PRACTICAL ASSIGNMENT

Course: Network Security

Name:- Aakash Mathur
Dept.:- Electronicd Engineering
Roll No.:- 17095001

PA0

Date:-22-5-2020
Name:- Aakash Mathur
Dept.:- Electronicd Engineering
Roll No.:- 17095001
Framework=Jupyter Notebook

Objective

To create a UI that perfroms Atbash cipher

Theory

Atbash cipher is a substitution cipher with just one specific key where all the letters are reversed that is A to Z and Z to A. It was originally used to encode the Hebrew alphabets but it can be modified to encode any alphabet.

A	B	C	D	E	F	G	H	I	J	K	L	M
Z	Y	X	W	V	U	T	S	R	Q	P	O	N

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
M	L	K	J	I	H	G	F	E	D	C	B	A

Basically, We have to replace each letter with the letter in the same position starting from the back of the alphabet series.
b->a
B->A

We can change it a little and follow the same for numbers
For eg.: 0->9, 1->8, 2->7,...., 9->9.

Code

```
import tkinter as tk
from tkinter import *
def show():# command that is called when "GO" is pressed
    f1=fu.get()
    if f1==1:
        outext.delete(0.0,END)# clears the output box
        i=intext.get("0.0",END)# takes input from intext box
        st=""
        for char in i:
            ch=char
            if char>='a' and char<='z':# char is between a-z invert
them z-a
                ch=chr(ord('z')-ord(char)+ord('a'))
            elif char>='A' and char<='Z':# char is between A-Z invert
them Z-A
                ch=chr(ord('Z')-ord(char)+ord('A'))
            elif char>='0' and char<='9':# char is between 0-9 invert
them 9-0
                ch=chr(ord('9')-ord(char)+ord('0'))

            st=st+ch
        outext.insert(0.0,st)#prints the text
    elif f1==2:# same function as above but linked to other button
        intext.delete(0.0,END)
        i=outext.get("0.0",END)
        st=""
        for char in i:
            ch=char
            if char>='a' and char<='z':
                ch=chr(ord('z')-ord(char)+ord('a'))
            elif char>='A' and char<='Z':
                ch=chr(ord('Z')-ord(char)+ord('A'))
            elif char>='0' and char<='9':
                ch=chr(ord('9')-ord(char)+ord('0'))
```

```
            st=st+ch
        intext.insert(0.0,st)



m= tk.Tk(screenName=None,  baseName=None,  className='Crpyter',
useTk=1)
# the Ui designing part for the functions

# creating the labels
l1=Label(m, text="Encrypted ")
l2=Label(m,text="Decrypted")
l3=Label(m,text="Function")
#positioning the labels
l1.grid(row=0,column=0)
l2.grid(row=0,column=2)
l3.grid(row=1,column=1)
#varible for int values
fu=IntVar()
# radiobuttons for selecting the functions
rd1=Radiobutton(m,text="Encrypt",variable=fu,value=1)
rd2=Radiobutton(m,text="Decrypt",variable=fu,value=2)
# positioning the radiobuttons
rd1.grid(row=1,column=0)
rd2.grid(row=1,column=2)
#inputs and output boxes and positioning them
intext=Text(m,width=30,height=20,wrap=WORD)
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wrap=WORD)
outext.grid(row=3,column=2)
# go button calls the function show
act=Button(m,text="GO",command=show)
act.grid(row=3,column=1)
# loops to lets the code continuously run
m.mainloop()
```

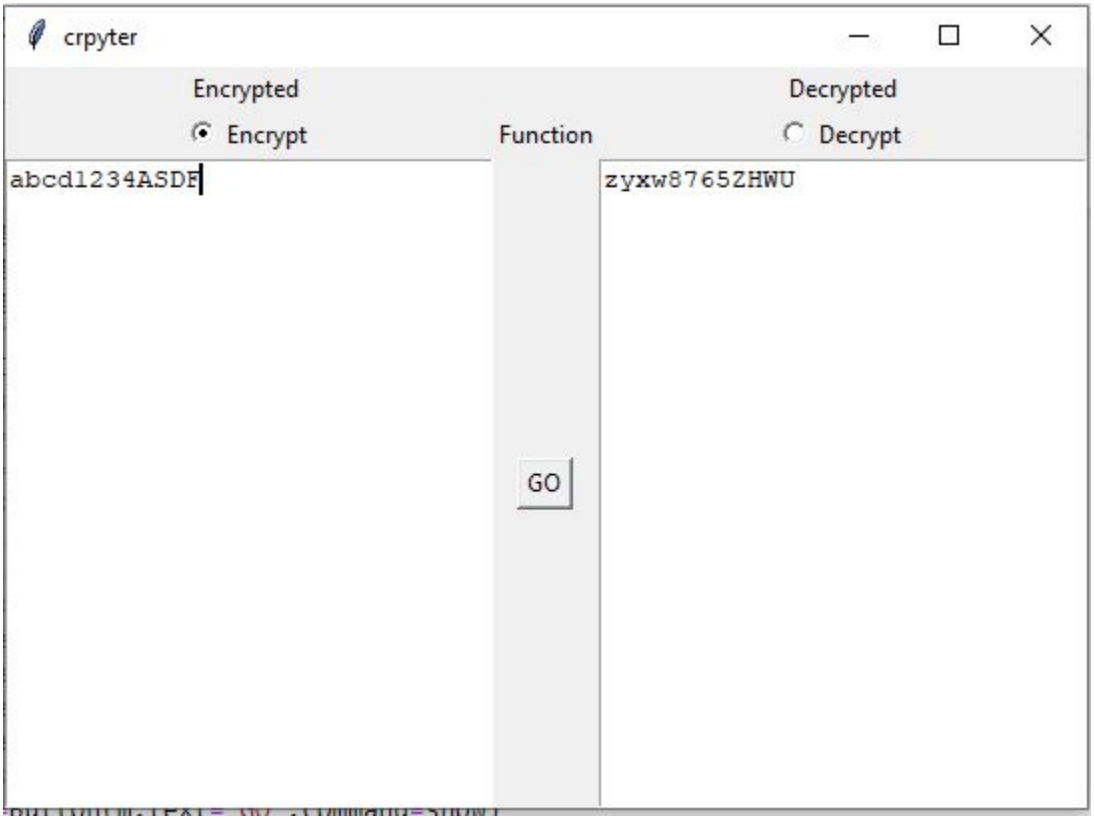
A WALK-THROUGH THE CODE

The code uses tkinter library to create a simple UI .
The function “Show” is called when we press
the button “GO” .

Also we choose what function we want to performed i.e. encrypt or decrypt otherwise “GO”
doesn’t work.
The encrypt function can work as decrypt function too if applied on the ciphertext.
Thus decrypt function is the same as the encrypt function.

Function:

The show function takes input the string of characters and check in which range every characters falls and accordingly it replaces that charater using if statements



Encryptor UI

PA-1

Date: 22/05/2020
Name: Aakash Mathur
Roll No: 17095001
Department: Electronics Engineering
Framework=Jupyter Notebook

Objective:

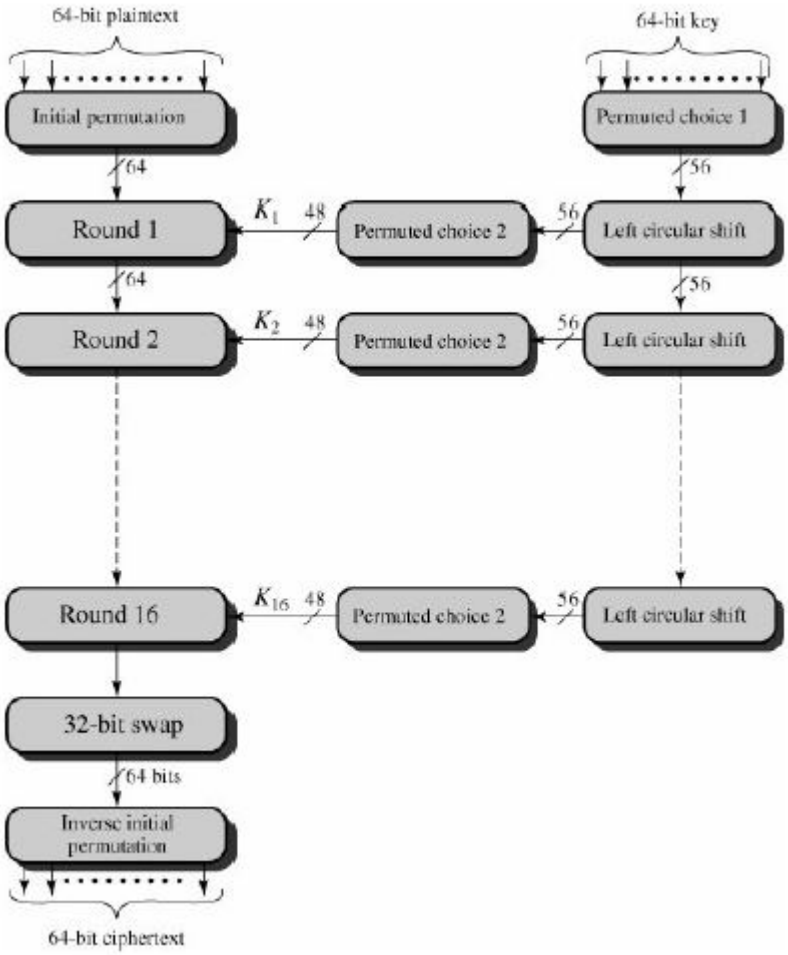
The objective is to implement and study DES algorithm with different parameters of block size(16,32,64 bit) and round numbers, accessible by a UI . Also to show the avalanche effect and the nature of weak keys .

Theory:

The Data Encryption Standard is a symmetric-key algorithm for the encryption of digital data. DES follows the feistel structure where a similar function is performed multiple times till the entropy reaches a certain level . In DES we use a 64 bit text block(2*32),64 bit key(56 effective) and 16 rounds as standard.
DES takes a 64 bit plain text which is split into two 32 bit blocks the output of each round is a 64 bit block of which one block is preserved each round while another is manipulated.

Feistel Structure of DES:

DES follows the Feistel structure in which a certain function is repeacted multiple times in rounds thus each round helps in increasing the unpredicatablisity of the message and is simple to implementation.
Feistel structure have the property that they need very less memory as repeated structure helps in using the same memory again and again.
Also in feistel structure certrain part of message is preserved for other round



The complete algorithm can be divided into following parts:

1. Key generation:

The algorithm is provided with a key input initially of 64 bits . The key generation process takes the 64 bit key , and reduces it into an effective key of 56bit in the beginning using a selection box . For the further rounds the 56 bits are split into size of 28 bit each goes through a particular amount of left and right rotations respectively . These rotated values are used for the future

rounds also in this round these are appended and passed through another permutation and selection box
which reduces it to a size of 48 bit single box.

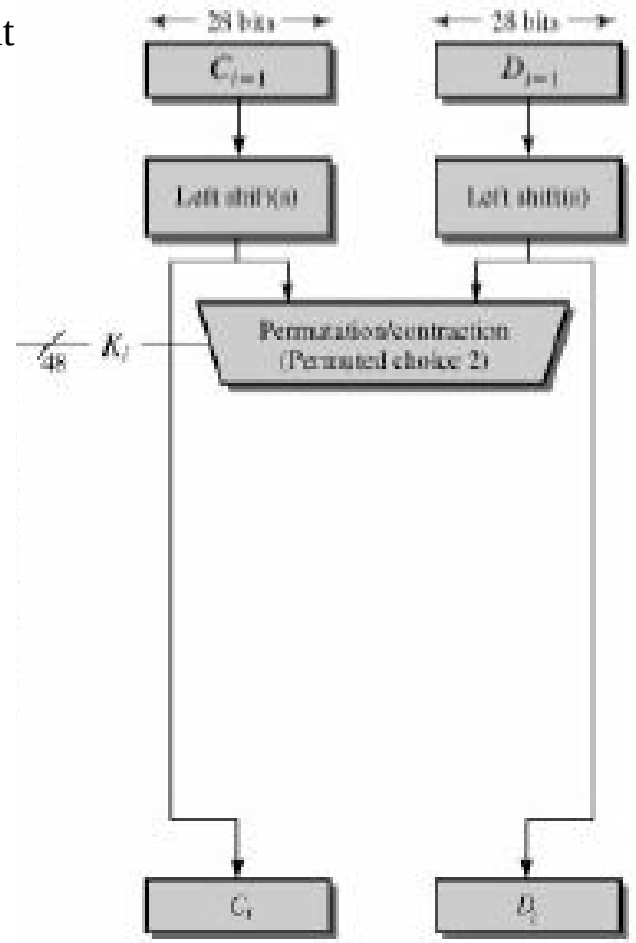


Fig: Key generation process.

2. Encryption process

The encryption process takes input of the round key and the input text blocks and outputs manipulated blocks of which one block is preserved as it is. The steps can be divided as follows.

a. Initial permutation : Before performing the first round the plain text is taken as input split into blocks of size 64 bits using padding if required and each 64 bit block is split into 2 blocks of 32 bits for rounds.

Round function:

in each round one of the block is preserved while the others passes through the round function . The round function has the following part

b. Expansion/permutation: The process takes a 32 bit half of the data and expand it to 48 bits to make it suitable for the keys to be used.
c. XORing : The expanded output is xored with the round key we produce every round.

d. Substitution : This is the only function which is not linear and provides large entropy in the data .
The sbox mapping process is used in which the sboxes are fixed world wide as they provides the best scheme.

A 48 bit block is broken into 8 blocks of 6 bits the first and the last bits are used to choose the rows while the middle 4 bits are used to choose the column and the 6 bits are replaced by the 4 bit number present at the point.

S_1	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

Fig: One of the 8 s boxes each used by one block

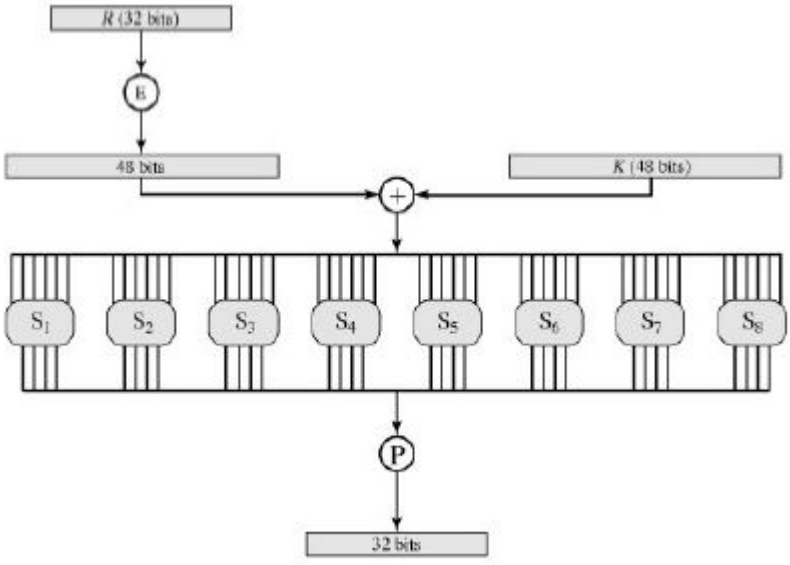


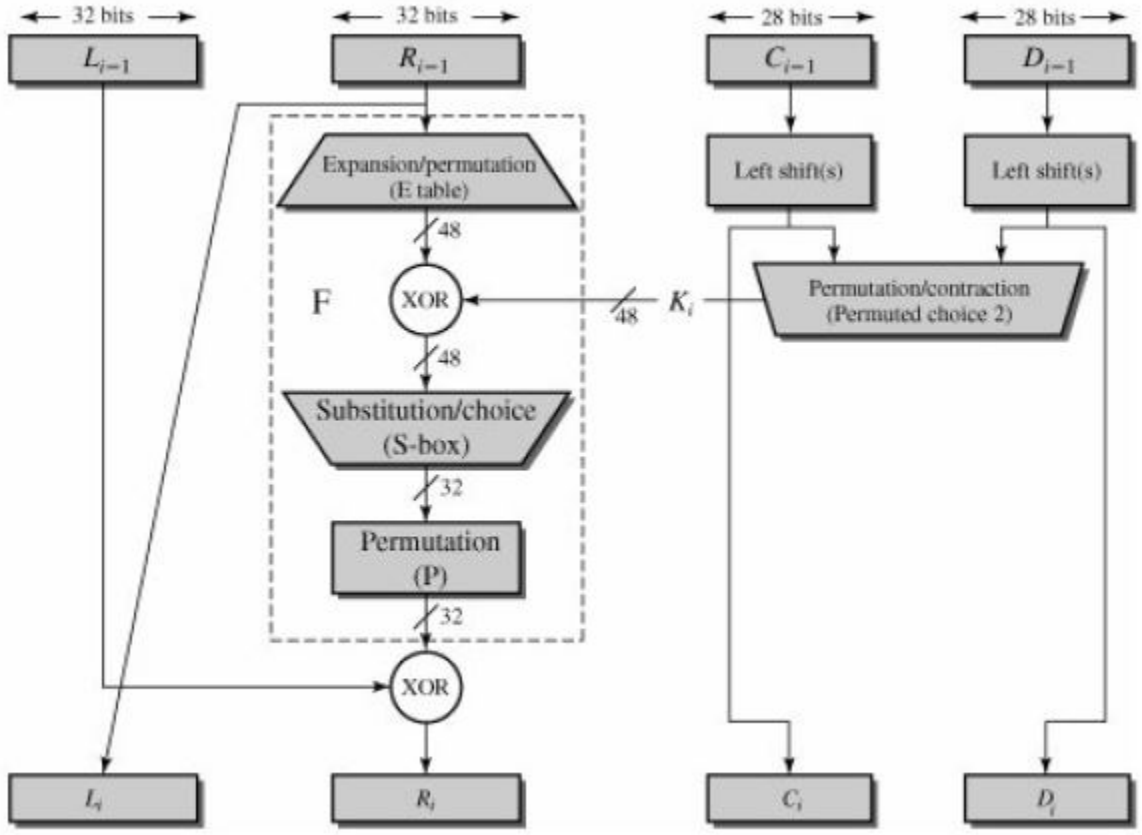
Fig : The substitution mechanism

e. Permutation/Compression: The 48 bit block is scrambled and 32 bits are choosen from it and this gives the output of the fiestel function.

f. Xoring/Switching: The output of the function is Xored with the other half block of 32 bit size and the block which went through the round function is preserved and the other is replaced by the output of the xor .Later there places are also switched.

These are the major steps in DES and are carried in rounds to increase randomness and reduce linearity and predictability.

g. Swapping and inverse permutation: At the last step the bits are swapped and a permutation box which is reverse of the original permutation box is used to scramble again .These 2 steps helps making the process identical for decryption and encryption.



Implementation:

The first important step is to generate the Tables(Permutation of plain Text block, Inverse Permutation, Initial permutation made on the key, Permutation applied on the shifted key to get Ki+1, Expand matrix to get a 48bits matrix of data to apply the xor with Ki, S-Boxes, Permutation made after each SBox substitution for each round, Shift table) according to the size of the

half-width as they change according to it.

After Generating these tables the next step is to use them in the DES algorithm but we have to keep this in mind that we have to implement DES separately for various possible Half-widths

The key and plain text should be of the same size as in the classical DES. (although we can pad them)

Now the most important Functions in DES implementation are The encrypt function and Key Generation function.

Code(Full code in appendix)

1. DES

```
1)  def run(self, key, text, action=ENCRYPT, padding=False):
2)      #run process is our des implementation
3)      # if the key is small an error occur, if large we take the first required bits
4)      if len(key) < kinw:
5)          raise "Key Should be 8 bytes long"
6)      elif len(key) > kinw:
7)          key = key[:kinw]
8)
9)      lx = len(text)
10)     if (lx%kinw)!=0 :
11)         for i in range(0,kinw-lx%kinw) :
12)             text=text+" "
13)     self.password = key
```

```
14)         self.text = text
15)
16)
17)         if padding and action==ENCRYPT:
18)             self.addPadding()
19)         elif len(self.text) % kinw != 0:
20)             raise "Data size should be multiple of 8"
21)
22)         self.generatekeys()
23)         # we generate all the keys required for the rounds by the above function
24)         text_blocks = nsplit(self.text, kinw)
25)         # create blocks of the plaintext of required size
26)
27)         result = list()
28)
29)         for block in text_blocks:
30)             acc.clear()
31)             for i in range(0,roun+1):
32)                 acc.append([])
33)             # acc contains the result after each round so that we can use it later in avalanche
    plots
34)             block = string_to_bit_array(block)#Convert the block in bit array
35)             acc[0]=block
36)             block = self.permut(block,PI)
37)             # it performs the initial permuation on the plain text
38)             g, d = nsplit(block, w)
39)             # create the two halves
40)             tmp = None
41)             # below is the rounds of des performs
42)             for i in range(0,roun):
43)
44)                 d_e = self.expand(d, E)#expansion of the second half
45)                 if action == ENCRYPT:
46)                     tmp = self.xor(self.keys[i], d_e)#xoring with the key
47)                 else:
48)                     tmp = self.xor(self.keys[roun-1-i], d_e)# for decrytion the keys are in reverse
49)                 tmp = self.substitute(tmp)# substition using s box
50)                 tmp = self.permut(tmp, P)# permuation choice of the values
51)                 tmp = self.xor(g, tmp)
52)                 g = d
53)                 d = tmp
54)                 acc[i+1]=self.permut(d+g, PI_1)
55)             # des rounds are performed above
56)
57)             result += self.permut(d+g, PI_1)
58)         final_res = bit_array_to_string(result)
59)         if padding and action==DECRYPT:
60)             return self.removePadding(final_res)
61)         else:
```

```
62)         return final_res
```

Line wise description of important code:
lines 2-8 : takes the key and choose the required key if key is small its discarded
line 13-14 : Set the values of text and key from input
line 17-20: Add padding if required
line 22: Generate all the keys for the upcoming rounds and store in keys[]
line 24 : Split the long text into blocks of required length(for example 64 bits)
line 42-57: The loops is run for each block for the DES. In each round we select a half of the message block expand it(line 44) ,xor it with round-key(45-48),substitute using sboxes(49),permute and choose(50),xor with the other half(51),swap the result and store for future use(21-56)

2.Key Generation:

```
1.  def generatekeys(self):#Algorithm that generates all the keys
2.      self.keys = []
3.      key = string_to_bit_array(self.password)
4.      key = self.permut(key, CP_1)
5.      g, d = nsplit(key, kfrh)
6.      for i in range(roun):
7.          g, d = self.shift(g, d, SHIFT[i])
8.          tmp = g + d
9.          self.keys.append(self.permut(tmp, CP_2))
10.
```

Line wise description of important code:
line 2: Create a list which stores all the round keys
line 3: The text key is converted into bits
line 4. The key is passed through the choice permutation box (CP_1)which reduces the size and scramble
line 5-9.Create 2 halves and in loop we shifts each half by some bits described in SHIFT[] list , for each round we join the shifter key halves and permute and choose some bits using CP_2

OUTPUTS AND RESULT:

We created a simple UI which can perform the functions:

Each character is coded into 8 bit value Total 256 character in out alphabet set

1. Guide to use the UI:

a. **For simple encrypt and decrypt : (Input only in the left box right box only to display output)**

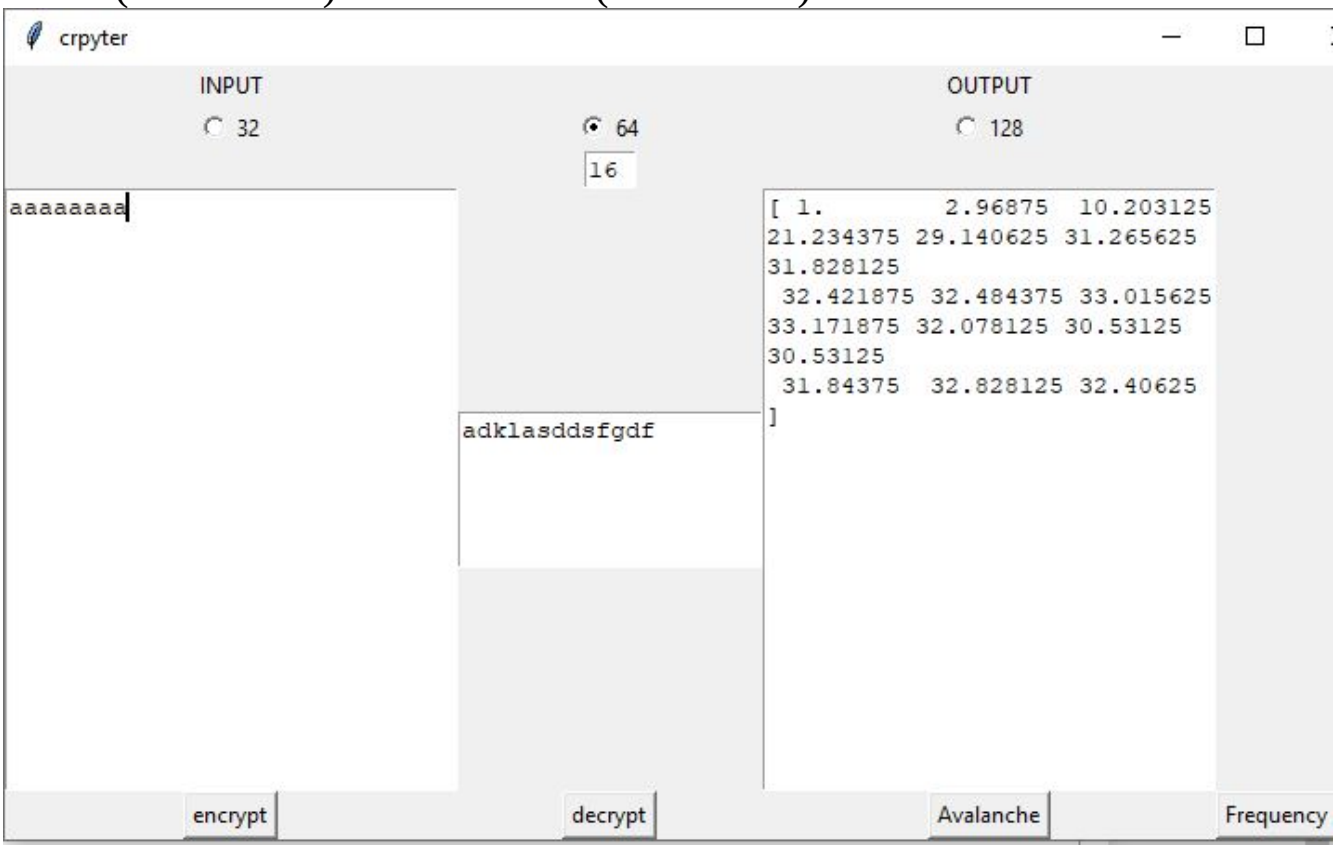
Input your text in the left inbox , choose the block size here 64 bit(32+32), input round number in the small box , Input the key in the middle large box (**make sure the size of key is greater than block size i.e. (block size/8)**) Press the appropriate button for encrypt or decrypt.

Imp* some characters are not visible because of tkinter limitation they can still be copied to check decryption copy it from outbox(right),put in the inbox

and use the same key to decrypt

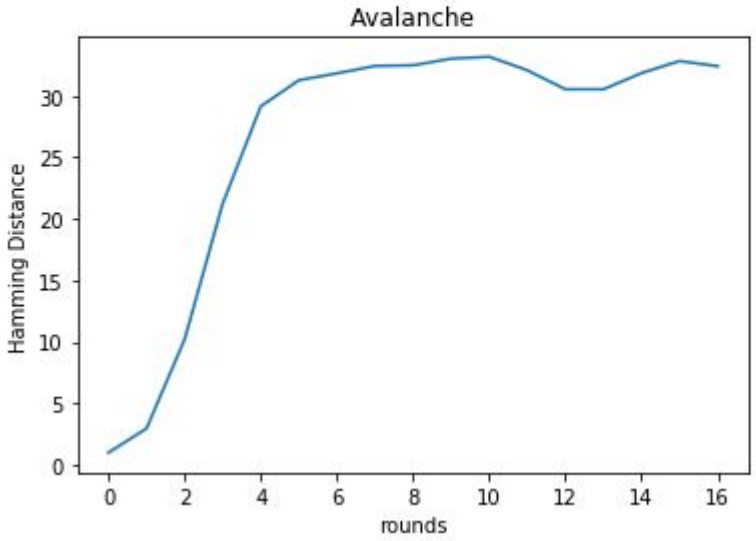
b.Avalanche and Frequency plot:

for these plots make sure the size of the text size equals to the size of blocks(blocksize/8) here 64/8=len(“aaaaaaaa”)



In the outbox we print the change in the average hamming distance through different rounds . It is same on the plot.

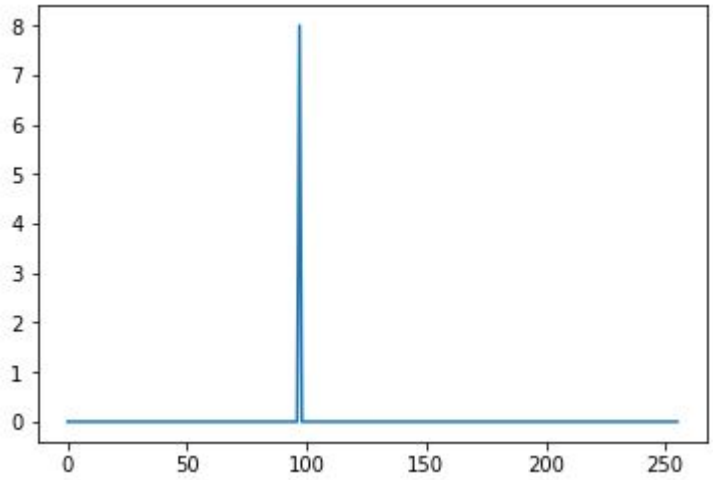
```
[ 1.        2.96875  10.203125 21.234375 29.140625 31.265625 31.828125
 32.421875 32.484375 33.015625 33.171875 32.078125 30.53125  30.53125
 31.84375  32.828125 32.40625 ]
```



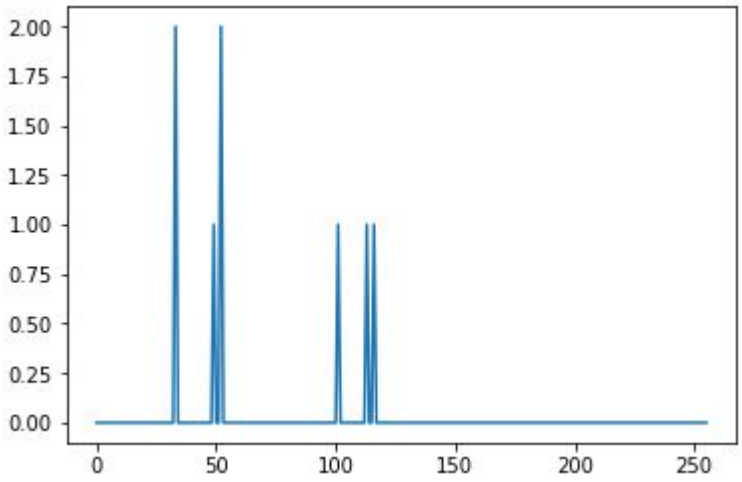
<u>Frequency plots:</u>

These shows the variation of the frequency of character through rounds (there are 256 character in our set)

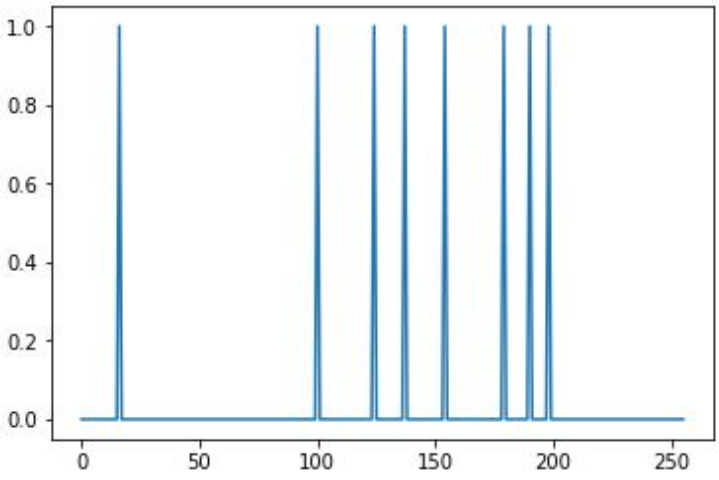
Round 0 : “aaaaaaaa”

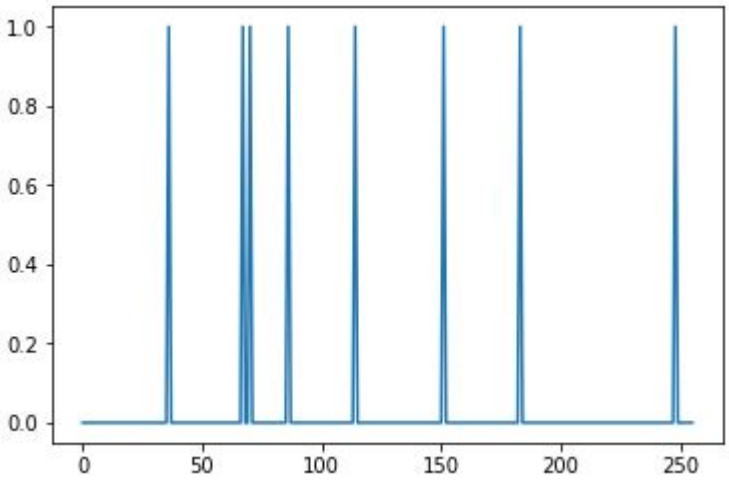


Round 1



Round 12:





Round 16

Conclusion :

From the observed pattern of an avalanche we can say that as we increase the size of half-width the confusion increases and as a result of the avalanche also peaks at a much higher value(at max (blocksize)/2) .We can also check the weak keys by checking the round keys generated using the weak key and we observe that after 1 or 2 rounds they start becoming same and thus do not add any confusion to the algorithm

Appendix

```
# initial permuatation box for the plain text
#PI1 for the 64 bit block size PI2 for 32 bit and PI 3 for 128 bit
PI1 = [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]
PI2=[27, 18, 13, 21, 19, 7, 28, 31, 26, 15, 32, 24, 3, 4, 22, 1, 11, 30, 5, 6, 25, 12, 20, 16, 10, 14, 17, 8, 9,
23, 29, 2]
PI3=[29,116,91,22,9,61,101,92,37,98,1,109,5,125,73,10,77,7,120,16,114,72,51,111,127,75,13,2,97,110
,49,52,123,121,108,40,45,70,99,85,90,34,124,47,119,118,96,94,112,107,31,126,81,115,105,17,88,68,7
4,103,102,59,79,57,128,28,86,93,66,83,104,117,21,60,19,106,58,71,89,84,53,26,95,55,46,62,80,20,44,
78,35,42,67,82,48,4,64,43,38,100,56,54,32,69,87,25,15,30,76,65,36,24,41,63,23,50,18,33,14,12,27,113
,39,122,6,11,3,8]
PI=PI3
#first permuation choice box for the key generation (1 for 64 bit,2 for 32 bit ,3 for 128 bit)
CP_11 = [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]
CP_12 = [12,5,6,19,20,3,30,4,26,25,31,22,23,18,21,15,14,28,1,13,27,17,10,7,11,9,2,29]
CP_13 =
[37,43,1,107,13,82,21,121,11,10,77,47,103,85,79,116,4,65,27,22,101,71,25,34,57,55,126,84,108,106,8
1,124,98,36,49,110,19,35,42,92,20,41,90,94,89,105,125,7,5,53,3,100,97,102,74,99,62,111,69,93,122,1
18,87,46,29,113,60,95,78,68,75,63,54,52,33,86,70,123,45,117,73,67,66,51,127,76,50,58,91,39,23,30,3
1,28,59,26,17,119,61,115,18,38,14,44,15,12,9,2,6,109,114,83]
CP_1=CP_13
```

```
# second permutation choice box for the key generation (1 for 64 bit,2 for 32 bit ,3 for 128 bit)
CP_21 = [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]
CP_22 = [24,1,20,10,4,3,6,25,26,22,21,14,28,18,15,7,16,19,17,27,12,23,8,5]
CP_23 =
[78,1,94,50,10,23,66,64,14,103,4,32,34,28,22,39,46,79,80,25,96,111,105,33,6,58,99,102,98,38,70,108,
104,112,40,109,84,100,83,74,92,77,90,72,73,68,86,110,65,91,56,97,75,9,87,76,57,61,81,13,82,85,60,6
2,69,89,48,19,88,106,49,29,52,63,42,55,54,41,51,43,95,93,71,35,67,36,101,59,47,107,44,37,26,3,27,30
,20,17,8,21,18,24,15,53,11,31,12,16,2,7,5,45]
CP_2=CP_23
# expansion box for the plain text (1 for 64 bit,2 for 32 bit ,3 for 128 bit)
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]
E2 = [16, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 1]
E3 = [64,1,2,3,4,5,
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,
28,29,30,31,32,33,
32,33,34,35,36,37,
36,37,38,39,40,41,
40,41,42,43,44,45,
44,45,46,47,48,49,
48,49,50,51,52,53,
52,53,54,55,56,57,
56,57,58,59,60,61,
60,61,62,63,64,1]
E=E3

# sboxes 64 bit requires 8 of them 32 bit 4 of them and 128 bit 16 of them (the first 8/4/16 boxes are
taken)
S_BOX = [
```

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

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

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

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


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

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

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


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

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

]

# permutation choice box for used after xor with key
P1 = [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]
P2 = [12,3,4,2,16,1,8,9,14,10,6,15,7,5,13,11]
```

```
P3 =
[56 ,37 ,30 ,61 ,4 ,55 ,10 ,52 ,16 ,9 ,58 ,7 ,36 ,34 ,51 ,23 ,40 ,64 ,42 ,19 ,20 ,45 ,47 ,17 ,24 ,54 ,14 ,49 ,2
8 ,59 ,50 ,31 ,48 ,62 ,60 ,39 ,46 ,29 ,27 ,35 ,57 ,43 ,44 ,32 ,33 ,38 ,26 ,41 ,8 ,22 ,2 ,18 ,15 ,25 ,12 ,1 ,21 ,
63 ,13 ,6 ,5 ,11 ,53 ,3]
P=P3
# inverse permuation box used at the end of the final round
PI_11 = [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]
PI_12 = [16, 32, 13, 14, 19, 20, 6, 28, 29, 25, 17, 22, 3, 26, 10, 24, 27, 2, 5, 23, 4, 15, 30, 12, 21, 9, 1, 7,
31, 18, 8, 11]
PI_13 = [11, 28, 127, 96, 13, 125, 18, 128, 5, 16, 126, 120, 27, 119, 107, 20, 56, 117, 75, 88, 73, 4, 115,
112, 106, 82, 121, 66, 1, 108, 51, 103, 118, 42, 91, 111, 9, 99, 123, 36, 113, 92, 98, 89, 37, 85, 44, 95,
31, 116, 23, 32, 81, 102, 84, 101, 64, 77, 62, 74, 6, 86, 114, 97, 110, 69, 93, 58, 104, 38, 78, 22, 15, 59,
26, 109, 17, 90, 63, 87, 53, 94, 70, 80, 40, 67, 105, 57, 79, 41, 3, 8, 68, 48, 83, 47, 29, 10, 39, 100, 7,
61, 60, 71, 55, 76, 50, 35, 12, 30, 24, 49, 122, 21, 54, 2, 72, 46, 45, 19, 34, 124, 33, 43, 14, 52, 25, 65]
PI_1 = PI_13
accum=[]
acc=[]

SHIFT =
[1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1,1,1,2,2,2,2,2
,2,1,2,2,2,2,2,2,1]
# shift array shift[i] denotes how many bits the blocks will be shifted at round i+1


def string_to_bit_array(text):
    # this function takes a text string and covert it into nubmers each character is mapped to a 8 bit value
    array = list()
    for char in text:
        binval = binvalue(char, 8)
        array.extend([int(x) for x in list(binval)])
    return array

def bit_array_to_string(array):
    #reverse of the above function
    res = ''.join([chr(int(y,2)) for y in [''.join([str(x) for x in _bytes]) for _bytes in  nsplit(array,8)]])
    return res

def binvalue(val, bitsize):
    # thus function takes the ASCII value of val and convert it into bitsize bits each
    binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]
    if len(binval) > bitsize:
        raise "binary value larger than the expected size"
```

```
    while len(binval) < bitsize:
        binval = "0"+binval
    return binval

def nsplit(s, n):
    # create  equal fragments of a array s each size n
    return [s[k:k+n] for k in range(0, len(s), n)]

ENCRYPT=1
DECRYPT=0

class des():
    # class which has the functions which performs des for us
    global roun
    def __init__(self):
        self.password = None
        self.text = None
        self.keys = list()
       # password is the first key that we recieve , text is the plain text,keys is a list of the keys generated
at each round
    def run(self, key, text, action=ENCRYPT, padding=False):
        #run process is our des implementation
        # if the key is small an error occur, if large we take the first required bits
        if len(key) < kinw:
            raise "Key Should be 8 bytes long"
        elif len(key) > kinw:
            key = key[:kinw]

        lx = len(text)
        if (lx%kinw)!=0 :
            for i in range(0,kinw-lx%kinw) :
                text=text+" "
        self.password = key
        self.text = text


        if padding and action==ENCRYPT:
            self.addPadding()
        elif len(self.text) % kinw != 0:
            raise "Data size should be multiple of 8"

        self.generatekeys()
        # we generate all the keys required for the rounds by the above function
        text_blocks = nsplit(self.text, kinw)
        # create blocks of the plaintext of required size

        result = list()

        for block in text_blocks:
```

```
        acc.clear()
        for i in range(0,roun+1):
            acc.append([])
        # acc contains the result after each value so that we can use it later in avalance plots
        block = string_to_bit_array(block)#Convert the block in bit array
        acc[0]=block
        block = self.permut(block,PI)
        # it performs the initial permuation on the plain text
        g, d = nsplit(block, w)
        # create the two halves
        tmp = None
        # below is the rounds of des performs
        for i in range(0,roun):

            d_e = self.expand(d, E)#expansion of the second half
            if action == ENCRYPT:
                tmp = self.xor(self.keys[i], d_e)#xoring with the key
            else:
                tmp = self.xor(self.keys[roun-1-i], d_e)# for decrytion the keys are in reverse
            tmp = self.substitute(tmp)# substition using s box
            tmp = self.permut(tmp, P)# permuation choice of the values
            tmp = self.xor(g, tmp)
            g = d
            d = tmp
            acc[i+1]=self.permut(d+g, PI_1)
        # des rounds are performed above

        result += self.permut(d+g, PI_1)
    final_res = bit_array_to_string(result)
    if padding and action==DECRYPT:
        return self.removePadding(final_res)
    else:
        return final_res

def substitute(self, d_e):
    # this function uses the subsitution boxes to create the output
    subblocks = nsplit(d_e, 6)
    result = list()
    for i in range(len(subblocks)):
        block = subblocks[i]
        row = int(str(block[0])+str(block[5]),2)
        column = int(''.join([str(x) for x in block[1:][:-1]]),2)
        val = S_BOX[i][row][column]
        bin = binvalue(val, 4)
        result += [int(x) for x in bin]
    return result

def permut(self, block, table):#Permut the given block using the given table (so generic method)
    return [block[x-1] for x in table]
```

```
    def expand(self, block, table):#Do the exact same thing than permut but for more clarity has been
renamed
        return [block[x-1] for x in table]

    def xor(self, t1, t2):#Apply a xor and return the resulting list
        return [x^y for x,y in zip(t1,t2)]

    def generatekeys(self):#Algorithm that generates all the keys
        self.keys = []
        key = string_to_bit_array(self.password)

        key = self.permut(key, CP_1)
        g, d = nsplit(key, kfrh)
        for i in range(roun):
            g, d = self.shift(g, d, SHIFT[i])
            tmp = g + d
            self.keys.append(self.permut(tmp, CP_2))


    def shift(self, g, d, n):
        return g[n:] + g[:n], d[n:] + d[:n]

    def addPadding(self):
        pad_len = 8 - (len(self.text) % 8)
        self.text += pad_len * chr(pad_len)

    def removePadding(self, data):
        pad_len = ord(data[-1])
        return data[:-pad_len]

    def encrypt(self, key, text, padding=False):
        return self.run(key, text, ENCRYPT, padding)

    def decrypt(self, key, text, padding=False):
        return self.run(key, text, DECRYPT, padding)


d = des()




import numpy as np

import matplotlib.pyplot as plt
def diff(row1,row0):# finds the hamming distance in data
```

```
    sum=0
    for f in range(0,len(row1)):
        sum=sum+abs(row1[f]-row0[f])
    return sum
def avalanche(roune,text,key,twow):#utility code to get avalnche effect and table
    arrin=string_to_bit_array(text)
    d=des()
    global accum
    global roun




    accum=np.zeros((twow+1,roune+1,twow))
  #  print(accum.shape)
    accum[0][0]=arrin
    for r in range(1,roune+1):

            roun=r
            rounvalue=d.encrypt(key,bit_array_to_string(arrin))
            accum[0][r] =(string_to_bit_array(rounvalue))


    for f in range(0,twow):

        arrin[f]=1-arrin[f]
        accum[f+1]=arrin

        for r in range(1,roune+1):

            roun=r


            rounvalue=d.encrypt(key,bit_array_to_string(arrin))
            accum[f+1][r]=(string_to_bit_array(rounvalue))
           # print(d.decrypt(key,d.encrypt(key,bit_array_to_string(arrin))))
           # print(bit_array_to_string(arrin))
            #print("rewwwfffffffffffffffffffff")

        arrin[f]=1-arrin[f]

  #  print(accum.shape)
    hero = np.zeros((twow+1,roun+1))

    for sample in range(0,twow+1):
          for row in range(0,roun+1):

              x=diff(accum[sample][row],accum[0][row])
```

```
            hero[sample][row]=x


    hero1 = np.delete(hero, (0), axis=0)

    av=np.mean(hero1, axis = 0)
    x = []
    for g1 in range(0,roun+1):
        x.append(g1)
    print(av)
    y = av.tolist()



    plt.plot(x, y)


    plt.xlabel('rounds')

    plt.ylabel('Hamming Distance')


    plt.title('Avalanche')
    plt.show()

    return av



  #avalanche(roun,text,key,twow)
def string_to_freqlist(tex):#calculates the frequency of a char in a string i.e."aaaaa" a=5
    lis=np.zeros((256))

    for char in tex:
        ba=string_to_bit_array(char)
        res = int("".join(str(x) for x in ba), 2)
        lis[res]=lis[res]+1
    return lis

def frequency(ro,te,ke):
    frew=np.zeros((ro+1,256))
    frew[0]=string_to_freqlist(te)
    global roun
    d=des()
    for ri in range(1,ro+1):
        roun=ri
        frew[ri]=string_to_freqlist(d.encrypt( ke, te))
    lis=np.arange(256)
```

```
    for ill in range(0,ro+1):
        plt.figure(ill+1)
        li = lis
        y = frew[ill]
        print("dfdfdsfdf")
        print(y.shape)
        plt.plot(li,y)
        plt.show()


    return



import tkinter as tk

from tkinter import *
def show():
    f1=fu.get()
    if f1==1:
        outext.delete(0.0,END)
        i=intext.get("0.0",END)
        st=""
        for char in i:
            ch=char
            if char>='a' and char<='z':
                ch=chr(ord('z')-ord(char)+ord('a'))
            elif char>='A' and char<='Z':
                ch=chr(ord('Z')-ord(char)+ord('A'))
            elif char>='0' and char<='9':
                ch=chr(ord('9')-ord(char)+ord('0'))

            st=st+ch
        outext.insert(0.0,st)
    elif f1==2:
        intext.delete(0.0,END)
        i=outext.get("0.0",END)
        st=""
        for char in i:
            ch=char
            if char>='a' and char<='z':
                ch=chr(ord('z')-ord(char)+ord('a'))
            elif char>='A' and char<='Z':
                ch=chr(ord('Z')-ord(char)+ord('A'))
            elif char>='0' and char<='9':
                ch=chr(ord('9')-ord(char)+ord('0'))

            st=st+ch
```

```
        intext.insert(0.0,st)
def encry():#encrypt push button function
    wid=fu.get()
    rs=roundbox.get("0.0",END)
    ro = int(rs)
    global w
    w=int(wid/2)
    global PI
    global PI1
    global PI3

    global P
    global P3
    global E
    global E3
    global CP_2
    global CP_23
    global CP_1
    global CP_13
    global PI_1
    if w==64:
        PI=PI3
        PI_1 = PI_13
        P=P3
        E=E3
        CP_2=CP_23
        CP_1=CP_13
    elif w==16:
        PI=PI2
        PI_1 = PI_12
        P=P2
        E=E2
        CP_2=CP_22
        CP_1=CP_12

    elif w==32:
        PI=PI1
        PI_1 = PI_11
        P=P1
        E=E1
        CP_2=CP_21
        CP_1=CP_11

    global roun
    roun=ro
    print("roun is"+str(roun))


    global twow
```

```
    twow=2*w
    global k
    k=2*w
    global kinw
    kinw=int(k/8)
    global kfr
    kfr=int((k*7)/8)
    global kfrh
    kfrh=int(kfr/2)

    outext.delete(0.0,END)
    i=intext.get("0.0",END)
    i=i[:-1]
    ke=keybox.get("0.0",END)
    ke=ke[:-1]
    d=des()
    #print("8888888888888")
    print(len(i))
    outi=d.encrypt(ke,i)
    print("output boloc is"+str(outi))
    print(len(outi))
    #print("8888888888888")

    outext.insert(0.0,outi)

def decry():# decrypt push button function
    wid=fu.get()
    rs=roundbox.get("0.0",END)
    ro = int(rs)
    global w
    w=int(wid/2)
    global PI
    global PI1
    global PI3

    global P
    global P3
    global E
    global E3
    global CP_2
    global CP_23
    global CP_1
    global CP_13
    global PI_1
    if w==64:
        PI=PI3
        PI_1 = PI_13
        P=P3
        E=E3
```

```
        CP_2=CP_23
        CP_1=CP_13
    elif w==16:
        PI=PI2
        PI_1 = PI_12
        P=P2
        E=E2
        CP_2=CP_22
        CP_1=CP_12
        print(CP_1)
    elif w==32:
        PI=PI1
        PI_1 = PI_11
        P=P1
        E=E1
        CP_2=CP_21
        CP_1=CP_11
        print(PI)
    global roun
    roun=ro
    print("roun is"+str(roun))


    global twow
    twow=2*w
    global k
    k=2*w
    global kinw
    kinw=int(k/8)
    global kfr
    kfr=int((k*7)/8)
    global kfrh
    kfrh=int(kfr/2)

    outext.delete(0.0,END)
    i=intext.get("0.0",END)
    i=i[:-1]
    ke=keybox.get("0.0",END)
    ke=ke[:-1]
    d=des()
    #print("8888888888888")
    print(len(i))
    outi=d.decrypt(ke,i)
    print(len(outi))
    #print("8888888888888")

    outext.insert(0.0,outi)
def ava():# avalance push button function
    wid=fu.get()
```

```
rs=roundbox.get("0.0",END)
ro = int(rs)
global w
w=int(wid/2)
global PI
global PI1
global PI3

global P
global P3
global E
global E3
global CP_2
global CP_23
global CP_1
global CP_13
global PI_1
if w==64:
    PI=PI3
    PI_1 = PI_13
    P=P3
    E=E3
    CP_2=CP_23
    CP_1=CP_13
elif w==16:
    PI=PI2
    PI_1 = PI_12
    P=P2
    E=E2
    CP_2=CP_22
    CP_1=CP_12
    print(CP_1)
elif w==32:
    PI=PI1
    PI_1 = PI_11
    P=P1
    E=E1
    CP_2=CP_21
    CP_1=CP_11
    print(PI)
global roun
roun=ro
print("roun is"+str(roun))


global twow
twow=2*w
global k
k=2*w
```

```
    global kinw
    kinw=int(k/8)
    global kfr
    kfr=int((k*7)/8)
    global kfrh
    kfrh=int(kfr/2)

    outext.delete(0.0,END)
    i=intext.get("0.0",END)
    i=i[:-1]
    ke=keybox.get("0.0",END)
    ke=ke[:-1]
    d=des()
    av=avalanche(roun,i,ke,twow)
    print("av is:"+str(av))
    outi=str(av)
    outext.insert(0.0,outi)
def free():#frequency function which runs when we press freqency button
    wid=fu.get()
    rs=roundbox.get("0.0",END)
    ro = int(rs)
    global w
    w=int(wid/2)
    global PI
    global PI1
    global PI3

    global P
    global P3
    global E
    global E3
    global CP_2
    global CP_23
    global CP_1
    global CP_13
    global PI_1
    if w==64:
        PI=PI3
        PI_1 = PI_13
        P=P3
        E=E3
        CP_2=CP_23
        CP_1=CP_13
    elif w==16:
        PI=PI2
        PI_1 = PI_12
        P=P2
        E=E2
        CP_2=CP_22
```

```
        CP_1=CP_12
       # print(CP_1)
    elif w==32:
        PI=PI1
        PI_1 = PI_11
        P=P1
        E=E1
        CP_2=CP_21
        CP_1=CP_11
      #  print(PI)
    global roun
    roun=ro
    global twow
    twow=2*w
    global k
    k=2*w
    global kinw
    kinw=int(k/8)
    global kfr
    kfr=int((k*7)/8)
    global kfrh
    kfrh=int(kfr/2)
    outext.delete(0.0,END)
    i=intext.get("0.0",END)
    i=i[:-1]
    ke=keybox.get("0.0",END)
    ke=ke[:-1]
    d=des()
    frequency(ro,i,ke)
    return


m = tk.Tk(screenName=None,  baseName=None,  className='Crpyter',  useTk=1)
#user interface code
l1=Label(m, text="INPUT")
l2=Label(m,text="OUTPUT")

l1.grid(row=0,column=0)
l2.grid(row=0,column=2)

fu=IntVar()
rd1=Radiobutton(m,text="32",variable=fu,value=32)
rd2=Radiobutton(m,text="64",variable=fu,value=64)
rd3=Radiobutton(m,text="128",variable=fu,value=128)

rd1.grid(row=1,column=0)
rd2.grid(row=1,column=1)
rd3.grid(row=1,column=2)
d = des()
```

```
intext=Text(m,width=30,height=20,wrap=WORD)
keybox=Text(m,width=20,height=5,wrap=WORD)
roundbox=Text(m,width=3,height=1,wrap=WORD)
intext.grid(row=3,column=0)
roundbox.grid(row=2,column=1)
keybox.grid(row=3,column=1)
outext=Text(m,width=30,height=20,wrap=WORD)
outext.grid(row=3,column=2)
act1=Button(m,text="encrypt",command=encry)
act2=Button(m,text="decrypt",command=decry)
act3=Button(m,text="Avalanche",command=ava)
act4=Button(m,text="Frequency plot",command=free)
act1.grid(row=4,column=0)
act2.grid(row=4,column=1)
act3.grid(row=4,column=2)
act4.grid(row=4,column=3)
m.mainloop()
```

PA-2

Date:-24-5-2020
Name:- Aakash Mathur
Dept.:- Electronics Engineering
Roll No.:- 17095001
Framework=Jupyter Notebook

Objective

To code the CBC(**Cipher** block chaining) and OFB(**Output** feedback mode) using the standard Blowfish implementation as basic encryption.

Theory

Blowfish is an encryption technique designed as an alternative to Data Encryption Standard(**DES**) 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 ciphers not subject to any patents and hence freely available for anyone to use. Basic properties of Blowfish technique:-

63) **blockSize**: 64-bits
64) **keySize**: 32-bits to 448-bits variable size
65) **number of subkeys**: 18 [P-array]
66) **number of rounds**: 16
67) **number of substitution boxes**: 4 [each having 512 entries of 32-bits each]

The Blowfish process proceeds with the following steps:-

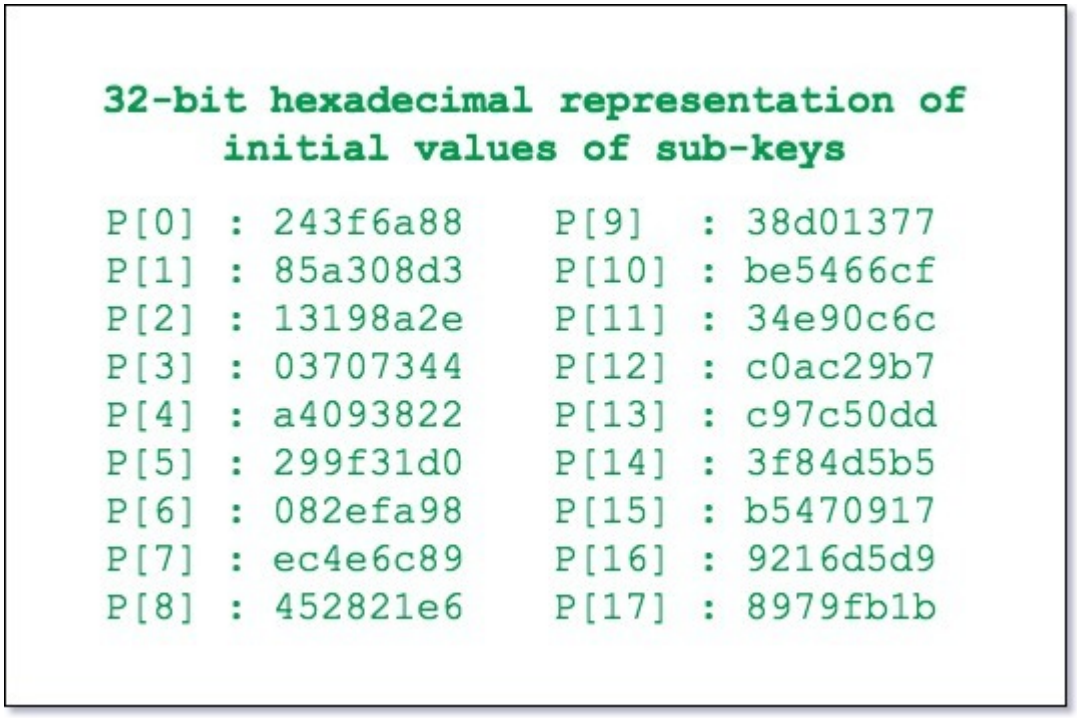
Generation of subkeys:

1. 18 subkeys{P[0]…P[17]} are needed in both encryption as well as the decryption process and the same subkeys are used for both the processes.

2. These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.

3. It is initialized with the digits of pi.

4. The hexadecimal representation of each of the subkeys is given by:



5. Then each of the subkeys is changed with respect to the input key as:

11. P[0] = P[0] XOR 1st 32-bits of input key
12. P[1] = P[1] XOR 2nd 32-bits of input key
13. P[i] = P[i] XOR (i+1)th 32-bits of input key
14. (rollover to 1st 32-bits depending on the key length)
15. P[17] = P[17] XOR 18th 32-bits of input key
16. (rollover to 1st 32-bits depending on key length)

6. The resultant P-array holds 18 subkeys that are used during the entire encryption process.

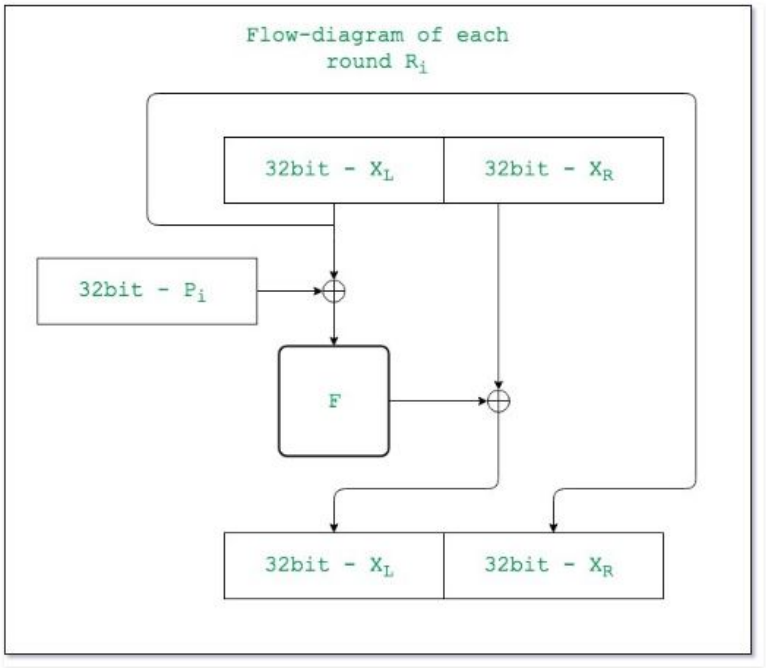
Initialize Substitution Boxes:

1. 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption aswell as decryption process with each S-box having 256 entries{S[i][0]… S[i][255], 0&lei&le4} where each entry is 32-bit.

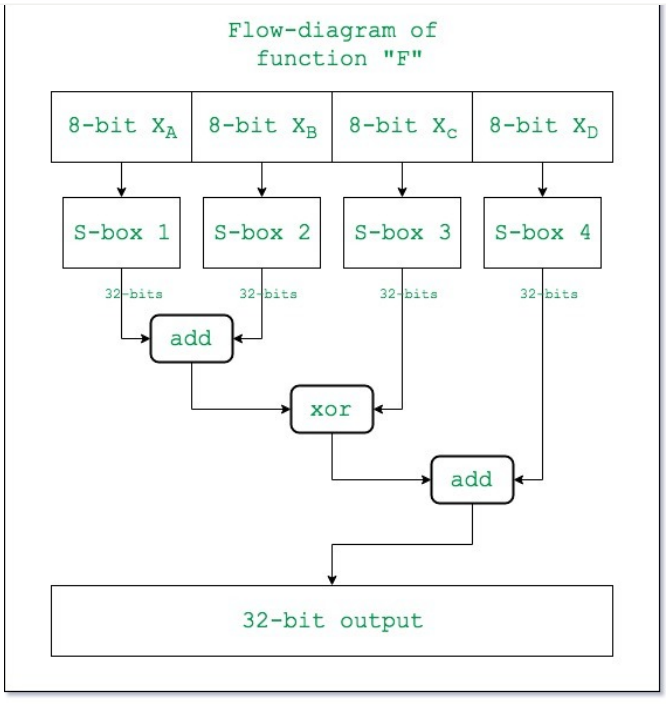
2.It is initialised with the digits of pi after initialising the P-array.

Encryption:

a) **Rounds:** The encryption consists of 16 rounds with each round(Ri) taking inputs the plainText(P.T.) from the previous round and corresponding subkey(Pi). The description of each round is as follows:



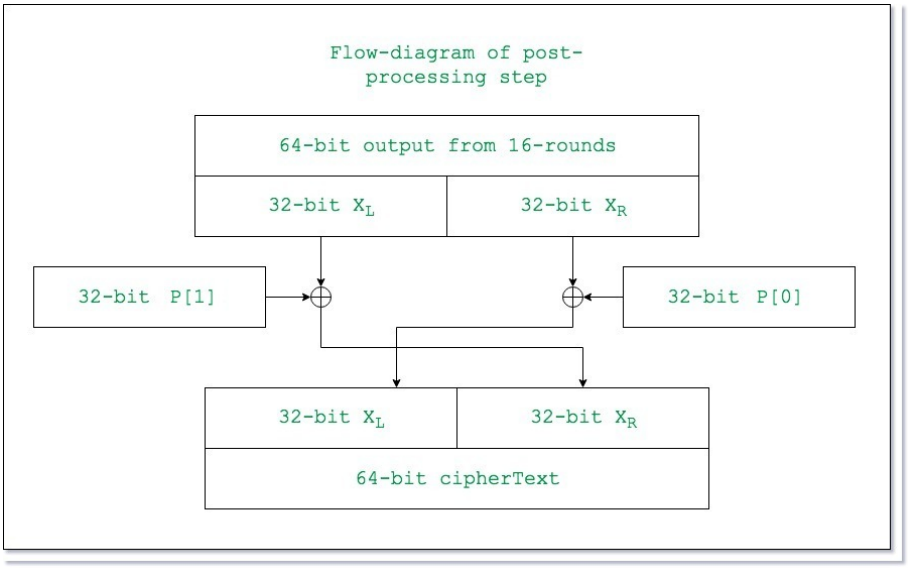
The description of the function ” F ” is as follows



Sbox functioning shown in fig

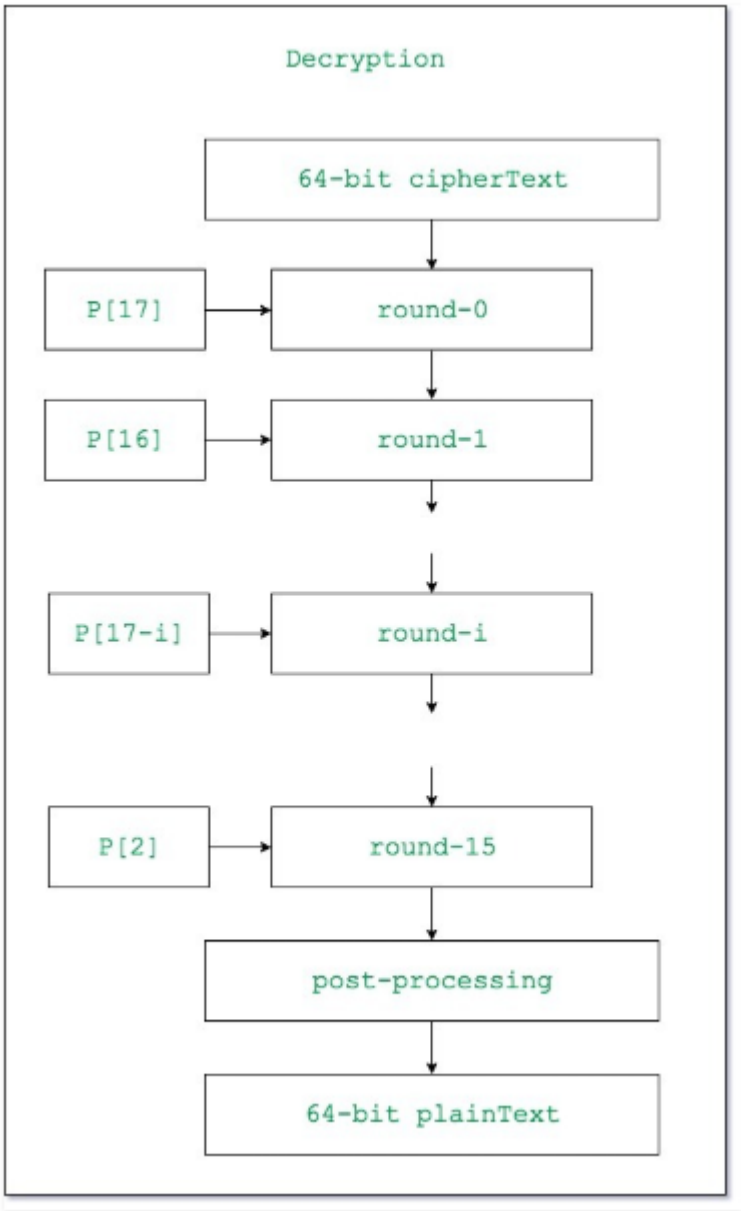
(add=add modulo 2^{32})

Post Processing: The output after the 16 rounds is processed as follows:



Decryption

The decryption process is similar to that of encryption and the subkeys are used in reverse{P[17] – P[0]}. The entire decryption process is shown



Cipher block chaining (CBC)

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector(IV) must be used in the first block.

If the first block has index 1, the mathematical formula for CBC encryption is

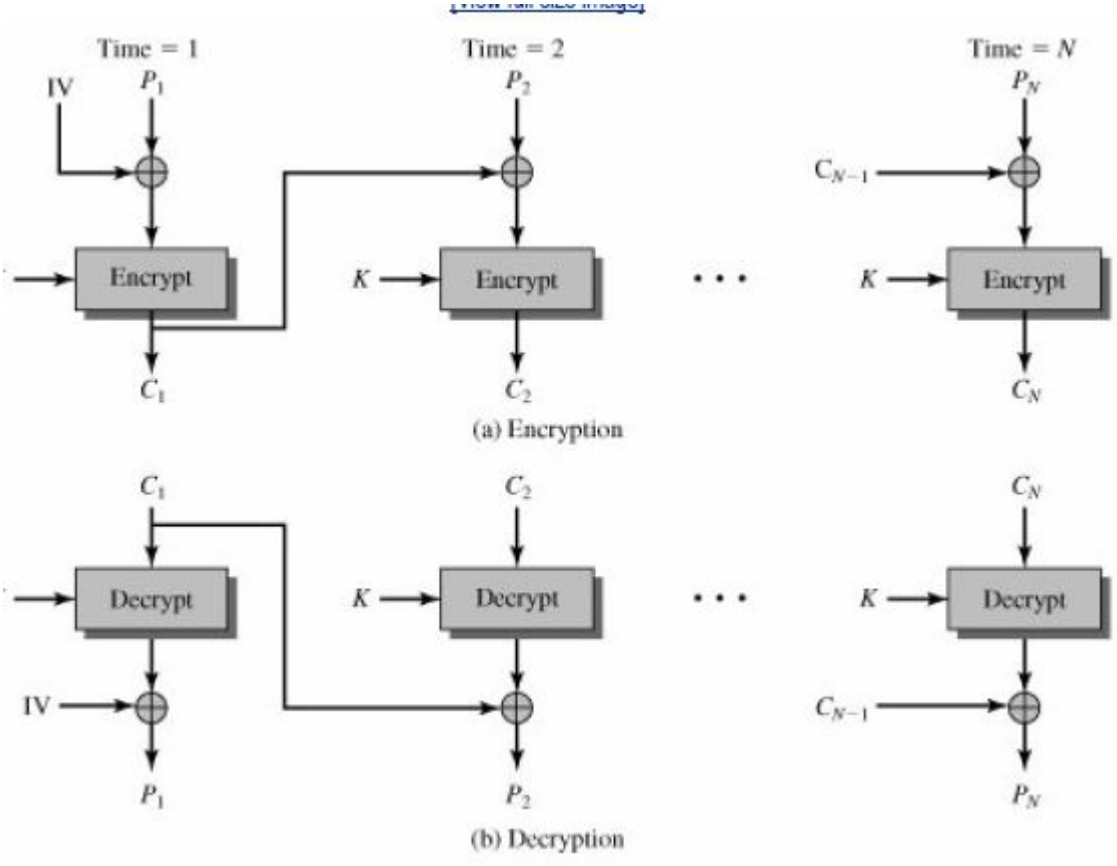
$$C_i = E_K(P_i \oplus C_{i-1}),$$

$$C_0 = IV,$$

while the mathematical formula for CBC decryption is

$$P_i = D_K(C_i) \oplus C_{i-1},$$

$$C_0 = IV.$$



Output feedback (OFB)

The *output feedback* (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error-correcting codes to function normally even when applied before encryption.

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:

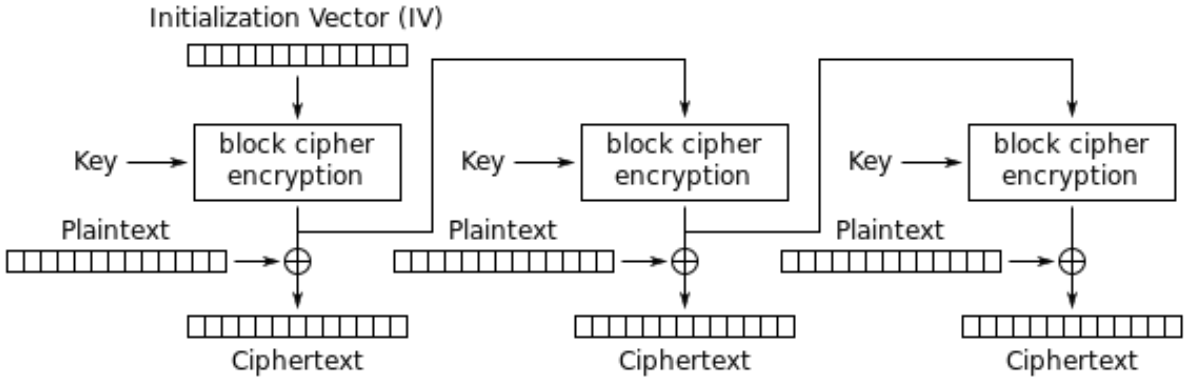
$$C_j = P_j \oplus O_j,$$

$$P_j = C_j \oplus O_j,$$

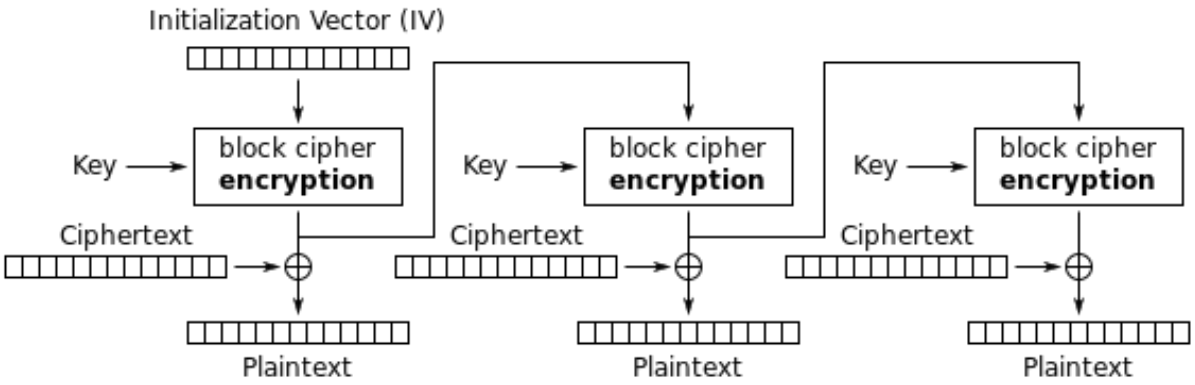
$$O_j = E_K(I_j),$$

$$I_j = O_{j-1},$$

$$I_0 = \mathrm{IV}.$$



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

Implementation:

we first implemented the blowfish algorithm which follows the feistel structure and is used in rounds , also keys are generated before hand for future rounds .

Once the blowfish encrypt and decrypt functions are ready OFB and CBC

are repetitive functions which are implemented using loops

Code(Complete code in appendix)

Important functions

A. KeyGeneration:

```
#generate subkeys.
def keyGenerate(key):
    j = 0
    subK=[]
    for i in range(0,len(P)):#takes the key to generate round keys
        subK.append(xor(P[i], key[j: j + 8]))
        j=(j + 8) % (len(key));
    return subK
```

In this function we take input of the key and generate all the subsequent keys using the P box where all the P boxes are xored with circular segments of key

B.round:

```
1. def rounds(time,plainText,subk):
2.     left = plainText[0:8]# in each round the blocks are split in equal halves
3.     right = plainText[8:16]
4.     left = xor(left, subk[time])#xored with key
5.     #output from F function
6.     fOut = f(left)
7.     right = xor(fOut, right)
8.     return (right + left)
```

In this function we take a plaintext split it into two halves ,each is then passed the left half is xored with key and swapped to right and also a function f create a value after using substitution of left half which is then xored with right

C. Encrypt and Decrypt:

In encryption and decryption function we take input a text in encrypt we pass it through rounds and final processing .In decrypt processing is done before revround function

```
1. def encrypt(plainText,subk):
2.     for i in range(0,16):
3.         plainText = rounds(i, plainText,subk) # repetitive rounds
4.     #postprocessing
5.     right = plainText[0:8]
6.     left = plainText[8:16]
7.     right = xor(right, subk[16])
8.     left = xor(left, subk[17])
```

```
    return (left + right)
#decryption
def decrypt(plainText,subk):
    #preprocessing
    right = plainText[8:16]
    left = plainText[0:8]
    right = xor(right, subk[16])
    left = xor(left, subk[17])
    cipher=right+left
    for i in range(0,16):
        cipher=revround(15-i,cipher,subk)
```

D. CBC encryption and Decryption

```
# cbc encryption function
def cbcencryption(plaintext,key,IV):
    sol=""
    s=len(plaintext)
    sn = s
    pt=plaintext.zfill(sn)#create plaintext by padding
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the plaintext blocks
        temp=pt[i:i+16]#the text block
        inp=xord(temp,IV)#xored
        IV=encrypt(inp,sbk)#IV value updated
        sol=sol+IV
```

```
    return sol
#cbc decryption function
def cbcdecryption(ciphertext,key,IV):
    sol=""
    s=len(ciphertext)
    sn = s
    ct=ciphertext.zfill(sn)#create ciphertext by padding
    sbk= keyGenerate(key)#keyset generated
    ans=[]
    for i in range(0,sn,16):#loop over the ciphertext blocks
        temp=ct[i:i+16]
        tiv=temp# the cipher text block
        inp=decrypt(temp,sbk)
        ans=xord(inp,IV)
        IV=tiv# the block is used as IV for future rounds
        sol=sol+ans
    return sol
```

The cbcencryptin function takes input of plaintext adds padding if required,create the keyset and then the loops are run on the plaintext blocks which take the plaintecxt xor it with IV then encrypt it this value is used as IV for other blocks and this value is also the cipher block.

In cbcdecryption we perform the reverse the input text is decrypted and xored with IV for future rounds Xored with previous input.ti get the plaintext

E. OFB encryption and Decryption

```
def ofbencryption(plaintext,key,IV):
    sol=""
    s=len(plaintext)
    sn = s
    pt=plaintext.zfill(sn)#create plaintext by padding
```

```
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the plaintext blocks
        temp=pt[i:i+16]
        inp=encrypt(IV,sbk)
        ans=xord(inp,temp)
        sol=sol+ans
        IV=inp #IV value updated

    return sol
#ofb decryption function
def ofbdecryption(text,key,IV):
    sol=""
    s=len(text)
    sn = s
    pt=text.zfill(sn)#create plaintext by padding
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the ciphertext blocks
        temp=pt[i:i+16]
        inp=encrypt(IV,sbk)
        ans=xord(inp,temp)
        sol=sol+ans
        IV=inp#IV value updated
    return sol
```

the OFB encryption and decryption are identical , the IV is encrypted then this value is used for future IV and this value is Xored with the text to get the desired output. In the encryption this text is plaintext and cipher text in other

F.f function

```
def f(plaintext):
    a=[]
    for i in range(0,8,2):
```

```
4.      temp=plaintext[i:i+2]# forms 4 blocks of planitext and substitute
5.      col=int(temp, 16)
6.      v=S[int(i / 2)][col]
7.      a.append(v)
8.  ans = addBin(a[0], a[1])#performs the functions
9.  #add the first 2 values
10. ans = xor(ans, a[2]) #xor then
11. ans = addBin(ans, a[3]) #then xor
12. return ans #returns an output of 32 bit
```

This is the only nonlinear part there the block of 32 bit is taken input which is then

broken in 4 continuous segments each segment uses a different s box and outputs a 32 bit number then they are performed operations on each other like

adding(in 32 bit field) and xoring . The final output is a 32 bit value

Outputs:

CBC:

Encryption(IV must be 16 char)

```
which encryption scheme do you want to use CBC / OFBCBC
Enter the text you want to modify in hex (min length 16 characters)abcdef1234567812
Enter the key you want to use in hex(min length 16 characters)ababcdcd12345678
Enter an IV(min length 16 characters)fedcba1234567890
Enter the function you want to perform (1) for encryption (2) for decryption1
404CFD289830A4DB
```

Decryption

```
which encryption scheme do you want to use CBC / OFBCBC
Enter the text you want to modify in hex (min length 16 characters)404CFD289830A4DB
Enter the key you want to use in hex(min length 16 characters)ababcdcd12345678
Enter an IV(min length 16 characters)fedcba1234567890
Enter the function you want to perform (1) for encryption (2) for decryption2
ABCDEF1234567812
```

OFB

Encryption

```
which encryption scheme do you want to use CBC / OFBOFB
Enter the text you want to modify in hex (min length 16 characters)abcdef1234567812
Enter the key you want to use in hex(min length 16 characters)ababcdcd12345678
Enter an IV(min length 16 characters)fedcba1234567890
Enter the function you want to perform (1) for encryption (2) for decryption1
AC99B792786B08EF
```

Decryption

```
which encryption scheme do you want to use CBC / OFBOFB
Enter the text you want to modify in hex (min length 16 characters)AC99B792786B08EF
Enter the key you want to use in hex(min length 16 characters)ababcdcd12345678
Enter an IV(min length 16 characters)fedcba1234567890
Enter the function you want to perform (1) for encryption (2) for decryption2
ABCDEF1234567812
```

Conclusion:

We successfully implemented OFB and CBC encryption and decryption using blowfish encryption standard

Appendix:

#Substitution boxes each string is a 32 bit hexadecimal value.

S= [["d1310ba6", "98dfb5ac", "2ffd72db", "d01adfb7", "b8e1afed",

"6a267e96", "ba7c9045", "f12c7f99", "24a19947", "b3916cf7",

"0801f2e2", "858efc16", "636920d8", "71574e69", "a458fea3",

"f4933d7e", "0d95748f", "728eb658", "718bcd58", "82154aee",

"7b54a41d", "c25a59b5", "9c30d539", "2af26013", "c5d1b023",

"286085f0", "ca417918", "b8db38ef", "8e79dcb0", "603a180e",

"6c9e0e8b", "b01e8a3e", "d71577c1", "bd314b27", "78af2fda",

"55605c60", "e65525f3", "aa55ab94", "57489862", "63e81440",

"55ca396a", "2aab10b6", "b4cc5c34", "1141e8ce", "a15486af",

"7c72e993", "b3ee1411", "636fbc2a", "2ba9c55d", "741831f6",

"ce5c3e16", "9b87931e", "afd6ba33", "6c24cf5c", "7a325381",
"28958677", "3b8f4898", "6b4bb9af", "c4bfe81b", "66282193",
"61d809cc", "fb21a991", "487cac60", "5dec8032", "ef845d5d",
"e98575b1", "dc262302", "eb651b88", "23893e81", "d396acc5",
"0f6d6ff3", "83f44239", "2e0b4482", "a4842004", "69c8f04a",
"9e1f9b5e", "21c66842", "f6e96c9a", "670c9c61", "abd388f0",
"6a51a0d2", "d8542f68", "960fa728", "ab5133a3", "6eef0b6c",
"137a3be4", "ba3bf050", "7efb2a98", "a1f1651d", "39af0176",
"66ca593e", "82430e88", "8cee8619", "456f9fb4", "7d84a5c3",
"3b8b5ebe", "e06f75d8", "85c12073", "401a449f", "56c16aa6",
"4ed3aa62", "363f7706", "1bfedf72", "429b023d", "37d0d724",
"d00a1248", "db0fead3", "49f1c09b", "075372c9", "80991b7b",
"25d479d8", "f6e8def7", "e3fe501a", "b6794c3b", "976ce0bd",
"04c006ba", "c1a94fb6", "409f60c4", "5e5c9ec2", "196a2463",
"68fb6faf", "3e6c53b5", "1339b2eb", "3b52ec6f", "6dfc511f",
"9b30952c", "cc814544", "af5ebd09", "bee3d004", "de334afd",
"660f2807", "192e4bb3", "c0cba857", "45c8740f", "d20b5f39",
"b9d3fbdb", "5579c0bd", "1a60320a", "d6a100c6", "402c7279",
"679f25fe", "fb1fa3cc", "8ea5e9f8", "db3222f8", "3c7516df",
"fd616b15", "2f501ec8", "ad0552ab", "323db5fa", "fd238760",
"53317b48", "3e00df82", "9e5c57bb", "ca6f8ca0", "1a87562e",
"df1769db", "d542a8f6", "287effc3", "ac6732c6", "8c4f5573",
"695b27b0", "bbca58c8", "e1ffa35d", "b8f011a0", "10fa3d98",
"fd2183b8", "4afcb56c", "2dd1d35b", "9a53e479", "b6f84565",
"d28e49bc", "4bfb9790", "e1ddf2da", "a4cb7e33", "62fb1341",
"cee4c6e8", "ef20cada", "36774c01", "d07e9efe", "2bf11fb4",
"95dbda4d", "ae909198", "eaad8e71", "6b93d5a0", "d08ed1d0",
"afc725e0", "8e3c5b2f", "8e7594b7", "8ff6e2fb", "f2122b64",
"8888b812", "900df01c", "4fad5ea0", "688fc31c", "d1cff191",
"b3a8c1ad", "2f2f2218", "be0e1777", "ea752dfe", "8b021fa1",
"e5a0cc0f", "b56f74e8", "18acf3d6", "ce89e299", "b4a84fe0",

"fd13e0b7", "7cc43b81", "d2ada8d9", "165fa266", "80957705",
"93cc7314", "211a1477", "e6ad2065", "77b5fa86", "c75442f5",
"fb9d35cf", "ebcdaf0c", "7b3e89a0", "d6411bd3", "ae1e7e49",
"00250e2d", "2071b35e", "226800bb", "57b8e0af", "2464369b",
"f009b91e", "5563911d", "59dfa6aa", "78c14389", "d95a537f",
"207d5ba2", "02e5b9c5", "83260376", "6295cfa9", "11c81968",
"4e734a41", "b3472dca", "7b14a94a", "1b510052", "9a532915",
"d60f573f", "bc9bc6e4", "2b60a476", "81e67400", "08ba6fb5",
"571be91f", "f296ec6b", "2a0dd915", "b6636521", "e7b9f9b6",
"ff34052e", "c5855664", "53b02d5d", "a99f8fa1", "08ba4799",
"6e85076a"],
["4b7a70e9", "b5b32944", "db75092e", "c4192623", "ad6ea6b0",
"49a7df7d", "9cee60b8", "8fedb266", "ecaa8c71", "699a17ff",
"5664526c", "c2b19ee1", "193602a5", "75094c29", "a0591340",
"e4183a3e", "3f54989a", "5b429d65", "6b8fe4d6", "99f73fd6",
"a1d29c07", "efe830f5", "4d2d38e6", "f0255dc1", "4cdd2086",
"8470eb26", "6382e9c6", "021ecc5e", "09686b3f", "3ebaefc9",
"3c971814", "6b6a70a1", "687f3584", "52a0e286", "b79c5305",
"aa500737", "3e07841c", "7fdeae5c", "8e7d44ec", "5716f2b8",
"b03ada37", "f0500c0d", "f01c1f04", "0200b3ff", "ae0cf51a",
"3cb574b2", "25837a58", "dc0921bd", "d19113f9", "7ca92ff6",
"94324773", "22f54701", "3ae5e581", "37c2dadc", "c8b57634",
"9af3dda7", "a9446146", "0fd0030e", "ecc8c73e", "a4751e41",
"e238cd99", "3bea0e2f", "3280bba1", "183eb331", "4e548b38",
"4f6db908", "6f420d03", "f60a04bf", "2cb81290", "24977c79",
"5679b072", "bcaf89af", "de9a771f", "d9930810", "b38bae12",
"dccf3f2e", "5512721f", "2e6b7124", "501adde6", "9f84cd87",
"7a584718", "7408da17", "bc9f9abc", "e94b7d8c", "ec7aec3a",
"db851dfa", "63094366", "c464c3d2", "ef1c1847", "3215d908",
"dd433b37", "24c2ba16", "12a14d43", "2a65c451", "50940002",
"133ae4dd", "71dff89e", "10314e55", "81ac77d6", "5f11199b",

"043556f1", "d7a3c76b", "3c11183b", "5924a509", "f28fe6ed",
"97f1fbfa", "9ebabf2c", "1e153c6e", "86e34570", "eae96fb1",
"860e5e0a", "5a3e2ab3", "771fe71c", "4e3d06fa", "2965dcb9",
"99e71d0f", "803e89d6", "5266c825", "2e4cc978", "9c10b36a",
"c6150eba", "94e2ea78", "a5fc3c53", "1e0a2df4", "f2f74ea7",
"361d2b3d", "1939260f", "19c27960", "5223a708", "f71312b6",
"ebadfe6e", "eac31f66", "e3bc4595", "a67bc883", "b17f37d1",
"018cff28", "c332ddef", "be6c5aa5", "65582185", "68ab9802",
"eecea50f", "db2f953b", "2aef7dad", "5b6e2f84", "1521b628",
"29076170", "ecdd4775", "619f1510", "13cca830", "eb61bd96",
"0334fe1e", "aa0363cf", "b5735c90", "4c70a239", "d59e9e0b",
"cbaade14", "eecc86bc", "60622ca7", "9cab5cab", "b2f3846e",
"648b1eaf", "19bdf0ca", "a02369b9", "655abb50", "40685a32",
"3c2ab4b3", "319ee9d5", "c021b8f7", "9b540b19", "875fa099",
"95f7997e", "623d7da8", "f837889a", "97e32d77", "11ed935f",
"16681281", "0e358829", "c7e61fd6", "96dedfa1", "7858ba99",
"57f584a5", "1b227263", "9b83c3ff", "1ac24696", "cdb30aeb",
"532e3054", "8fd948e4", "6dbc3128", "58ebf2ef", "34c6ffea",
"fe28ed61", "ee7c3c73", "5d4a14d9", "e864b7e3", "42105d14",
"203e13e0", "45eee2b6", "a3aaabea", "db6c4f15", "facb4fd0",
"c742f442", "ef6abbb5", "654f3b1d", "41cd2105", "d81e799e",
"86854dc7", "e44b476a", "3d816250", "cf62a1f2", "5b8d2646",
"fc8883a0", "c1c7b6a3", "7f1524c3", "69cb7492", "47848a0b",
"5692b285", "095bbf00", "ad19489d", "1462b174", "23820e00",
"58428d2a", "0c55f5ea", "1dadf43e", "233f7061", "3372f092",
"8d937e41", "d65fecf1", "6c223bdb", "7cde3759", "cbee7460",
"4085f2a7", "ce77326e", "a6078084", "19f8509e", "e8efd855",
"61d99735", "a969a7aa", "c50c06c2", "5a04abfc", "800bcadc",
"9e447a2e", "c3453484", "fdd56705", "0e1e9ec9", "db73dbd3",
"105588cd", "675fda79", "e3674340", "c5c43465", "713e38d8",
"3d28f89e", "f16dff20", "153e21e7", "8fb03d4a", "e6e39f2b",

"db83adf7"],

["e93d5a68", "948140f7", "f64c261c", "94692934", "411520f7",
"7602d4f7", "bcf46b2e", "d4a20068", "d4082471", "3320f46a",
"43b7d4b7", "500061af", "1e39f62e", "97244546", "14214f74",
"bf8b8840", "4d95fc1d", "96b591af", "70f4ddd3", "66a02f45",
"bfbc09ec", "03bd9785", "7fac6dd0", "31cb8504", "96eb27b3",
"55fd3941", "da2547e6", "abca0a9a", "28507825", "530429f4",
"0a2c86da", "e9b66dfb", "68dc1462", "d7486900", "680ec0a4",
"27a18dee", "4f3ffea2", "e887ad8c", "b58ce006", "7af4d6b6",
"aace1e7c", "d3375fec", "ce78a399", "406b2a42", "20fe9e35",
"d9f385b9", "ee39d7ab", "3b124e8b", "1dc9faf7", "4b6d1856",
"26a36631", "eae397b2", "3a6efa74", "dd5b4332", "6841e7f7",
"ca7820fb", "fb0af54e", "d8feb397", "454056ac", "ba489527",
"55533a3a", "20838d87", "fe6ba9b7", "d096954b", "55a867bc",
"a1159a58", "cca92963", "99e1db33", "a62a4a56", "3f3125f9",
"5ef47e1c", "9029317c", "fdf8e802", "04272f70", "80bb155c",
"05282ce3", "95c11548", "e4c66d22", "48c1133f", "c70f86dc",
"07f9c9ee", "41041f0f", "404779a4", "5d886e17", "325f51eb",
"d59bc0d1", "f2bcc18f", "41113564", "257b7834", "602a9c60",
"dff8e8a3", "1f636c1b", "0e12b4c2", "02e1329e", "af664fd1",
"cad18115", "6b2395e0", "333e92e1", "3b240b62", "eebeb922",
"85b2a20e", "e6ba0d99", "de720c8c", "2da2f728", "d0127845",
"95b794fd", "647d0862", "e7ccf5f0", "5449a36f", "877d48fa",
"c39dfd27", "f33e8d1e", "0a476341", "992eff74", "3a6f6eab",
"f4f8fd37", "a812dc60", "a1ebddf8", "991be14c", "db6e6b0d",
"c67b5510", "6d672c37", "2765d43b", "dcd0e804", "f1290dc7",
"cc00ffa3", "b5390f92", "690fed0b", "667b9ffb", "cedb7d9c",
"a091cf0b", "d9155ea3", "bb132f88", "515bad24", "7b9479bf",
"763bd6eb", "37392eb3", "cc115979", "8026e297", "f42e312d",
"6842ada7", "c66a2b3b", "12754ccc", "782ef11c", "6a124237",
"b79251e7", "06a1bbe6", "4bfb6350", "1a6b1018", "11caedfa",

"3d25bdd8", "e2e1c3c9", "44421659", "0a121386", "d90cec6e",
"d5abea2a", "64af674e", "da86a85f", "bebfe988", "64e4c3fe",
"9dbc8057", "f0f7c086", "60787bf8", "6003604d", "d1fd8346",
"f6381fb0", "7745ae04", "d736fccc", "83426b33", "f01eab71",
"b0804187", "3c005e5f", "77a057be", "bde8ae24", "55464299",
"bf582e61", "4e58f48f", "f2ddfda2", "f474ef38", "8789bdc2",
"5366f9c3", "c8b38e74", "b475f255", "46fcd9b9", "7aeb2661",
"8b1ddf84", "846a0e79", "915f95e2", "466e598e", "20b45770",
"8cd55591", "c902de4c", "b90bace1", "bb8205d0", "11a86248",
"7574a99e", "b77f19b6", "e0a9dc09", "662d09a1", "c4324633",
"e85a1f02", "09f0be8c", "4a99a025", "1d6efe10", "1ab93d1d",
"0ba5a4df", "a186f20f", "2868f169", "dcb7da83", "573906fe",
"a1e2ce9b", "4fcd7f52", "50115e01", "a70683fa", "a002b5c4",
"0de6d027", "9af88c27", "773f8641", "c3604c06", "61a806b5",
"f0177a28", "c0f586e0", "006058aa", "30dc7d62", "11e69ed7",
"2338ea63", "53c2dd94", "c2c21634", "bbcbee56", "90bcb6de",
"ebfc7da1", "ce591d76", "6f05e409", "4b7c0188", "39720a3d",
"7c927c24", "86e3725f", "724d9db9", "1ac15bb4", "d39eb8fc",
"ed545578", "08fca5b5", "d83d7cd3", "4dad0fc4", "1e50ef5e",
"b161e6f8", "a28514d9", "6c51133c", "6fd5c7e7", "56e14ec4",
"362abfce", "ddc6c837", "d79a3234", "92638212", "670efa8e",
"406000e0"],
["3a39ce37", "d3faf5cf", "abc27737", "5ac52d1b", "5cb0679e",
"4fa33742", "d3822740", "99bc9bbe", "d5118e9d", "bf0f7315",
"d62d1c7e", "c700c47b", "b78c1b6b", "21a19045", "b26eb1be",
"6a366eb4", "5748ab2f", "bc946e79", "c6a376d2", "6549c2c8",
"530ff8ee", "468dde7d", "d5730a1d", "4cd04dc6", "2939bbdb",
"a9ba4650", "ac9526e8", "be5ee304", "a1fad5f0", "6a2d519a",
"63ef8ce2", "9a86ee22", "c089c2b8", "43242ef6", "a51e03aa",
"9cf2d0a4", "83c061ba", "9be96a4d", "8fe51550", "ba645bd6",
"2826a2f9", "a73a3ae1", "4ba99586", "ef5562e9", "c72fefd3",

"f752f7da", "3f046f69", "77fa0a59", "80e4a915", "87b08601",
"9b09e6ad", "3b3ee593", "e990fd5a", "9e34d797", "2cf0b7d9",
"022b8b51", "96d5ac3a", "017da67d", "d1cf3ed6", "7c7d2d28",
"1f9f25cf", "adf2b89b", "5ad6b472", "5a88f54c", "e029ac71",
"e019a5e6", "47b0acfd", "ed93fa9b", "e8d3c48d", "283b57cc",
"f8d56629", "79132e28", "785f0191", "ed756055", "f7960e44",
"e3d35e8c", "15056dd4", "88f46dba", "03a16125", "0564f0bd",
"c3eb9e15", "3c9057a2", "97271aec", "a93a072a", "1b3f6d9b",
"1e6321f5", "f59c66fb", "26dcf319", "7533d928", "b155fdf5",
"03563482", "8aba3cbb", "28517711", "c20ad9f8", "abcc5167",
"ccad925f", "4de81751", "3830dc8e", "379d5862", "9320f991",
"ea7a90c2", "fb3e7bce", "5121ce64", "774fbe32", "a8b6e37e",
"c3293d46", "48de5369", "6413e680", "a2ae0810", "dd6db224",
"69852dfd", "09072166", "b39a460a", "6445c0dd", "586cdecf",
"1c20c8ae", "5bbef7dd", "1b588d40", "ccd2017f", "6bb4e3bb",
"dda26a7e", "3a59ff45", "3e350a44", "bcb4cdd5", "72eacea8",
"fa6484bb", "8d6612ae", "bf3c6f47", "d29be463", "542f5d9e",
"aec2771b", "f64e6370", "740e0d8d", "e75b1357", "f8721671",
"af537d5d", "4040cb08", "4eb4e2cc", "34d2466a", "0115af84",
"e1b00428", "95983a1d", "06b89fb4", "ce6ea048", "6f3f3b82",
"3520ab82", "011a1d4b", "277227f8", "611560b1", "e7933fdc",
"bb3a792b", "344525bd", "a08839e1", "51ce794b", "2f32c9b7",
"a01fbac9", "e01cc87e", "bcc7d1f6", "cf0111c3", "a1e8aac7",
"1a908749", "d44fbd9a", "d0dadecb", "d50ada38", "0339c32a",
"c6913667", "8df9317c", "e0b12b4f", "f79e59b7", "43f5bb3a",
"f2d519ff", "27d9459c", "bf97222c", "15e6fc2a", "0f91fc71",
"9b941525", "fae59361", "ceb69ceb", "c2a86459", "12baa8d1",
"b6c1075e", "e3056a0c", "10d25065", "cb03a442", "e0ec6e0e",
"1698db3b", "4c98a0be", "3278e964", "9f1f9532", "e0d392df",
"d3a0342b", "8971f21e", "1b0a7441", "4ba3348c", "c5be7120",
"c37632d8", "df359f8d", "9b992f2e", "e60b6f47", "0fe3f11d",

```
"e54cda54", "1edad891", "ce6279cf", "cd3e7e6f", "1618b166",
"fd2c1d05", "848fd2c5", "f6fb2299", "f523f357", "a6327623",
"93a83531", "56cccd02", "acf08162", "5a75ebb5", "6e163697",
"88d273cc", "de966292", "81b949d0", "4c50901b", "71c65614",
"e6c6c7bd", "327a140a", "45e1d006", "c3f27b9a", "c9aa53fd",
"62a80f00", "bb25bfe2", "35bdd2f6", "71126905", "b2040222",
"b6cbcf7c", "cd769c2b", "53113ec0", "1640e3d3", "38abbd60",
"2547adf0", "ba38209c", "f746ce76", "77afa1c5", "20756060",
"85cbfe4e", "8ae88dd8", "7aaaf9b0", "4cf9aa7e", "1948c25c",
"02fb8a8c", "01c36ae4", "d6ebe1f9", "90d4f869", "a65cdea0",
"3f09252d", "c208e69f", "b74e6132", "ce77e25b", "578fdfe3",
"3ac372e6" ] ]

# Subkeys initialisation with digits of pi.
P= [ "243f6a88", "85a308d3", "13198a2e", "03707344", "a4093822",
"299f31d0", "082efa98", "ec4e6c89", "452821e6", "38d01377",
"be5466cf", "34e90c6c", "c0ac29b7", "c97c50dd", "3f84d5b5",
"b5470917", "9216d5d9", "8979fb1b" ]
# xor two hexadecimal strings of the same length.
def xor(a,b):
   a="{0:08b}".format(int(a, 16))
   a=str(a)
   a=format(int(a), '032d')
   b ="{0:08b}".format(int(b, 16))
   b=str(b)
   b=format(int(b), '032d')
   ans = ""
   for i in range(len(a)):
      if (a[i] == b[i]):
         ans += "0"
      else:
```

```
            ans += "1"
    ans = "{0:0>4X}".format(int(ans, 2))
    ans= ans.zfill(8)
    ans=str(ans)
    return ans


##addition modulo 2^32 of two hexadecimal strings.
def addBin(a,b):
    n1 = int(a, 16)
    n2 = int(b, 16)
    n1 = (n1 + n2) %(pow(2,32))
    st = "0x%0.16X" % n1
    st=st[2:18]
    ans = st
    return ans
# This is the round function which modifies the left half
def f(plaintext):
    a=[]
    for i in range(0,8,2):
        temp=plaintext[i:i+2]# forms 4 blocks of planitext and substitute
        col=int(temp, 16)
        v=S[int(i / 2)][col]
        a.append(v)
    ans = addBin(a[0], a[1])#performs the functions
    #add the first 2 values
    ans = xor(ans, a[2]) #xor then
    ans = addBin(ans, a[3]) #then xor
    return ans #returns an output of 32 bits
#generate subkeys.
def keyGenerate(key):
```

```
    j = 0
    subK=[]
    for i in range(0,len(P)):#takes the key to generate round keys
        subK.append(xor(P[i], key[j: j + 8]))
        j=(j + 8) % (len(key));
    return subK
# round function
def rounds(time,plainText,subk):
    left = plainText[0:8]# in each round the blocks are split in equal halves
    right = plainText[8:16]
    left = xor(left, subk[time])#xored with key
    #output from F function
    fOut = f(left)
    right = xor(fOut, right)
    return (right + left)
# round function for decryption
def revround(i,cipher,subk):
    left = cipher[0:8]
    right = cipher[8:16]
    fout=f(right)
    tl=xor(right,subk[i])
    tr=xor(fout,left)
    #print(i)
    #print(right+ri)
    return (tl+tr)
# encryption
def encrypt(plainText,subk):
    for i in range(0,16):
        plainText = rounds(i, plainText,subk) # repetitive rounds
     #postprocessing
    right = plainText[0:8]
```

```
    left = plainText[8:16]
    right = xor(right, subk[16])
    left = xor(left, subk[17])
    return (left + right)
#decryption
def decrypt(plainText,subk):
    #preprocessing
    right = plainText[8:16]
    left = plainText[0:8]
    right = xor(right, subk[16])
    left = xor(left, subk[17])
    cipher=right+left
    for i in range(0,16):
        cipher=revround(15-i,cipher,subk)

    return cipher
# a general xor function for variable length string
def xord(a,b):
    al=len(a)
    a="{0:08b}".format(int(a, 16))
    a=str(a)
    a=a.zfill(4*al)
    b ="{0:08b}".format(int(b, 16))
    b=str(b)
    b=format(int(b), '064d')
    b=b.zfill(4*al)
    ans = ""
    for i in range(len(a)):
        if (a[i] == b[i]):
            ans += "0"
        else:
```

```
            ans += "1"
    ans = "{0:0>4X}".format(int(ans, 2))
    ans= ans.zfill(al)
    ans=str(ans)
    return ans
# cbc encryption function
def cbcencryption(plaintext,key,IV):
    sol=""
    s=len(plaintext)
    sn = s
    pt=plaintext.zfill(sn)#create plaintext by padding
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the plaintext blocks
        temp=pt[i:i+16]#the text block
        inp=xord(temp,IV)#xored
        IV=encrypt(inp,sbk)#IV value updated
        sol=sol+IV
    return sol
#cbc decryption function
def cbcdecryption(ciphertext,key,IV):
    sol=""
    s=len(ciphertext)
    sn = s
    ct=ciphertext.zfill(sn)#create ciphertext by padding
    sbk= keyGenerate(key)#keyset generated
    ans=[]
    for i in range(0,sn,16):#loop over the ciphertext blocks
        temp=ct[i:i+16]
        tiv=temp# the cipher text block
        inp=decrypt(temp,sbk)
        ans=xord(inp,IV)
```

```
        IV=tiv# the block is used as IV for future rounds
        sol=sol+ans
    return sol
#ofb encryption function
def ofbencryption(plaintext,key,IV):
    sol=""
    s=len(plaintext)
    sn = s
    pt=plaintext.zfill(sn)#create plaintext by padding
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the plaintext blocks
        temp=pt[i:i+16]
        inp=encrypt(IV,sbk)
        ans=xord(inp,temp)
        sol=sol+ans
        IV=inp #IV value updated

    return sol
#ofb decryption function
def ofbdecryption(text,key,IV):
    sol=""
    s=len(text)
    sn = s
    pt=text.zfill(sn)#create plaintext by padding
    sbk= keyGenerate(key)#keyset generated
    for i in range(0,sn,16):#loop over the ciphertext blocks
        temp=pt[i:i+16]
        inp=encrypt(IV,sbk)
        ans=xord(inp,temp)
        sol=sol+ans
        IV=inp#IV value updated
```

```
    return sol



if __name__ == '__main__':
    fc=input("how many queries")
    fc=int(fc)
    for i in range(0,fc):
        c=input("which encryption scheme do you want to use CBC / OFB")
        textin=input("Enter the text you want to modify in hex (min length 16 characters)")
        keyin=input("Enter the key you want to use in hex(min length 16 characters)")
        IVin=input("Enter an IV(min length 16 characters)")
        en=input("Enter the function you want to perform (1) for encryption (2) for decryption")
        en=int(en)
        sol=""
        if(c=="CBC"):
            if(en==1):

                sol=cbcencryption(textin,keyin,IVin)
            elif(en==2):
                sol=cbcdecryption(textin,keyin,IVin)
        elif(c=="OFB"):
             if(en==1):
                sol=ofbencryption(textin,keyin,IVin)

             elif(en==2):
                sol=ofbdecryption(textin,keyin,IVin)
                                        print(sol)
```

PA – 3 .1

Date:-28/05/2020
Name : Aakash Mathur
Roll No: 17095001
Department: Electronics Engineering
Framework Jupyter NoteBook.

Objective: To implement a simple voting protocol for online voting system with properties as described later.

Theory:
various properties required to implement a good but simple voting protocol these are as follows:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else vote. (This turns out to be the hardest requirement.)
5. No one can change anyone else vote without being discovered.
6. Every voter can make sure that his vote has been taken into account in the final tabulation.
7. Everyone knows who voted and who didn’t.

In the simplistic version of protocol for step 3 it is assumed tjat the ctf is a reliable and authorized server.

Design of the protocol*:*
To implement this protocol we use the digital signature for the authentication of the voter which is followed by RSA encryption technique for the safety of the message.

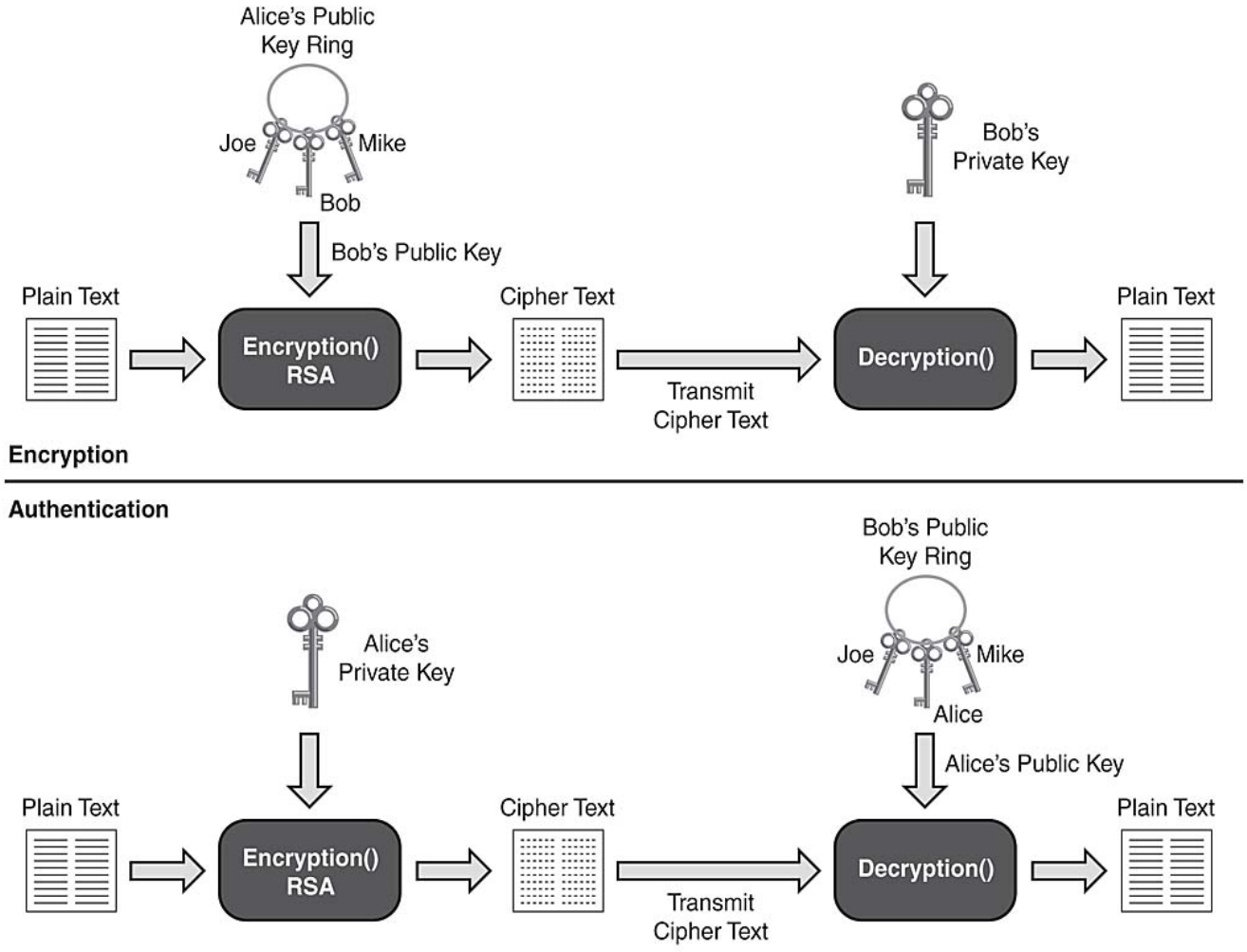
Below is the procedure :

(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.

RSA encryption *:*

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission . It is an asymmetric encryption scheme . A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. . Anyone can use the public key to encrypt a message, but only someone with knowledge of the prime numbers can decode the message. There are no published methods to defeat the system if a large enough key is used.



Confidentiality :

In our case the voter A produces a vote message in plain text
X =[X1 , X2 ,..., XM,].
These X values are numbers generated after hashing and mapping algorithms which are linked to our vote in one to one relationship
This message is intended for destination CTF server .
B generates a related pair of keys: a public key , ser_pub, and a private key, ser_pri.
ser_pri is known only to CTF, whereas ser_pub is publicly available and therefore accessible by voter A also.
With the message X and the encryption key ser_pub as input, A forms the ciphertext Y = [Y1 , Y2 ,..., YN]

$$Y = E(ser\_pub, X)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation

$$X = D(ser\_pri , Y)$$

An adversary, observing Y and having access to ser_pub but not having access to ser_pri or X, attempts to recover X and/or ser_pri .
It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms.
Thus the vote message confidentiality is provided

Authentication*:*
voter A prepares a message to CTF and encrypts it using voter A's private key before transmitting it.
CTF can decrypt the message using A's public key.
Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature.

In addition, it is impossible to meaningfully alter the vote without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.
The authen1tication

It is possible to provide both authentication and confidentiality by a double use of the public-key scheme:
X is the original vote message of the voter A and Z is the message which travels through the channel.
$Z = E(Ser\_Pub , E(PRa , X))$
$X = D(PUa , D(Ser\_pri , Z))$

Mathematical Concept of RSA*:*

Key generation process:
The following steps are involved in the generation of keys :
1.Choose two different large random prime numbers p and q.
2.Calculate $n=p.q$
3.Calculate $\Phi(n) = (p-1)*(q-1)$
4.Choose an interger e such that $1 < e < \Phi(n)$ and $gcd(\Phi(n),e) = 1$.
5. d is calculate as the mod inverse of e in field of $\Phi(n)$.
The private key consists of {d, n} and the public key consists of {e, n}.

Message generation for alphabet field:

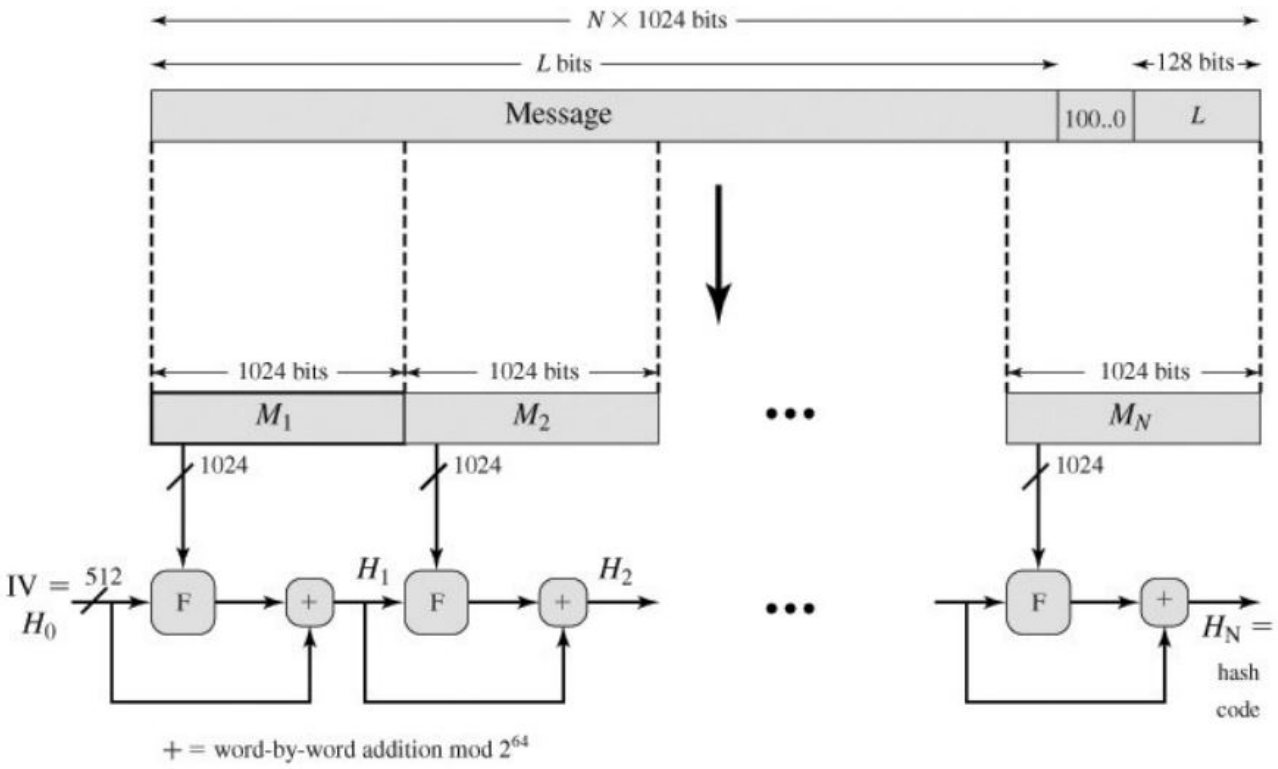
Secure Hash Function(SHA) : is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001.

SHA are used to create a message of fixed length that is easy for further processing .

The algorithm takes as input a message with a maximum length of less than 2^128 bits and produces as output a 512-bit message digest. • The input is processed in 1024-bit blocks

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256
Notes: 1. All sizes are measured in bits. 2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$				

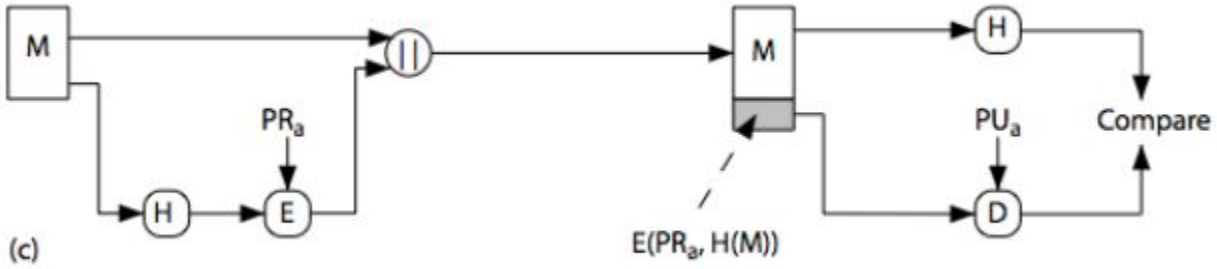
The given message is made into a multiple of 1024 bits (n*1024) by adding padding and a number denoting the length of the original message .
Each 1024 bits are passed through are repetitive function



The output of the SHA algorithm is expected to be collision resistant.

Authentication Mechanism :

Digital signature is used to authenticate the message in our example is the vote so the voters sign it with there private key before sending . The hash value of the message is actually signed along with we send the message so that they can be compared to check that the signature is authentic .



In our case we send the hash value of the message except of message for easy of computation it also serves the same purpose.

Encryption :

The signed message is encrypted with the servers public key for the confidentiality of the message .

The mathematical formula used is:

$$c = (m^e) \bmod n$$

where m is the message , {e,n} is the server public key and c is the encrypted message .

Decryption :

The encrypted message is decrpyted with the servers private key
The mathematical formula used is:

$$m= (c^d) \bmod n$$

where m is the message , {e,n} is the server public key and c is the encrypted message .

The message m which is received is containing the original message along with the signed message . The signed message is unsigned with the voters public key and the unsigned message is compared with the appended message if they are same it means the signer is same as expected.

Code :

```
import random
from Crypto.Hash import SHA256
from random import SystemRandom
import copy
import hashlib

def gcd(a, b):
    #simple gcd code using euclidean therom
    while b != 0:
        a, b = b, a % b
    return a

def power(x, y, p) :
    # modulo exponentiation function return (x^y)mod p
    res = 1
    x = x % p
    if (x == 0) :
        return 0
    while (y > 0) :
        # If y is odd, multiply
        # x with result
        if ((y & 1) == 1) :
            res = (res * x) % p
        # y must be even now
        y = y >> 1      # y = y/2
        x = (x * x) % p
    return res

def multiplicative_inverse(a, m):
    #multiplicative inverse of a in  m field (a^(-1))mod m
```

```
    m0 = m
    y = 0
    x = 1
    if (m == 1) :
        return 0
    while (a > 1) :
        q = a // m
        t = m
        m = a % m
        a = t
        t = y
        y = x - q * y
        x = t
    if (x < 0) :
        x = x + m0
    return x

def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True


primes = [i for i in range(10,100) if is_prime(i)]
# above is a list of prime numbers from 10 to 100 will be used in creation of key

def generate_keypair(p, q):
    # this function is used to generate key pair that is private and public for the pair of prime numbers
    n = p * q
    phi = (p-1) * (q-1)
    # totient function
    e = random.randrange(1, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)
    # above code is used to find a number e coprime to phi
    d = multiplicative_inverse(e, phi)
    # d is inverse of e
    return ((e, n), (d, n))

def encrypt(pk, plaintext):
    # this function takes input a list of numbers and encrypt each element with the key
    key, n = pk
```

```
    cipher=[]
    for i in plaintext :
        cipher.append(power(i, key, n))
    return cipher

def decrypt(pk, ciphertext):
  # this function takes input a list of numbers and decrypt each element with the key
    key, n = pk
    plain=[]
    for i in ciphertext :
        plain.append(power(i, key, n))
    return plain


def hstintarray(hexstring):
    # the hash function create a string of 256 bits which is converted into a hexstring
    # the hexstring is digit by digit converted into a list of numbers
    # example if the hash value of M is "ff0110ffabcd1234" thus 256 bit value is converted into
    # [15,15,0,1,1,0,15,15,10,11,12,13,1,2,3,4]
    # so that we have numbers which can be processed
    s=[]
    for char in hexstring:
        if (char>='a')&(char<='f'):
            s.append(ord(char)-ord('a')+10)
        else:
            s.append(ord(char)-ord('0'))
    return s

CTR = dict()
# ctr  is the serverside dictionary of counting and verifing vote it contains the allowed voters name
VB = dict()
# vote bank VB for counting votes and keeping them
pubkeymap = dict()
prikeymap=dict()
# maps to store pybkey and privkey     *imp the public key map will be shared where as prikeymap is
just for the voter to get
#his private key (it is not shared)
pubkeymap["one"],prikeymap["one"]=generate_keypair(23,29)
pubkeymap["two"],prikeymap["two"]=generate_keypair(31,37)
pubkeymap["three"],prikeymap["three"]=generate_keypair(41,43)
pubkeymap["four"],prikeymap["four"]=generate_keypair(47,53)
# some voters have thier keys already created while others can manually create in program
CTR["one"]="NV"
CTR["two"]="NV"
CTR["three"]="NV"
CTR["four"]="NV"
CTR["five"]="NV"
# list of authentic voters
ser_pub,ser_pri = generate_keypair(71,73)
```

```
# servers key created will remain constant
print(CTR)
print(prikeymap)
# just printed for execution sake
#you might not know your private key you can look into it this will be offcourse from others

def sendvote(vot,private,nam):
    # 'this function create a message packed signed and encrypted ready to share in channel
    msg = hashlib.sha256(vot.encode())
    msg = msg.hexdigest()
    # msg is the hash value of the choosen vote
    hexary =hstintarray(msg)
    # we convert it into a list of numbers which can be mathematically operated
    hexarysign = encrypt(private, hexary)
    # we sign the hash value for authentication with voters private key
    hexaryen=encrypt(ser_pub,hexary)
    # we encrypt the hash value unsigned this value will be also send for later verification of the user
    # in general method we send the message that is vote here but here we are sending the hashvalue of
easy of working
    hexarysignen=encrypt(ser_pub,hexarysign)
    # we also encrypt the signed msg for confidentiality
    # encrypting done using servers public key
    channelmsg=(hexarysignen,hexaryen)
    # a message ready to pass throught the channel and meet the hackers
    return channelmsg,nam

def serversideporcessing(channelmsg,nam):
    # this is the only function which occurs at the server side
    # voters name is also transferred to CTR
    hexarysignen,hexaryen = channelmsg
    # the encrypted hash value signed and unsigned both are recieved
    hexarysign=decrypt(ser_pri,hexarysignen)
    hexary = decrypt(ser_pri,hexaryen)
    # the values are decrypted to get the signed and unsigned hash values using servers private key
    hexaryunsign=decrypt(pubkeymap[nam],hexarysign)
    # the singed value is unsinged for checking using voters public key
    if(hexaryunsign!=hexary):
        # if the values that is singed(later unsinged ) and the original doesnot match authentication failed
        print("signature is incorrect")
        print(hexary)
        print(hexaryunsign)
    else:
        # singature verified
        print("signature verifed")
        if nam in CTR:
            # if the voter is in CTR list then only it can vote
            if CTR[nam]=="NV":
                # if the voters has not voted before then only it can vote
                CTR[nam]="V"
```

```
            VB[nam]=hexary
            # the vote is registered and the CTR notes the name also
            print("vote successfully added")
        else:
            print("you have already voted")
    else:
        print("your name is not in list ")
return


def votecount(VB):
    # this function is used to count the votes
    # the voters name are known to the vote bank / CTR
    vot ="BJP"
    msg = hashlib.sha256(vot.encode())
    msg = msg.hexdigest()
    bjphexary =hstintarray(msg)
    vot ="Trump"
    msg = hashlib.sha256(vot.encode())
    msg = msg.hexdigest()
    Trumphexary =hstintarray(msg)
    vot ="JSR"
    msg = hashlib.sha256(vot.encode())
    msg = msg.hexdigest()
    jsrhexary =hstintarray(msg)
    vot ="NOTA"
    msg = hashlib.sha256(vot.encode())
    msg = msg.hexdigest()
    dhexary =hstintarray(msg)

    dvb = dict()
    count = dict()
    count["BJP"]=0

    count["Trump"]=0

    count["JSR"]=0

    count["NOTA"]=0
    count["trash"]=0
    for i in VB:
        if(VB[i]==bjphexary):
            dvb[i]="BJP"
            count["BJP"]=count["BJP"]+1
        elif (VB[i]==Trumphexary):
            dvb[i]="Trump"
            count["Trump"]=count["Trump"]+1
        elif (VB[i]==jsrhexary):
            dvb[i]="JSR"
```

```
            count["JSR"]=count["JSR"]+1
        elif (VB[i]==dhexary):
            dvb[i]="NOTA"
            count["NOTA"]=count["NOTA"]+1
        else :
            count["trash"]=count["trash"]+0


    print(count)
    #print(dvb)
    return


if __name__ == '__main__':
    j = input("How many queries ")
    # we enter the number of queries we want to perform
    j =int(j)
    while(j>0):
        j=j-1
        bo =input("Do you want to cast a vote:y/n ")
        if bo == "y":
            nam = input("Enter your voter id name")
            # if user want to vote he input his voter id here it is its name.# in real cases it could be voter id
number
            if(nam not in pubkeymap):
                # if the voter has not create his keys these lines of code creates for him and uploads the
pubkey to the
                # common map
                pn = random.choice(primes)
                qn=  random.choice(primes)
                while(qn==pn):
                    qn=randomChoices(primes)
                pubkeymap[nam],prikeymap[nam]=generate_keypair(pn,qn)
                print("we created keys for you and your private key is")
                print(prikeymap[nam])
                # it also outputs the prikey for the voter only known to him


            # the voter selects the party name he want to vote this is our base message
            vot =input("enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)")
            print("your vote is taken input your private key to continue first enter d then n")
            privd=input()
            privd = int(privd)
            privn=input()
            privn=int(privn)
            # voter inputs its private key
            private=(privd,privn)
            print("now we are creating a portable encrypted  message  'your name' will be send to the
server")
```

```
        # all the above part of code runs on the voters computer only
        channelmsg,nam=sendvote(vot,private,nam)
        # a encrypted and signed message for the channel is created
        # this passes through the channel and reaches the server/CTF  a hacker can get it if the network
is compromissed
        serversideporcessing(channelmsg,nam)
        # above code runs on CTR it decrpt the message checks the signature also
        # this function doesnot allow unregisted to vote also it doesnot allow fake votes
      else:
        print("why are you here than you dont want to vote so get lost and let others come")

  votecount(VB)
  # this function count the votes for you
```

<u>UNDERSTANDING THE CODE</u>:

1. The code is a basic coded representation of two parallel processes running on two different computers one is the voter/client side and other is the CTR/server side
2.The code is in a serial manner not in parallel and the process are created in form of functions which “represent” the program that would be running in the systems in the code they are not parallel process but part of same code but in function format.
3.The process running at the server side is serversideporcessing .
All other parts would be running on the voters computer
4. Lets take a look over the code

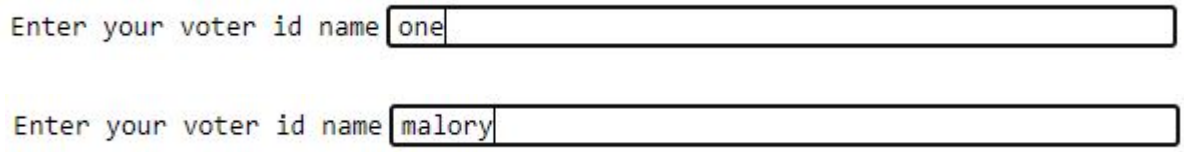
<u>WORKING</u> :

1.The code first asks for the number of queries this is analogues to the time interval we provide for vote g larger interval more voters can vote more queries

queries = Total number of votes passes (correct+incorrect+malicious)
These queries are for creating and sending a message.
2.Then for each query the voter computer ask for if the user wants to vote.
3.If yes it continues else continues to other queries.

```
How many queries 10
Do you want to cast a vote:y/n y
```

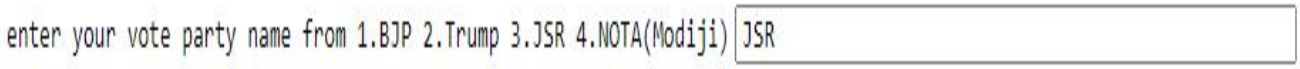
4.Then it ask for voter identification here it is name in general it can be voter id number.
You can put any name i.e. it may be in list or not
thus representing anyone can create a message

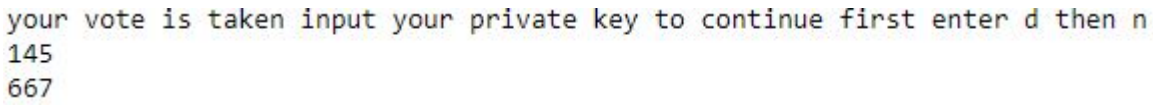


Here mallory's name is not in the voter list

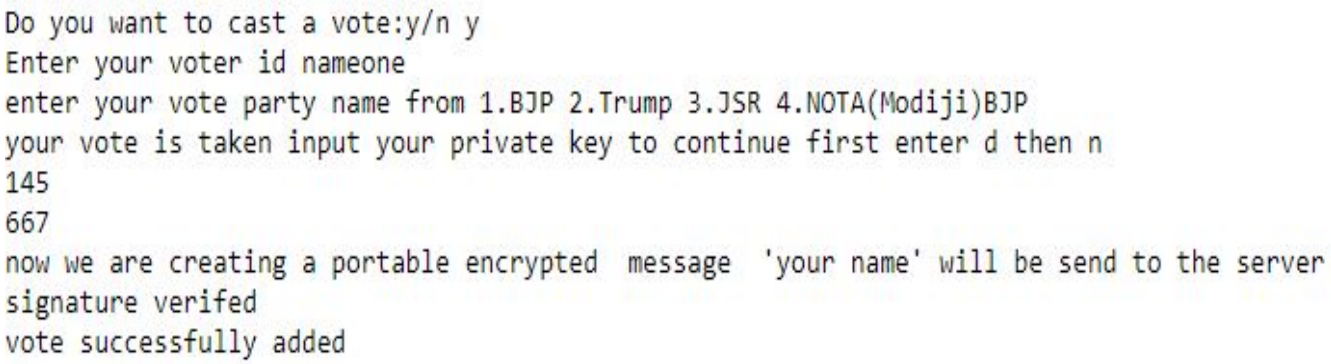
5. Now we see an example of an authentic voter.

Voter one as above is asked to choose the parties name like JSR here which is our message create .
Any string input will be create into message but at server side all the string which are not a parties name will be send into trash





Voter is asked to input his private key remember all this process is running at the voter computer and this private key is only known to the voter.
For the sake of program we have printed the private keys of each user in starting of a run if you want to act like the user and want to know the key
Input the d and n values one by one.



Now if the voter computer takes in the values of message, name of user, private key and creates a signed and encrypted message using the functions

channelmsg,nam=sendvote(vot,private,nam)

Line 274 Jupyter notebook , Code description at Line 133

This create a message which can pass through a channel and maybe corrupted.

The channel message is received by the CTF/Server and it checks for the confidentiality and authentication of the message and notes the voters name and the message he sent

serversideporcessing(channelmsg,nam)

Code description at Line 152, Code called at274 in Jupyter notebook

If the voter is authentic the vote is added in the CTF dictionary else discarded.

6. If mallory tries to fake someone

Suppose in one query a data thief wants to use someones username for example

```
Do you want to cast a vote:y/n y
Enter your voter id nameone
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)JSR
your vote is taken input your private key to continue first enter d then n
103
1763
now we are creating a portable encrypted  message  'your name' will be send to the server
signature is incorrect
[12, 11, 12, 12, 0, 9, 8, 6, 6, 6, 3, 6, 10, 7, 9, 7, 2, 5, 9, 14, 4, 3, 2, 4, 0, 10, 2, 9, 8, 5, 5, 12, 12, 10, 10, 10, 3, 0,
12, 12, 5, 4, 2, 5, 7, 10, 7, 10, 7, 2, 10, 5, 7, 13, 12, 4, 2, 3, 10, 5, 5, 4, 5, 6]
[245, 203, 245, 245, 0, 524, 534, 464, 464, 464, 566, 464, 206, 164, 524, 164, 89, 596, 524, 362, 321, 566, 89, 321, 0, 206, 8
9, 524, 534, 596, 596, 245, 245, 206, 206, 206, 566, 0, 245, 245, 596, 321, 89, 596, 164, 206, 164, 206, 164, 89, 206, 596, 16
4, 379, 245, 321, 89, 566, 206, 596, 596, 321, 596, 464]
```

here a uses uses the name of voter one than at the time of voting when he is asked to input the private key which is not know to him the message is created and at the server it checks for the authentication where this query fails.

7. If mallory himself wants to try using his private key
Mallory is not in the voters list

```
Do you want to cast a vote:y/n y
Enter your voter id nameMallory
we created keys for you and your private key is
(181, 221)
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)BJP
your vote is taken input your private key to continue first enter d then n
181
221
now we are creating a portable encrypted  message  'your name' will be send to the server
signature verifed
your name is not in list
```

If the user has not created its key the system also create a key for him only known to the user and not to the network.

8. If the user has already submitted his vote successfully the system ignores further requests.

```
Do you want to cast a vote:y/n y
Enter your voter id nameone
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)Trump
your vote is taken input your private key to continue first enter d then n
145
667
now we are creating a portable encrypted  message  'your name' will be send to the server
signature verifed
you have already voted
```

9. Finally the votes are printed and made visible to all.

```
{'BJP': 3, 'Trump': 0, 'JSR': 1, 'NOTA': 1, 'trash': 0}
```

If any message was not a party name it counts in trash.

<u>Important Functions used:</u>

1.**sendvote(vot,private,nam)**: Line 133

This function create a message which is signed by the person(nam) private key .

2.serversideporcessing(channelmsg,nam): Line 152
This function represents the working at the CTF server it takes the signed and encrpyted message and checks for confidentiality and authentication .
It decrypts the message and then compares the signed and the hash values for confidentiality.
3. encrypt(pk, plaintext) and decrypt(pk, ciphertext):Line 77 and Line 85
These functions take a list of numbers and perform the modulor exponentiation function on each of them
4. generate_keypair(p, q): Line 62
This function takes 2 prime numbers and then uses them to generate the private and public key. For reducing the chances of repeating prime numbers we use a prime number array (Line 59).
5. hashlib.sha256 : This is an inbuilt function which is used to create a hash value.

Imp**
We have printed the names of valid users and their private keys for the sake of imitating the user(So you can use its values).
In general these will be only known to particular user.

Conclusion :
Implemented a simple RSA voting protocol for online voting .
All the properties for confidentiality and authentication of votes were implemented using RSA asymmetric encrytion.

PA – 3.2

Date:-28/05/2020
Name: Aakash Mathur
Roll No: 17095001
Department: Electronics Engineering
Framework=Jupyter Notebook

Objective: To implement a Voting Protocol ensuring all the requirements of the ideal online voting protocol.

Theory:

various properties required to implement a good but simple voting protocol these are as follows:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else vote. (This turns out to be the hardest requirement.)
5. No one can change anyone else vote without being discovered.
6. Every voter can make sure that his vote has been taken into account in the final tabulation.
7. Everyone knows who voted and who didn't.

In this near-ideal voting protocol, we have assumed that we receive the votes in such a manner so that we can not find any relation between voter and vote.
Somewhat analogous to submitting the votes in a closed ballot box.

Design of the protocol:

The most important part of this complete process is blinding the signature as and the unique id number so that the CTF can not relate any particular vote id number to a voter.

The steps of the process are listed below:

(1) Each voter generates 10 sets of messages, each set containing a valid vote for each possible outcome (e.g., if the vote is a yes or no question, each set contains two votes, one for “yes” and the other for “no”). Each message also contains a randomly generated identification number, large enough to avoid duplicates with other voters.

(2) Each voter individually blinds all of the messages (see following note on blinding) and sends them, with their blinding factors, to the CTF.

(3) The CTF checks its database to make sure the voter has not submitted his blinded votes for signature previously. It opens nine of the sets to check that they

are properly formed. Then it individually signs each message in the set. It sends them back to the voter, storing the name of the voter in its database.

(4) The voter unblinds the messages and is left with a set of votes signed by the CTF. (These votes are signed but unencrypted, so the voter can easily see which vote is “yes” and which is“no.”)

(5) The voter chooses one of the votes and encrypts it with the CTF’s public key.

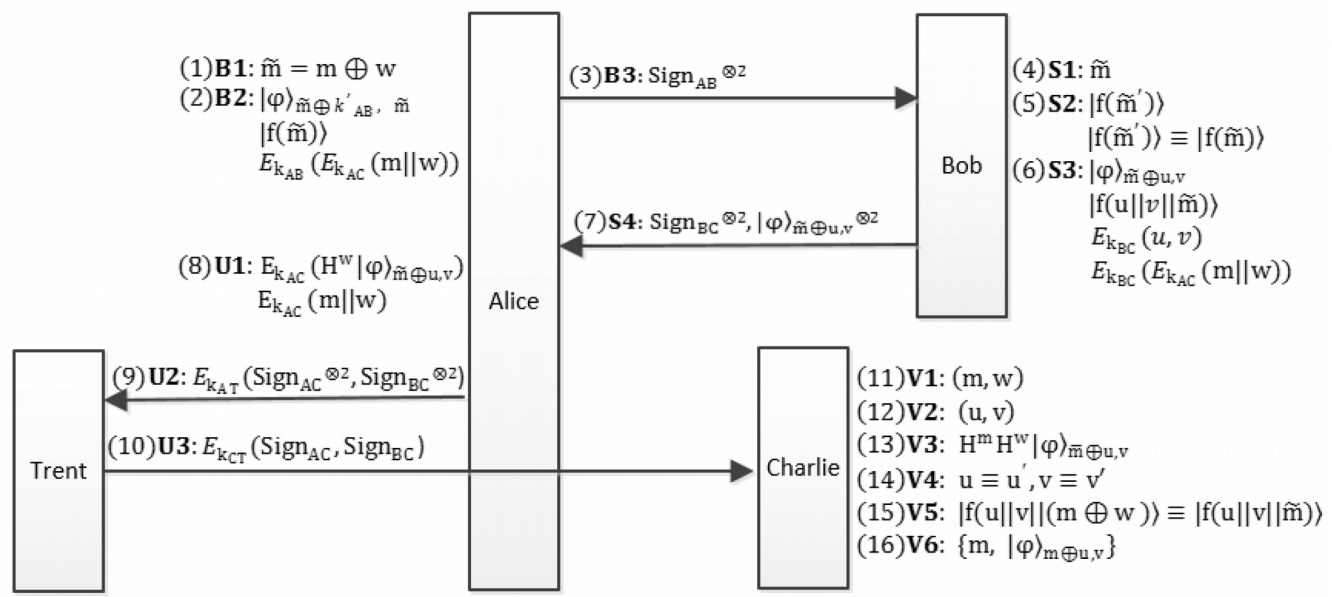
(6) The voter sends his vote in.

(7) The CTF decrypts the votes, checks the signatures, checks its database for a
duplicate identification number, saves the serial number, and tabulates the votes. It
publishes the results of the election, along with every serial number and its associated vote.

THEORY:

Blind Signatures :

a Blind signature is a form of digital signature in which the content
of a message is disguised (blinded) before it is signed. The resulting blind signature can be publicly verified against the original, unblinded message in the
manner of a regular digital signature. Blind signatures are typically employed in
privacy-related protocols where the signer and message author are different parties.
Examples include cryptographic election systems and digital cash schemes.



MATHEMATICAL EXPLANATION OF RSA BLIND SIGNATURES:

Suppose C wants D to sign the message m blindly. Suppose D's public key is (e,n) and her private key is (d,n).

C first blinds the message m, by multiplying it by k^e mod n, where k is a randomly chosen number called the blinding factor. C sends the blinded message m. k^e mod n to D.
Next, D signs the blinded message, resulting in $(m . k^e)^d$ mod n, and sends the signed blinded message back to C.
Finally, C unblinds the message by dividing by k mod n, resulting in $(m . k^e)^d$ / k mod n.

For RSA: $K^{ed} = K^1$

$(m . k^e)^d$ / k mod	**=**	**$m^d.k^{ed}$ / k mod n**	**(mod n)**	**elementary equivalence**
	=	**m^d. k / k mod n**	**(mod n)**	**by the reasoning given above**
	=	**m^d mod n**	**(mod n)**	**which is the D's signature on m**

Confidentiality:

In the process, Voter A sends the 10 sets of messages (signed and blinded) encrypted by CTF's Public key so that only CTF can decrypt the sets of messages as only CTF knows its private key.

X=E(M_signed_Blinded,CTF_pubk)
M_signed_Blinded=D(X,CTF_prik)

Where CTF_pubk= CTF’s Public key
CTF_prik=CTF’s Private key.

Since the decrypted message is itself blinded it provides an extra layer of confidentiality to the process.

Authentication:

voter A prepares a Blinded message to CTF and encrypts it using voter A's private key before transmitting it.
CTF can decrypt the message using A's public key.
Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature.
In addition, it is impossible to meaningfully alter the vote without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

It is possible to provide both authentication and confidentiality by a double use of the public-key scheme:
M_Blinded is the original (Blinded) vote message of voter A and Z is the message which travels through the channel.

Z = E(CTF_pubk , E(PRa , M_Blinded))
M_Blinded= D(PUa , D(CTF_prik , Z))

Message generation for alphabet field:

Secure Hash Function(SHA): is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001.

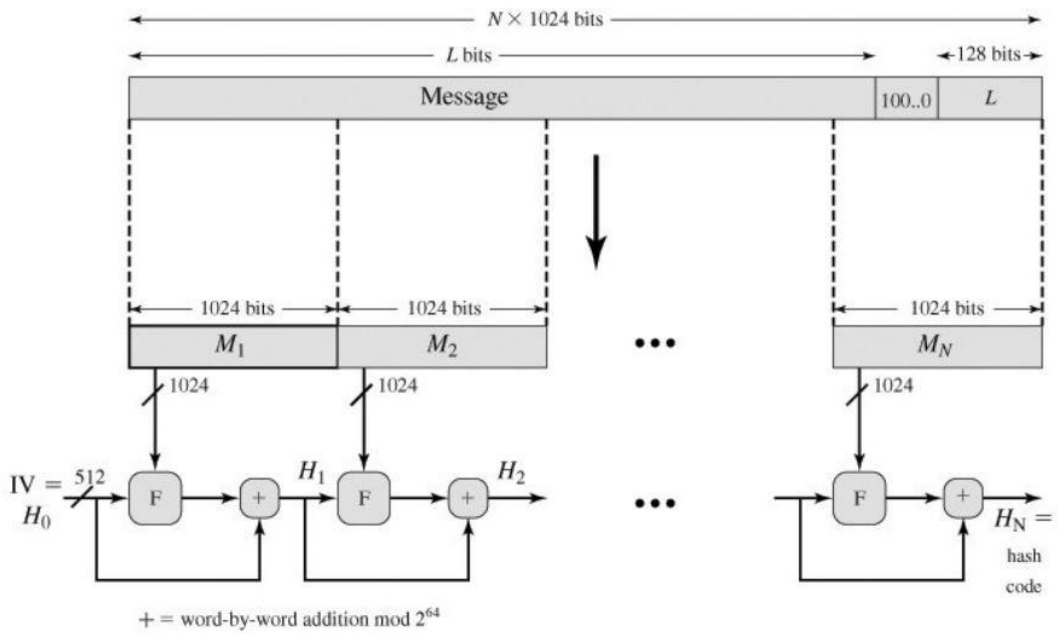
SHA are used to create a message of fixed length that is easy for further processing.

The algorithm takes as input a message with a maximum length of less than 2^128 bits and produces as output a 512-bit message digest. • The input is processed in 1024-bit blocks

Some Information for various variants of SHA

The given message is made into a multiple of 1024 bits (n*1024) by adding padding and a number denoting the length of the original message.Every 1024 bits are passed through are repetitive functio

n



Working of SHA

The output of the SHA algorithm is expected to be collision-resistant

Encryption:

First of all the message is blinded and signed by the voter A Then the Blind and signed message is encrypted with the public key of CTF so that only CTF can decrypt it

We have used 2 different keys for server one for Blinding and others for confidentiality:

Equation:

c=(m*r^eb(mod nb)^e)mod n

where m is the message, {e,n} is the server public key (for confidentiality), {eb, nb} is the server public key for confidentiality, and c is the encrypted message.

Decryption:

CTF needs to decrypt the message 2 times in the process once when the user sends the votes to be signed by CTF and then when the user sends the unblinded unsigned vote.

In both cases :

m= (c^d)mod n

where m is the message, {d,n} is the server private key, and c is the encrypted message
When checking the 9 sets of the total 10.
CTF will further have to unblind the m.

m_unblind=(m^d*r^-1) mod n

Where r is the blinding factor.

Then we can check the m_unblind for any error.

Code(Complete code is in appendix):

The above described are the most important functions of code.

They are explained below:

(A)def blindgenerator(gtid,r, text):

```
# takes in a list of text (contains the expected partynames) and convert each into a corresponding hash value
# and adds gtid to a vote also blinds this list
#for refernce("BJP",gitd) ("MJP",gtid) these votes are hashed anf then blinded
# made into a set of these votes(blind) msgset

def blindgenerator(gtid,r,text):

    msgset=[]
    for i in text:
        msg = hashlib.sha256(i.encode())
        msg = msg.hexdigest()

        msg= hstintarray(msg)
        msg.append(gtid)

        bl=blindalist(msg,r,ser_bkpub)

        msgset.append(bl)
return msgset
```

This function is used to generate the blinded sets of messages and attach a unique id to the set. We first convert the hash into hexadecimal and then do the blind operation on it using the blind factor r.

(B)def blindalist(msg,r, key):

```
def blindalist(msg,r,key):
    e,m = key
    rpow = power(r,e,m)
    bl=[]
    for i in msg:
        x=(i*rpow)%m
        bl.append(x)
    # performs the blind operation simple to understand
return bl
```

this functoin takes a list and blind it using r factor (m*(r^e))%n

(C)def signenncrypter(blindsets,rset,private):

```
def signencrypter(blindsets,rset,private):
    siblindsets=[]#voter signed blind sets

    for i in blindsets:
        simsgset=[]
        for l in i:
```

```
7.              temp=encrypt(private,l)
8.              simsgset.append(temp)
9.          siblindsets.append(simsgset)
10.
11.     sirset=encrypt(private,rset)#voter signed r values
12.
13.     ensiblindsets=[]#encrypted signed blindvotes
14.     enblindsets=[]#encrypted non signed blind votes
15.     enrset=[]# encrypted r values
16.     ensirset=[]#encrypted signed r values
17.
18.     for i in blindsets:
19.         enmsgset=[]
20.         for l in i:
21.             temp=encrypt(ser_pub,l)
22.             enmsgset.append(temp)
23.         enblindsets.append(enmsgset)
24.     for i in siblindsets:
25.         ensimsgset=[]
26.         for l in i:
27.             temp=encrypt(ser_pub,l)
28.             ensimsgset.append(temp)
29.         ensiblindsets.append(ensimsgset)
30.
31.     enrset=encrypt(ser_pub,rset)
32.     ensirset=encrypt(ser_pub,sirset)
33. return (ensiblindsets,enblindsets,enrset,ensirset)# returns the channel msg which is
    encrypted and signed
```

This function is used sign the messages and encrypt for channel movement :

Signs the blindsets and r values

The blinds and r values are singed by users private key for authentication(line4-11)

Encrypted Signed blinded sets of the message:

Encrypted using the public key of CTF (line 24-29)

Encrypted blind sets of messages:

Encrypted using the Public key of CTF(line 18-23)

Encrypted Signed Set of R:

Encrypted using CTF public key (line 32)

Encrypted Set of R:

Encrypted using CTF public key (line 33)

(D)def serverside(channelpack,nam):

```
def serverside(channelpack,nam):
    if(nam not in CTR) or (CTR[nam]!="bmnotcreated"):#checks if the name is in list and blinds are not created already
        temp=[]
        print("Either you have already create blinds or your name is not in list")
        return temp,0
    CTR[nam]="blindsalreadycreated"

    ensiblindsets,enblindsets,enrset,ensirset=channelpack
    #******************this part decrypts the message and then chech the authentication**********
    #decryption using server private key
    siblindsets=[]
    blindsets=[]
    rset=[]
    sirset=[]
    for i in enblindsets:
        msgset=[]
        for l in i:
            temp=decrypt(ser_pri,l)
            msgset.append(temp)
        blindsets.append(msgset)
    for i in ensiblindsets:
        simsgset=[]
        for l in i:
            temp=decrypt(ser_pri,l)
            simsgset.append(temp)
        siblindsets.append(simsgset)

    rset=decrypt(ser_pri,enrset)
    sirset=decrypt(ser_pri,ensirset)


    print("decryption done")
    #sign is checked for authentication
    unsiblindsets=[]
    unsirset=[]

    for i in siblindsets:
        unsimsgset=[]
        for l in i:
            temp=decrypt(pubkeymap[nam],l)
            unsimsgset.append(temp)
        unsiblindsets.append(unsimsgset)

```

```
unsirset=decrypt(pubkeymap[nam],sirset)
temp=[]

retr=0
if unsirset==rset and unsiblindsets==blindsets:

    # if blinds are authentic
    print("User verified")
    print("now we will open nine blinds")
    no=random.randrange(0,no_of_blinds)
    # a random blind is not opened while other are opened
    print("we will not open this index")
    print(no)
    j=0
    c=0
    retmsgset=[]

    for i in blindsets:
        if(j!=no):
            c=c+blindchecking(i,rset[j])
            #checks if other blinds are correct in format
        else :


            retmsgset=i
            retr=rset[no]

        j=j+1
    if(c==no_of_blinds-1):
        #if the number of correct blinds c = one less than total blinds
        # we assume format is correct

        # the selected message is now signed by server private key

        for i in retmsgset:
            tm= encrypt(ser_bkpri,i)
            temp.append(tm)
        retr=rset[no]
         # **********message is signed above**********


    else :
        print("you tried to send an incorrect message  query failed please check format")
        retr=0



else:
            print("verification failed ")
```

```
101.            retr=0
102.
103.
104.        return temp,retr
```

This function represents the process happening at the server after receiving the vote
First, we check whether the voter has already voted or not if he has then we do not count his repeated vote otherwise update his status(line 2-6)
Then we decrypt the signed , unsigned blind votes and r array using server's private key(line 10-32)

We unsign the blinds and unsign r(blind factor) array also(line 34-44)

we then check for the authentication (line 48)

Then we will choose a random number between 0 to 9 and we will not open that index we will open/unblind all other 9 sets of messages to check the structure (line 61-64)

If the number of correct structures is 9 then we proceed forward and sign the remaining one (still blind) set of the message and return it to Voter (line 78-96)

(E)def blindchecking(msgset,r):

```
1.  def blindchecking(msgset,r):
2.      rin=multiplicative_inverse(r,n)
3.      unblimsgset=[]#contains the unblinded message
4.      for i in msgset:
5.          temp2 = blindalist(i,rin,ser_bkpub)
6.          gtidset.add(temp2[-1])# the corressponding gtid is stored in the server
7.          temp2.pop()
8.          unblimsgset.append(temp2)
9.      c=0
10.     phash=dict()
11.     for p in partyname:
12.         msg = hashlib.sha256(p.encode())
13.         msg = msg.hexdigest()
14.         hp= hstintarray(msg)
15.         phash[p]=hp
16.         if hp in unblimsgset:
17.             c=c+1
18.     # checks if all the messages are in correct format
19.     if c==4:
20.         return 1
21.     else:
22.         return 0
```

This function first stores the unique id of the set so that no repetition could occur(line 6) it also unblinds the message (line 4-8)
After that, it checks whether all the messages in the set are legitimate parties or not (line 9-18)

If the structure is correct it returns 1 else 0 (line 19-22)

(F)def servervotecount(channelvote):

```
1.  def servervotecount(channelvote):
2.      # this function recieves the vote and then counts if it is a valid vote also adds the gtid to
    gtid set for future refernce
3.      vote =decrypt(ser_pri,channelvote)
4.      #print("reciveed and decryted")
5.      #print(vote)
6.      print("now we will reading the vote")
7.      #decrypts the message vote
8.      read=decrypt(ser_bkpub,vote)
9.      print(read)
10.     if(read[-1] in gtidset):
11.         # if the gtid is already used the server ignore it
12.         print("blind message set already used")
13.     else:
14.         #else counts a valid vote
15.         gtidset.add(read[-1])
16.         read.pop()
17.         f=0
18.         for p in votemap:
19.             msg = hashlib.sha256(p.encode())
20.             msg = msg.hexdigest()
21.             hp= hstintarray(msg)
22.             if(hp==read):
23.                 votemap[p]=votemap[p]+1
24.                 f=1
25.         if(f==0):
26.             votemap["trash"]=votemap["trash"]+1
```

This function takes in the message and decrypts it(line -3) and then remove the servers blind sign to read the message(line -8) , also it gets the transaction id and if this id was used before it ignores the message else add the id into its set then the message is matched with the possible cases if the message is one of the praty name it is counted else counted in trash

CODE WALK-THROUGH :

The code provides an interface for voters to vote to make it real if also provide possibility of malicious acts that also through its interface and stopping them

Case1:Lets first consider the ideal case of a registered voter whose name is in the list (if any voter has not created his keys over system creates one for him)

1.We first input the number of total queries and then ask for voting

```
Input the numbers of query10
Do you want to create vote y/ny

Enter your voter id name two
```

2. We ask for the user id (here name)
3. we give the user to creates his own message(like mallory can do) which may or may not be correct in format if the voter choose some value the remaining votes of 10 are created in correct format

```
if you want to create your own message enter number n<10 0
```

since our voter is not a bad guy he will create all 10 in correct format and not create his own format at all

4. The blinds get created and asks for private key
The private key is only accessible to the voter we have printed it for user reference only as you may not know it.

```
Input the numbers of query10
Do you want to create vote y/ny
Enter your voter id nametwo
(17957, 103459)
we will first create blind votes for you
if you want to create your own message enter number n<10
your blinds are created
now we will sign your messages
Input your private key to continue first enter d then n
17957

103459
```

5. Now the singed and encrypted message is sent to the server which decrypts it and verify the user as in this case also It decides randomly not to open vote no 4. and check others for format.

```
now we are sending your message to server with your name
decryption done
User verified
now we will open nine blinds
we will not open this index
4
```

6. if the blinds are in correct format no error occurs and **blindly signs one set** and sends it back
7. The interface asks how may votes the user wants to cast a good user will only cast one vote but mallory is cunning she may try to use these votes in the set to cast multiple vote

```
number of votes you want to send we expect n=1 1
```

8. Since our user is nice he will only use one vote
here he give index 0 which is of BJP (1 for MJP,2 for JSR ,3 Default)

```
now give the index of the vote you want to pass
0
now we will send owr hidden vote to server
now we will reading the vote
[11, 15, 13, 7, 7, 4, 3, 7, 8, 3, 14, 12, 2, 14, 11, 15, 8, 14, 0, 15, 1, 9, 9, 8, 9, 8, 5, 5, 0, 1, 13, 0, 3, 11, 5, 7, 10, 1,
5, 4, 0, 4, 10, 6, 2, 5, 8, 11, 13, 6, 7, 2, 7, 13, 11, 5, 11, 2, 1, 10, 0, 11, 0, 7, 945]
printing votemap for better reach to process
{'trash': 0, 'BJP': 1, 'MJP': 0, 'JSR': 0, 'Default': 0}
```

His vote gets counted this is only printed voter user reference votes are actually counted at the last of all query

Case 2.Voter creates wrong set format:
It is possible for a user to change is vote format which can be done by our interface to show how our code tackles it.

1. We ask if the voter want to create fake sets and he input that 6 sets will be created by him and 4 other will be of correct format

```
Do you want to create vote y/ny
Enter your voter id namefive
we created keys for you and your private key is
(126687, 134041)
we will first create blind votes for you

if you want to create your own message enter number n<10 6
```

2. Each set contains 4 votes so total of 24 fake votes are created we input them as strings

```
please input 4 votes of a set
enter vote textabcd
enter vote textefgh
enter vote textasf
enter vote textdsfsdf
please input 4 votes of a set
enter vote text1
enter vote text2
enter vote text3
enter vote text4
please input 4 votes of a set
enter vote text5
enter vote text6
enter vote textfad
enter vote textadf
please input 4 votes of a set
enter vote textad
enter vote textdfa
enter vote textdaf
enter vote textadf
please input 4 votes of a set
enter vote textadf
enter vote textdaf
enter vote textadf
enter vote textgsb
please input 4 votes of a set
enter vote textdafg
enter vote textwradf
enter vote textadgg
enter vote textadgg
your blinds are created
now we will sign your messages
Input your private key to continue first enter d then n
```

3. The user is asked to give keys and we can see as his name was in list he gets verified and the server choose not two open blind set 6 while other 9 are open of which atleat 5 are of wrong format our server checks it and reports this

```
126687
134041
now we are sending your message to server with your name
decryption done
User verified
now we will open nine blinds
we will not open this index
6
you tried to send an incorrect message  query failed please check format
query failed for some reason
printing votemap for better reach to process
{'trash': 0, 'BJP': 1, 'MJP': 0, 'JSR': 0, 'Default': 0}
```

Case 3 .If the user tries to send more then one vote :

1. The user gets his set singed and is asked how many votes he want to send which he replies 2 (second last line n=1 <-expected, 2 ← input)

```
Do you want to create vote y/ny
Enter your voter id nameone
(61939, 63383)
we will first create blind votes for you
if you want to create your own message enter number n<100
your blinds are created
now we will sign your messages
Input your private key to continue first enter d then n
61939
63383
now we are sending your message to server with your name
decryption done
User verified
now we will open nine blinds
we will not open this index
6
number of votes you want to send we expect n=12
now give the index of the vote you want to pass
```

2.Then he gives 2 indexes same or different which it want to pass to server

```
now give the index of the vote you want to pass
0
now we will send owr hidden vote to server
now we will reading the vote
[11, 15, 13, 7, 7, 4, 3, 7, 8, 3, 14, 12, 2, 14, 11, 15, 8, 14, 0, 15, 1, 9, 9, 8, 9, 8, 5, 5, 0, 1, 13, 0, 3, 11, 5, 7, 10, 1,
5, 4, 0, 4, 10, 6, 2, 5, 8, 11, 13, 6, 7, 2, 7, 13, 11, 5, 11, 2, 1, 10, 0, 11, 0, 7, 1374]
now give the index of the vote you want to pass
1
now we will send owr hidden vote to server
now we will reading the vote
[8, 13, 0, 15, 6, 12, 0, 11, 0, 3, 11, 7, 9, 6, 10, 10, 2, 13, 9, 1, 3, 1, 0, 4, 11, 9, 6, 2, 2, 9, 13, 4, 10, 6, 7, 12, 10, 1
3, 10, 8, 10, 13, 7, 11, 8, 11, 9, 14, 10, 9, 8, 0, 4, 9, 2, 9, 8, 6, 5, 5, 0, 0, 7, 10, 1374]
blind message set already used
```

The server takes the first vote and ignores the second one

Remainging cases:

All these cases are the general cases which were discussed in part 1 also solved by RSA

1) If the user is not in list: His message reaches the server and if checks in the list
where it cancels it
2) if the user input wrong key or fakes some other user: The authentication fails
3) if the user keys are not created: our client creates key and share the public part
4) if user tries more than once to blind votes: Server checks his name in CTR where the blind vote status is if it is create already if skips the query

CONCLUSION:

I implemented the Online elections using the RSA blind signatures, SHA and RSA Cryptosystem which follows all the protocols of ideal elections.

APPENDIX:

```
import random
from Crypto.Hash import SHA256
from random import SystemRandom
import hashlib

#required libraries
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
# modulo exponentiation function return (x^y)%p
def power(x, y, p) :
    res = 1
    x = x % p

    if (x == 0) :
        return 0

    while (y > 0) :


        if ((y & 1) == 1) :
            res = (res * x) % p
```

```
        y = y >> 1
        x = (x * x) % p

    return res
# returns (a^-1)%m
def multiplicative_inverse(a, m):
    m0 = m
    y = 0
    x = 1
    if (m == 1) :
        return 0

    while (a > 1) :


        q = a // m

        t = m

        m = a % m
        a = t
        t = y


        y = x - q * y
        x = t

    if (x < 0) :
        x = x + m0

    return x

#checks if the number is prime
def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True
# create a array of all the primes between 200 to 700 for use in key generation
primes = [i for i in range(200,700) if is_prime(i)]
# returns the keys by taking 2 primes as input
def generate_keypair(p, q):
    n = p * q# n is our base of modulo
    phi = (p-1) * (q-1)#totient function
    e = random.randrange(2, phi)
```

```
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(2, phi)
        g = gcd(e, phi)
    d = multiplicative_inverse(e, phi)
    # returns two tandom keys such that the (e,d) are coprime to phi
    return ((e, n), (d, n))

no_of_blinds=10
CTR = dict()
pubkeymap = dict()# a key table which will be broadcaseted for pubkey refernce
prikeymap=dict()# a kay table only for the reference of the user in case he doesn't no the private key
gtidset=set()# gtid = global transaction id . Each vote set has one and gtidset contains gtid that the server has used till now
partyname =[]
partyname.append("BJP")
partyname.append("MJP")
partyname.append("JSR")
partyname.append("Default")
# this contains the pubkey and privates of the users .
#pubkeymap is braodcasted privatekeymak only for user reference
pubkeymap["one"],prikeymap["one"]=generate_keypair(241,263)
pubkeymap["two"],prikeymap["two"]=generate_keypair(307,337)
pubkeymap["three"],prikeymap["three"]=generate_keypair(419,433)
pubkeymap["four"],prikeymap["four"]=generate_keypair(577,599)

#CTR knows who has generated blind or not (only this)
CTR["one"]="bmnotcreated"
CTR["two"]="bmnotcreated"
CTR["three"]="bmnotcreated"
CTR["four"]="bmnotcreated"
CTR["five"]="bmnotcreated"
# server keys for enryption adn decryption (Confidentiallity)
ser_pub,ser_pri = generate_keypair(3307,3313)
# server key for blinding and unblinding
ser_bkpub,ser_bkpri=generate_keypair(47,53)
#*******both sets are different because of the problems the modulo propety was bringing .ps they still act the same

N = ser_pub[1]
n= ser_bkpub[1]

print(CTR)# contains the list of voters and status of blind generation
#print(prikeymap)

votemap=dict()#contains votes
votemap["trash"]=0
# we set the vote count of each party =0
```

```
for p in partyname:
    votemap[p]=0
# this function encrypt a list (plaintext) using key(pk)
def encrypt(pk, plaintext):
    key, n = pk
    cipher=[]
    for i in plaintext :
        cipher.append(power(i, key, n))
    return cipher

# this function decrypt a list (ciphertext) using key(pk)
def decrypt(pk, ciphertext):
    key, n = pk
    plain=[]
    for i in ciphertext :
        plain.append(power(i, key, n))
    return plain


# converts a string to array of interget for mathematical calculations
def hstintarray(hexstring):

    s=[]
    for char in hexstring:

        if (char>='a')&(char<='f'):
            s.append(ord(char)-ord('a')+10)
        else:
            s.append(ord(char)-ord('0'))
    return s
# performs the reverse of above operation
def hsfintarray(ls):
    s=""
    for i in ls:
        if(i>9):
            #s=''.join(('','s',(chr(ord('a')-10+i))))
            s=s+(chr(ord('a')-10+i))
        else:
            #s=''.join(('','s',(chr(ord('0')+i))))
            s=s+(chr(ord('0')+i))
    return s

# this functoin takes a list and blind it using r factor (m*(r^e))%n
def blindalist(msg,r,key):
    e,m = key
    rpow = power(r,e,m)
    bl=[]
    for i in msg:
        x=(i*rpow)%m
```

```
        bl.append(x)
    # performs the blind operation simple to understand
    return bl

# takes in a list of text (contains the expected partynames) and convert each into a
corresponding hash value
# and adds gtid to a vote also blinds this list
#for refernce("BJP",gitd) ("MJP",gtid) these votes are hashed anf then blinded
# made into a set of these votes(blind) msgset
def blindgenerator(gtid,r,text):

    msgset=[]
    for i in text:
        msg = hashlib.sha256(i.encode())
        msg = msg.hexdigest()

        msg= hstintarray(msg)
        msg.append(gtid)

        bl=blindalist(msg,r,ser_bkpub)

        msgset.append(bl)
    return msgset
#takes the messages and values sign them with voters private key followed by encryption
for
#authentication and confidentiallity of blinds
# also does the same with r values as they are also set to CTR
def signencrypter(blindsets,rset,private):
    siblindsets=[]#voter signed blind sets

    for i in blindsets:
        simsgset=[]
        for l in i:
            temp=encrypt(private,l)
            simsgset.append(temp)
        siblindsets.append(simsgset)

    sirset=encrypt(private,rset)#voter signed r values

    ensiblindsets=[]#encrypted signed blindvotes
    enblindsets=[]#encrypted non signed blind votes
    enrset=[]# encrypted r values
    ensirset=[]#encrypted signed r values

    for i in blindsets:
        enmsgset=[]
        for l in i:
            temp=encrypt(ser_pub,l)
            enmsgset.append(temp)
```

```
        enblindsets.append(enmsgset)
    for i in siblindsets:
        ensimsgset=[]
        for l in i:
            temp=encrypt(ser_pub,l)
            ensimsgset.append(temp)
        ensiblindsets.append(ensimsgset)

    enrset=encrypt(ser_pub,rset)
    ensirset=encrypt(ser_pub,sirset)
    return (ensiblindsets,enblindsets,enrset,ensirset)# returns the channel msg which is
encrypted and signed

# this function takes in blindvotes and r values to check there formats
def blindchecking(msgset,r):
    rin=multiplicative_inverse(r,n)
    unblimsgset=[]#contains the unblinded message
    for i in msgset:
        temp2 = blindalist(i,rin,ser_bkpub)
        gtidset.add(temp2[-1])# the corressponding gtid is stored in the server
        temp2.pop()
        unblimsgset.append(temp2)
    c=0
    phash=dict()
    for p in partyname:
        msg = hashlib.sha256(p.encode())
        msg = msg.hexdigest()
        hp= hstintarray(msg)
        phash[p]=hp
        if hp in unblimsgset:
            c=c+1
    # checks if all the messages are in correct format
    if c==4:
        return 1
    else:
        return 0




def serverside(channelpack,nam):
    if(nam not in CTR) or (CTR[nam]!="bmnotcreated"):#checks if the name is in list and
blinds are not created already
        temp=[]
        print("Either you have already create blinds or your name is not in list")
        return temp,0
    CTR[nam]="blindsalreadycreated"
```

```
    ensiblindsets,enblindsets,enrset,ensirset=channelpack
    #******************this part decrypts the message and then chech the
authentication**********
    #decryption using server private key
    siblindsets=[]
    blindsets=[]
    rset=[]
    sirset=[]
    for i in enblindsets:
        msgset=[]
        for l in i:
            temp=decrypt(ser_pri,l)
            msgset.append(temp)
        blindsets.append(msgset)
    for i in ensiblindsets:
        simsgset=[]
        for l in i:
            temp=decrypt(ser_pri,l)
            simsgset.append(temp)
        siblindsets.append(simsgset)

    rset=decrypt(ser_pri,enrset)
    sirset=decrypt(ser_pri,ensirset)


    print("decryption done")
    #sign is checked for authentication
    unsiblindsets=[]
    unsirset=[]

    for i in siblindsets:
        unsimsgset=[]
        for l in i:
            temp=decrypt(pubkeymap[nam],l)
            unsimsgset.append(temp)
        unsiblindsets.append(unsimsgset)

    unsirset=decrypt(pubkeymap[nam],sirset)
    temp=[]

    retr=0
    if unsirset==rset and unsiblindsets==blindsets:

        # if blinds are authentic
        print("User verified")
        print("now we will open nine blinds")
        no=random.randrange(0,no_of_blinds)
        # a random blind is not opened while other are opened
```

```
        print("we will not open this index")
        print(no)
        j=0
        c=0
        retmsgset=[]

        for i in blindsets:
            if(j!=no):
                c=c+blindchecking(i,rset[j])
                #checks if other blinds are correct in format
            else :


                retmsgset=i
                retr=rset[no]

            j=j+1
        if(c==no_of_blinds-1):
            #if the number of correct blinds c = one less than total blinds
            # we assume format is correct

            # the selected message is now signed by server private key

            for i in retmsgset:
                tm= encrypt(ser_bkpri,i)
                temp.append(tm)
            retr=rset[no]
             # **************message is signed above*************


        else :
            print("you tried to send an incorrect message  query failed please check format")
            retr=0



    else:
        print("verification failed ")
        retr=0



    return temp,retr

def sendvote(vote):
    # this function takes one vote message and encrypts it with server pub key for
confidentiality of vote
    # note the user name is not given
    check=encrypt(ser_pub,vote)
```

```
    #print("the second channel vote encryped")
    #print(check)
    return check



def servervotecount(channelvote):
    # this function recieves the vote and then counts if it is a valid vote also adds the gtid to
gtid set for future refernce
    vote =decrypt(ser_pri,channelvote)
    #print("reciveed and decryted")
    #print(vote)
    print("now we will reading the vote")
    #decrypts the message vote
    read=decrypt(ser_bkpub,vote)
    print(read)
    if(read[-1] in gtidset):
        # if the gtid is already used the server ignore it
        print("blind message set already used")
    else:
        #else counts a valid vote
        gtidset.add(read[-1])
        read.pop()
        f=0
        for p in votemap:
            msg = hashlib.sha256(p.encode())
            msg = msg.hexdigest()
            hp= hstintarray(msg)
            if(hp==read):
                votemap[p]=votemap[p]+1
                f=1
        if(f==0):
            votemap["trash"]=votemap["trash"]+1







if __name__ == '__main__':

    q=input("Input the numbers of query")
    blind = dict()
    q= int(q)
    while(q>0):

        q=q-1
```

```
        f=input("Do you want to create vote y/n")
        if(f=="y"):
            nam = input("Enter your voter id name")
            # if the voter has not created keys we create it for him
            if(nam not in pubkeymap):
                pn = random.choice(primes)
                qn=  random.choice(primes)
                while(qn==pn):
                    qn=randomChoices(primes)
                pubkeymap[nam],prikeymap[nam]=generate_keypair(pn,qn)
                print("we created keys for you and your private key is")

            print(prikeymap[nam])
            # we create some blinds sets of the expected format that is the votes are correct in
them
            print("we will first create blind votes for you")
            blindsets=[]# contains the 10 sets of blind votes(each set has 4 votes)
            rset=[]
            for i in range(0,no_of_blinds):
                gtid = random.randrange(1,n)
                r = random.randrange(2,n)
                g = gcd(r, n)

                while g != 1:
                    r = random.randrange(1, n)
                    g = gcd(r, n)
                # we create random gtid and r value for a set of messages
                msgset=blindgenerator(gtid,r,partyname)
                # create a blinded set of messages with correct format using partynames
                blindsets.append(msgset)
                rset.append(r)
            # if our hacker wants to change some votese he can change them
            fr=input("if you want to create your own message enter number n<10")
            fr = int(fr)
            for i in range(0,fr):
                gtid = random.randrange(1,n)
                r = random.randrange(2,n)
                g = gcd(r, n)
                while g != 1:
                    r = random.randrange(1, n)
                    g = gcd(r, n)
                print("please input 4 votes of a set")
                text=[]
                for qm in range(0,4):
                    inpy=input("enter vote text")
                    text.append(inpy)
                msgset=blindgenerator(gtid,r,text)
                # create a blinded set of messages with  format as per user request
                blindsets[i]=msgset
```

```
            rset[i]=r
        #print("the blind sets created")
        #print(blindsets)
        print("your blinds are created ")

        print("now we will sign your messages")

        print("Input your private key to continue first enter d then n")
        #privd=input()
        #privd = int(privd)
        #privn=input()
        #privn=int(privn)
        #private=(privd,privn)
        channelpack=signencrypter(blindsets,rset,prikeymap[nam])
        # above function create a signed and encrypted copy of blinds for channel use
        print("now we are sending your message to server with your name")
        # we send the message through channe; with name
        blisiset,re=serverside(channelpack,nam)
        # the serve checks for confidentiality ,authentication,random blinds are open
        # if the votes are of correct format the remaining blind vote set is signed and
returned to the voter
        if(re):
            # if the vote is returned
            rin = multiplicative_inverse(re,n)
            votec=input("number of votes you want to send we expect n=1")
            votec=int(votec)
            # this allows to send multiple votes
            for vc in range(0,votec):
                print("now give the index of the vote you want to pass")
                inp=input()
                inp=int(inp)
                tmp=blisiset[inp]
                arr=[]
                for i in tmp:
                    arr.append((i*rin)%n)
                # the vote is selected and its r factor is removed and then send to the server
                #print("lets print the m^d")
                #print(arr)
                vote =arr
                print("now we will send owr hidden vote to server")
                # the vote is send to the server vote = ((m)^d)log n i.e it is blindly singed by
server
                # the below function encrypts it with server public key for confidentiality
                channelvote=sendvote(vote)
                # the server then checks the vote
                servervotecount(channelvote)
        else:
            print("query failed for some reason")
        print("printing votemap for better reach to process")
```

```
        print(votemap)
print("the final result is printed")
print(votemap)
```