Practical Assignment - 0

Name : Gaurav Singh
Roll no : 17085035
Department : Electrical Engineering

Objective:
The objective of this assignment was to reverse the alphabetical order I.e ‘a’ will be written as ‘z’ and ‘b’ will be written as ‘y’ and so on.
Theory:
Operating System : linux
Language used : Python
Library used : Tkinter
The first letter in the alphabetical order should be converted to the first letter from the last in the alphabetical order and hence we use the relative position of the letter as compared to letter ‘a’ if it is not a capital letter and use ‘A’ if the given letter is capital letter.
Implementation Details:
Here is my GitHub link for source code of all assignments.
(https://github.com/gaurav6225/NetSec-Assignments)
Below is my source code written in Python

```
from tkinter import *

#This Function will be executed when Encrypt Button is clicked
def Encrypt():
    dec.delete(1.0,END)        #Empty the text present in dec text box
    s = enc.get("1.0",'end-1c')  #Take the input from enc text box
    ans = ""
    for i in s:                #Iterate through every character in the input
        if i>='a' and i<='z':
            i = chr(ord('z')+ord('a')-ord(i))
        elif i>='A' and i<='Z':
            i = chr(ord('A')+ord('Z')-ord(i))
        ans = ans + str(i)
    dec.insert(0.0,ans)

#This Function will be executed when Decrypt Button is clicked
def Decrypt():
    enc.delete(1.0,END)        #Empty the text present inn enc text box
    s = dec.get("1.0",'end-1c')  #Take the input from dec text box
    ans = ""
    for i in s:                #Iterate through every character in the input
        if i>='a' and i<='z':
            i = chr(219-ord(i))
        elif i>='A' and i<='Z':
```

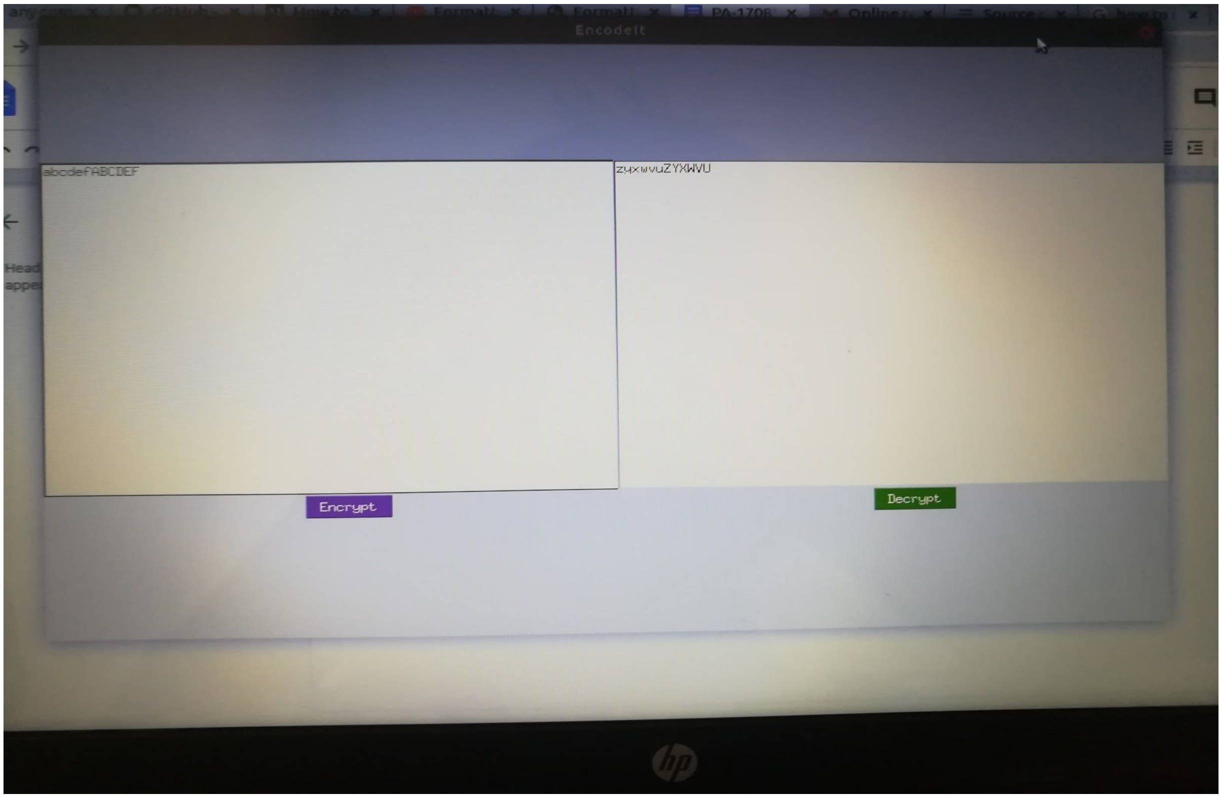
```
            i = chr(155-ord(i))
        ans = ans + str(i)
    enc.insert(0.0,ans)

# Used tkinter for creating the UI

root = Tk()
root.geometry("1000x600")
root.title('EncodeIt')

#Creating the Frames in which we will insert Text Boxes and Buttons
frame1 = Frame(root)
frame2 = Frame(root)

frame1.grid(row=0,column=0,sticky=W)
frame2.grid(row=0,column=1,sticky=E)

root.rowconfigure(0, weight=1)
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=1)

#Creating the Text Boxes
enc = Text(frame1,height=30,width=75,wrap=WORD)
dec = Text(frame2,height=30,width=75,wrap=WORD)
enc.grid(sticky=W)
dec.grid(sticky=E)

#Creating the Buttons
btn1 = Button(frame1,text='Encrypt',bg="purple",fg="white",command=Encrypt)
btn2 = Button(frame2,text='Decrypt',bg="green",fg="white",command=Decrypt)
btn1.grid(sticky=S)
btn2.grid(sticky=S)

root.mainloop()
```

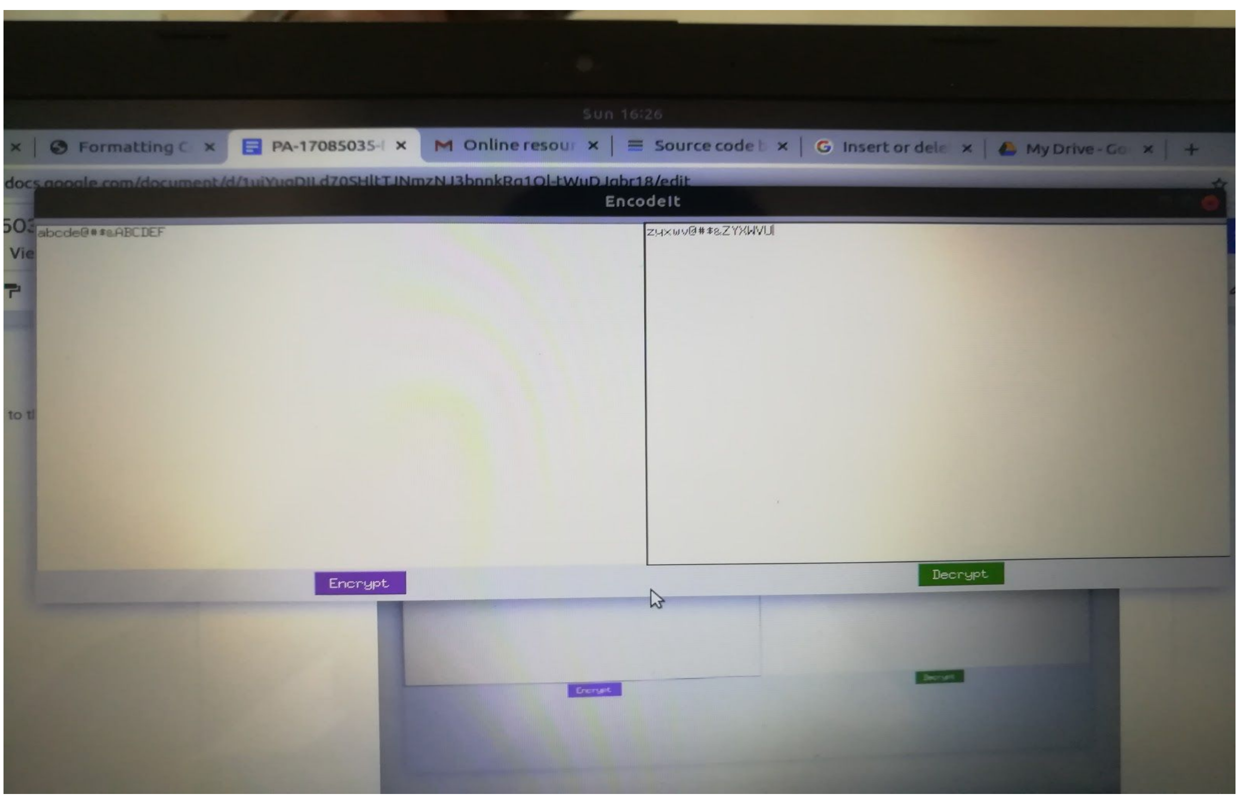
Encrypt function will be executed when Encrypt Button is clicked. It first erases all the text from the Decrypt text Box and encrypts the given input provided which is then written in the decrypt text box and Similarly the Decrypt function will work exactly the same as the Encrypt function.

Screenshots of output:

In the first image abcdefABCDEF is encrypted as zyxwvuZYXWVU

In the second image we can note that any character other than english alphabet will neither be encrypted nor decrypted.


EncodeIt
abcdefABCDEF
zyxwvuZYXWVU
Encrypt
Decrypt


Sun 16:26
Formatting C
PA-17085035-
Online resou
Source code
Insert or dele
My Drive - Go
EncodeIt
abcde@#$&ABCDEF
zyxwv@#$&ZYXWVU
Encrypt
Decrypt

Practical Assignment - 1

Name : Gaurav Singh
Roll no : 17085035
Department : Electrical Engineering

Objective:
To write a program with a nice UI to implement and study DEA with different hyper parameters. Number of rounds n=1,8,16,32; half width of data block w=16,32,64. Pick suitable entries for P- and S- boxes.Demonstrate the avalanche effect with different hyper parameter choices. Demonstrate how weak keys supplied by the user affects the round keys.

Theory:

DES Algorithm-

DES is a block cipher, and encrypts data in blocks of size of 64 bit each, meaning 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

Avalanche Effect:

In cryptography, the avalanche effect is a term associated with a specific behavior of mathematical functions used for encryption. Avalanche effect is considered as one of the desirable properties of any encryption algorithm. A slight change in either the key or the plain-text should result in a significant change in the cipher-text. This property is termed an **avalanche effect**.

Implementation details:

Operating System : linux
Language used : Python
Library used : Tkinter
Below is my source code written in Python

```
from tkinter import *

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

```
    63, 55, 47, 39, 31, 23, 15, 7
)
IP_INV = (
    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
)
PC1 = (
    57, 49, 41, 33, 25, 17, 9,
    1,  58, 50, 42, 34, 26, 18,
    10, 2,  59, 51, 43, 35, 27,
    19, 11, 3,  60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7,  62, 54, 46, 38, 30, 22,
    14, 6,  61, 53, 45, 37, 29,
    21, 13, 5,  28, 20, 12, 4
)
PC2 = (
    14, 17, 11, 24, 1,  5,
    3,  28, 15, 6,  21, 10,
    23, 19, 12, 4,  26, 8,
    16, 7,  27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
)

E1  = (
    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
)

Sboxes = {
    0: (
        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
    ),
    1: (
        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
    ),
    2: (
        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
    ),
    3: (
        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
    ),
    4: (
        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
    ),
    5: (
        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
    ),
    6: (
        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
    ),
    7: (
        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
    )
}
```

```
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
)

def Encrypt():
    # only encrypt single blocks
    dec.delete(1.0,END)
    msg = int(enc.get("1.0",'end-1c'))
    key = int(k1.get("1.0",'end-1c'))
    n = int(rnd1.get("1.0",'end-1c'))
    assert isinstance(msg, int) and isinstance(key, int)
    assert not msg.bit_length() > 64
    assert not key.bit_length() > 64

    # permute by table PC1
    key = permutation_by_table(key, 64, PC1) # 64bit -> PC1 -> 56bit

    # split up key in two halves
    # generate the 16 round keys
    C0 = key >> 28
    D0 = key & (2**28-1)
    round_keys = generate_round_keys(C0, D0) # 56bit -> PC2 -> 48bit

    msg_block = permutation_by_table(msg, 64, IP)
    L0 = msg_block >> 32
    R0 = msg_block & (2**32-1)

    # apply thr round function 16 times in following scheme (feistel cipher):
    L_last = L0
    R_last = R0
    for i in range(1,n+1):
        #if decrypt: # just use the round keys in reversed order
        #    i = 17-i
        L_round = R_last
        R_round = L_last ^ round_function(R_last, round_keys[i])
        L_last = L_round
        R_last = R_round

    # concatenate reversed
```

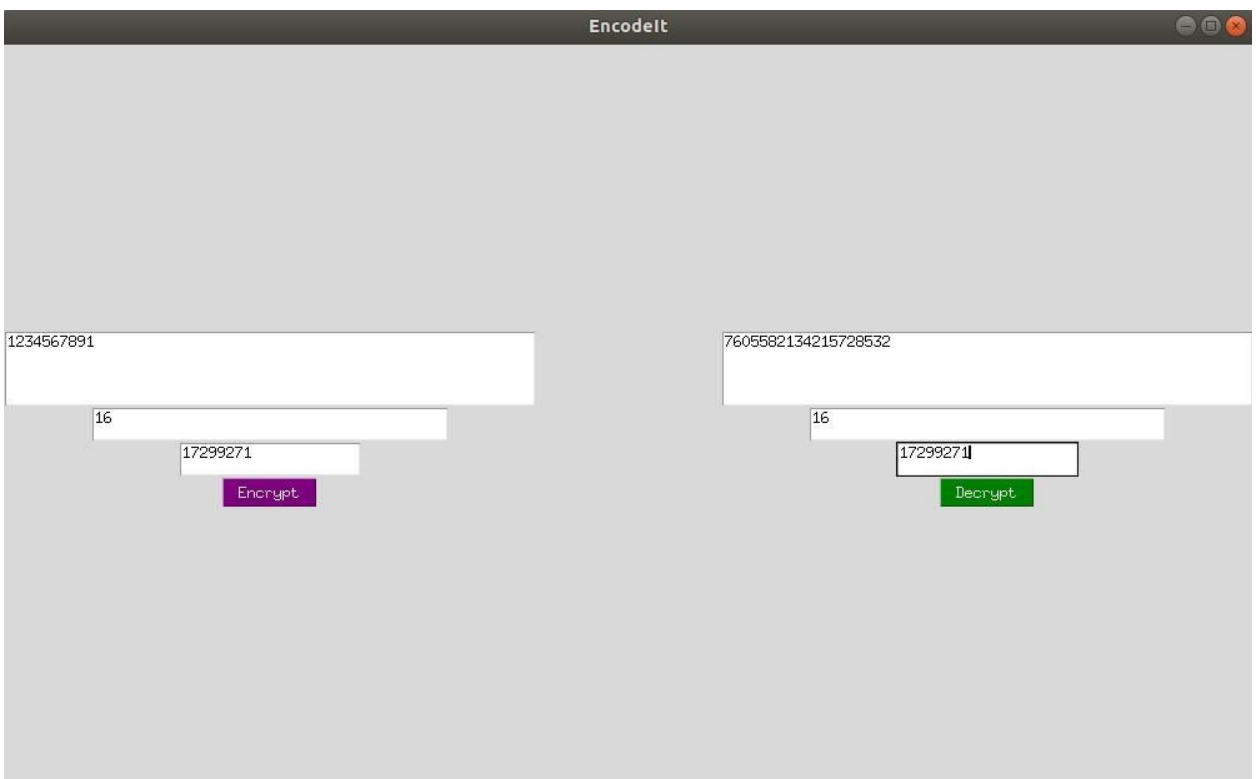
```
    cipher_block = (R_round<<32) + L_round

    # final permutation
    cipher_block = permutation_by_table(cipher_block, 64, IP_INV)
    ans = str(cipher_block)
    dec.insert(0.0,ans)


def Decrypt():
    # only encrypt single blocks
    enc.delete(1.0,END)
    msg = int(dec.get("1.0",'end-1c'))
    key = int(k2.get("1.0",'end-1c'))
    n = int(rnd2.get("1.0",'end-1c'))
    assert isinstance(msg, int) and isinstance(key, int)
    assert not msg.bit_length() > 64
    assert not key.bit_length() > 64

    # permutate by table PC1
    key = permutation_by_table(key, 64, PC1) # 64bit -> PC1 -> 56bit

    # split up key in two halves
    # generate the 16 round keys
    C0 = key >> 28
    D0 = key & (2**28-1)
    round_keys = generate_round_keys(C0, D0) # 56bit -> PC2 -> 48bit

    msg_block = permutation_by_table(msg, 64, IP)
    L0 = msg_block >> 32
    R0 = msg_block & (2**32-1)

    # apply thr round function 16 times in following scheme (feistel cipher):
    L_last = L0
    R_last = R0
    for i in range(1,n+1):
        # just use the round keys in reversed order
        i = n+1-i
        L_round = R_last
        R_round = L_last ^ round_function(R_last, round_keys[i])
        L_last = L_round
        R_last = R_round

    # concatenate reversed
    cipher_block = (R_round<<32) + L_round

    # final permutation
    cipher_block = permutation_by_table(cipher_block, 64, IP_INV)
```

```
    ans = str(cipher_block)
    enc.insert(0.0,ans)


def round_function(Ri, Ki):
    # expand Ri from 32 to 48 bit using table E
    Ri = permutation_by_table(Ri, 32, E1)

    # xor with round key
    Ri ^= Ki

    # split Ri into 8 groups of 6 bit
    Ri_blocks = [((Ri & (0b111111 << shift_val)) >> shift_val) for shift_val in
(42,36,30,24,18,12,6,0)]

    # interpret each block as address for the S-boxes
    for i, block in enumerate(Ri_blocks):
        # grab the bits we need
        row = ((0b100000 & block) >> 4) + (0b1 & block)
        col = (0b011110 & block) >> 1
        # sboxes are stored as one-dimensional tuple, so we need to calc the index this way
        Ri_blocks[i] = Sboxes[i][16*row+col]

    # pack the blocks together again by concatenating
    Ri_blocks = zip(Ri_blocks, (28,24,20,16,12,8,4,0))
    Ri = 0
    for block, lshift_val in Ri_blocks:
        Ri += (block << lshift_val)

    # another permutation 32bit -> 32bit
    Ri = permutation_by_table(Ri, 32, P)

    return Ri

def permutation_by_table(block, block_len, table):
    # quick and dirty casting to str
    block_str = bin(block)[2:].zfill(block_len)
    perm = []
    for pos in range(len(table)):
        perm.append(block_str[table[pos]-1])
    return int(''.join(perm), 2)

def generate_round_keys(C0, D0):
    # returns dict of 16 keys (one for each round)

    round_keys = dict.fromkeys(range(0,17))
```

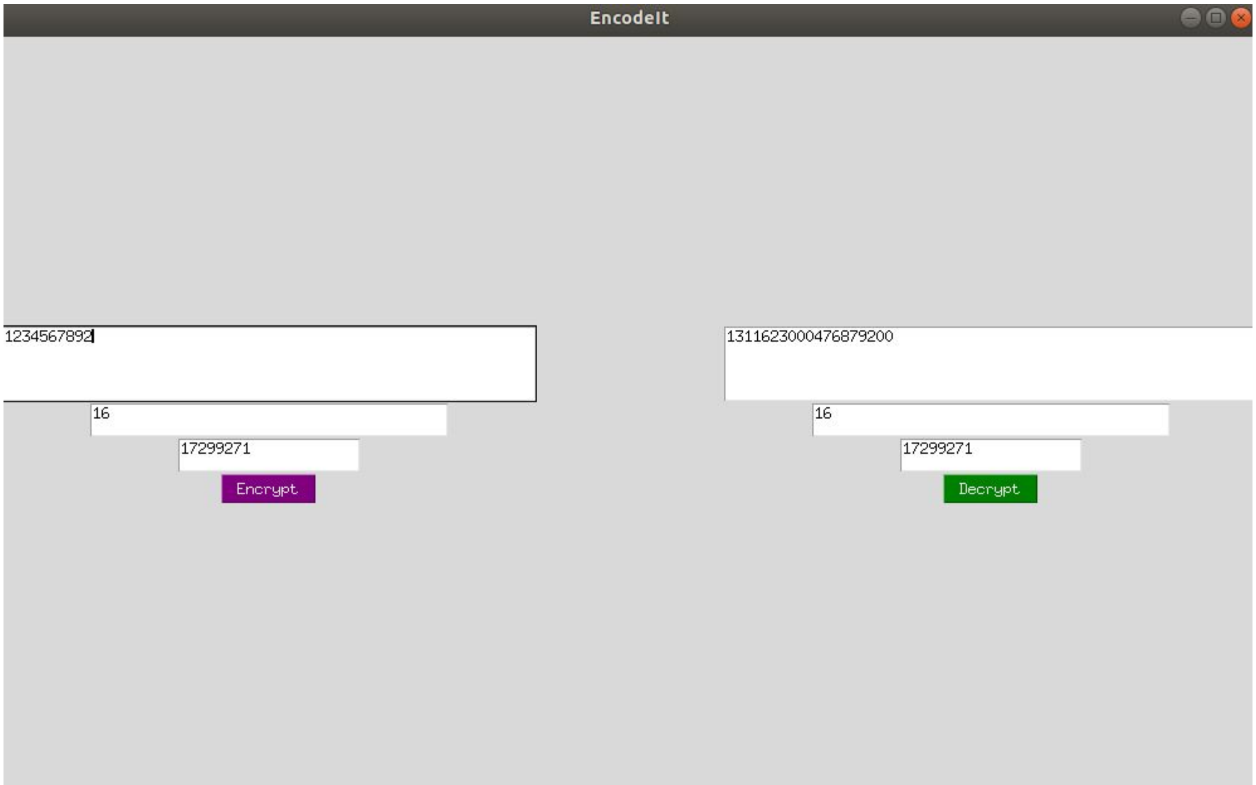
```
    lrot_values = (1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1)

    # left-rotation function
    lrot = lambda val, r_bits, max_bits: \
    (val << r_bits%max_bits) & (2**max_bits-1) | \
    ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

    # initial rotation
    C0 = lrot(C0, 0, 28)
    D0 = lrot(D0, 0, 28)
    round_keys[0] = (C0, D0)

    # create 16 more different key pairs
    for i, rot_val in enumerate(lrot_values):
        i+=1
        Ci = lrot(round_keys[i-1][0], rot_val, 28)
        Di = lrot(round_keys[i-1][1], rot_val, 28)
        round_keys[i] = (Ci, Di)

    # round_keys[1] for first round
    #           [16] for 16th round
    # dont need round_keys[0] anymore, remove
    del round_keys[0]

    # now form the keys from concatenated CiDi 1<=i<=16 and by apllying PC2
    for i, (Ci, Di) in round_keys.items():
        Ki = (Ci << 28) + Di
        round_keys[i] = permutation_by_table(Ki, 56, PC2) # 56bit -> 48bit

    return round_keys


root = Tk()
root.geometry("1000x600")
root.title('EncodeIt')

frame1 = Frame(root)
frame2 = Frame(root)

frame1.grid(row=0,column=0,sticky=W)
frame2.grid(row=0,column=1,sticky=E)

root.rowconfigure(0, weight=1)
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=1)

enc = Text(frame1,height=5,width=60,wrap=WORD)
```

```
dec = Text(frame2,height=5,width=60,wrap=WORD)
enc.grid(sticky=W)
dec.grid(sticky=E)

rnd1 = Text(frame1, height=2,width=40,wrap=WORD)
rnd2 = Text(frame2, height=2,width=40,wrap=WORD)
rnd1.grid(sticky=S)
rnd2.grid(sticky=S)

k1 = Text(frame1, height=2,width=20,wrap=WORD)
k2 = Text(frame2, height=2,width=20,wrap=WORD)
k1.grid(sticky=S)
k2.grid(sticky=S)

btn1 = Button(frame1,text='Encrypt',bg="purple",fg="white",command=Encrypt)
btn2 = Button(frame2,text='Decrypt',bg="green",fg="white",command=Decrypt)
btn1.grid(sticky=S)
btn2.grid(sticky=S)

root.mainloop()
```

Screenshots of output:



- In the above image we can see that the plain text 1234567891 is encrypted to 7605582134215728532 with 17299271 as key and no.of rounds equal to 16. And we can observe that when we click the decrypt button with name no.of rounds and same key we get the same plain text that was initially encrypted.



- In the above image we can see that the plain text 1234567892 is encrypted as 1311623000476879200 with the same key and same number of rounds. And hence we can observe that there are a lot of changes in the encrypted text as compared to earlier. This is known as Avalanche Effect. A little change in plain text will result in lot changes in cipher text.

Practical Assignment - 2

Name : Gaurav Singh
Roll no : 17085035
Department : Electrical Engineering

Objective:
Using a standard Blowfish implementation as basic encryption, write code for the CBC and OFB modes.

Theory:
Blowfish is an encryption technique designed by Bruce Schneier in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.

Implementation Details:
Operating System : linux
Language used : Python
Library used : Tkinter
Below is my source code written in Python

```
IV = 0xFFFFFFFFFFFFFFFD   #initialization vector
#p and s boxes
p = [
    0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
    0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
    0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
    0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
    0x9216D5D9, 0x8979FB1B
  ]

s = [
    [
      0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
      0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
      0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
      0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
      0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
      0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
      0xC5D1B023, 0x286085F0, 0xCA417918, 0xB8DB38EF,
      0x8E79DCB0, 0x603A180E, 0x6C9E0E8B, 0xB01E8A3E,
      0xD71577C1, 0xBD314B27, 0x78AF2FDA, 0x55605C60,
      0xE65525F3, 0xAA55AB94, 0x57489862, 0x63E81440,
      0x55CA396A, 0x2AAB10B6, 0xB4CC5C34, 0x1141E8CE,
      0xA15486AF, 0x7C72E993, 0xB3EE1411, 0x636FBC2A,
      0x2BA9C55D, 0x741831F6, 0xCE5C3E16, 0x9B87931E,
```

```
0xAFD6BA33, 0x6C24CF5C, 0x7A325381, 0x28958677,
0x3B8F4898, 0x6B4BB9AF, 0xC4BFE81B, 0x66282193,
0x61D809CC, 0xFB21A991, 0x487CAC60, 0x5DEC8032,
0xEF845D5D, 0xE98575B1, 0xDC262302, 0xEB651B88,
0x23893E81, 0xD396ACC5, 0x0F6D6FF3, 0x83F44239,
0x2E0B4482, 0xA4842004, 0x69C8F04A, 0x9E1F9B5E,
0x21C66842, 0xF6E96C9A, 0x670C9C61, 0xABD388F0,
0x6A51A0D2, 0xD8542F68, 0x960FA728, 0xAB5133A3,
0x6EEF0B6C, 0x137A3BE4, 0xBA3BF050, 0x7EFB2A98,
0xA1F1651D, 0x39AF0176, 0x66CA593E, 0x82430E88,
0x8CEE8619, 0x456F9FB4, 0x7D84A5C3, 0x3B8B5EBE,
0xE06F75D8, 0x85C12073, 0x401A449F, 0x56C16AA6,
0x4ED3AA62, 0x363F7706, 0x1BFEDF72, 0x429B023D,
0x37D0D724, 0xD00A1248, 0xDB0FEAD3, 0x49F1C09B,
0x075372C9, 0x80991B7B, 0x25D479D8, 0xF6E8DEF7,
0xE3FE501A, 0xB6794C3B, 0x976CE0BD, 0x04C006BA,
0xC1A94FB6, 0x409F60C4, 0x5E5C9EC2, 0x196A2463,
0x68FB6FAF, 0x3E6C53B5, 0x1339B2EB, 0x3B52EC6F,
0x6DFC511F, 0x9B30952C, 0xCC814544, 0xAF5EBD09,
0xBEE3D004, 0xDE334AFD, 0x660F2807, 0x192E4BB3,
0xC0CBA857, 0x45C8740F, 0xD20B5F39, 0xB9D3FBDB,
0x5579C0BD, 0x1A60320A, 0xD6A100C6, 0x402C7279,
0x679F25FE, 0xFB1FA3CC, 0x8EA5E9F8, 0xDB3222F8,
0x3C7516DF, 0xFD616B15, 0x2F501EC8, 0xAD0552AB,
0x323DB5FA, 0xFD238760, 0x53317B48, 0x3E00DF82,
0x9E5C57BB, 0xCA6F8CA0, 0x1A87562E, 0xDF1769DB,
0xD542A8F6, 0x287EFFC3, 0xAC6732C6, 0x8C4F5573,
0x695B27B0, 0xBBCA58C8, 0xE1FFA35D, 0xB8F011A0,
0x10FA3D98, 0xFD2183B8, 0x4AFCB56C, 0x2DD1D35B,
0x9A53E479, 0xB6F84565, 0xD28E49BC, 0x4BFB9790,
0xE1DDF2DA, 0xA4CB7E33, 0x62FB1341, 0xCEE4C6E8,
0xEF20CADA, 0x36774C01, 0xD07E9EFE, 0x2BF11FB4,
0x95DBDA4D, 0xAE909198, 0xEAAD8E71, 0x6B93D5A0,
0xD08ED1D0, 0xAFC725E0, 0x8E3C5B2F, 0x8E7594B7,
0x8FF6E2FB, 0xF2122B64, 0x8888B812, 0x900DF01C,
0x4FAD5EA0, 0x688FC31C, 0xD1CFF191, 0xB3A8C1AD,
0x2F2F2218, 0xBE0E1777, 0xEA752DFE, 0x8B021FA1,
0xE5A0CC0F, 0xB56F74E8, 0x18ACF3D6, 0xCE89E299,
0xB4A84FE0, 0xFD13E0B7, 0x7CC43B81, 0xD2ADA8D9,
0x165FA266, 0x80957705, 0x93CC7314, 0x211A1477,
0xE6AD2065, 0x77B5FA86, 0xC75442F5, 0xFB9D35CF,
0xEBCDAF0C, 0x7B3E89A0, 0xD6411BD3, 0xAE1E7E49,
0x00250E2D, 0x2071B35E, 0x226800BB, 0x57B8E0AF,
0x2464369B, 0xF009B91E, 0x5563911D, 0x59DFA6AA,
0x78C14389, 0xD95A537F, 0x207D5BA2, 0x02E5B9C5,
0x83260376, 0x6295CFA9, 0x11C81968, 0x4E734A41,
0xB3472DCA, 0x7B14A94A, 0x1B510052, 0x9A532915,
```

```
        0xD60F573F, 0xBC9BC6E4, 0x2B60A476, 0x81E67400,
        0x08BA6FB5, 0x571BE91F, 0xF296EC6B, 0x2A0DD915,
        0xB6636521, 0xE7B9F9B6, 0xFF34052E, 0xC5855664,
        0x53B02D5D, 0xA99F8FA1, 0x08BA4799, 0x6E85076A
    ],
    [
        0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
        0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
        0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
        0x193602A5, 0x75094C29, 0xA0591340, 0xE4183A3E,
        0x3F54989A, 0x5B429D65, 0x6B8FE4D6, 0x99F73FD6,
        0xA1D29C07, 0xEFE830F5, 0x4D2D38E6, 0xF0255DC1,
        0x4CDD2086, 0x8470EB26, 0x6382E9C6, 0x021ECC5E,
        0x09686B3F, 0x3EBAEFC9, 0x3C971814, 0x6B6A70A1,
        0x687F3584, 0x52A0E286, 0xB79C5305, 0xAA500737,
        0x3E07841C, 0x7FDEAE5C, 0x8E7D44EC, 0x5716F2B8,
        0xB03ADA37, 0xF0500C0D, 0xF01C1F04, 0x0200B3FF,
        0xAE0CF51A, 0x3CB574B2, 0x25837A58, 0xDC0921BD,
        0xD19113F9, 0x7CA92FF6, 0x94324773, 0x22F54701,
        0x3AE5E581, 0x37C2DADC, 0xC8B57634, 0x9AF3DDA7,
        0xA9446146, 0x0FD0030E, 0xECC8C73E, 0xA4751E41,
        0xE238CD99, 0x3BEA0E2F, 0x3280BBA1, 0x183EB331,
        0x4E548B38, 0x4F6DB908, 0x6F420D03, 0xF60A04BF,
        0x2CB81290, 0x24977C79, 0x5679B072, 0xBCAF89AF,
        0xDE9A771F, 0xD9930810, 0xB38BAE12, 0xDCCF3F2E,
        0x5512721F, 0x2E6B7124, 0x501ADDE6, 0x9F84CD87,
        0x7A584718, 0x7408DA17, 0xBC9F9ABC, 0xE94B7D8C,
        0xEC7AEC3A, 0xDB851DFA, 0x63094366, 0xC464C3D2,
        0xEF1C1847, 0x3215D908, 0xDD433B37, 0x24C2BA16,
        0x12A14D43, 0x2A65C451, 0x50940002, 0x133AE4DD,
        0x71DFF89E, 0x10314E55, 0x81AC77D6, 0x5F11199B,
        0x043556F1, 0xD7A3C76B, 0x3C11183B, 0x5924A509,
        0xF28FE6ED, 0x97F1FBFA, 0x9EBABF2C, 0x1E153C6E,
        0x86E34570, 0xEAE96FB1, 0x860E5E0A, 0x5A3E2AB3,
        0x771FE71C, 0x4E3D06FA, 0x2965DCB9, 0x99E71D0F,
        0x803E89D6, 0x5266C825, 0x2E4CC978, 0x9C10B36A,
        0xC6150EBA, 0x94E2EA78, 0xA5FC3C53, 0x1E0A2DF4,
        0xF2F74EA7, 0x361D2B3D, 0x1939260F, 0x19C27960,
        0x5223A708, 0xF71312B6, 0xEBADFE6E, 0xEAC31F66,
        0xE3BC4595, 0xA67BC883, 0xB17F37D1, 0x018CFF28,
        0xC332DDEF, 0xBE6C5AA5, 0x65582185, 0x68AB9802,
        0xEECEA50F, 0xDB2F953B, 0x2AEF7DAD, 0x5B6E2F84,
        0x1521B628, 0x29076170, 0xECDD4775, 0x619F1510,
        0x13CCA830, 0xEB61BD96, 0x0334FE1E, 0xAA0363CF,
        0xB5735C90, 0x4C70A239, 0xD59E9E0B, 0xCBAADE14,
        0xEECC86BC, 0x60622CA7, 0x9CAB5CAB, 0xB2F3846E,
        0x648B1EAF, 0x19BDF0CA, 0xA02369B9, 0x655ABB50,
```

```
    0x40685A32, 0x3C2AB4B3, 0x319EE9D5, 0xC021B8F7,
    0x9B540B19, 0x875FA099, 0x95F7997E, 0x623D7DA8,
    0xF837889A, 0x97E32D77, 0x11ED935F, 0x16681281,
    0x0E358829, 0xC7E61FD6, 0x96DEDFA1, 0x7858BA99,
    0x57F584A5, 0x1B227263, 0x9B83C3FF, 0x1AC24696,
    0xCDB30AEB, 0x532E3054, 0x8FD948E4, 0x6DBC3128,
    0x58EBF2EF, 0x34C6FFEA, 0xFE28ED61, 0xEE7C3C73,
    0x5D4A14D9, 0xE864B7E3, 0x42105D14, 0x203E13E0,
    0x45EEE2B6, 0xA3AAABEA, 0xDB6C4F15, 0xFACB4FD0,
    0xC742F442, 0xEF6ABBB5, 0x654F3B1D, 0x41CD2105,
    0xD81E799E, 0x86854DC7, 0xE44B476A, 0x3D816250,
    0xCF62A1F2, 0x5B8D2646, 0xFC8883A0, 0xC1C7B6A3,
    0x7F1524C3, 0x69CB7492, 0x47848A0B, 0x5692B285,
    0x095BBF00, 0xAD19489D, 0x1462B174, 0x23820E00,
    0x58428D2A, 0x0C55F5EA, 0x1DADF43E, 0x233F7061,
    0x3372F092, 0x8D937E41, 0xD65FECF1, 0x6C223BDB,
    0x7CDE3759, 0xCBEE7460, 0x4085F2A7, 0xCE77326E,
    0xA6078084, 0x19F8509E, 0xE8EFD855, 0x61D99735,
    0xA969A7AA, 0xC50C06C2, 0x5A04ABFC, 0x800BCADC,
    0x9E447A2E, 0xC3453484, 0xFDD56705, 0x0E1E9EC9,
    0xDB73DBD3, 0x105588CD, 0x675FDA79, 0xE3674340,
    0xC5C43465, 0x713E38D8, 0x3D28F89E, 0xF16DFF20,
    0x153E21E7, 0x8FB03D4A, 0xE6E39F2B, 0xDB83ADF7
],
[
    0xE93D5A68, 0x948140F7, 0xF64C261C, 0x94692934,
    0x411520F7, 0x7602D4F7, 0xBCF46B2E, 0xD4A20068,
    0xD4082471, 0x3320F46A, 0x43B7D4B7, 0x500061AF,
    0x1E39F62E, 0x97244546, 0x14214F74, 0xBF8B8840,
    0x4D95FC1D, 0x96B591AF, 0x70F4DDD3, 0x66A02F45,
    0xBFBC09EC, 0x03BD9785, 0x7FAC6DD0, 0x31CB8504,
    0x96EB27B3, 0x55FD3941, 0xDA2547E6, 0xABCA0A9A,
    0x28507825, 0x530429F4, 0x0A2C86DA, 0xE9B66DFB,
    0x68DC1462, 0xD7486900, 0x680EC0A4, 0x27A18DEE,
    0x4F3FFEA2, 0xE887AD8C, 0xB58CE006, 0x7AF4D6B6,
    0xAACE1E7C, 0xD3375FEC, 0xCE78A399, 0x406B2A42,
    0x20FE9E35, 0xD9F385B9, 0xEE39D7AB, 0x3B124E8B,
    0x1DC9FAF7, 0x4B6D1856, 0x26A36631, 0xEAE397B2,
    0x3A6EFA74, 0xDD5B4332, 0x6841E7F7, 0xCA7820FB,
    0xFB0AF54E, 0xD8FEB397, 0x454056AC, 0xBA489527,
    0x55533A3A, 0x20838D87, 0xFE6BA9B7, 0xD096954B,
    0x55A867BC, 0xA1159A58, 0xCCA92963, 0x99E1DB33,
    0xA62A4A56, 0x3F3125F9, 0x5EF47E1C, 0x9029317C,
    0xFDF8E802, 0x04272F70, 0x80BB155C, 0x05282CE3,
    0x95C11548, 0xE4C66D22, 0x48C1133F, 0xC70F86DC,
    0x07F9C9EE, 0x41041F0F, 0x404779A4, 0x5D886E17,
    0x325F51EB, 0xD59BC0D1, 0xF2BCC18F, 0x41113564,
```

```
    0x257B7834, 0x602A9C60, 0xDFF8E8A3, 0x1F636C1B,
    0x0E12B4C2, 0x02E1329E, 0xAF664FD1, 0xCAD18115,
    0x6B2395E0, 0x333E92E1, 0x3B240B62, 0xEEBEB922,
    0x85B2A20E, 0xE6BA0D99, 0xDE720C8C, 0x2DA2F728,
    0xD0127845, 0x95B794FD, 0x647D0862, 0xE7CCF5F0,
    0x5449A36F, 0x877D48FA, 0xC39DFD27, 0xF33E8D1E,
    0x0A476341, 0x992EFF74, 0x3A6F6EAB, 0xF4F8FD37,
    0xA812DC60, 0xA1EBDDF8, 0x991BE14C, 0xDB6E6B0D,
    0xC67B5510, 0x6D672C37, 0x2765D43B, 0xDCD0E804,
    0xF1290DC7, 0xCC00FFA3, 0xB5390F92, 0x690FED0B,
    0x667B9FFB, 0xCEDB7D9C, 0xA091CF0B, 0xD9155EA3,
    0xBB132F88, 0x515BAD24, 0x7B9479BF, 0x763BD6EB,
    0x37392EB3, 0xCC115979, 0x8026E297, 0xF42E312D,
    0x6842ADA7, 0xC66A2B3B, 0x12754CCC, 0x782EF11C,
    0x6A124237, 0xB79251E7, 0x06A1BBE6, 0x4BFB6350,
    0x1A6B1018, 0x11CAEDFA, 0x3D25BDD8, 0xE2E1C3C9,
    0x44421659, 0x0A121386, 0xD90CEC6E, 0xD5ABEA2A,
    0x64AF674E, 0xDA86A85F, 0xBEBFE988, 0x64E4C3FE,
    0x9DBC8057, 0xF0F7C086, 0x60787BF8, 0x6003604D,
    0xD1FD8346, 0xF6381FB0, 0x7745AE04, 0xD736FCCC,
    0x83426B33, 0xF01EAB71, 0xB0804187, 0x3C005E5F,
    0x77A057BE, 0xBDE8AE24, 0x55464299, 0xBF582E61,
    0x4E58F48F, 0xF2DDFDA2, 0xF474EF38, 0x8789BDC2,
    0x5366F9C3, 0xC8B38E74, 0xB475F255, 0x46FCD9B9,
    0x7AEB2661, 0x8B1DDF84, 0x846A0E79, 0x915F95E2,
    0x466E598E, 0x20B45770, 0x8CD55591, 0xC902DE4C,
    0xB90BACE1, 0xBB8205D0, 0x11A86248, 0x7574A99E,
    0xB77F19B6, 0xE0A9DC09, 0x662D09A1, 0xC4324633,
    0xE85A1F02, 0x09F0BE8C, 0x4A99A025, 0x1D6EFE10,
    0x1AB93D1D, 0x0BA5A4DF, 0xA186F20F, 0x2868F169,
    0xDCB7DA83, 0x573906FE, 0xA1E2CE9B, 0x4FCD7F52,
    0x50115E01, 0xA70683FA, 0xA002B5C4, 0x0DE6D027,
    0x9AF88C27, 0x773F8641, 0xC3604C06, 0x61A806B5,
    0xF0177A28, 0xC0F586E0, 0x006058AA, 0x30DC7D62,
    0x11E69ED7, 0x2338EA63, 0x53C2DD94, 0xC2C21634,
    0xBBCBEE56, 0x90BCB6DE, 0xEBFC7DA1, 0xCE591D76,
    0x6F05E409, 0x4B7C0188, 0x39720A3D, 0x7C927C24,
    0x86E3725F, 0x724D9DB9, 0x1AC15BB4, 0xD39EB8FC,
    0xED545578, 0x08FCA5B5, 0xD83D7CD3, 0x4DAD0FC4,
    0x1E50EF5E, 0xB161E6F8, 0xA28514D9, 0x6C51133C,
    0x6FD5C7E7, 0x56E14EC4, 0x362ABFCE, 0xDDC6C837,
    0xD79A3234, 0x92638212, 0x670EFA8E, 0x406000E0
  ],
  [
    0x3A39CE37, 0xD3FAF5CF, 0xABC27737, 0x5AC52D1B,
    0x5CB0679E, 0x4FA33742, 0xD3822740, 0x99BC9BBE,
    0xD5118E9D, 0xBF0F7315, 0xD62D1C7E, 0xC700C47B,
```

```
0xB78C1B6B, 0x21A19045, 0xB26EB1BE, 0x6A366EB4,
0x5748AB2F, 0xBC946E79, 0xC6A376D2, 0x6549C2C8,
0x530FF8EE, 0x468DDE7D, 0xD5730A1D, 0x4CD04DC6,
0x2939BBDB, 0xA9BA4650, 0xAC9526E8, 0xBE5EE304,
0xA1FAD5F0, 0x6A2D519A, 0x63EF8CE2, 0x9A86EE22,
0xC089C2B8, 0x43242EF6, 0xA51E03AA, 0x9CF2D0A4,
0x83C061BA, 0x9BE96A4D, 0x8FE51550, 0xBA645BD6,
0x2826A2F9, 0xA73A3AE1, 0x4BA99586, 0xEF5562E9,
0xC72FEFD3, 0xF752F7DA, 0x3F046F69, 0x77FA0A59,
0x80E4A915, 0x87B08601, 0x9B09E6AD, 0x3B3EE593,
0xE990FD5A, 0x9E34D797, 0x2CF0B7D9, 0x022B8B51,
0x96D5AC3A, 0x017DA67D, 0xD1CF3ED6, 0x7C7D2D28,
0x1F9F25CF, 0xADF2B89B, 0x5AD6B472, 0x5A88F54C,
0xE029AC71, 0xE019A5E6, 0x47B0ACFD, 0xED93FA9B,
0xE8D3C48D, 0x283B57CC, 0xF8D56629, 0x79132E28,
0x785F0191, 0xED756055, 0xF7960E44, 0xE3D35E8C,
0x15056DD4, 0x88F46DBA, 0x03A16125, 0x0564F0BD,
0xC3EB9E15, 0x3C9057A2, 0x97271AEC, 0xA93A072A,
0x1B3F6D9B, 0x1E6321F5, 0xF59C66FB, 0x26DCF319,
0x7533D928, 0xB155FDF5, 0x03563482, 0x8ABA3CBB,
0x28517711, 0xC20AD9F8, 0xABCC5167, 0xCCAD925F,
0x4DE81751, 0x3830DC8E, 0x379D5862, 0x9320F991,
0xEA7A90C2, 0xFB3E7BCE, 0x5121CE64, 0x774FBE32,
0xA8B6E37E, 0xC3293D46, 0x48DE5369, 0x6413E680,
0xA2AE0810, 0xDD6DB224, 0x69852DFD, 0x09072166,
0xB39A460A, 0x6445C0DD, 0x586CDECF, 0x1C20C8AE,
0x5BBEF7DD, 0x1B588D40, 0xCCD2017F, 0x6BB4E3BB,
0xDDA26A7E, 0x3A59FF45, 0x3E350A44, 0xBCB4CDD5,
0x72EACEA8, 0xFA6484BB, 0x8D6612AE, 0xBF3C6F47,
0xD29BE463, 0x542F5D9E, 0xAEC2771B, 0xF64E6370,
0x740E0D8D, 0xE75B1357, 0xF8721671, 0xAF537D5D,
0x4040CB08, 0x4EB4E2CC, 0x34D2466A, 0x0115AF84,
0xE1B00428, 0x95983A1D, 0x06B89FB4, 0xCE6EA048,
0x6F3F3B82, 0x3520AB82, 0x011A1D4B, 0x277227F8,
0x611560B1, 0xE7933FDC, 0xBB3A792B, 0x344525BD,
0xA08839E1, 0x51CE794B, 0x2F32C9B7, 0xA01FBAC9,
0xE01CC87E, 0xBCC7D1F6, 0xCF0111C3, 0xA1E8AAC7,
0x1A908749, 0xD44FBD9A, 0xD0DADECB, 0xD50ADA38,
0x0339C32A, 0xC6913667, 0x8DF9317C, 0xE0B12B4F,
0xF79E59B7, 0x43F5BB3A, 0xF2D519FF, 0x27D9459C,
0xBF97222C, 0x15E6FC2A, 0x0F91FC71, 0x9B941525,
0xFAE59361, 0xCEB69CEB, 0xC2A86459, 0x12BAA8D1,
0xB6C1075E, 0xE3056A0C, 0x10D25065, 0xCB03A442,
0xE0EC6E0E, 0x1698DB3B, 0x4C98A0BE, 0x3278E964,
0x9F1F9532, 0xE0D392DF, 0xD3A0342B, 0x8971F21E,
0x1B0A7441, 0x4BA3348C, 0xC5BE7120, 0xC37632D8,
0xDF359F8D, 0x9B992F2E, 0xE60B6F47, 0x0FE3F11D,
```

```
        0xE54CDA54, 0x1EDAD891, 0xCE6279CF, 0xCD3E7E6F,
        0x1618B166, 0xFD2C1D05, 0x848FD2C5, 0xF6FB2299,
        0xF523F357, 0xA6327623, 0x93A83531, 0x56CCCD02,
        0xACF08162, 0x5A75EBB5, 0x6E163697, 0x88D273CC,
        0xDE966292, 0x81B949D0, 0x4C50901B, 0x71C65614,
        0xE6C6C7BD, 0x327A140A, 0x45E1D006, 0xC3F27B9A,
        0xC9AA53FD, 0x62A80F00, 0xBB25BFE2, 0x35BDD2F6,
        0x71126905, 0xB2040222, 0xB6CBCF7C, 0xCD769C2B,
        0x53113EC0, 0x1640E3D3, 0x38ABBD60, 0x2547ADF0,
        0xBA38209C, 0xF746CE76, 0x77AFA1C5, 0x20756060,
        0x85CBFE4E, 0x8AE88DD8, 0x7AAAF9B0, 0x4CF9AA7E,
        0x1948C25C, 0x02FB8A8C, 0x01C36AE4, 0xD6EBE1F9,
        0x90D4F869, 0xA65CDEA0, 0x3F09252D, 0xC208E69F,
        0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6
    ]
]

key = [ 0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
      0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
      0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
      0x193602A5, 0x75094C29]

p_new = p.copy()

def swap(a,b):
   temp = a
   a = b
   b = temp
   return a,b

def blowfish_encrypt(data):
        #break plaintext according to leftmost and rightmost 32 bits
   L = data>>32
   R = data & 0xffffffff
   for i in range(0,16):
      L = p[i]^L
      L1 = func(L)
      R = R^func(L1)
      L,R = swap(L,R)
   #undo swap
   L,R = swap(L,R)
   L = L^p[17]
   R = R^p[16]
   #push l bits and r bits
   encrypted = (L<<32) ^ R
   return encrypted
```

```
#round function
def func(L):
    temp = s[0][L >> 24]
    temp = (temp + s[1][L >> 16 & 0xff]) % 2**32
    temp = temp ^ s[2][L >> 8 & 0xff]
    temp = (temp + s[3][L & 0xff]) % 2**32
    return temp

def blowfish_decrypt(data):
        #break plaintext according to leftmost and rightmost 32 bits
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
        L = p[i]^L
        L1 = func(L)
        R = R^func(L1)
        L,R = swap(L,R)
    #undo swap
    L,R = swap(L,R)
    L = L^p[0]
    R = R^p[1]
    decrypted_data1 = (L<<32) ^ R
    return decrypted_data1

for i in range(0,18):
    p[i] = p[i]^key[i%14]
k = 0
data = 0
for i in range(0,9):
    temp = blowfish_encrypt(data)
    p[k] = temp >> 32
    k+=1
    p[k] = temp & 0xffffffff
    k+=1
    data = temp

#this encryption uses 5 rounds and returns a list containing ciphertext
#plaintext is a list
def cbcencrypt(plaintext):
    ciphertext = []
    for i in range(0,5):
        #xor operation
        if i == 0:
            plaintext[i] = plaintext[i]^IV
        else:
            plaintext[i] = plaintext[i]^ciphertext[i-1]
        ciphertext.append(blowfish_encrypt(plaintext[i]))
```

```
    return ciphertext

#cbc decryption returns plaintext in the form of a list
def cbcdecrypt(ciphertext):
    plaintext = []
    for i in range(0,5):
        ct = blowfish_decrypt(ciphertext[i])
        if i == 0:
            plaintext.append(ct^IV)
        else:
            plaintext.append(ct^ciphertext[i-1])
    return plaintext

#for ofb a single 64 bit integer was transformed into 16 bits blocks
#s == 16 and b == 64
def ofbencrypt(plaintext):
    ciphertext = []
    register = IV
    uwu = IV
    for i in range(0,4):
            if i>0:
                        #for shifting of register
                        l = register & (2**48 - 1)
                        r = uwu
                        register = (l<<16)^r
            uwu = blowfish_encrypt(register)
            # take leftmost s bits
            uwu = uwu>>48
            #breaking plaintext into blocks by taking leftmost s bits and pushing the remaining bits
            r = plaintext & (2**48 - 1)
            l = plaintext>>48
            ciphertext.append(uwu^l)
            plaintext = r<<16
    return ciphertext


def ofbdecrypt(ciphertext):
    plaintext = 0
    register = IV
    uwu = IV
    for i in range(0,4):
        if i>0:
            #shifting of register
            l = register & (2**48 - 1)
            r = uwu
            register = (l<<16)^r
        uwu = blowfish_encrypt(register)
```

```
        #take leftmost s bits
        uwu = uwu>>48
        #reconstructing plaintext by pushing 16 bit blocks right to left
        add = (ciphertext[i]^uwu)
        plaintext = plaintext<<16
        plaintext+=add
    return plaintext

#driver function
def main():
    plain_text = []
    cipher_text_cbc = []
    cipher_text_ofb = []
    plain_text_cbc = []
    plain_text_ofb = 0
    plain_text = list(map(int,input("Yos! Enter 5 numbers not exceeding 64 bits for cbc mode:
").split()))
    plain_text_1 = int(input("Now enter one integer not exceeding 64 bits for OFB mode: "))
    cipher_text_cbc = cbcencrypt(plain_text)
    cipher_text_ofb = ofbencrypt(plain_text_1)
    plain_text_cbc = cbcdecrypt(cipher_text_cbc)
    plain_text_ofb = ofbdecrypt(cipher_text_ofb)
    print("CBC MODE:" + '\n')
    print(str(cipher_text_cbc) + '\n')
    print(str(plain_text_cbc) + '\n')
    print("OFB MODE:" + '\n')
    print(str(cipher_text_ofb) + '\n')
    print(str(plain_text_ofb) + '\n')

if __name__ == "__main__":
    main()
```

Screenshots of output:

Below is the screenshot of my terminal in the Ubuntu operating system. In which we can see the output for CBC mode and OFB mode separately.

gaurav@gaurav: ~/Desktop/Network security_assignment

File Edit View Search Terminal Help

```
(base) gaurav@gaurav:~/Desktop/Network security_assignment$ python block.py
Hello! Enter 5 numbers not exceeding 64 bits: 1 2 3 4 5
Now enter one integer not exceeding 64 bits: 15
CBC MODE:

[3780344556988293572, 16076579887062894640, 17085020721142942656, 13630855368809
859849, 18315971078175772672]

[1, 2, 3, 4, 5]

OFB MODE:

[62586, 52765, 47353, 16057]

15

(base) gaurav@gaurav:~/Desktop/Network security_assignment$
```

gaurav@gaurav: ~/Desktop/Network security_assignment

File Edit View Search Terminal Help

```
(base) gaurav@gaurav:~/Desktop/Network security_assignment$ python block.py
Hello! Enter 5 numbers not exceeding 64 bits: 48 59 54 99 54
Now enter one integer not exceeding 64 bits: 595996
CBC MODE:

[10357059628092617279, 13801367789738876147, 10855064722787196262, 1121196103391
9861050, 13898518692077684859]

[48, 59, 54, 99, 54]

OFB MODE:

[62586, 52765, 47344, 9898]

595996

(base) gaurav@gaurav:~/Desktop/Network security_assignment$
```

Practical Assignment - 3

Name : Gaurav Singh
Roll no : 17085035
Department : Electrical Engineering

Objective:
In the post-pandemic world the election commission decides to hold elections online to maintain social distancing. The task is to design and implement a secure protocol used for online elections that both maintains individual privacy and prevents cheating.

Theory:

[A]Simplistic Voting Protocol #1

(1) Each voter signs his vote with his private key.
(2) Each voter encrypts his signed vote with the CTF's public key.
(3) Each voter sends his vote to the CTF.
(4) The CTF decrypts the votes, checks the signatures, tabulates the votes, and makes the results public.

Poll start and end time has to be specified. This protocol satisfies properties one and two: Only authorized voters can vote and no one can vote more than once—the CTF would record votes received in step (3). Each vote is signed with the voter's private key, so the CTF knows who voted, who didn't, and how often each voter voted. If a vote comes in that isn't signed by an eligible voter, or if a second vote comes in signed by a voter who has already voted (CTF maintains a flag), the facility ignores it. No one can change anyone else's vote either, even if they intercept it in step (3), because of the digital signature. The problem with this protocol is that the signature is attached to the vote; the CTF knows who voted for whom. Encrypting the votes with the CTF's public key prevents anyone from eavesdropping on the protocol and figuring out who voted for whom, but you have to trust the CTF completely. It's analogous to having an election officer starting over your shoulder in the voting booth.

Implementation Details:

Operating System : linux
Language used : Python
Library used : Tkinter,libnum,pycryptodome
Below is my source code written in Python

```
import tkinter as tk
from tkinter import *
from tkinter.ttk import *
import Crypto
from Crypto.Util import Counter
from Crypto import Random
from Crypto.Util.number import *
import libnum
```

```
root = Tk()
auth_user = ["Ravi", "Bob",  "Ram", "Ayush", "John", "Kelly"]
uwu = {"Ravi":390314650276167803, "Bob":483710515075373321,
"Ram":860209867719162361, "Ayush":581633013168578947, "John":640621416611553599,
"Kelly":488455929614637841}
candidate = ["Bob Ross", "Van Gogh"]
candidate_res = {"Bob Ross":0, "Van Gogh":0}
msg = StringVar()
msg2 = StringVar()
bits = 30

'''f = open("voterauth.txt","r")

for i in f:
    auth_user.append(i)

f.close()'''

# Function to generate key which will be stored in a separate txt file
def keygenerate():
    p = Crypto.Util.number.getPrime(bits,randfunc = Crypto.Random.get_random_bytes)
    q = Crypto.Util.number.getPrime(bits,randfunc= Crypto.Random.get_random_bytes)

    phi = (p-1)*(q-1)
    n = p*q

    e = 65537
    d = (libnum.invmod(e,phi))
    return ((e,n),(d,n))

def encrypt(privkkey,plaintext):
    key,n = privkkey
    cipher = [pow(ord(char),key,n) for char in plaintext]
    return cipher

def decrypt(pubkkey,ciphertext):
    key,n = pubkkey
    plain = [chr((char ** key) % n) for char in ciphertext]
    return ''.join(plain)

def public_enc(privkkey,data):
    key,n = privkkey
    S = [pow(i,key,n) for i in data]
    return S

def public_dec(pubkkey,data):
```

```
    key,n = pubkkey
    D = [pow(i,key,n) for i in data]
    return D

CTFpub,CTFpriv = keygenerate()
user_priv = (0,0)
user_pub = (0,0)

def check(oof,name):
    for i in oof:
        if i == name:
            return True
    return False

def getkey():
    msg2 = ip1.get("1.0",'end-1c')
    msg = ip3.get("1.0",'end-1c')
    msg = int(msg)
    pub = (65537,uwu[msg2])
    priv = (msg,uwu[msg2])
    return (pub,priv)


def BR():
    global  user_priv
    global user_pub
    msg = ip1.get("1.0",'end-1c')
    if check(auth_user,msg):
        if check(existing_user,msg) == False:
            user_pub,user_priv = getkey()
            pt = "Bob Ross"
            user_sign = encrypt(user_priv,pt)
            ct_enc = public_enc(CTFpub,user_sign)
            ct_dec = public_dec(CTFpriv,ct_enc)
            user_dec = decrypt(user_pub,ct_dec)
            if user_dec == pt:
                candidate_res["Bob Ross"]+=1
                existing_user.append(msg)

def VG():
    global user_priv
    global user_pub
    msg = ip1.get("1.0",'end-1c')
    if check(auth_user,msg):
        if check(existing_user,msg) == False:
            user_pub, user_priv = getkey()
            pt = "Van Gogh"
```

```
        user_sign = encrypt(user_priv,pt)
        ct_enc = public_enc(CTFpub,user_sign)
        ct_dec = public_dec(CTFpriv,ct_enc)
        user_dec = decrypt(user_pub,ct_dec)
        if user_dec == pt:
            candidate_res["Van Gogh"]+=1
            existing_user.append(msg)



def get_result():
    ip2.delete("1.0",'end-1c')
    for i in candidate:
        upd = i + " " + str(candidate_res[i])
        ip2.insert(END,upd + '\n')


root.geometry("1000x600")
root.title('Vote')

frame1 = Frame(root)
frame2 = Frame(root)

frame1.grid(row=0,column=0,sticky=W)
frame2.grid(row=0,column=1,sticky=E)

root.rowconfigure(0, weight=1)
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=1)

ip1 = Text(frame1,height=5,width=60,wrap=WORD)
ip1.grid(sticky=N)

ip3 = Text(frame1, height = 5, width=60, wrap = WORD)
ip3.grid(sticky=N)

br = tk.Button(frame1, text = "Bob Ross",bg="purple",fg="white", command = BR)
br.grid(sticky = S)
vg = tk.Button(frame1,text = "Van Gogh",bg="green",fg="white",command = VG)
vg.grid(sticky = S)

ip2 = Text(frame2, height = 4, width = 60, wrap = WORD)
ip2.grid(sticky=N)
result = tk.Button(frame2, text = "RANKING",bg="orange",fg="white",command = get_result)
result.grid(sticky = S)
root.mainloop()
```

Here is the python script to generate the key in a separate text file:

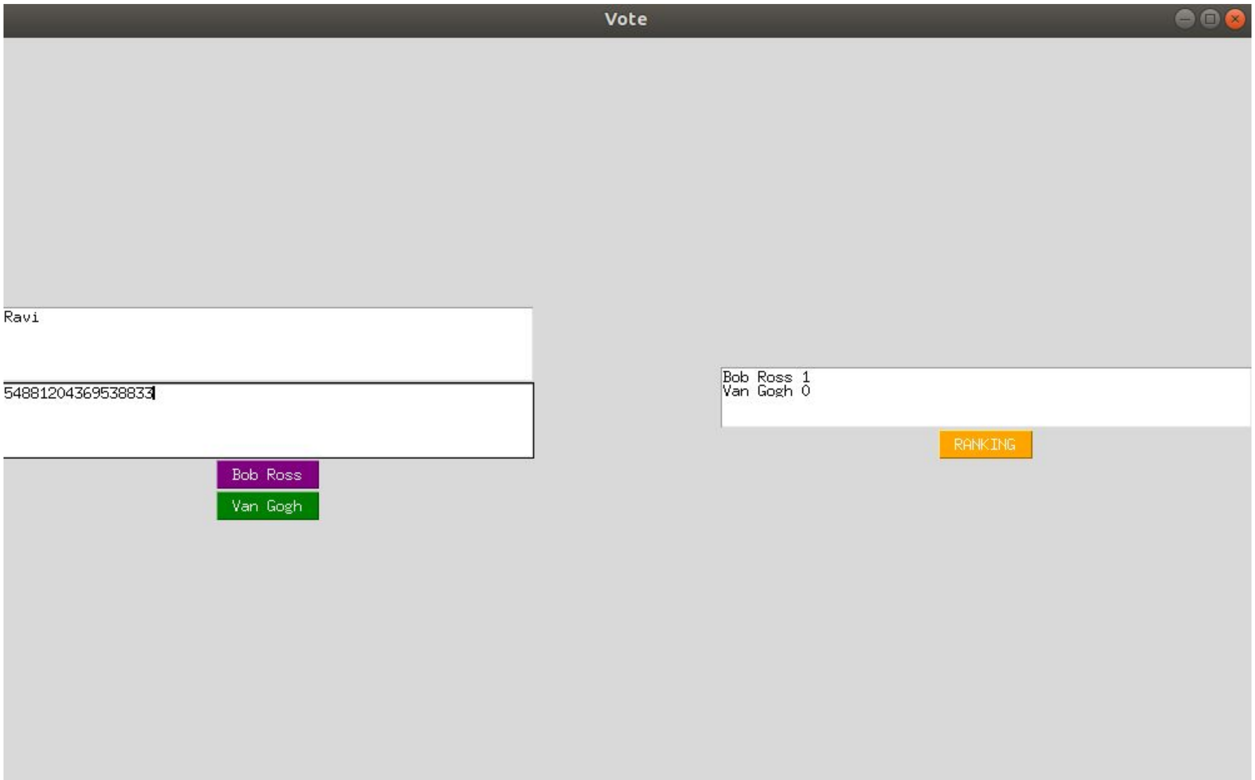
```
import Crypto
from Crypto.Util import Counter
from Crypto import Random
from Crypto.Util.number import *
import libnum

bits = 30

def keygenerate():
    p = Crypto.Util.number.getPrime(bits,randfunc =
Crypto.Random.get_random_bytes)
    q = Crypto.Util.number.getPrime(bits,randfunc=
Crypto.Random.get_random_bytes)

    phi = (p-1)*(q-1)
    n = p*q

    e = 65537
    d = (libnum.invmod(e,phi))
    return ((e,n),(d,n))

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

for i in range(6):
    ((e,n),(d,n)) = keygenerate()
    f.write(str(d) + " " + str(n)+ "\n")
f.close()
```

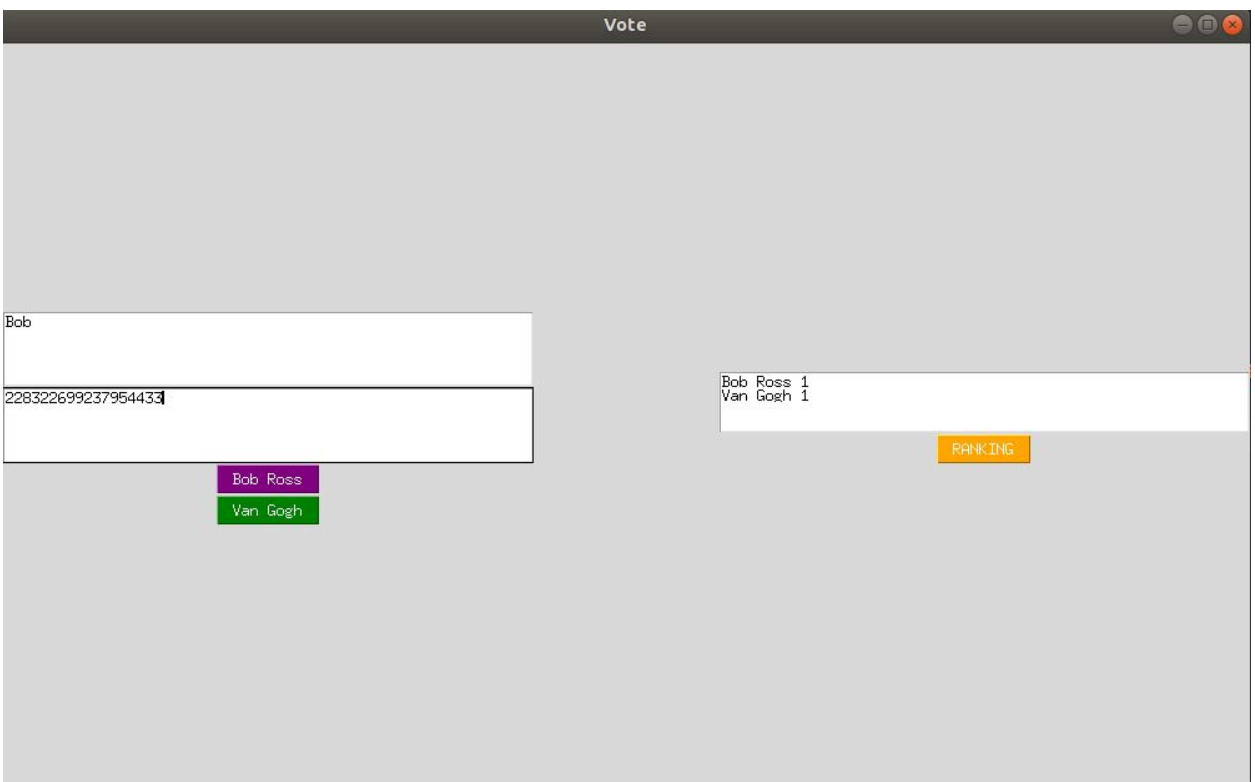
Here is the text file after running the above script:

54881204369538833 390314650276167803
228322699237954433 483710515075373321
432329715450147841 860209867719162361
540977186275161921 581633013168578947
12795419872360025 640621416611553599
341047878590691473 488455929614637841

Screenshots:

We can see that after clicking the buttons multiple times also the output does not change and hence only one per can vote only once and we even if enter a wrong username or wrong private key i.e other than auth_user then his or her vote will not be counted.And every user has his own private key which is generated using the python script and written in a separate text file.


Vote
Ravi
54881204369538833
Bob Ross
Van Gogh
Bob Ross 1
Van Gogh 0
RANKING


Vote
Bob
228322699237954433
Bob Ross
Van Gogh
Bob Ross 1
Van Gogh 1
RANKING