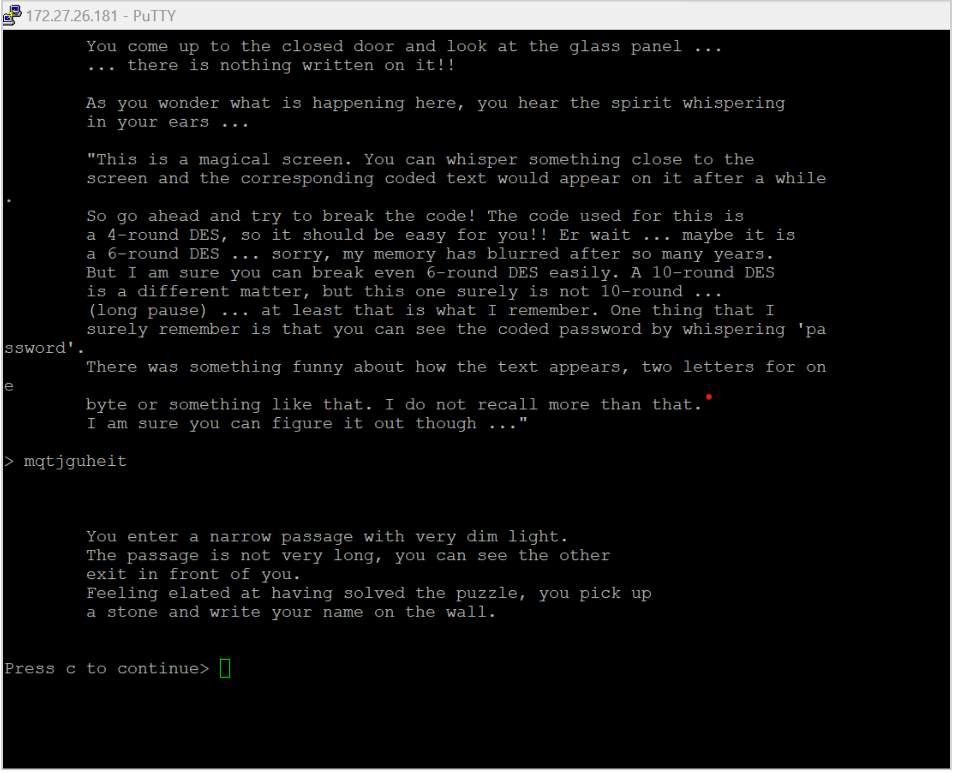
Q5 Code & figures

0 Points

Upload any codes that you have used to solve this level. You may also upload figures (only).

▾ Solved-1.png

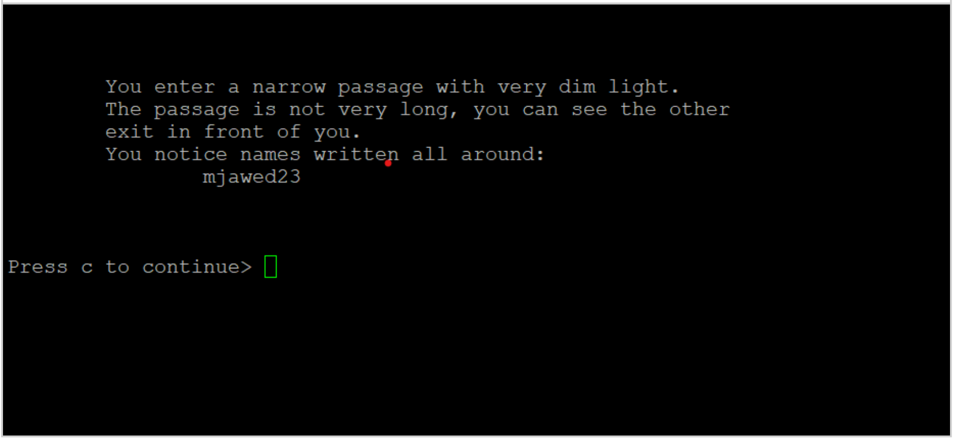
Download



```
        You come up to the closed door and look at the glass panel ...
        ... there is nothing written on it!!

        As you wonder what is happening here, you hear the spirit whispering
        in your ears ...

        "This is a magical screen. You can whisper something close to the
        screen and the corresponding coded text would appear on it after a while
.
        So go ahead and try to break the code! The code used for this is
        a 4-round DES, so it should be easy for you!! Er wait ... maybe it is
        a 6-round DES ... sorry, my memory has blurred after so many years.
        But I am sure you can break even 6-round DES easily. A 10-round DES
        is a different matter, but this one surely is not 10-round ...
        (long pause) ... at least that is what I remember. One thing that I
        surely remember is that you can see the coded password by whispering 'pa
ssword'.
        There was something funny about how the text appears, two letters for on
e
        byte or something like that. I do not recall more than that.
        I am sure you can figure it out though ..."

> mqtjguheit



        You enter a narrow passage with very dim light.
        The passage is not very long, you can see the other
        exit in front of you.
        Feeling elated at having solved the puzzle, you pick up
        a stone and write your name on the wall.


Press c to continue>
```

▾ Final-Name.png

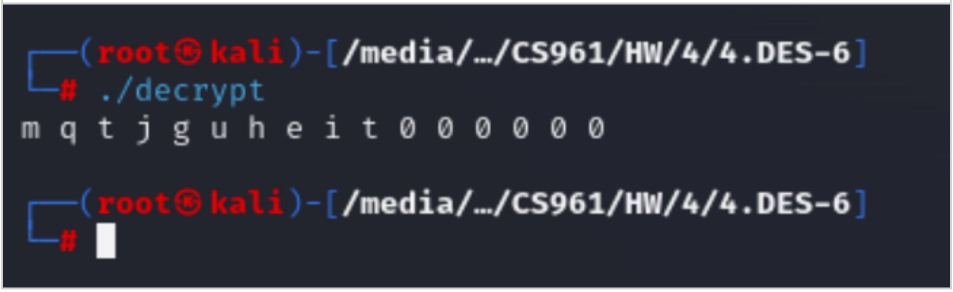
Download



```
        You enter a narrow passage with very dim light.
        The passage is not very long, you can see the other
        exit in front of you.
        You notice names written all around:
                mjawed23


Press c to continue>
```

▾ Passcode-Command.png

Download



```
┌──(root㉿kali)-[/media/…/CS961/HW/4/4.DES-6]
└─# ./decrypt
m q t j g u h e i t 0 0 0 0 0 0

┌──(root㉿kali)-[/media/…/CS961/HW/4/4.DES-6]
└─#
```

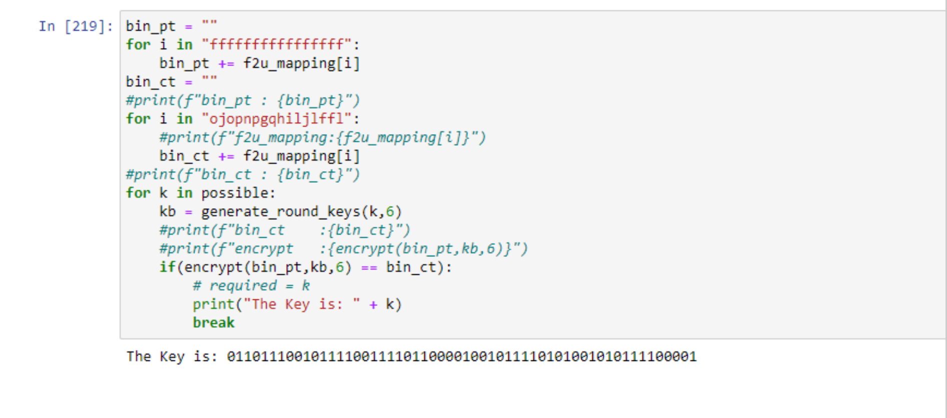
4.DES-6.zip

Download

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

ActualKey.png

Download

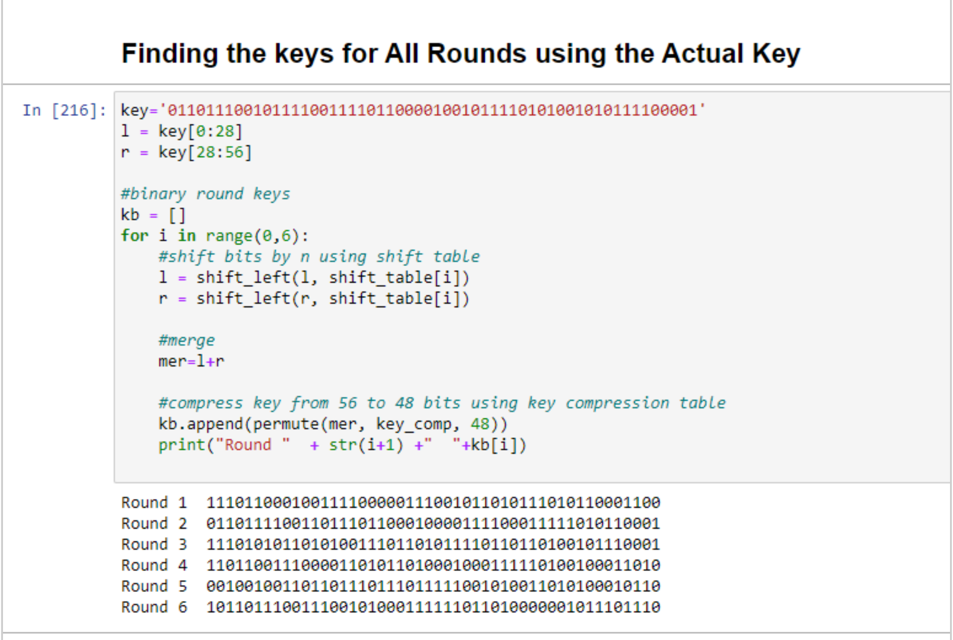


In [219]:
```
bin_pt = ""
for i in "ffffffffffffffff":
    bin_pt += f2u_mapping[i]
bin_ct = ""
#print(f"bin_pt : {bin_pt}")
for i in "ojopnpgqhiljlffl":
    #print(f"f2u_mapping:{f2u_mapping[i]}")
    bin_ct += f2u_mapping[i]
#print(f"bin_ct : {bin_ct}")
for k in possible:
    kb = generate_round_keys(k,6)
    #print(f"bin_ct    :{bin_ct}")
    #print(f"encrypt   :{encrypt(bin_pt,kb,6)}")
    if(encrypt(bin_pt,kb,6) == bin_ct):
        # required = k
        print("The Key is: " + k)
        break
```

```
The Key is: 01101110010111100111101100001001011110101001010111100001
```

RoundKeys.png

Download



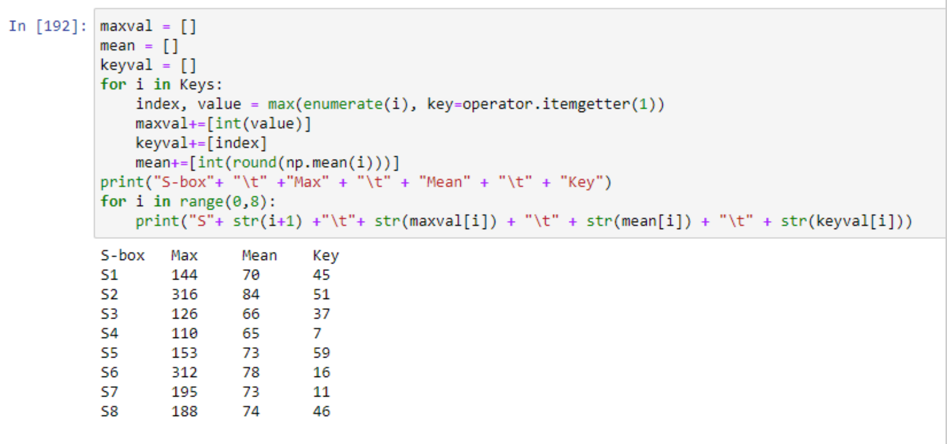
Finding the keys for All Rounds using the Actual Key

In [216]:
```
key='01101110010111100111101100001001011110101001010111100001'
l = key[0:28]
r = key[28:56]

#binary round keys
kb = []
for i in range(0,6):
    #shift bits by n using shift table
    l = shift_left(l, shift_table[i])
    r = shift_left(r, shift_table[i])

    #merge
    mer=l+r

    #compress key from 56 to 48 bits using key compression table
    kb.append(permute(mer, key_comp, 48))
    print("Round "  + str(i+1) +"  "+kb[i])
```

```
Round 1  111011000100111100000111001011010111010110001100
Round 2  011011110011011101100010000111100011111010110001
Round 3  111010101101010011101101011110110110100101110001
Round 4  110110011100001101011010001000111110100100011010
Round 5  001001001101101110111011111001010011010100010110
Round 6  101101110011100101000111111011010000001011101110
```

S-Box-Key-Set1.png

Download

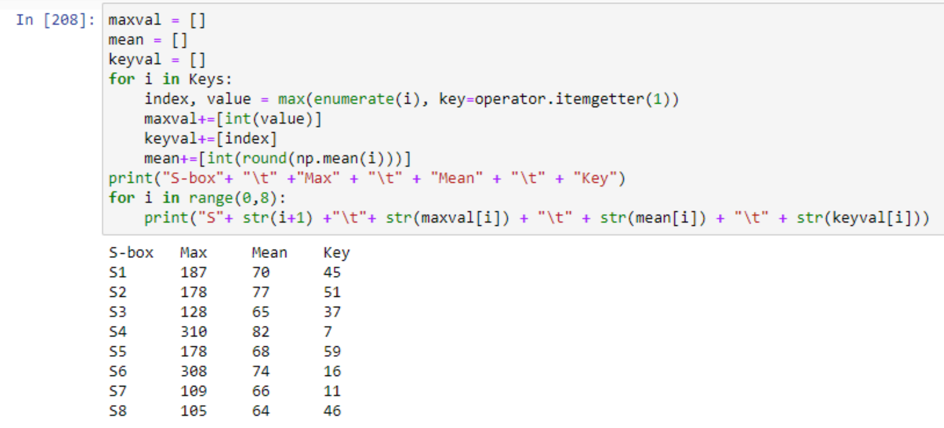


In [192]:
```
maxval = []
mean = []
keyval = []
for i in Keys:
    index, value = max(enumerate(i), key=operator.itemgetter(1))
    maxval+=[int(value)]
    keyval+=[index]
    mean+=[int(round(np.mean(i)))]
print("S-box"+ "\t" +"Max" + "\t" + "Mean" + "\t" + "Key")
for i in range(0,8):
    print("S"+ str(i+1) +"\t"+ str(maxval[i]) + "\t" + str(mean[i]) + "\t" + str(keyval[i]))
```

```
S-box   Max     Mean    Key
S1      144     70      45
S2      316     84      51
S3      126     66      37
S4      110     65      7
S5      153     73      59
S6      312     78      16
S7      195     73      11
S8      188     74      46
```

▾ S-Box-Key-Set2.png

Download



```
In [208]: maxval = []
          mean = []
          keyval = []
          for i in Keys:
              index, value = max(enumerate(i), key=operator.itemgetter(1))
              maxval+=[int(value)]
              keyval+=[index]
              mean+=[int(round(np.mean(i)))]
          print("S-box"+ "\t" +"Max" + "\t" + "Mean" + "\t" + "Key")
          for i in range(0,8):
              print("S"+ str(i+1) +"\t"+ str(maxval[i]) + "\t" + str(mean[i]) + "\t" + str(keyval[i]))
```

```
S-box   Max     Mean    Key
S1      187     70      45
S2      178     77      51
S3      128     65      37
S4      310     82      7
S5      178     68      59
S6      308     74      16
S7      109     66      11
S8      105     64      46
```

Assignment 4

● Ungraded

Student

Mohammed Jawed

View or edit group

Total Points

- / 100 pts

Question 1

Commands 0 pts

Question 2

Cryptosystem 10 pts

Question 3

Analysis 80 pts

Question 4

Password 10 pts

Question 5

Code & figures 0 pts