PA0(Practical assignment 0)

Objective:

To write a program with a nice UI to Encipher / Decipher the substitution cipher which maps each letter in reverse alphabetical order(atbash cipher).

Theory:

The atbash cipher is a simple substitution cipher from Biblical times;
it reverses the alphabet such that each letter is mapped to the letter in the same position in the reverse of the alphabet (A -> Z, B -> Y).
The atbash cipher is trivial to crack, once you realize that you're dealing with a substitution cipher,
and is highly vulnerable to letter frequency analysis. It's primary modern application is puzzles and games.

mapping->

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA

Implementation details :

To encipher a message, find the letter you wish to encipher in the top row,
then replace it with the letter in the bottom row.
In the example below, we encipher the message 'ATTACK AT DAWN'.
The first letter we wish to encipher is 'A', which is above 'Z', so the first ciphertext letter is 'Z'.
The next letter is 'T', which is above 'G', so that comes next. The whole message is enciphered:

ATTACK AT DAWN
ZGGZXP ZG WZDM

To decipher a message, the exact same procedure is followed.
Find 'Z' in the top row, which is 'A' in the bottom row. Continue until the whole message is deciphered.

Code walkthrough :

python code :

```
# using tkinter module for user interface , so we have to import the tkinter module
#Python offers multiple options for developing GUI (Graphical User Interface).
#Out of all the GUI methods, tkinter is the most commonly used method.
#It is a standard Python interface to the Tk GUI toolkit shipped with Python.
#Python with tkinter is the fastest and easiest way to create the GUI applications.


#To create a tkinter app:

#                       1)Import the module – tkinter
#                       2)Create the main window (container)
#                       3)Add any number of widgets to the main window
#                       4)Apply the event Trigger on the widgets.
```

```
#step 1) importing the module-tkinter
import tkinter as tk
from tkinter import *

# we define a function show which encrypts and decrypts
def show():
    f1=fu.get()
#encrytion
    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)
#decrytion
    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)



m= tk.Tk(screenName=None,  baseName=None,  className='Atbash_Cipher',  useTk=1)

#basic user interface is implemented using tkinter module
#two buttons are provided to either encrypt or decrypt the message

l1=Label(m, text="Encrypted ")
l2=Label(m,text="Decrypted")
l1.grid(row=0,column=0)
l2.grid(row=0,column=2)
fu=IntVar()
rd1=Radiobutton(m,text="Encrypt",variable=fu,value=1)
rd2=Radiobutton(m,text="Decrypt",variable=fu,value=2)
rd1.grid(row=1,column=0)
rd2.grid(row=1,column=2)
intext=Text(m,width=30,height=20,wrap=WORD)
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wrap=WORD)
```

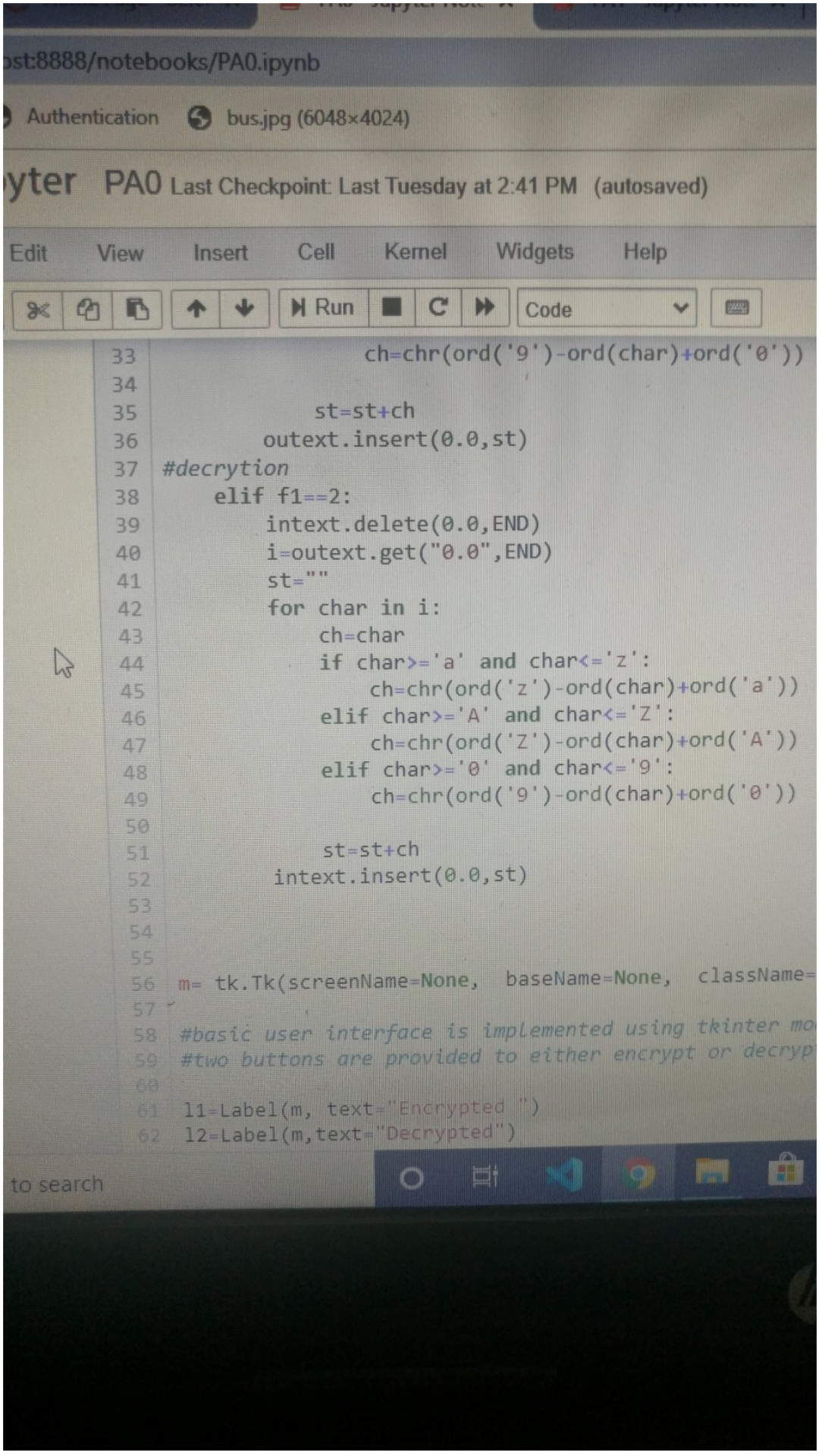
```
outext.grid(row=3,column=2)
act=Button(m,text="ok",command=show)
act.grid(row=3,column=1)
m.mainloop()
```

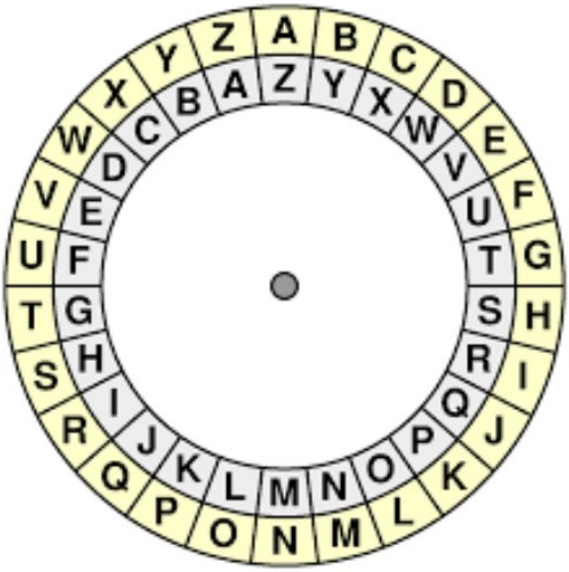
Conclusion:

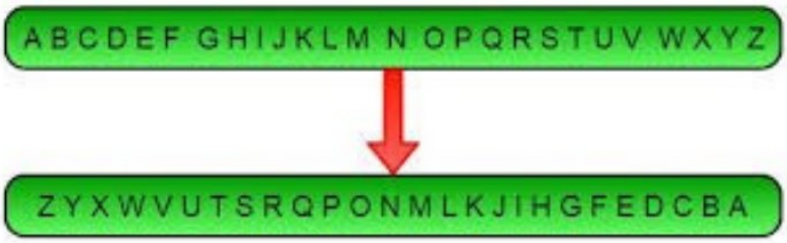
Cryptanalysis ->

The Atbash cipher is trivial to break since there is no key,
 as soon as you know it is an Atbash cipher you can simply decrypt it.
If you didn't know it was an Atbash cipher,
you could break it by assuming the ciphertext is a substitution cipher
, which can still be easily broken, see here.
Alternatively, it can be broken if it is assumed to be an Affine cipher.

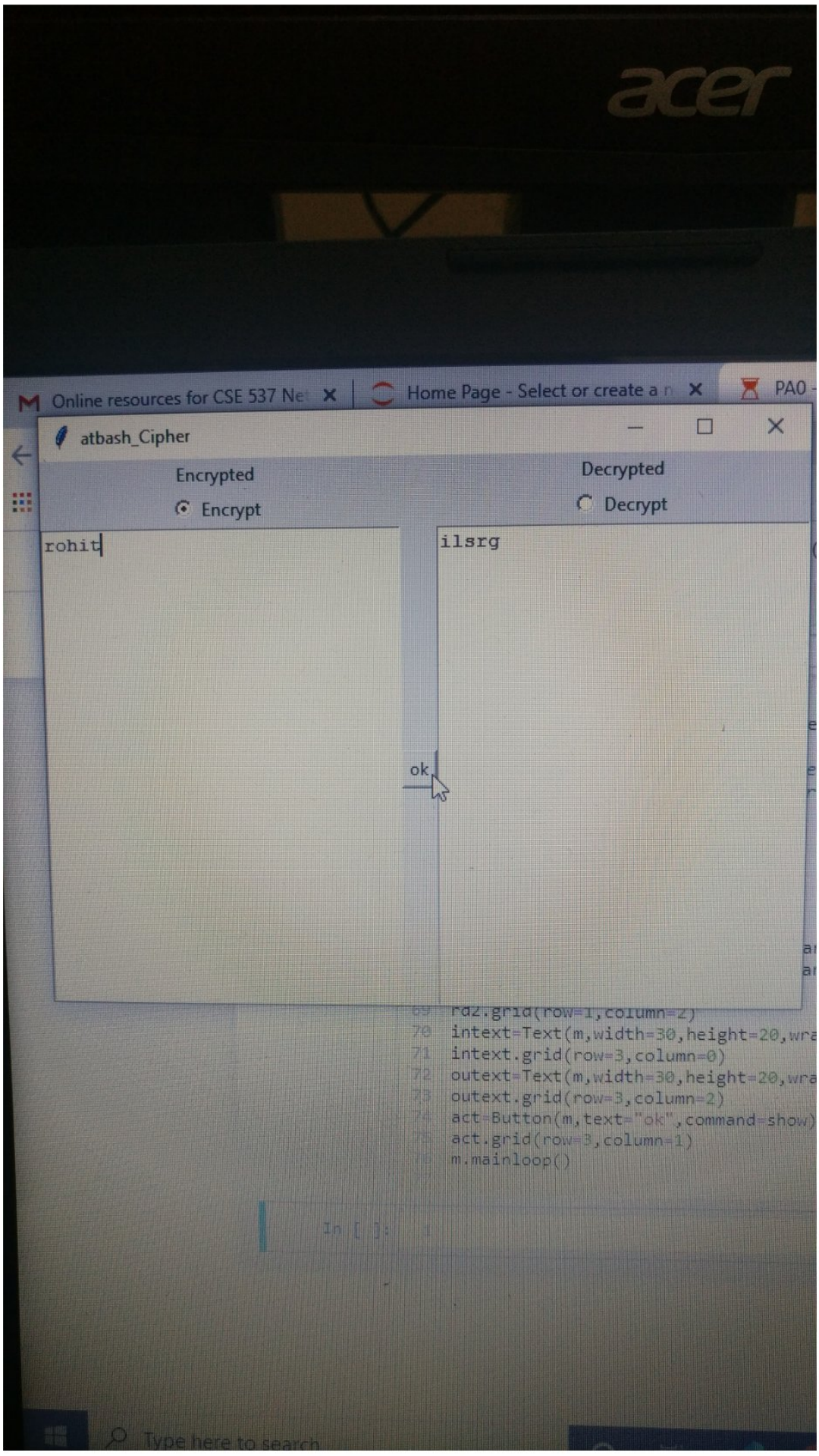
Screenshots and images :

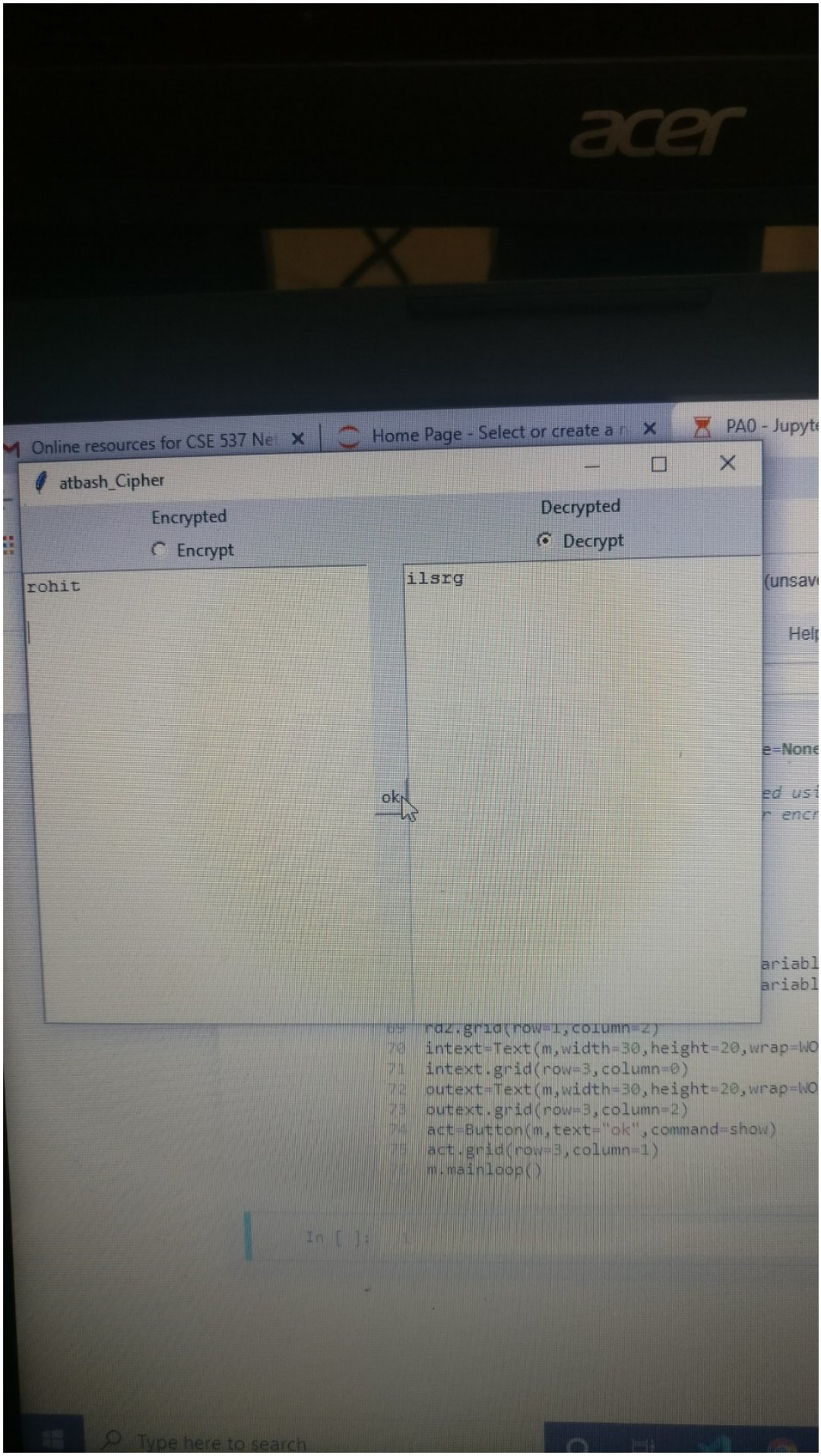

st:8888/notebooks/PA0.ipynb
Authentication
bus.jpg (6048×4024)
yter PA0 Last Checkpoint: Last Tuesday at 2:41 PM (autosaved)
Edit View Insert Cell Kernel Widgets Help
Run
Code
ch=chr(ord('9')-ord(char)+ord('0'))
st=st+ch
outext.insert(0.0,st)
#decrytion
elif f1==2:
intext.delete(0.0,END)
i=outext.get("0.0",END)
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)
m= tk.Tk(screenName=None, baseName=None, className=
#basic user interface is implemented using tkinter mo
#two buttons are provided to either encrypt or decryp
l1=Label(m, text="Encrypted ")
l2=Label(m,text="Decrypted")
to search

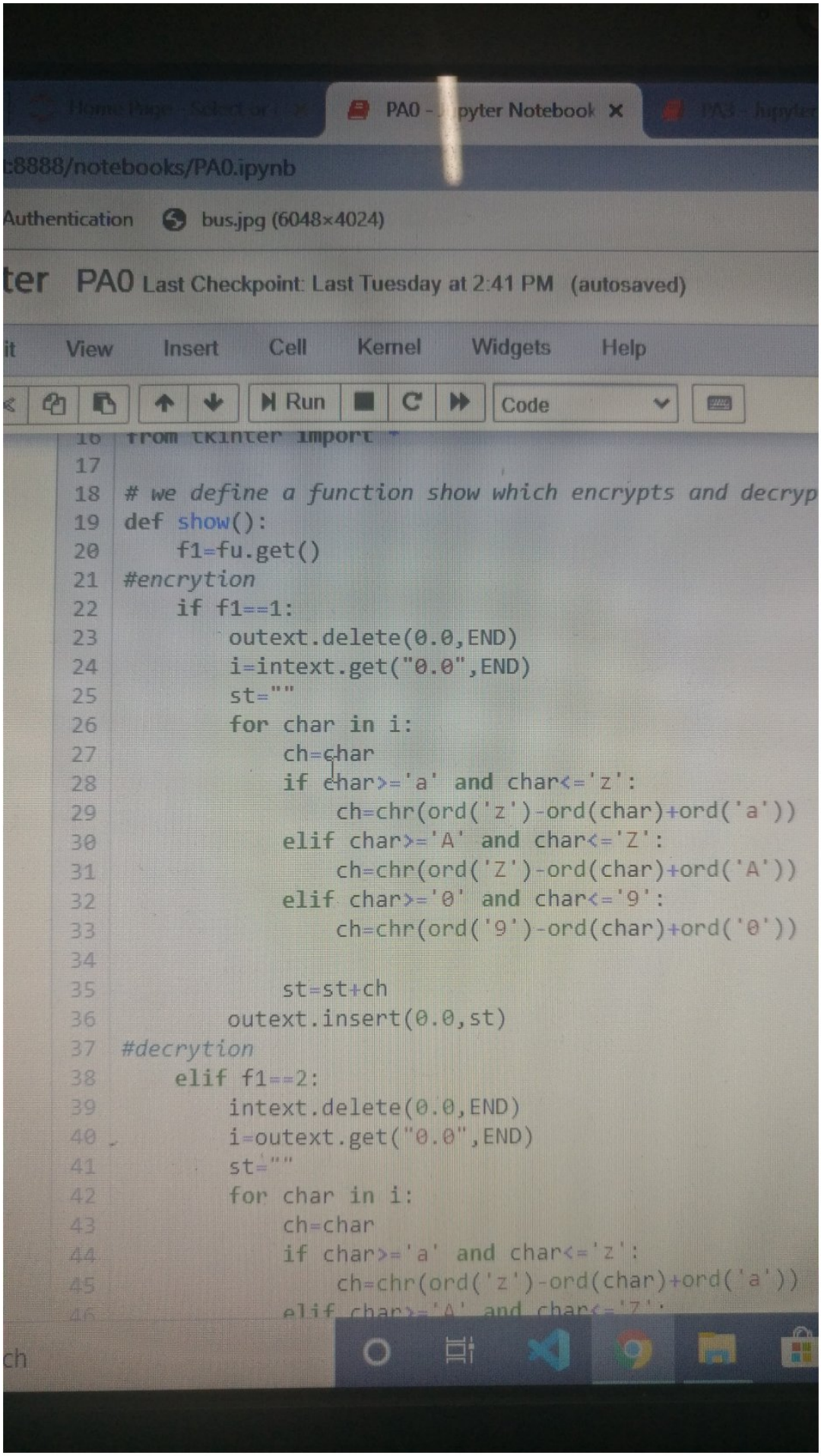

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y

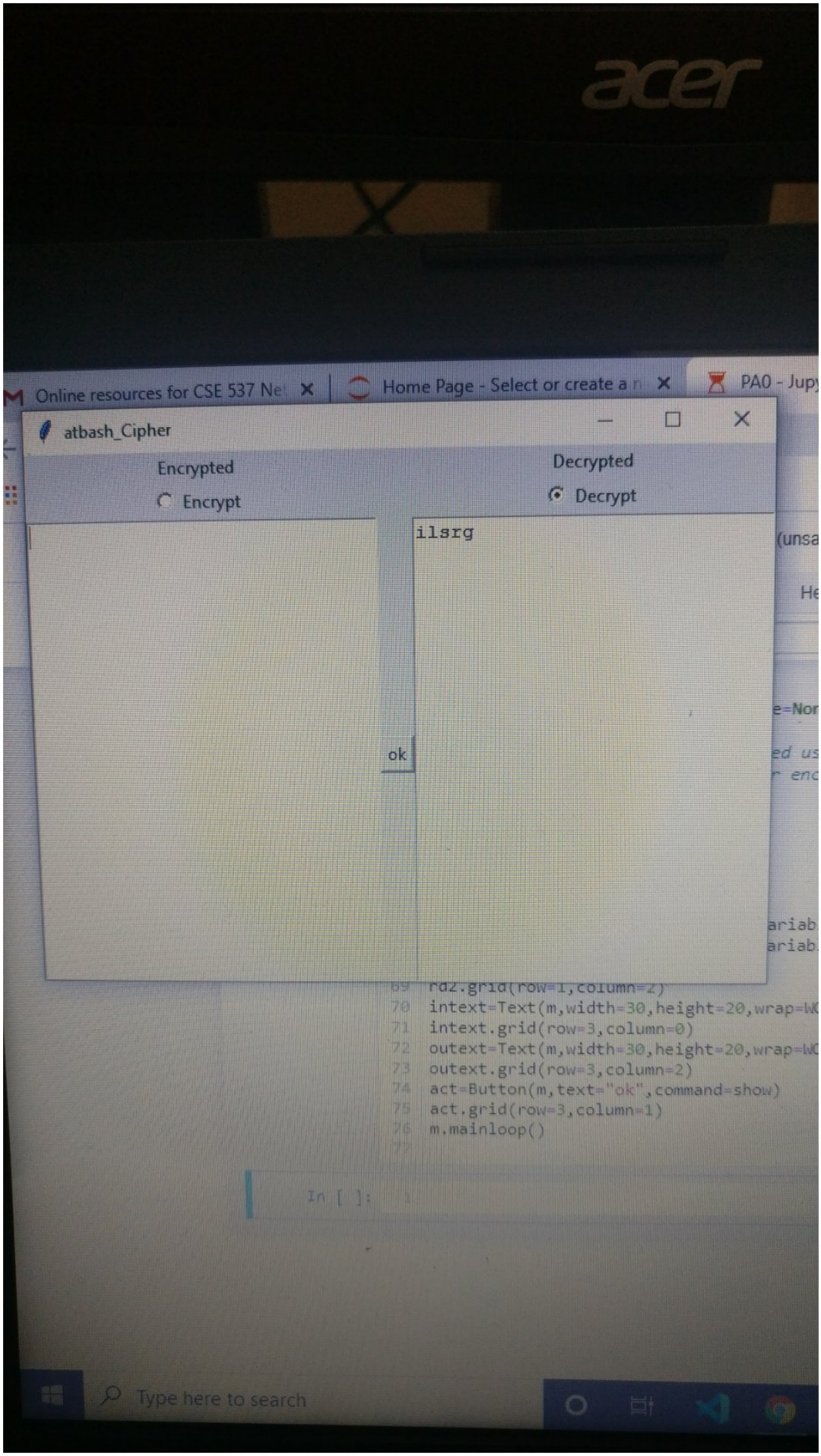

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

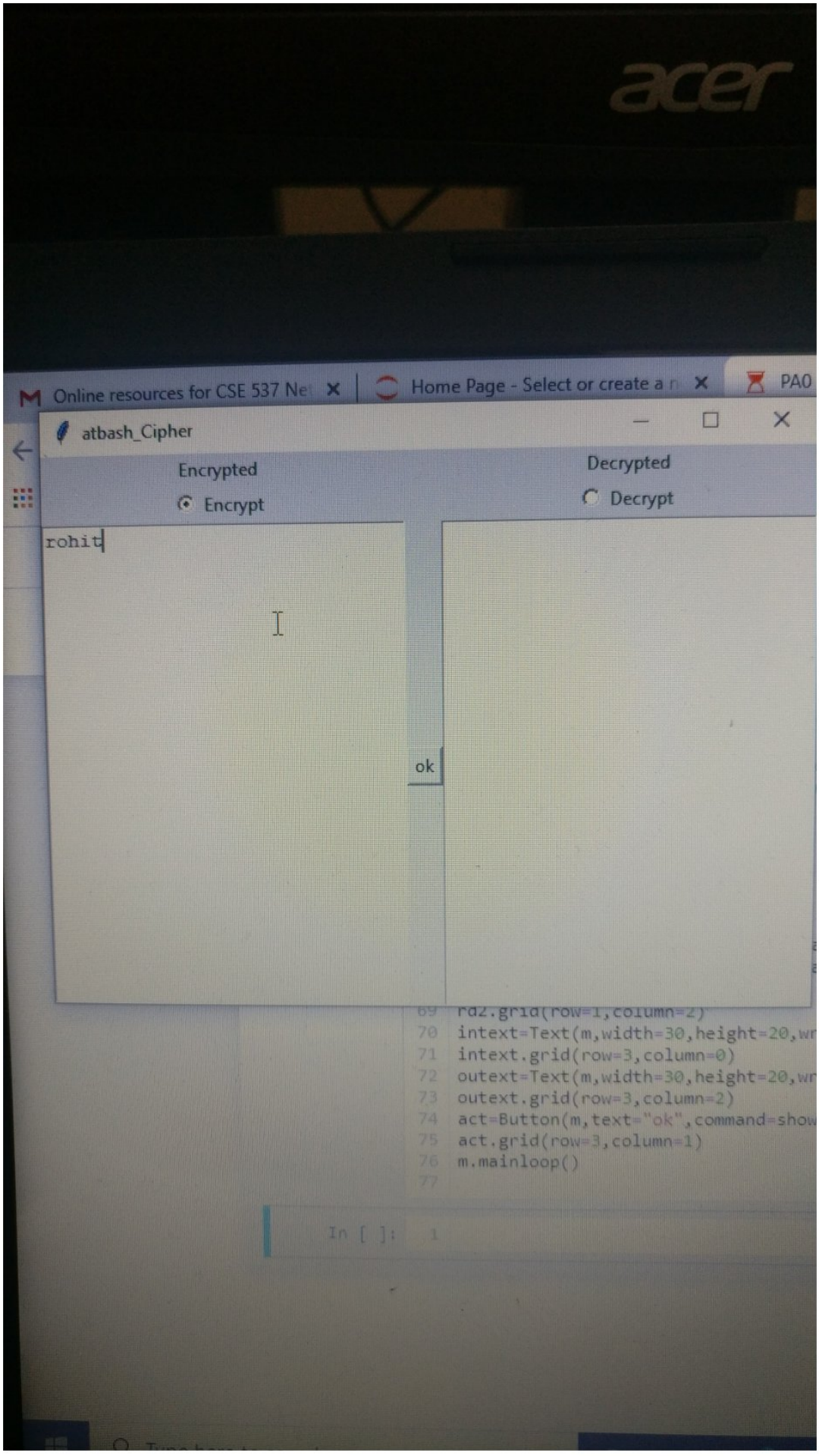
The key

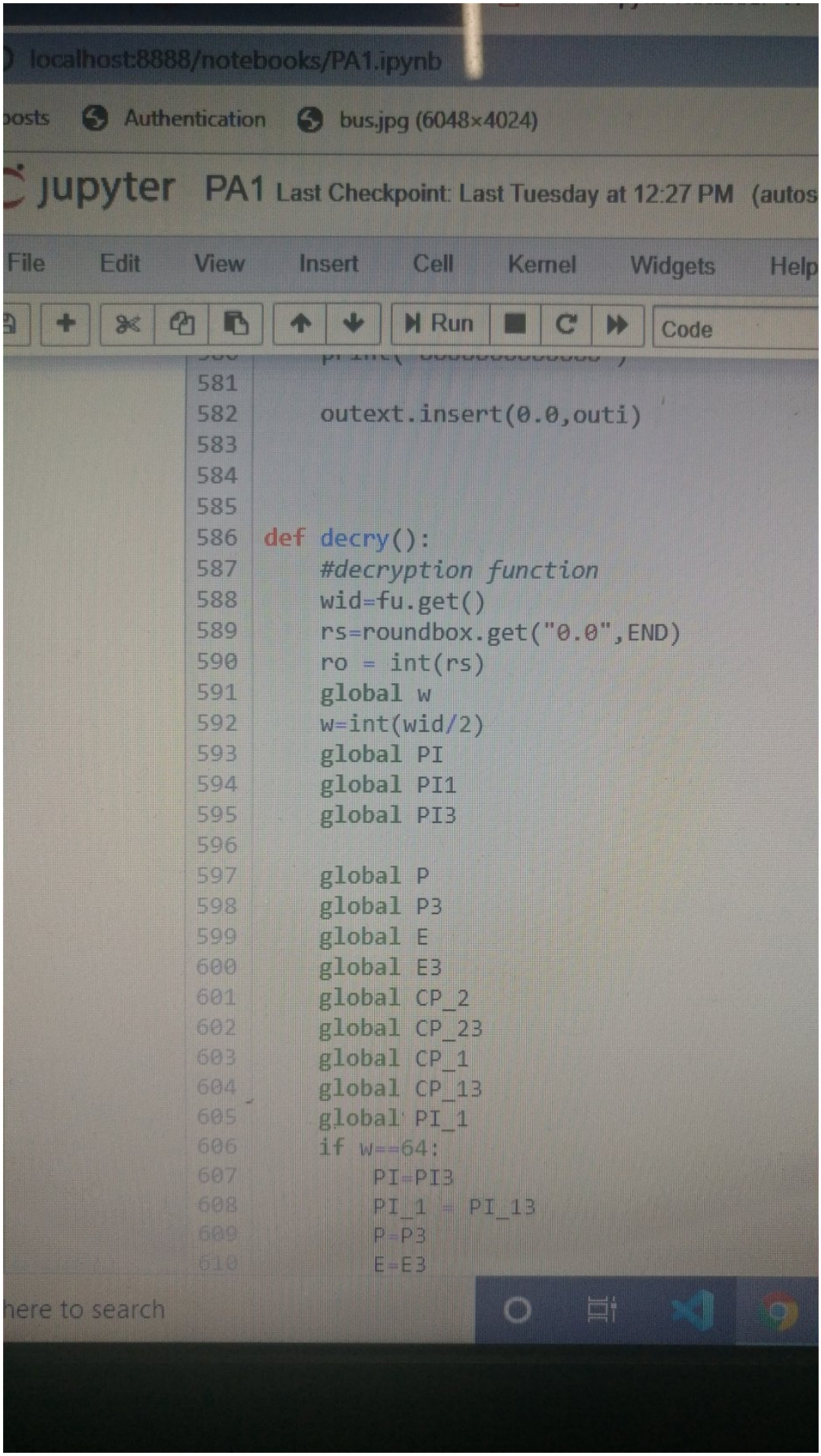

Online resources for CSE 537 Ne
Home Page - Select or create a n
PA0 -
atbash_Cipher
Encrypted
Encrypt
Decrypted
Decrypt
rohit
ilsrg
ok
raz.grid(row=1,column=2)
intext=Text(m,width=30,height=20,wra
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wra
outext.grid(row=3,column=2)
act=Button(m,text="ok",command=show)
act.grid(row=3,column=1)
m.mainloop()
In [ ]:
Type here to search

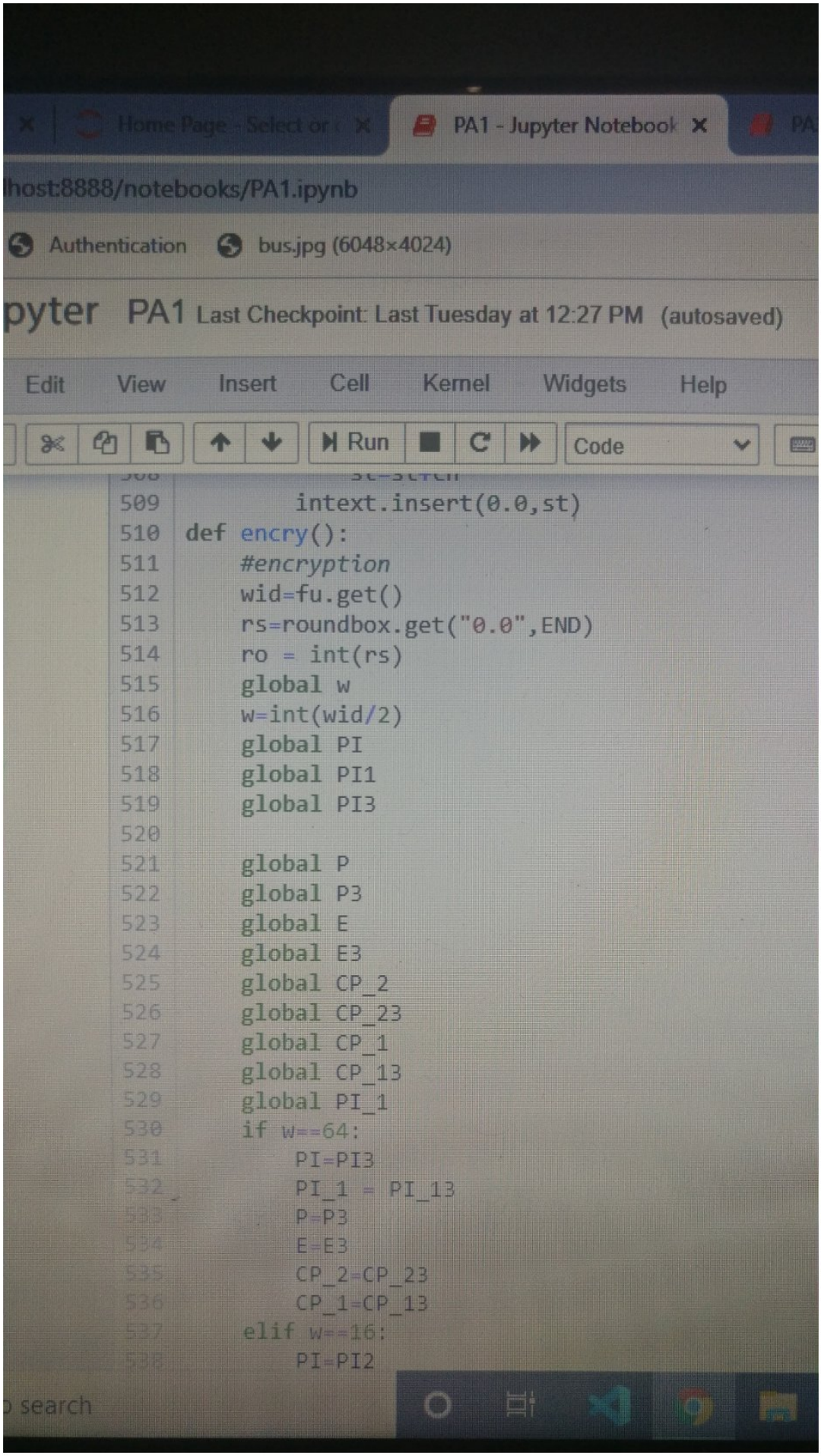

Online resources for CSE 537 Ne
Home Page - Select or create a n
PA0 - Jupyte
atbash_Cipher
Encrypted
Encrypt
Decrypted
Decrypt
rohit
ilsrg
ok
rd2.grid(row=1,column=2)
intext=Text(m,width=30,height=20,wrap=WO
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wrap=WO
outext.grid(row=3,column=2)
act=Button(m,text="ok",command=show)
act.grid(row=3,column=1)
m.mainloop()
In [ ]:
Type here to search

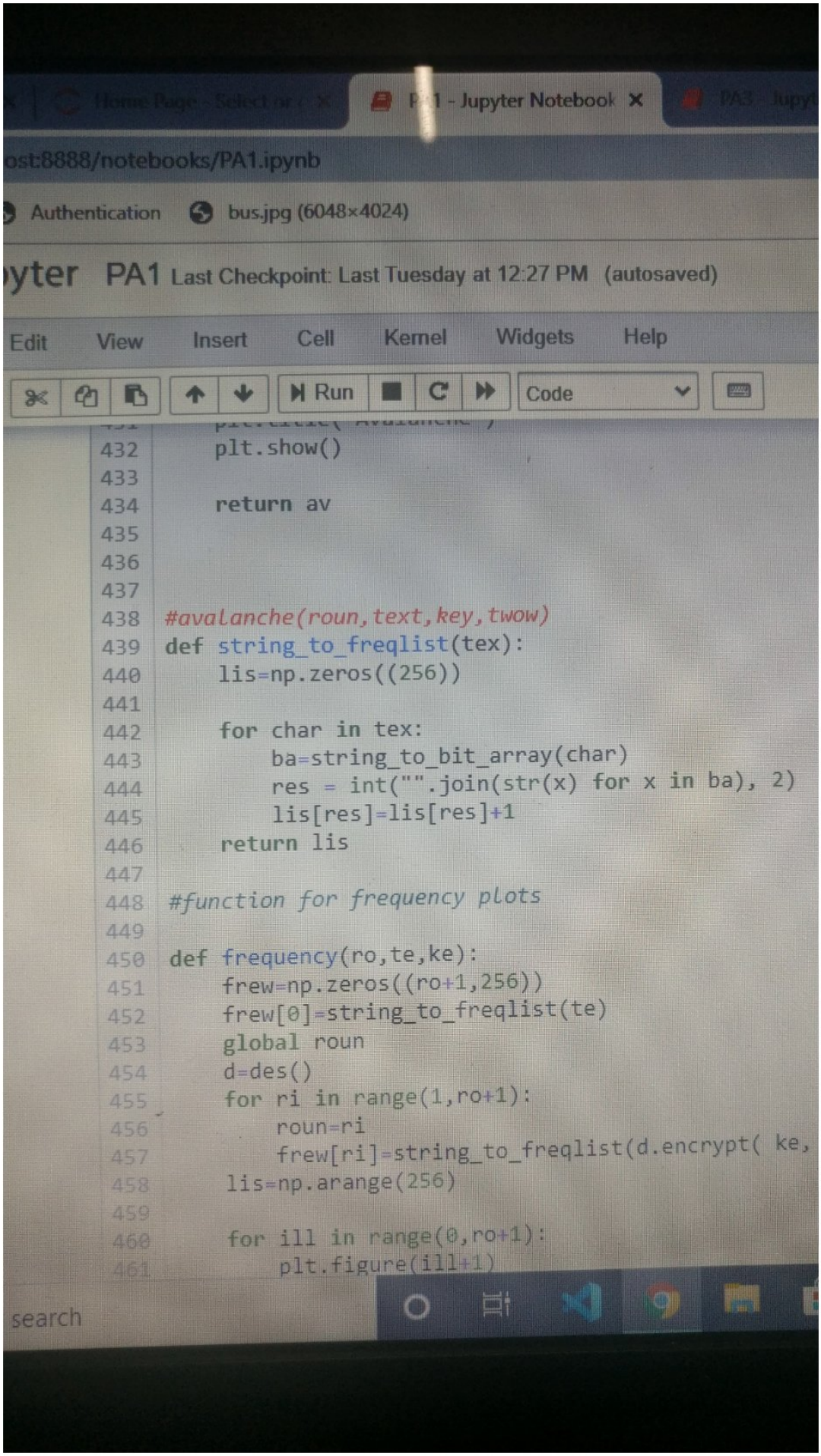

PA0 - Jupyter Notebook
:8888/notebooks/PA0.ipynb
Authentication
bus.jpg (6048×4024)
ter PA0 Last Checkpoint: Last Tuesday at 2:41 PM (autosaved)
it View Insert Cell Kernel Widgets Help
Run
Code
from tkinter import *
# we define a function show which encrypts and decryp
def show():
f1=fu.get()
#encrytion
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)
#decrytion
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'))

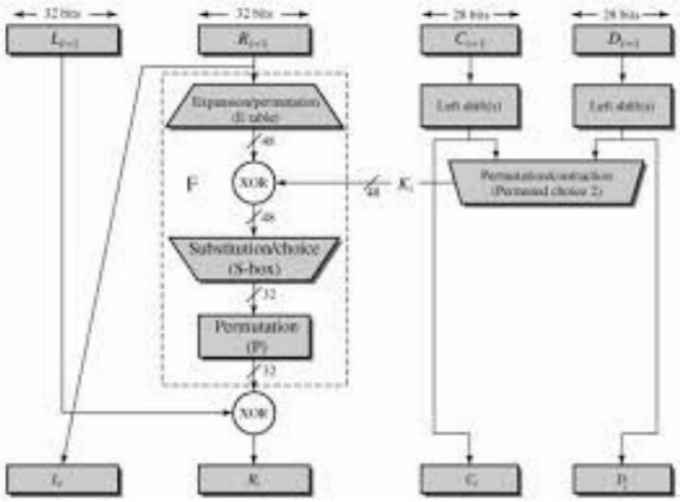

Online resources for CSE 537 Ne
Home Page - Select or create a n
PA0 - Jupy
atbash_Cipher
Encrypted
Encrypt
Decrypted
Decrypt
ilsrg
ok
rd2.grid(row=1,column=2)
intext=Text(m,width=30,height=20,wrap=W
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wrap=W
outext.grid(row=3,column=2)
act=Button(m,text="ok",command=show)
act.grid(row=3,column=1)
m.mainloop()
In [ ]:
Type here to search

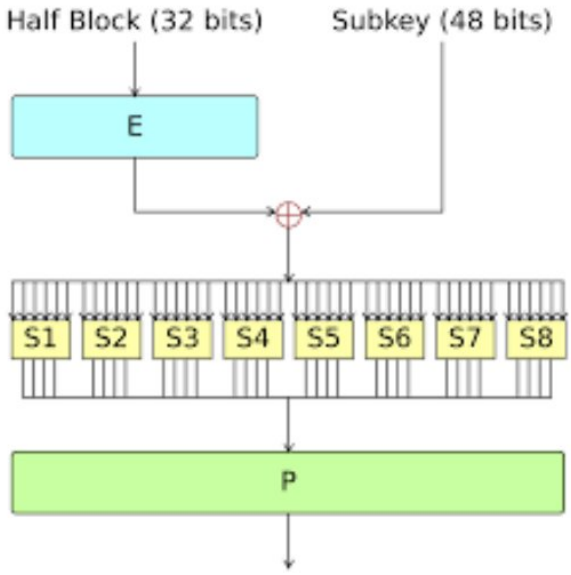

Online resources for CSE 537 Ne
Home Page - Select or create a n
PA0
atbash_Cipher
Encrypted
Encrypt
Decrypted
Decrypt
rohit
ok
rd2.grid(row=1,column=2)
intext=Text(m,width=30,height=20,wr
intext.grid(row=3,column=0)
outext=Text(m,width=30,height=20,wr
outext.grid(row=3,column=2)
act=Button(m,text="ok",command=show
act.grid(row=3,column=1)
m.mainloop()
In [ ]:

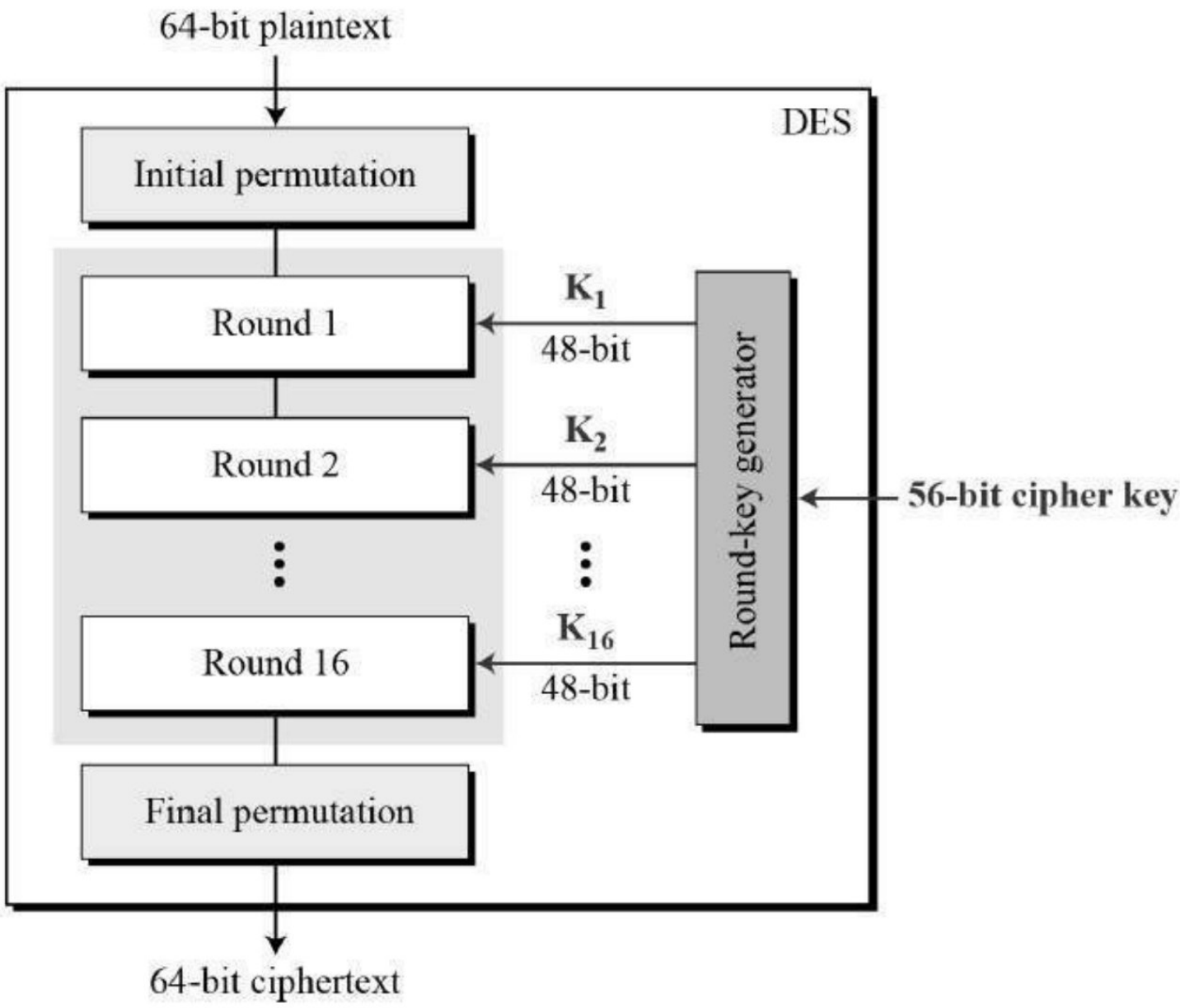

localhost:8888/notebooks/PA1.ipynb
posts
Authentication
bus.jpg (6048×4024)
jupyter
PA1
Last Checkpoint: Last Tuesday at 12:27 PM (autos
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code
outext.insert(0.0,outi)
def decry():
#decryption function
wid=fu.get()
rs=roundbox.get("0.0",END)
ro = int(rs)
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
here to search

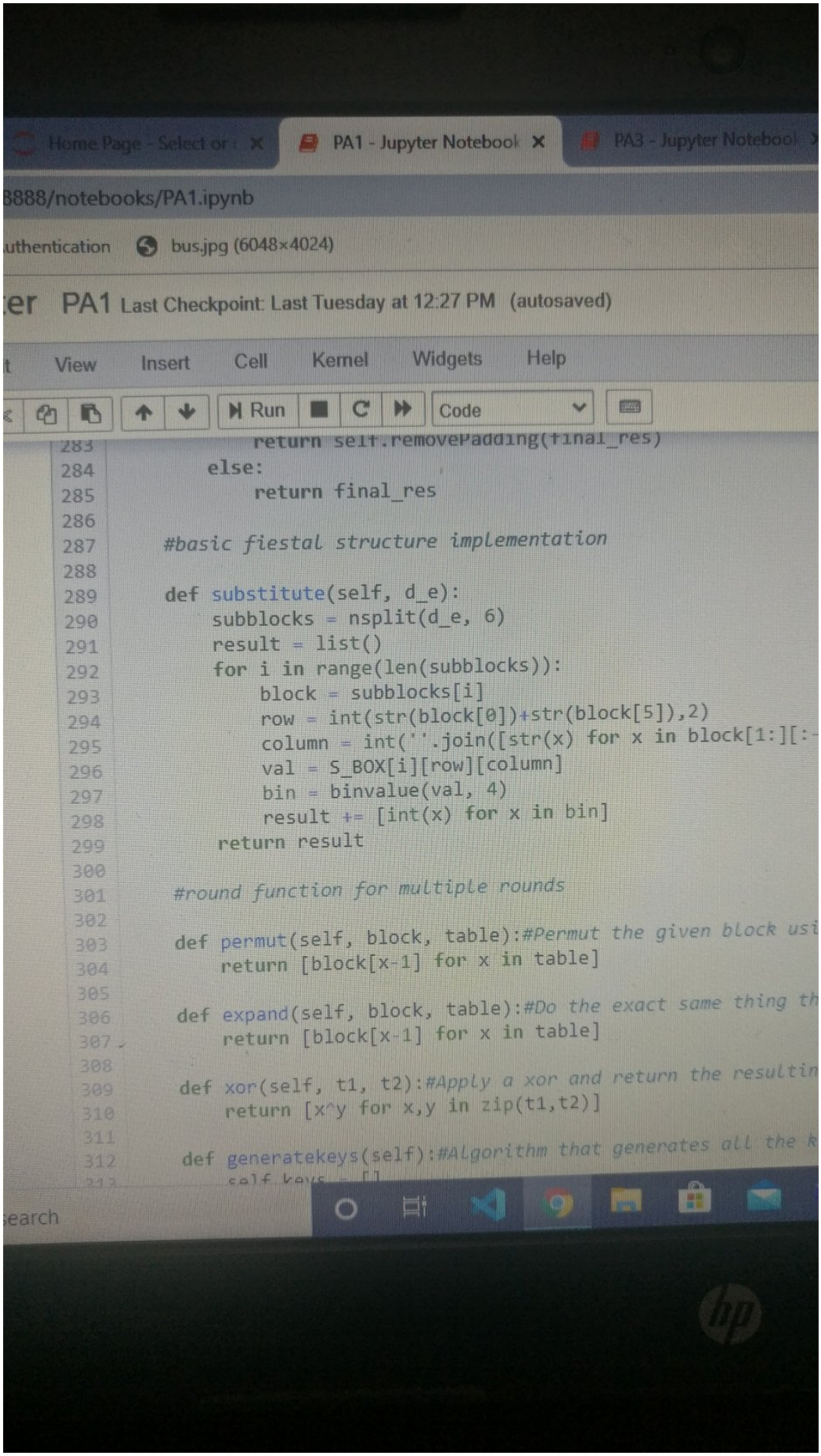

PA1 - Jupyter Notebook
host:8888/notebooks/PA1.ipynb
Authentication
bus.jpg (6048×4024)
pyter PA1 Last Checkpoint: Last Tuesday at 12:27 PM (autosaved)
Edit View Insert Cell Kernel Widgets Help
Run
Code
intext.insert(0.0,st)
def encry():
#encryption
wid=fu.get()
rs=roundbox.get("0.0",END)
ro = int(rs)
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
search

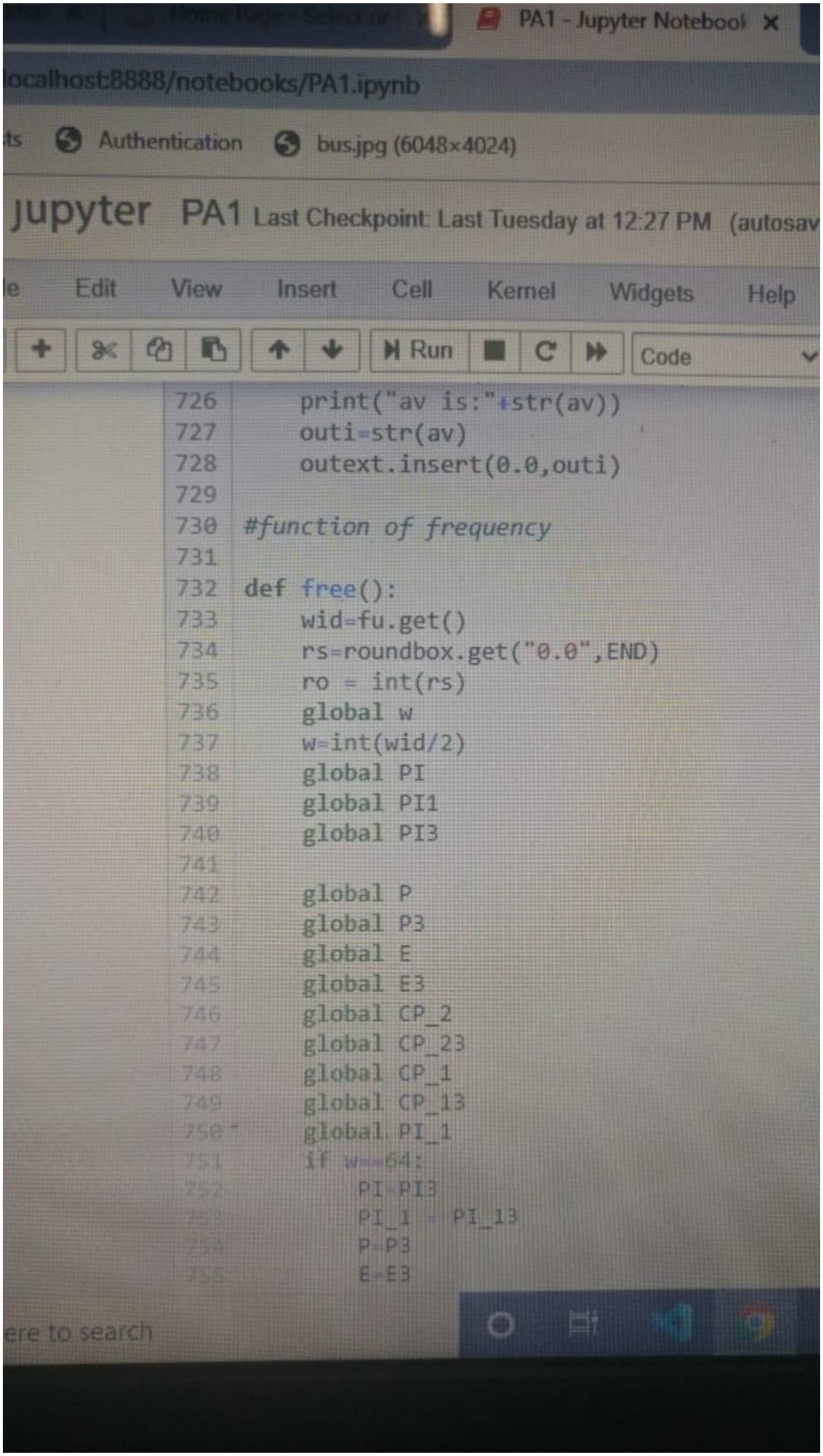

PA1 - Jupyter Notebook
ost:8888/notebooks/PA1.ipynb
Authentication
bus.jpg (6048×4024)
yter PA1 Last Checkpoint: Last Tuesday at 12:27 PM (autosaved)
Edit View Insert Cell Kernel Widgets Help
Run
Code
plt.show()
return av
#avalanche(roun,text,key,twow)
def string_to_freqlist(tex):
lis=np.zeros((256))
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
#function for frequency plots
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,
lis=np.arange(256)
for ill in range(0,ro+1):
plt.figure(ill+1)
search


32 bits
32 bits
28 bits
28 bits
Expansion/permutation
(E table)
48
F
XOR
48
Permutation/contraction
(Permuted choice 2)
Substitution/choice
(S-box)
32
Permutation
(P)
32
XOR


Half Block (32 bits)
Subkey (48 bits)
E
S1
S2
S3
S4
S5
S6
S7
S8
P


64-bit plaintext
DES
Initial permutation
Round 1
Round 2
Round 16
Final permutation
K_1
48-bit
K_2
48-bit
K_{16}
48-bit
Round-key generator
56-bit cipher key
64-bit ciphertext


Home Page - Select or
PA1 - Jupyter Notebook
PA3 - Jupyter Notebook
8888/notebooks/PA1.ipynb
uthentication
bus.jpg (6048×4024)
er PA1 Last Checkpoint: Last Tuesday at 12:27 PM (autosaved)
View
Insert
Cell
Kernel
Widgets
Help
Run
Code
283 return self.removePadding(final_res)
284 else:
285 return final_res
286
287 #basic fiestal structure implementation
288
289 def substitute(self, d_e):
290 subblocks = nsplit(d_e, 6)
291 result = list()
292 for i in range(len(subblocks)):
293 block = subblocks[i]
294 row = int(str(block[0])+str(block[5]),2)
295 column = int(''.join([str(x) for x in block[1:][:-
296 val = S_BOX[i][row][column]
297 bin = binvalue(val, 4)
298 result += [int(x) for x in bin]
299 return result
300
301 #round function for multiple rounds
302
303 def permut(self, block, table):#Permut the given block usi
304 return [block[x-1] for x in table]
305
306 def expand(self, block, table):#Do the exact same thing th
307 return [block[x-1] for x in table]
308
309 def xor(self, t1, t2):#Apply a xor and return the resultin
310 return [x^y for x,y in zip(t1,t2)]
311
312 def generatekeys(self):#Algorithm that generates all the k
search

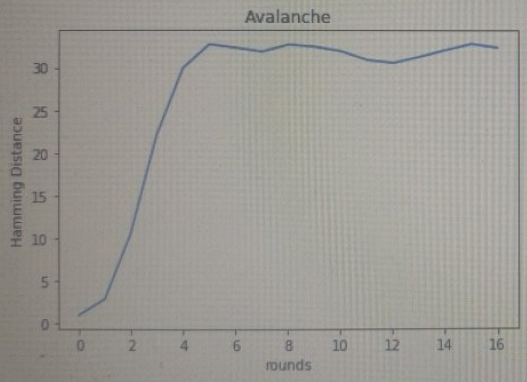

PA1 - Jupyter Notebook
localhost:8888/notebooks/PA1.ipynb
Authentication
bus.jpg (6048×4024)
jupyter PA1 Last Checkpoint: Last Tuesday at 12:27 PM (autosav
Edit View Insert Cell Kernel Widgets Help
Run
Code
print("av is:"+str(av))
outi=str(av)
outext.insert(0.0,outi)
#function of frequency
def free():
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
ere to search

File Edit View Insert Cell Kernel Widgets Help Trusted

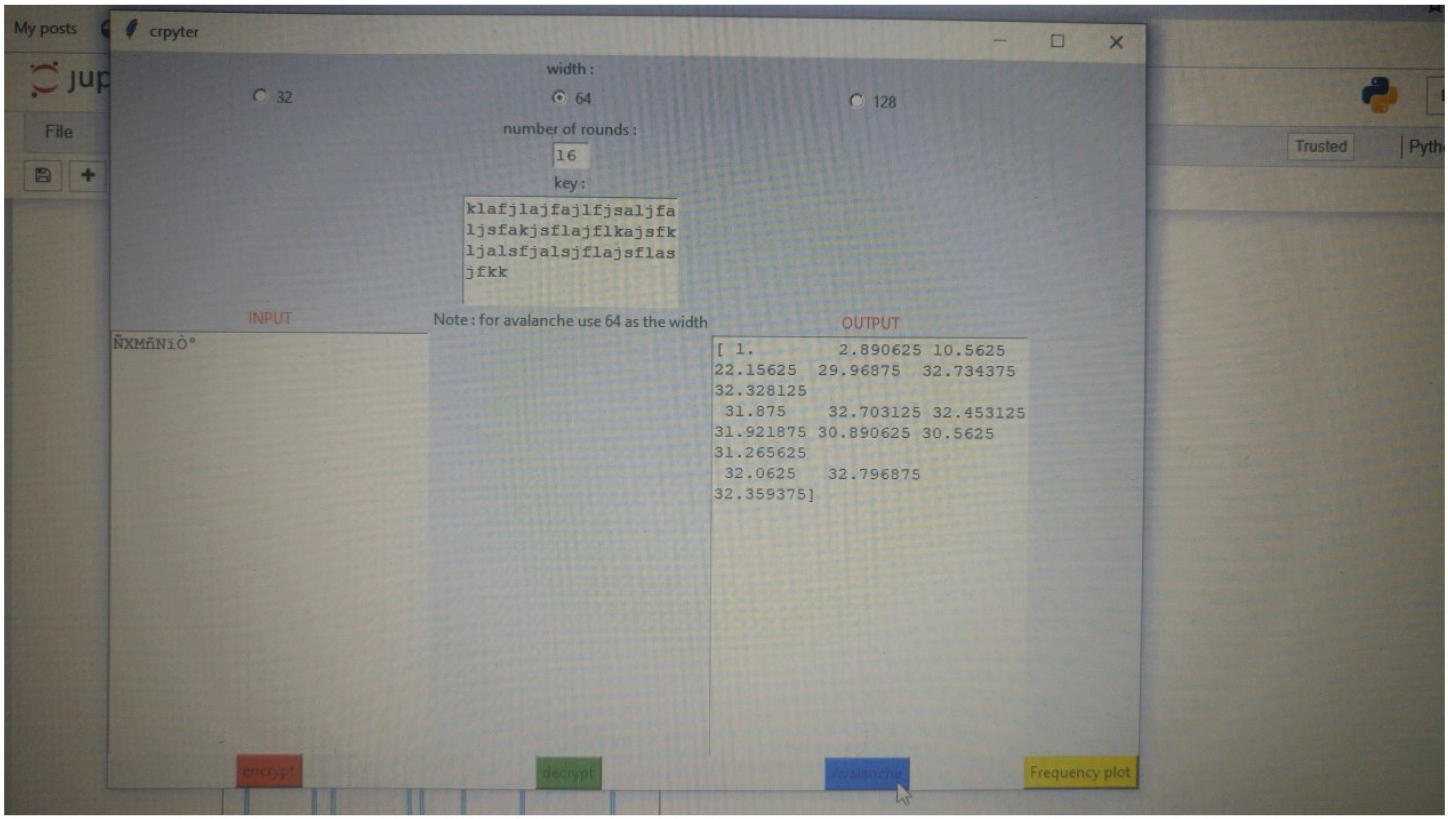
Run Code

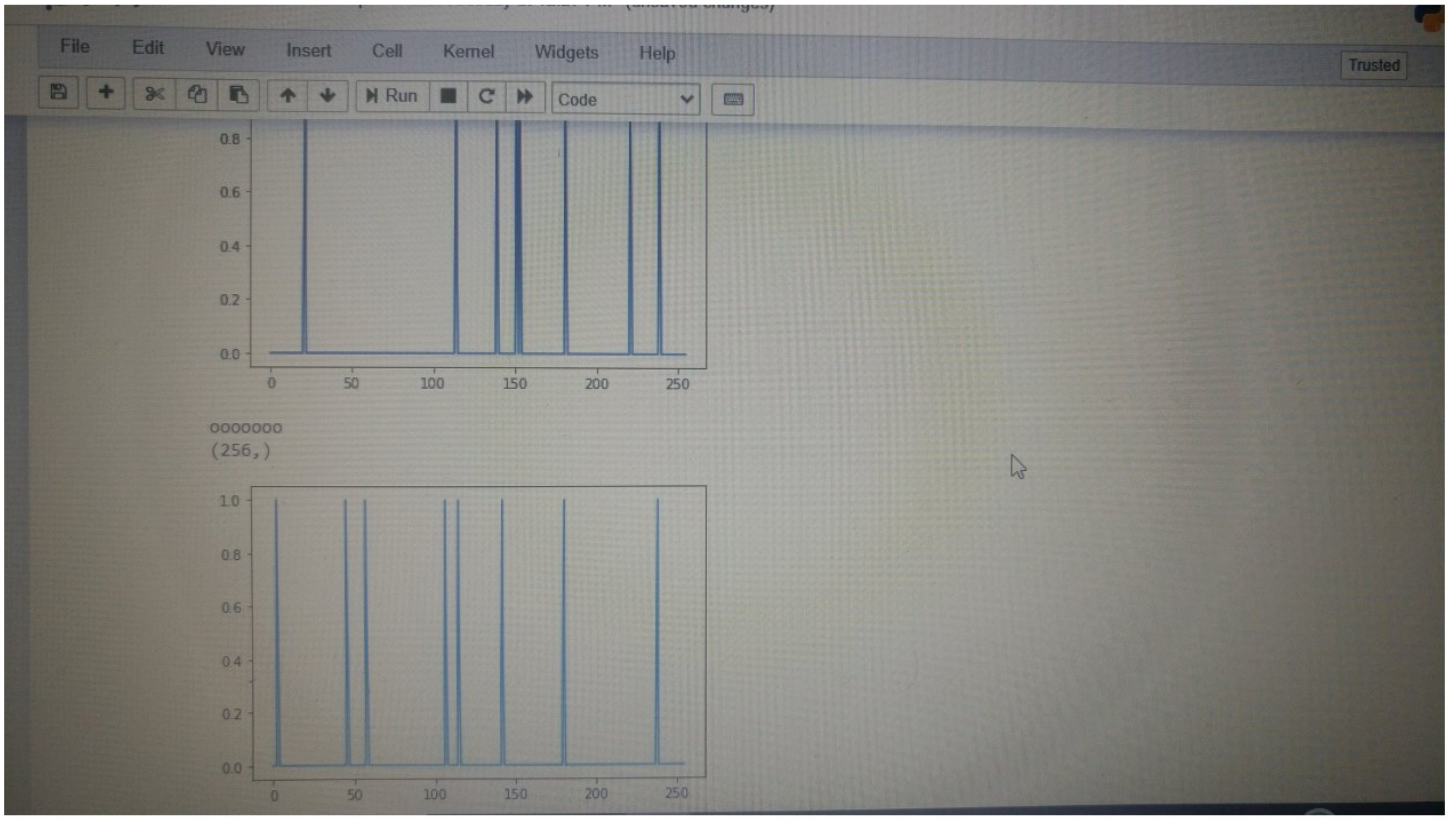
0 50 100 150 200 250

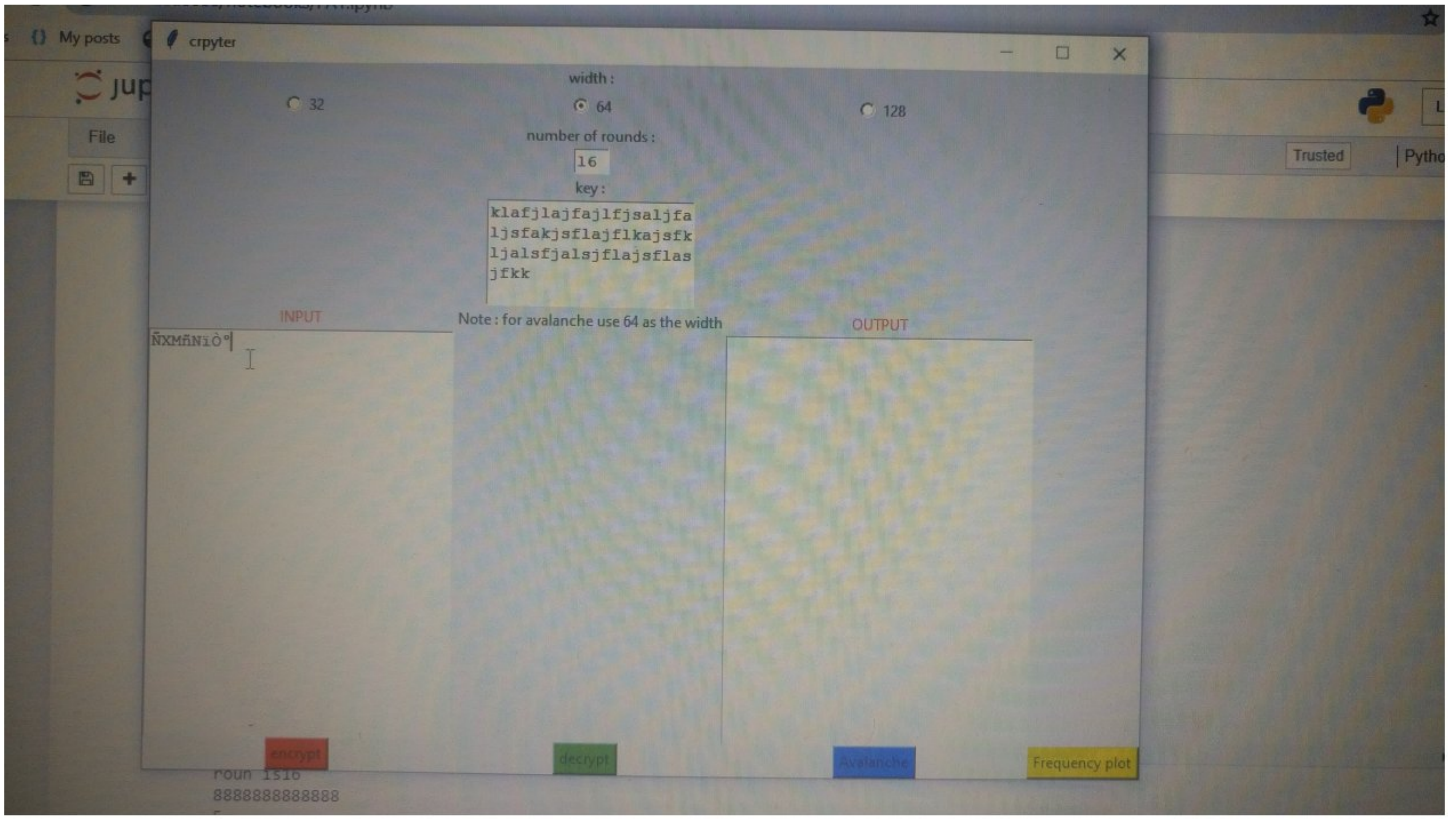
```
[58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24,
7, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15
roun is16
[ 1.        2.890625 10.5625   22.15625  29.96875  32.734375 32.328125
 31.875    32.703125 32.453125 31.921875 30.890625 30.5625   31.265625
 32.0625   32.796875 32.359375]
```

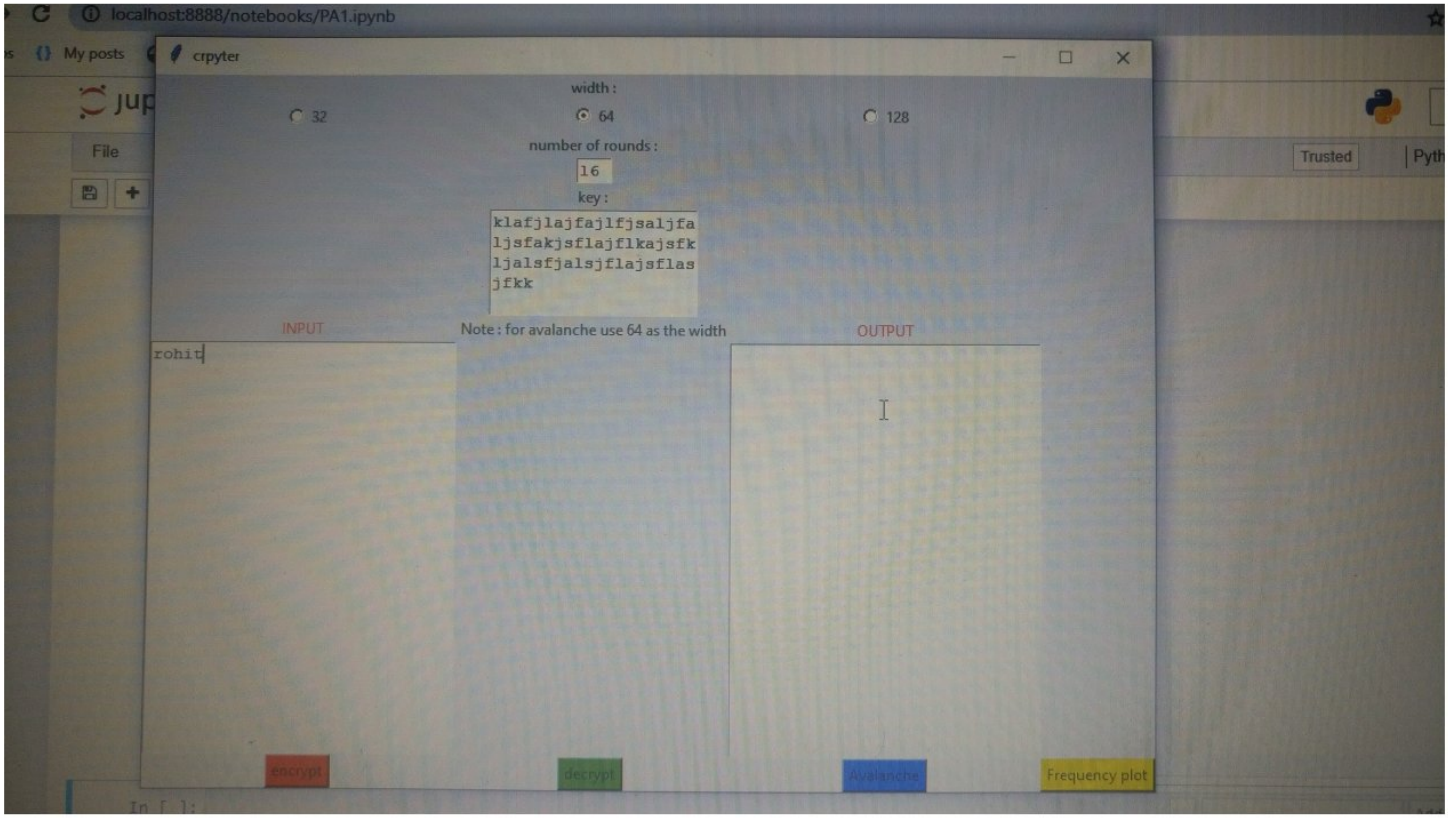


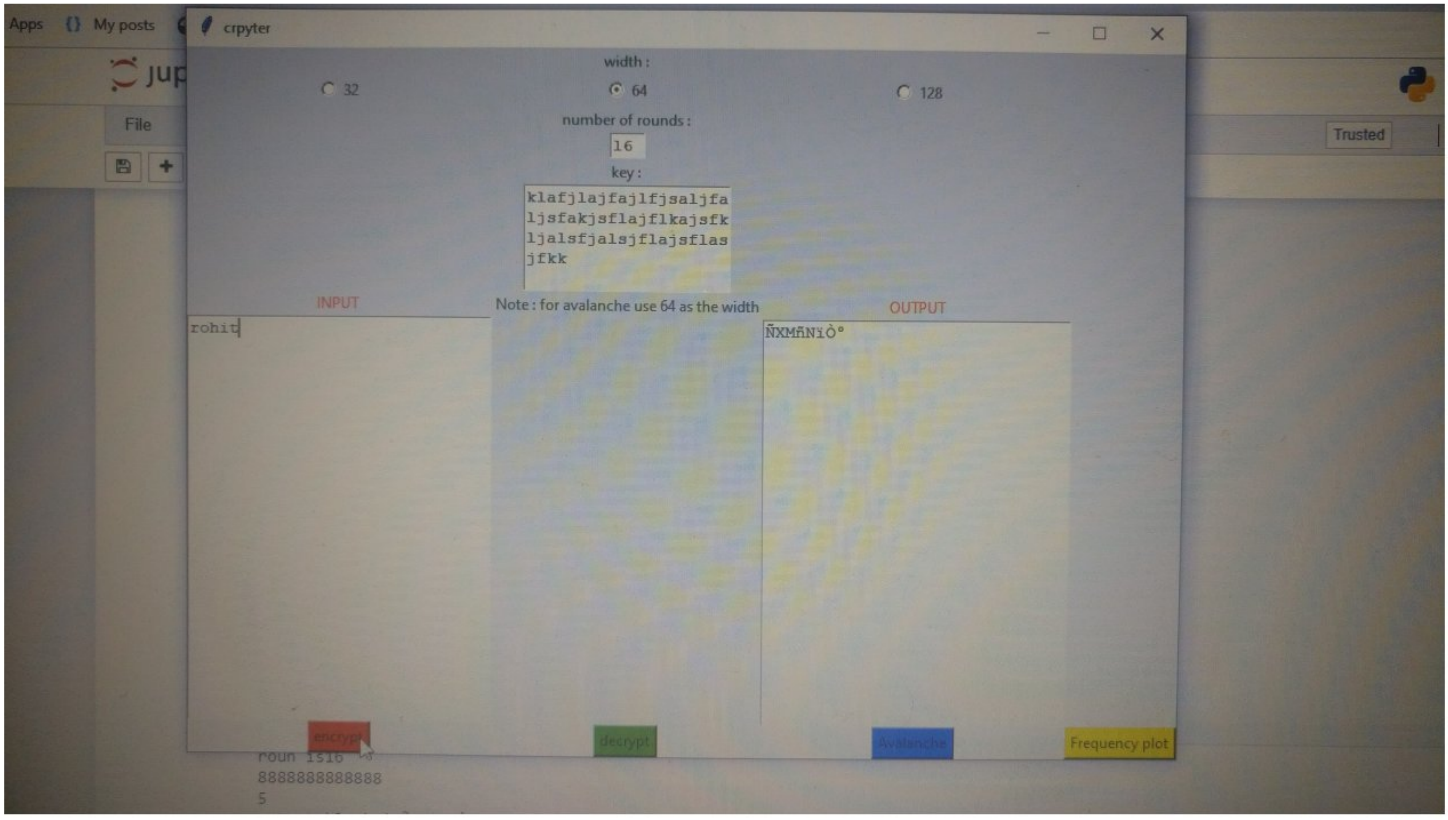

```
av is:[ 1.        2.890625 10.5625   22.15625  29.96875  32.734375 32.328125
 31.875    32.703125 32.453125 31.921875 30.890625 30.5625   31.265625
 32.0625   32.796875 32.359375]
```

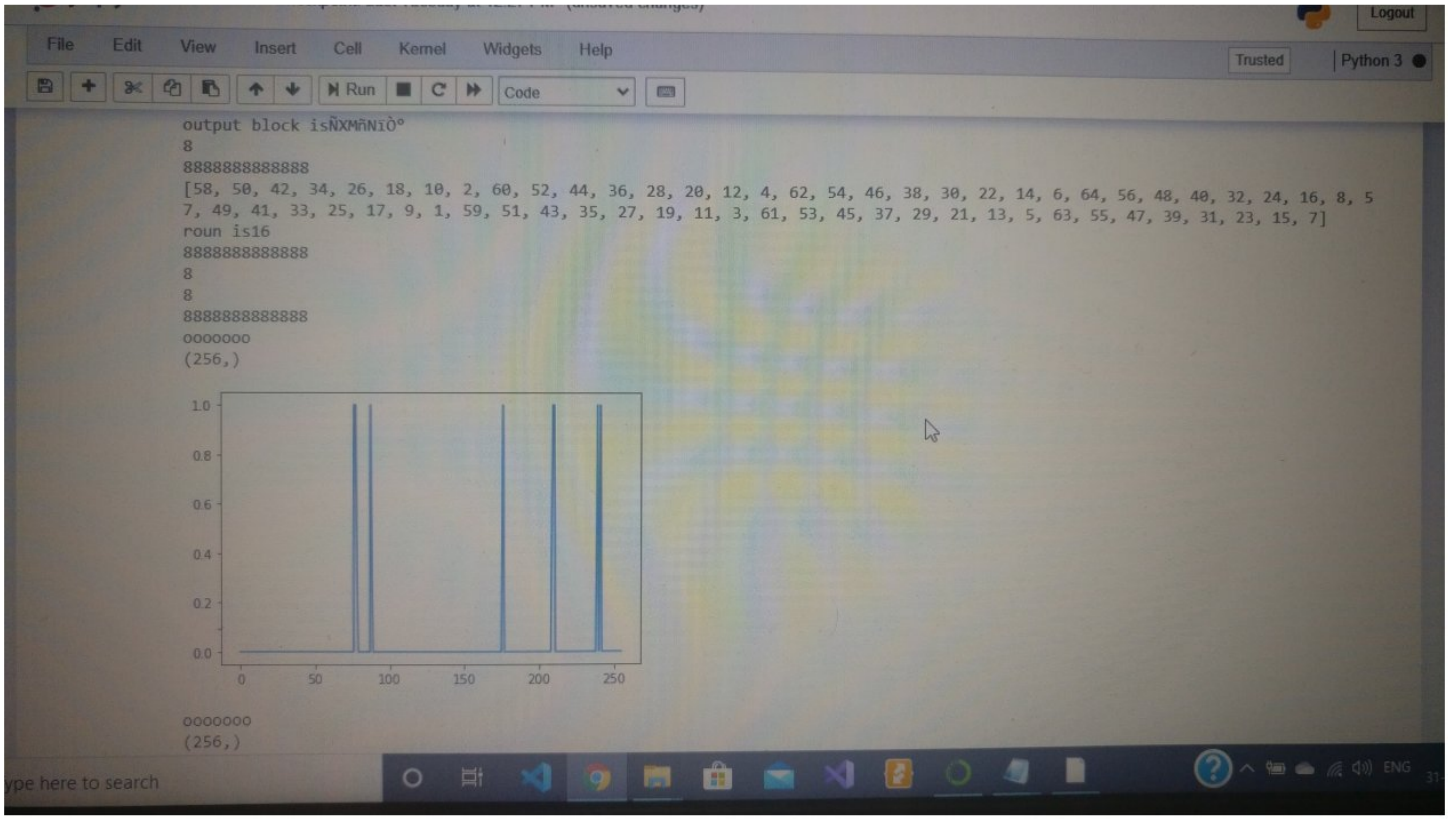

My posts
crpyter
File
Trusted
width :
32
64
128
number of rounds :
16
key :
klafjlajfajlfjsaljfa
ljsfakjsflajflkajsfk
ljalsfjalsjflajsflas
jfkk
INPUT
ÑXMñNiÒ°
Note : for avalanche use 64 as the width
OUTPUT
[1. 2.890625 10.5625
22.15625 29.96875 32.734375
32.328125
31.875 32.703125 32.453125
31.921875 30.890625 30.5625
31.265625
32.0625 32.796875
32.359375]
encrypt
decrypt
avalanche
Frequency plot

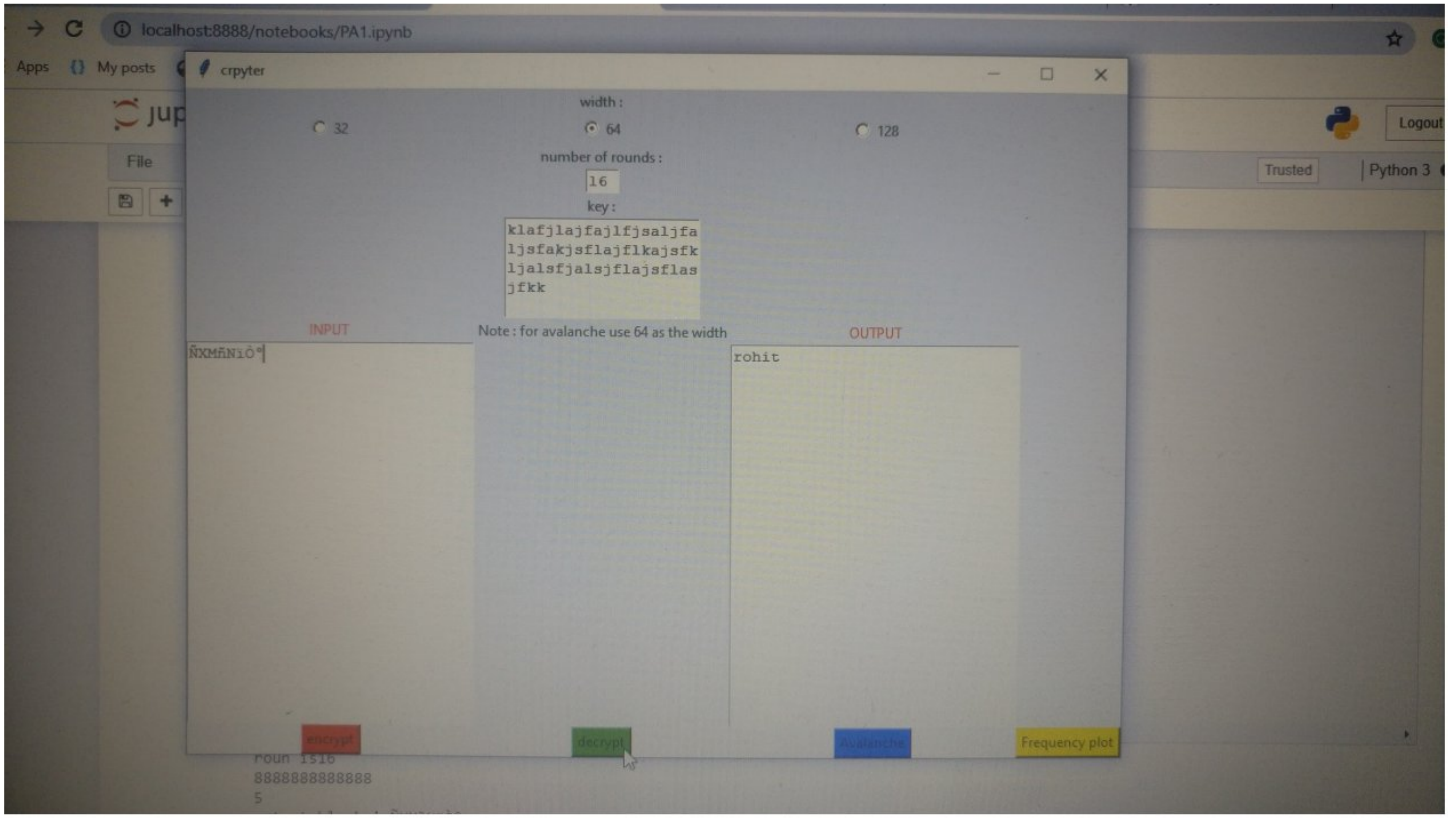

File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Run
Code
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
oooooooo
(256,)
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250


My posts
File
Trusted
crpyter
width :
32
64
128
number of rounds :
16
key :
klafjlajfajlfjsaljfa
ljsfakjsflajflkajsfk
ljalsfjalsjflajsflas
jfkk
Note : for avalanche use 64 as the width
INPUT
ÑXMñNìÒ°
OUTPUT
encrypt
decrypt
Avalanche
Frequency plot
roun 1516
8888888888888


localhost:8888/notebooks/PA1.ipynb
My posts
File
Trusted
crpyter
width :
32
64
128
number of rounds :
16
key :
klafjlajfajlfjsaljfa
ljsfakjsflajflkajsfk
ljalsfjalsjflajsflas
jfkk
Note : for avalanche use 64 as the width
INPUT
rohit
OUTPUT
encrypt
decrypt
Avalanche
Frequency plot


Apps
My posts
File
Trusted
crpyter
width :
32
64
128
number of rounds :
16
key :
klafjlajfajlfjsaljfa
ljsfakjsflajflkajsfk
ljalsfjalsjflajsflas
jfkk
INPUT
Note : for avalanche use 64 as the width
OUTPUT
rohit
ÑXMãNìÒ°
encrypt
decrypt
Avalanche
Frequency plot
8888888888888
5


File Edit View Insert Cell Kernel Widgets Help
Logout
Trusted
Python 3
Run
Code
output block isÑXMñNìÒ°
8
8888888888888
[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, 5
7, 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]
roun is16
8888888888888
8
8
8888888888888
oooooo
(256,)
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
oooooo
(256,)
ype here to search
ENG


localhost:8888/notebooks/PA1.ipynb
Apps
My posts
crpyter
width :
32
64
128
number of rounds :
16
key :
klafjlajfajlfjsaljfa
ljsfakjsflajflkajsfk
ljalsfjalsjflajsflas
jfkk
Note : for avalanche use 64 as the width
INPUT
ÑXMñNiÒ°
OUTPUT
rohit
encrypt
decrypt
avalanche
Frequency plot
Logout
Trusted
Python 3
File

PA1(Practical assignment 1)

Objective:

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

Theory:

The Data Encryption Standard (DES) is an outdated symmetric-key method of data encryption.
DES works by using the same key to encrypt and decrypt a message,
so both the sender and the receiver must know and use the same private key.
Once the go-to, symmetric-key algorithm for the encryption of electronic data,
DES has been superseded by the more secure Advanced Encryption Standard (AES) algorithm.
Originally designed by researchers at IBM in the early 1970s,
DES was adopted by the U.S. government as an official
Federal Information Processing Standard (FIPS) in 1977 for the
encryption of commercial and sensitive yet unclassified government computer data.
It was the first encryption algorithm approved by the U.S. government for public disclosure.
This ensured that DES was quickly adopted by industries such as
financial services, where the need for strong encryption is high.
The simplicity of DES also saw it used in a wide variety of
embedded systems, smart cards, SIM cards and network devices requiring
encryption like modems, set-top boxes and routers.

DES key length and brute-force attacks:

The Data Encryption Standard is a block cipher, meaning a
cryptographic key and algorithm are applied to a block of data simultaneously
rather than one bit at a time. To encrypt a plaintext message, DES groups it into 64-bit blocks.
Each block is enciphered using the secret key into a 64-bit ciphertext by means of permutation and substitution. The process involves
16 rounds and can run in four different modes,
encrypting blocks individually or making each
cipher block dependent on all the previous blocks.
Decryption is simply the inverse of encryption,
following the same steps but reversing the order in which the
keys are applied. For any cipher, the most basic method of attack is brute force,
which involves trying each key until you find the right one.
The length of the key determines the number of
possible keys -- and hence the feasibility -- of this type of attack.
DES uses a 64-bit key, but eight of those bits are used for parity checks,
effectively limiting the key to 56-bits. Hence,
it would take a maximum of 2^56, or 72,057,594,037,927,936, attempts to find the correct key.

Implementation details :

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure.

The block size is 64-bit. Though, key length is 64-bit,
DES has an effective key length of 56 bits, since 8 of the
64 bits of the key are not used by the encryption algorithm
Since DES is based on the Feistel Cipher, all that is required to specify DES is:

Round function
Key schedule
Any additional processing − Initial and final permutation

Round function:

The heart of this cipher is the DES function, f.
The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

Initial and Final Permutation:

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other.
They have no cryptography significance in DES.

Expansion Permutation Box:

Since right input is 32-bit and round key is a 48-bit,
we first need to expand right input to 48 bits.

Key Generation:

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key.

DES Analysis:

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.
Avalanche effect − A small change in plaintext results in the very great change in the ciphertext.

Completeness − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys.
These keys shall be avoided.
DES has proved to be a very well designed block cipher.
There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Code walkthrough :

python code :

```
#permutation boxes

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
,31,126,81,115,105,17,88,68,74,103,102,59,79,57,128,28,86,93,66,83,104,117,21,60,19
,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

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 =
[37,43,1,107,13,82,21,121,11,10,77,47,103,85,79,116,4,65,27,22,101,71,25,34,57,55,1
26,84,108,106,81,124,98,36,49,110,19,35,42,92,20,41,90,94,89,105,125,7,5,53,3,100,9
7,102,74,99,62,111,69,93,122,118,87,46,29,113,60,95,78,68,75,63,54,52,33,86,70,123,
45,117,73,67,66,51,127,76,50,58,91,39,23,30,31,28,59,26,17,119,61,115,18,38,14,44,1
5,12,9,2,6,109,114,83]
CP_1=CP_13


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 =
[78,1,94,50,10,23,66,64,14,103,4,32,34,28,22,39,46,79,80,25,96,111,105,33,6,58,99,1
02,98,38,70,108,104,112,40,109,84,100,83,74,92,77,90,72,73,68,86,110,65,91,56,97,75
,9,87,76,57,61,81,13,82,85,60,62,69,89,48,19,88,106,49,29,52,63,42,55,54,41,51,43,9
5,93,71,35,67,36,101,59,47,107,44,37,26,3,27,30,20,17,8,21,18,24,15,53,11,31,12,16,
2,7,5,45]
CP_2=CP_23

#expansion boxes

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

#s-boxes

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]],

]

#p-boxes

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 ,4
5 ,47 ,17 ,24 ,54 ,14 ,49 ,28 ,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

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=[]

#left shift for sub key generation

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]



def string_to_bit_array(text):
    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):
    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):
    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):
    return [s[k:k+n] for k in range(0, len(s), n)]
```

```
ENCRYPT=1
DECRYPT=0

class des():
    global roun
    def __init__(self):
        self.password = None
        self.text = None
        self.keys = list()

    #encryption

    def run(self, key, text, action=ENCRYPT, padding=False):
        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()
        text_blocks = nsplit(self.text, kinw)


        result = list()

        for block in text_blocks:
            acc.clear()
            for i in range(0,roun+1):
                acc.append([])
            #print("input block is "+str(block))
            #print("printing blocks of text :"+str(block)+"\n")
            block = string_to_bit_array(block)#Convert the block in bit array
            acc[0]=block
            block = self.permut(block,PI)
            g, d = nsplit(block, w)
            tmp = None
            #print("running for roun value"+str(roun))
            for i in range(0,roun):
              #  print("the round number is "+str(i))
                d_e = self.expand(d, E)
                if action == ENCRYPT:
                    tmp = self.xor(self.keys[i], d_e)
                else:
                    tmp = self.xor(self.keys[roun-1-i], d_e)
                tmp = self.substitute(tmp)
                tmp = self.permut(tmp, P)
                tmp = self.xor(g, tmp)
                g = d
```

```
                d = tmp
                acc[i+1]=self.permut(d+g, PI_1)
               # print(str(i)+" acci"+str(acc[i]))

            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

    #basic fiestal structure implementation

    def substitute(self, d_e):
        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

    #round function for multiple rounds

    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)
        #print("key:"+str(len(key))+"  "+str(key)+"\n")
        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)

    #decrption

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


d = des()




import numpy as np

#graph for avalanche

import matplotlib.pyplot as plt
def diff(row1,row0):
    sum=0
    for f in range(0,len(row1)):
        sum=sum+abs(row1[f]-row0[f])
    return sum

#function for avalanche effect

def avalanche(roune,text,key,twow):
    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("rewwwfffffffffffffffffff")

        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()

    #avalanche plots with hamming distance

    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):
    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

#function for frequency plots

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("ooooooo")
        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():
    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 block is"+str(outi))
    print(len(outi))
    print("8888888888888")

    outext.insert(0.0,outi)



def decry():
    #decryption 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)

#avalanche function

def ava():
    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)

#function of frequency

def free():
    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)

#basic user interface is implemented by tkinter

l1=Label(m, text="INPUT",fg='red')
l2=Label(m,text="OUTPUT",fg='red')
l3=Label(m,text="width : ")
l4=Label(m,text="number of rounds : ")
l5=Label(m,text="key : ")
```

```
l6=Label(m,text="Note : for avalanche use 64 as the width")

l1.grid(row=6,column=0)
l2.grid(row=6,column=2)
l3.grid(row=0,column=1)
l4.grid(row=2,column=1)
l5.grid(row=4,column=1)
l6.grid(row=6,column=1)

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=7,column=0)
roundbox.grid(row=3,column=1)
keybox.grid(row=5,column=1)
outext=Text(m,width=30,height=20,wrap=WORD)
outext.grid(row=7,column=2)
act1=Button(m,text="encrypt",command=encry,bg='red')
act2=Button(m,text="decrypt",command=decry,bg='green')
act3=Button(m,text="Avalanche",command=ava,bg='blue')
act4=Button(m,text="Frequency plot",command=free,bg='yellow')
act1.grid(row=8,column=0)
act2.grid(row=8,column=1)
act3.grid(row=8,column=2)
act4.grid(row=8,column=3)
m.mainloop()
```

Conclusion:

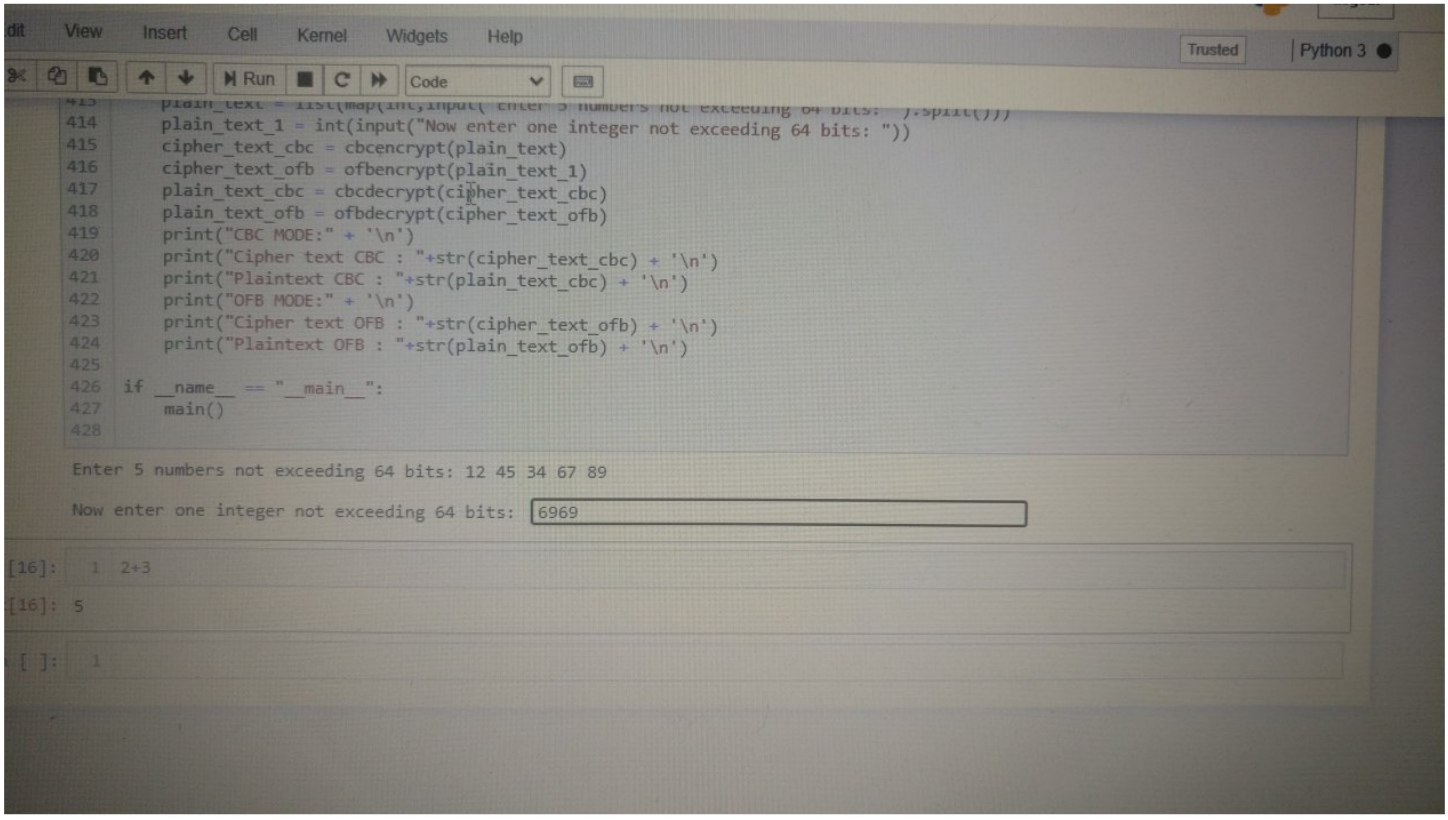
Advantages:

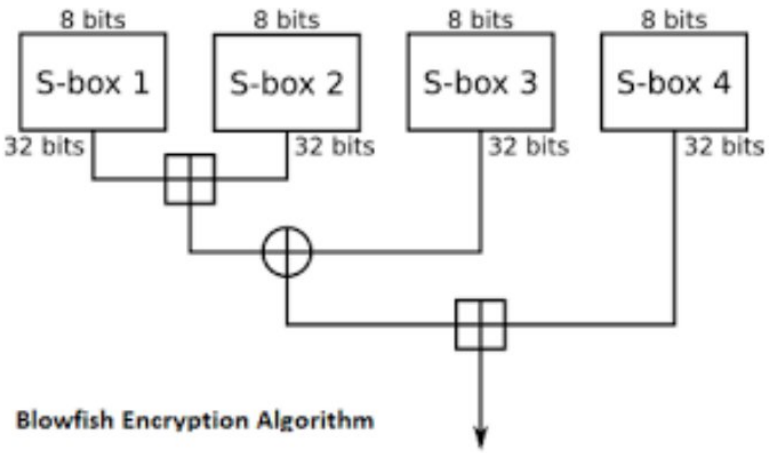
DES is broken; however, 3DES is currently considered a secure cipher.
DES does have the desirable properties of confusion and diffusion:
each bit of ciphertext is based upon multiple bits of the key and changing a single bit of plaintext changes, on average, half of the bits of ciphertext.
Due to its Feistel structure and uncomplicated logic, DES is relatively easy to implement.
However, it uses eight distinct S-Boxes, which increases its footprint (AES uses a single S-Box).

Disadvantages:

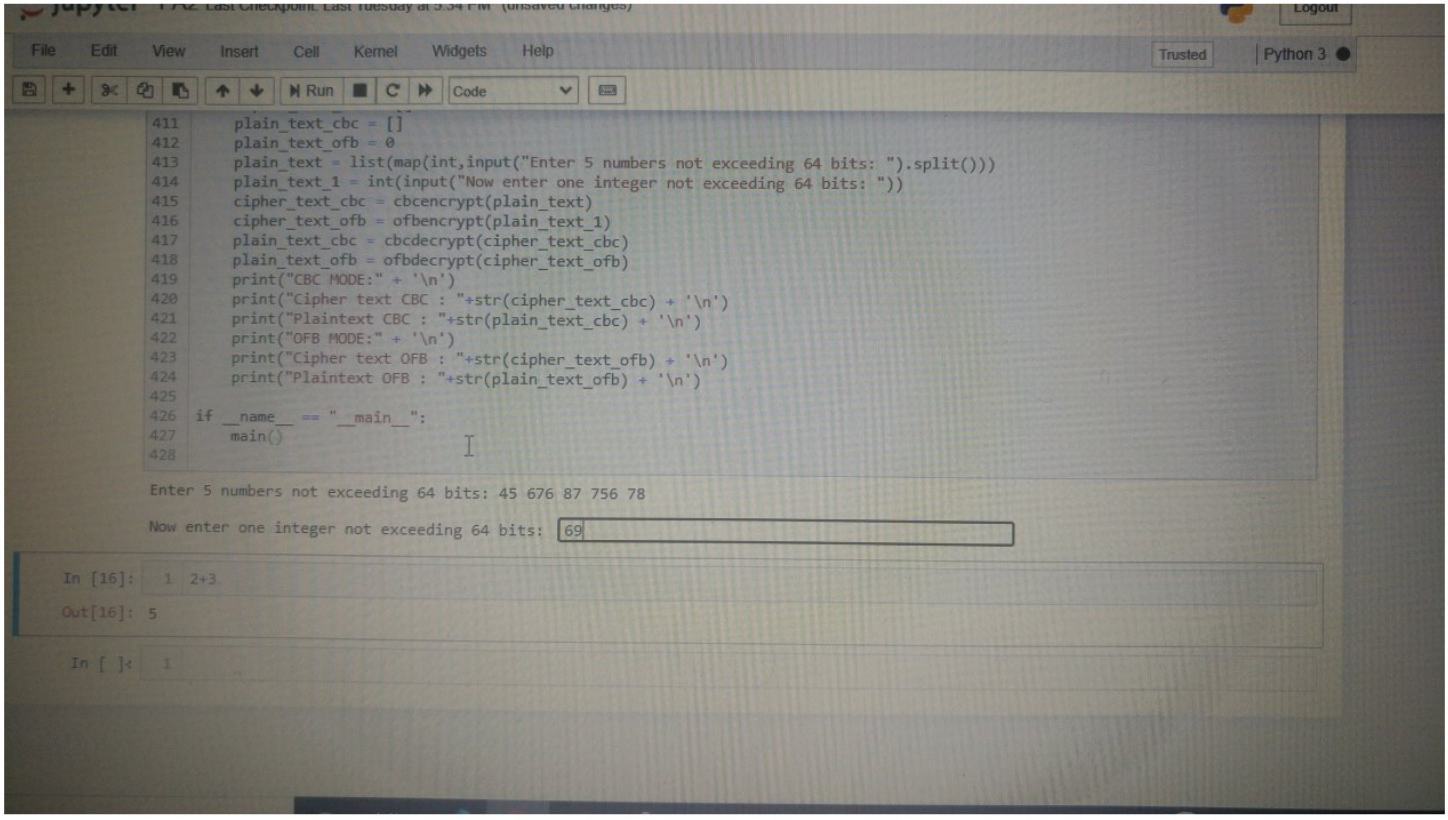
The main disadvantage to DES is that it is broken using brute-force search.
However, using 3DES mitigates this issue at the cost of increasing execution time.
DES is also vulnerable to attacks using linear cryptanalysis. However, it takes 247 known plaintexts to break DES in this manner.

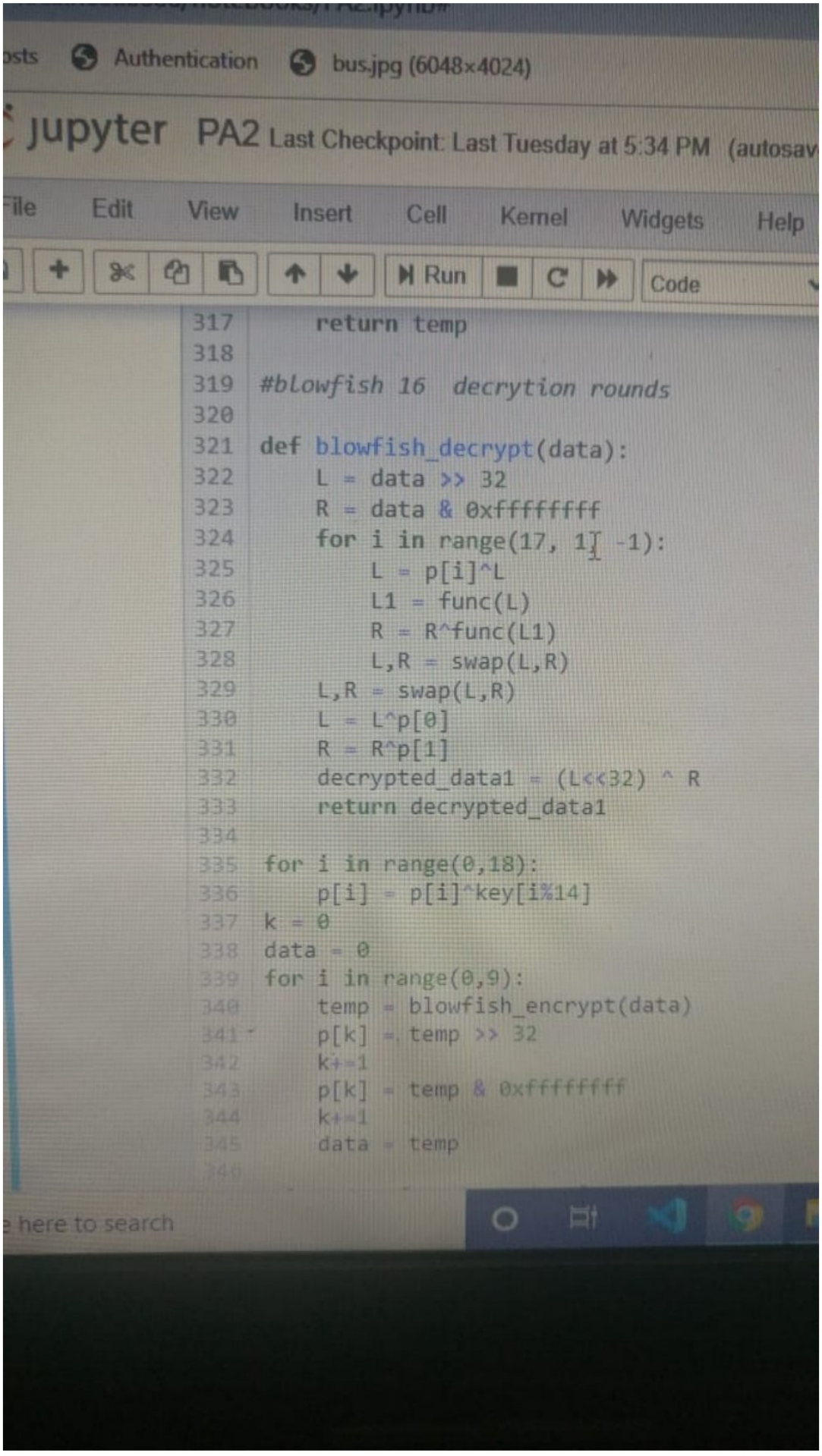
Screenshots and images :

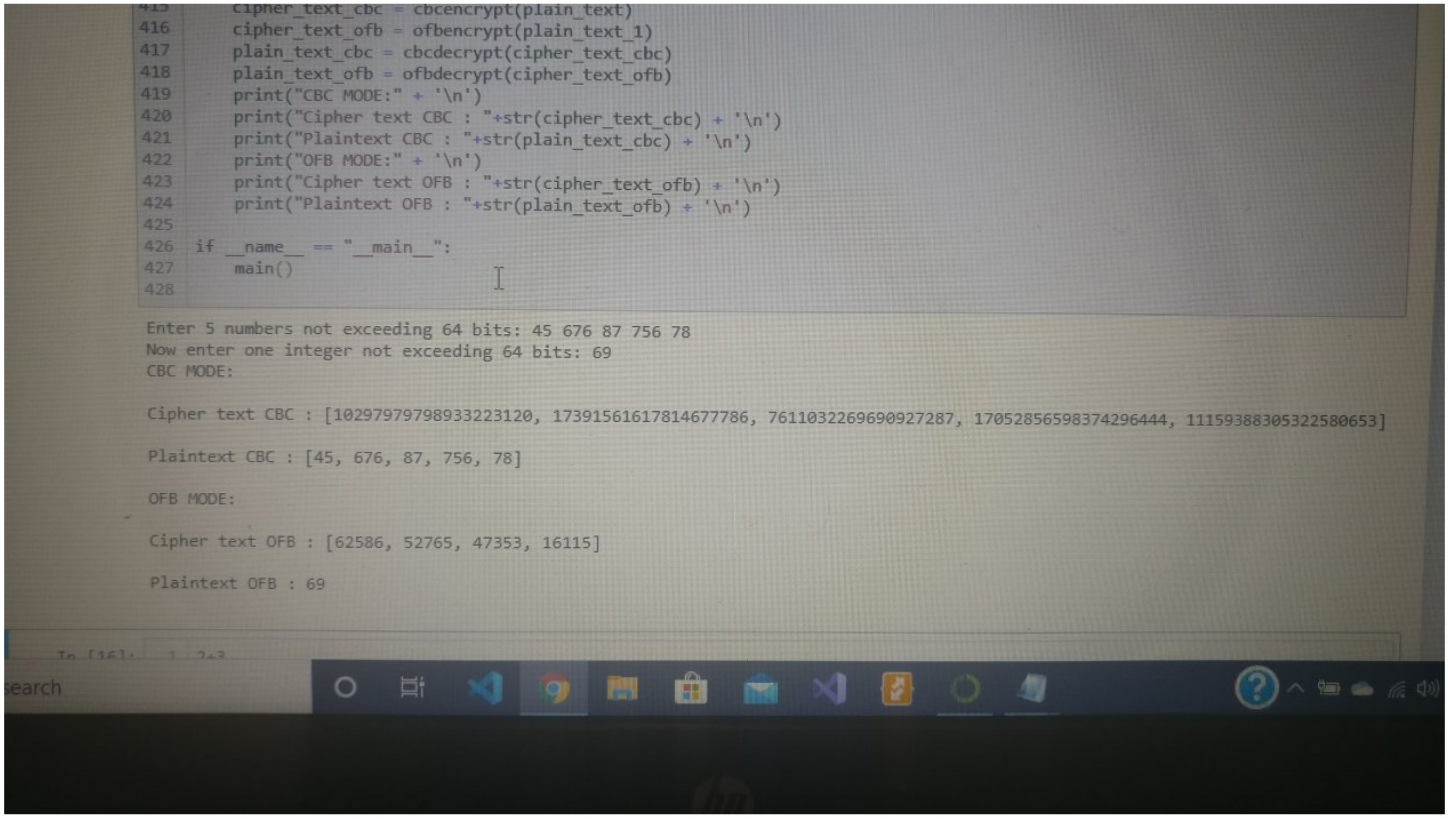

dit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
plain_text_1 = int(input("Now enter one integer not exceeding 64 bits: "))
cipher_text_cbc = cbcencrypt(plain_text)
cipher_text_ofb = ofbencrypt(plain_text_1)
plain_text_cbc = cbcdecrypt(cipher_text_cbc)
plain_text_ofb = ofbdecrypt(cipher_text_ofb)
print("CBC MODE:" + '\n')
print("Cipher text CBC : "+str(cipher_text_cbc) + '\n')
print("Plaintext CBC : "+str(plain_text_cbc) + '\n')
print("OFB MODE:" + '\n')
print("Cipher text OFB : "+str(cipher_text_ofb) + '\n')
print("Plaintext OFB : "+str(plain_text_ofb) + '\n')
if __name__ == "__main__":
main()
Enter 5 numbers not exceeding 64 bits: 12 45 34 67 89
Now enter one integer not exceeding 64 bits: 6969
[16]: 1 2+3
[16]: 5
[ ]: 1



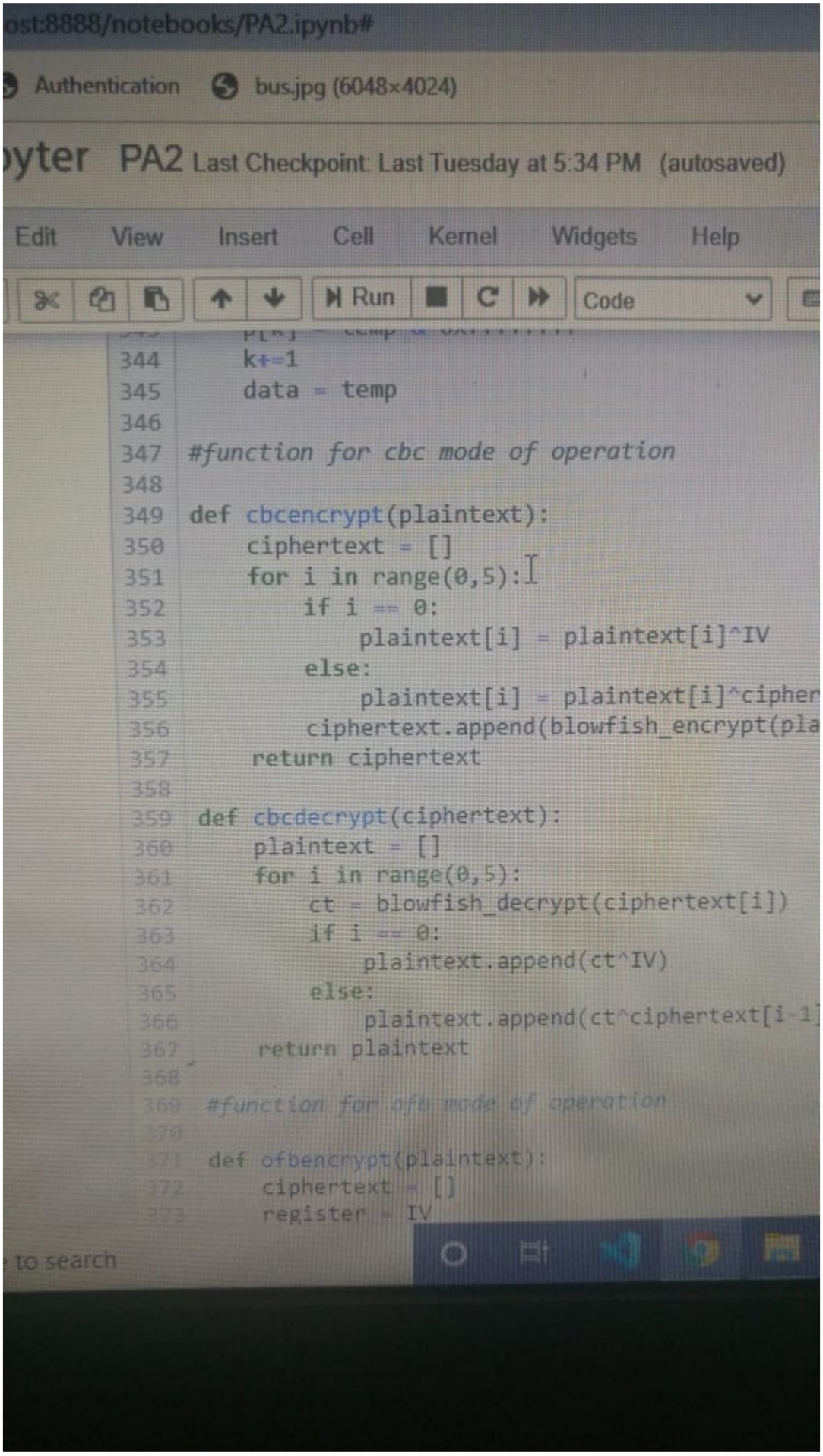
Blowfish Encryption Algorithm

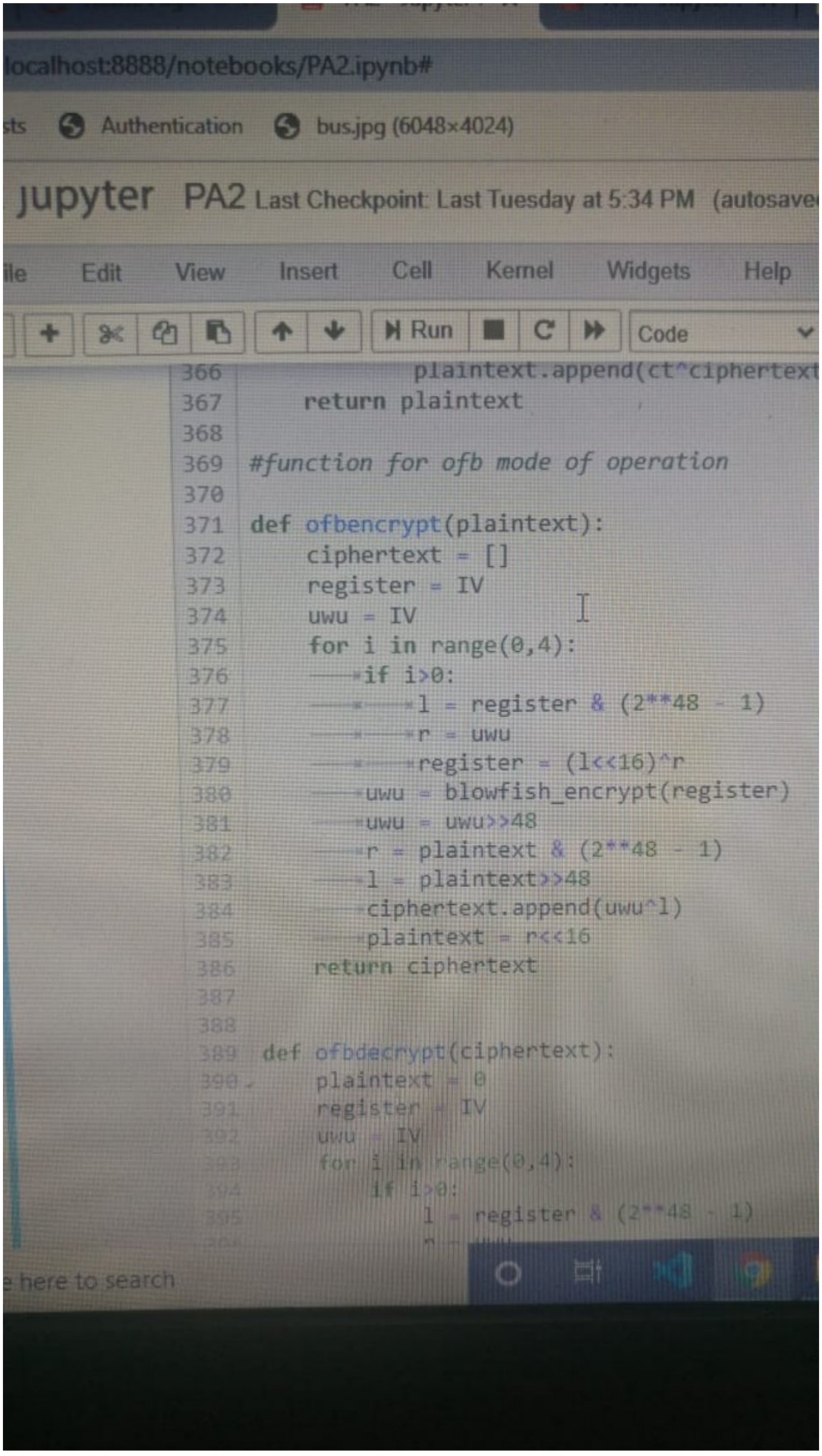

File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
plain_text_cbc = []
plain_text_ofb = 0
plain_text = list(map(int,input("Enter 5 numbers not exceeding 64 bits: ").split()))
plain_text_1 = int(input("Now enter one integer not exceeding 64 bits: "))
cipher_text_cbc = cbcencrypt(plain_text)
cipher_text_ofb = ofbencrypt(plain_text_1)
plain_text_cbc = cbcdecrypt(cipher_text_cbc)
plain_text_ofb = ofbdecrypt(cipher_text_ofb)
print("CBC MODE:" + '\n')
print("Cipher text CBC : "+str(cipher_text_cbc) + '\n')
print("Plaintext CBC : "+str(plain_text_cbc) + '\n')
print("OFB MODE:" + '\n')
print("Cipher text OFB : "+str(cipher_text_ofb) + '\n')
print("Plaintext OFB : "+str(plain_text_ofb) + '\n')
if __name__ == "__main__":
main()
Enter 5 numbers not exceeding 64 bits: 45 676 87 756 78
Now enter one integer not exceeding 64 bits: 69
In [16]: 2+3
Out[16]: 5
In [ ]:


Authentication
bus.jpg (6048×4024)
jupyter PA2 Last Checkpoint: Last Tuesday at 5:34 PM (autosav
File Edit View Insert Cell Kernel Widgets Help
Run
Code
317 return temp
318
319 #blowfish 16 decrytion rounds
320
321 def blowfish_decrypt(data):
322 L = data >> 32
323 R = data & 0xffffffff
324 for i in range(17, 1, -1):
325 L = p[i]^L
326 L1 = func(L)
327 R = R^func(L1)
328 L,R = swap(L,R)
329 L,R = swap(L,R)
330 L = L^p[0]
331 R = R^p[1]
332 decrypted_data1 = (L<<32) ^ R
333 return decrypted_data1
334
335 for i in range(0,18):
336 p[i] = p[i]^key[i%14]
337 k = 0
338 data = 0
339 for i in range(0,9):
340 temp = blowfish_encrypt(data)
341 p[k] = temp >> 32
342 k+=1
343 p[k] = temp & 0xffffffff
344 k+=1
345 data = temp
here to search



```
    cipher_text_cbc = cbcencrypt(plain_text)
    cipher_text_ofb = ofbencrypt(plain_text_1)
    plain_text_cbc = cbcdecrypt(cipher_text_cbc)
    plain_text_ofb = ofbdecrypt(cipher_text_ofb)
    print("CBC MODE:" + '\n')
    print("Cipher text CBC : "+str(cipher_text_cbc) + '\n')
    print("Plaintext CBC : "+str(plain_text_cbc) + '\n')
    print("OFB MODE:" + '\n')
    print("Cipher text OFB : "+str(cipher_text_ofb) + '\n')
    print("Plaintext OFB : "+str(plain_text_ofb) + '\n')

if __name__ == "__main__":
    main()

Enter 5 numbers not exceeding 64 bits: 45 676 87 756 78
Now enter one integer not exceeding 64 bits: 69
CBC MODE:

Cipher text CBC : [10297979798933223120, 17391561617814677786, 7611032269690927287, 17052856598374296444, 11159388305322580653]

Plaintext CBC : [45, 676, 87, 756, 78]

OFB MODE:

Cipher text OFB : [62586, 52765, 47353, 16115]

Plaintext OFB : 69
```


ost:8888/notebooks/PA2.ipynb#
Authentication
bus.jpg (6048×4024)
yter PA2 Last Checkpoint: Last Tuesday at 5:34 PM (autosaved)
Edit View Insert Cell Kernel Widgets Help
Run
Code
k+=1
data = temp
#function for cbc mode of operation
def cbcencrypt(plaintext):
ciphertext = []
for i in range(0,5):
if i == 0:
plaintext[i] = plaintext[i]^IV
else:
plaintext[i] = plaintext[i]^cipher
ciphertext.append(blowfish_encrypt(pla
return ciphertext
def cbcdecrypt(ciphertext):
plaintext = []
for i in range(0,5):
ct = blowfish_decrypt(ciphertext[i])
if i == 0:
plaintext.append(ct^IV)
else:
plaintext.append(ct^ciphertext[i-1]
return plaintext
#function for ofb mode of operation
def ofbencrypt(plaintext):
ciphertext = []
register = IV
to search


localhost:8888/notebooks/PA2.ipynb#
Authentication
bus.jpg (6048×4024)
jupyter PA2 Last Checkpoint: Last Tuesday at 5:34 PM (autosave
File Edit View Insert Cell Kernel Widgets Help
Run
Code
366 plaintext.append(ct^ciphertext
367 return plaintext
368
369 #function for ofb mode of operation
370
371 def ofbencrypt(plaintext):
372 ciphertext = []
373 register = IV
374 uwu = IV
375 for i in range(0,4):
376 if i>0:
377 l = register & (2**48 - 1)
378 r = uwu
379 register = (l<<16)^r
380 uwu = blowfish_encrypt(register)
381 uwu = uwu>>48
382 r = plaintext & (2**48 - 1)
383 l = plaintext>>48
384 ciphertext.append(uwu^l)
385 plaintext = r<<16
386 return ciphertext
387
388
389 def ofbdecrypt(ciphertext):
390 plaintext = 0
391 register = IV
392 uwu = IV
393 for i in range(0,4):
394 if i>0:
395 l = register & (2**48 - 1)
here to search

PA2(Practical Assignment 2)

Objective:

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

Theory :

Blowfish is a symmetric encryption algorithm developed by Bruce Schneier to replace Data Encryption Standard (DES).
At the time of its development, most encryption algorithms were protected by patents, government secrecy, or company intellectual property.
Schneier placed Blowfish in the public domain making it freely available for anyone to use.
Blowfish is a 16-round Feistel cipher. It's block size is 64-bit and key sizes range from 32 to 448 bits.

Implementation Details:

Blowfish is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date.
It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.

blockSize: 64-bits
keySize: 32-bits to 448-bits variable size
number of subkeys: 18 [P-array]
number of rounds: 16
number of subsitution boxes: 4 [each having 512 entries of 32-bits each]

Blowfish Encryption Algorithm:

The entire encryption process can be elaborated as:

Step1: Generation of subkeys:

18 subkeys{P[0]…P[17]} are needed in both encryption aswell as decryption process and the same subkeys are used for both the processes.
These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
It is initialised with the digits of pi(?).
The hexadecimal representation of each of the subkeys is given by:
P[0] = "243f6a88"
..
P[17] = "8979fb1b"

Step2: initialise Substitution Boxes:

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.
It is initialised with the digits of pi(?) after initialising the P-array.

Step3: Encryption:

The encryption function consists of two parts:
Rounds: The encryption consists of 16 rounds with each round(Ri) taking inputs the

plainText(P.T.) from previous round and corresponding subkey(Pi).
Post-processing: The output after the 16 rounds is processed further.

Blowfish Decryption Algorithm:

18 subkeys{P[0]…P[17]} are needed in both encryption aswell as decryption process and the same subkeys are used for both the processes.
These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
It is initialised with the digits of pi(?).
The hexadecimal representation of each of the subkeys is given by:
P[0] = "243f6a88"
..
P[17] = "8979fb1b"

Step2: initialise Substitution Boxes:

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.
It is initialised with the digits of pi(?) after initialising the P-array.

Code Walkthrough :

Pyhton code :

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

#s-box initialization

s = [
      [
          0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
          0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
          0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
          0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
          0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
          0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
          0xC5D1B023, 0x286085F0, 0xCA417918, 0xB8DB38EF,
          0x8E79DCB0, 0x603A180E, 0x6C9E0E8B, 0xB01E8A3E,
          0xD71577C1, 0xBD314B27, 0x78AF2FDA, 0x55605C60,
          0xE65525F3, 0xAA55AB94, 0x57489862, 0x63E81440,
          0x55CA396A, 0x2AAB10B6, 0xB4CC5C34, 0x1141E8CE,
          0xA15486AF, 0x7C72E993, 0xB3EE1411, 0x636FBC2A,
          0x2BA9C55D, 0x741831F6, 0xCE5C3E16, 0x9B87931E,
          0xAFD6BA33, 0x6C24CF5C, 0x7A325381, 0x28958677,
          0x3B8F4898, 0x6B4BB9AF, 0xC4BFE81B, 0x66282193,
          0x61D809CC, 0xFB21A991, 0x487CAC60, 0x5DEC8032,
          0xEF845D5D, 0xE98575B1, 0xDC262302, 0xEB651B88,
```

```
    0x23893E81, 0xD396ACC5, 0x0F6D6FF3, 0x83F44239,
    0x2E0B4482, 0xA4842004, 0x69C8F04A, 0x9E1F9B5E,
    0x21C66842, 0xF6E96C9A, 0x670C9C61, 0xABD388F0,
    0x6A51A0D2, 0xD8542F68, 0x960FA728, 0xAB5133A3,
    0x6EEF0B6C, 0x137A3BE4, 0xBA3BF050, 0x7EFB2A98,
    0xA1F1651D, 0x39AF0176, 0x66CA593E, 0x82430E88,
    0x8CEE8619, 0x456F9FB4, 0x7D84A5C3, 0x3B8B5EBE,
    0xE06F75D8, 0x85C12073, 0x401A449F, 0x56C16AA6,
    0x4ED3AA62, 0x363F7706, 0x1BFEDF72, 0x429B023D,
    0x37D0D724, 0xD00A1248, 0xDB0FEAD3, 0x49F1C09B,
    0x075372C9, 0x80991B7B, 0x25D479D8, 0xF6E8DEF7,
    0xE3FE501A, 0xB6794C3B, 0x976CE0BD, 0x04C006BA,
    0xC1A94FB6, 0x409F60C4, 0x5E5C9EC2, 0x196A2463,
    0x68FB6FAF, 0x3E6C53B5, 0x1339B2EB, 0x3B52EC6F,
    0x6DFC511F, 0x9B30952C, 0xCC814544, 0xAF5EBD09,
    0xBEE3D004, 0xDE334AFD, 0x660F2807, 0x192E4BB3,
    0xC0CBA857, 0x45C8740F, 0xD20B5F39, 0xB9D3FBDB,
    0x5579C0BD, 0x1A60320A, 0xD6A100C6, 0x402C7279,
    0x679F25FE, 0xFB1FA3CC, 0x8EA5E9F8, 0xDB3222F8,
    0x3C7516DF, 0xFD616B15, 0x2F501EC8, 0xAD0552AB,
    0x323DB5FA, 0xFD238760, 0x53317B48, 0x3E00DF82,
    0x9E5C57BB, 0xCA6F8CA0, 0x1A87562E, 0xDF1769DB,
    0xD542A8F6, 0x287EFFC3, 0xAC6732C6, 0x8C4F5573,
    0x695B27B0, 0xBBCA58C8, 0xE1FFA35D, 0xB8F011A0,
    0x10FA3D98, 0xFD2183B8, 0x4AFCB56C, 0x2DD1D35B,
    0x9A53E479, 0xB6F84565, 0xD28E49BC, 0x4BFB9790,
    0xE1DDF2DA, 0xA4CB7E33, 0x62FB1341, 0xCEE4C6E8,
    0xEF20CADA, 0x36774C01, 0xD07E9EFE, 0x2BF11FB4,
    0x95DBDA4D, 0xAE909198, 0xEAAD8E71, 0x6B93D5A0,
    0xD08ED1D0, 0xAFC725E0, 0x8E3C5B2F, 0x8E7594B7,
    0x8FF6E2FB, 0xF2122B64, 0x8888B812, 0x900DF01C,
    0x4FAD5EA0, 0x688FC31C, 0xD1CFF191, 0xB3A8C1AD,
    0x2F2F2218, 0xBE0E1777, 0xEA752DFE, 0x8B021FA1,
    0xE5A0CC0F, 0xB56F74E8, 0x18ACF3D6, 0xCE89E299,
    0xB4A84FE0, 0xFD13E0B7, 0x7CC43B81, 0xD2ADA8D9,
    0x165FA266, 0x80957705, 0x93CC7314, 0x211A1477,
    0xE6AD2065, 0x77B5FA86, 0xC75442F5, 0xFB9D35CF,
    0xEBCDAF0C, 0x7B3E89A0, 0xD6411BD3, 0xAE1E7E49,
    0x00250E2D, 0x2071B35E, 0x226800BB, 0x57B8E0AF,
    0x2464369B, 0xF009B91E, 0x5563911D, 0x59DFA6AA,
    0x78C14389, 0xD95A537F, 0x207D5BA2, 0x02E5B9C5,
    0x83260376, 0x6295CFA9, 0x11C81968, 0x4E734A41,
    0xB3472DCA, 0x7B14A94A, 0x1B510052, 0x9A532915,
    0xD60F573F, 0xBC9BC6E4, 0x2B60A476, 0x81E67400,
    0x08BA6FB5, 0x571BE91F, 0xF296EC6B, 0x2A0DD915,
    0xB6636521, 0xE7B9F9B6, 0xFF34052E, 0xC5855664,
    0x53B02D5D, 0xA99F8FA1, 0x08BA4799, 0x6E85076A
],
[
    0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
    0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
    0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
    0x193602A5, 0x75094C29, 0xA0591340, 0xE4183A3E,
    0x3F54989A, 0x5B429D65, 0x6B8FE4D6, 0x99F73FD6,
    0xA1D29C07, 0xEFE830F5, 0x4D2D38E6, 0xF0255DC1,
    0x4CDD2086, 0x8470EB26, 0x6382E9C6, 0x021ECC5E,
    0x09686B3F, 0x3EBAEFC9, 0x3C971814, 0x6B6A70A1,
    0x687F3584, 0x52A0E286, 0xB79C5305, 0xAA500737,
    0x3E07841C, 0x7FDEAE5C, 0x8E7D44EC, 0x5716F2B8,
```

```
    0xB03ADA37, 0xF0500C0D, 0xF01C1F04, 0x0200B3FF,
    0xAE0CF51A, 0x3CB574B2, 0x25837A58, 0xDC0921BD,
    0xD19113F9, 0x7CA92FF6, 0x94324773, 0x22F54701,
    0x3AE5E581, 0x37C2DADC, 0xC8B57634, 0x9AF3DDA7,
    0xA9446146, 0x0FD0030E, 0xECC8C73E, 0xA4751E41,
    0xE238CD99, 0x3BEA0E2F, 0x3280BBA1, 0x183EB331,
    0x4E548B38, 0x4F6DB908, 0x6F420D03, 0xF60A04BF,
    0x2CB81290, 0x24977C79, 0x5679B072, 0xBCAF89AF,
    0xDE9A771F, 0xD9930810, 0xB38BAE12, 0xDCCF3F2E,
    0x5512721F, 0x2E6B7124, 0x501ADDE6, 0x9F84CD87,
    0x7A584718, 0x7408DA17, 0xBC9F9ABC, 0xE94B7D8C,
    0xEC7AEC3A, 0xDB851DFA, 0x63094366, 0xC464C3D2,
    0xEF1C1847, 0x3215D908, 0xDD433B37, 0x24C2BA16,
    0x12A14D43, 0x2A65C451, 0x50940002, 0x133AE4DD,
    0x71DFF89E, 0x10314E55, 0x81AC77D6, 0x5F11199B,
    0x043556F1, 0xD7A3C76B, 0x3C11183B, 0x5924A509,
    0xF28FE6ED, 0x97F1FBFA, 0x9EBABF2C, 0x1E153C6E,
    0x86E34570, 0xEAE96FB1, 0x860E5E0A, 0x5A3E2AB3,
    0x771FE71C, 0x4E3D06FA, 0x2965DCB9, 0x99E71D0F,
    0x803E89D6, 0x5266C825, 0x2E4CC978, 0x9C10B36A,
    0xC6150EBA, 0x94E2EA78, 0xA5FC3C53, 0x1E0A2DF4,
    0xF2F74EA7, 0x361D2B3D, 0x1939260F, 0x19C27960,
    0x5223A708, 0xF71312B6, 0xEBADFE6E, 0xEAC31F66,
    0xE3BC4595, 0xA67BC883, 0xB17F37D1, 0x018CFF28,
    0xC332DDEF, 0xBE6C5AA5, 0x65582185, 0x68AB9802,
    0xEECEA50F, 0xDB2F953B, 0x2AEF7DAD, 0x5B6E2F84,
    0x1521B628, 0x29076170, 0xECDD4775, 0x619F1510,
    0x13CCA830, 0xEB61BD96, 0x0334FE1E, 0xAA0363CF,
    0xB5735C90, 0x4C70A239, 0xD59E9E0B, 0xCBAADE14,
    0xEECC86BC, 0x60622CA7, 0x9CAB5CAB, 0xB2F3846E,
    0x648B1EAF, 0x19BDF0CA, 0xA02369B9, 0x655ABB50,
    0x40685A32, 0x3C2AB4B3, 0x319EE9D5, 0xC021B8F7,
    0x9B540B19, 0x875FA099, 0x95F7997E, 0x623D7DA8,
    0xF837889A, 0x97E32D77, 0x11ED935F, 0x16681281,
    0x0E358829, 0xC7E61FD6, 0x96DEDFA1, 0x7858BA99,
    0x57F584A5, 0x1B227263, 0x9B83C3FF, 0x1AC24696,
    0xCDB30AEB, 0x532E3054, 0x8FD948E4, 0x6DBC3128,
    0x58EBF2EF, 0x34C6FFEA, 0xFE28ED61, 0xEE7C3C73,
    0x5D4A14D9, 0xE864B7E3, 0x42105D14, 0x203E13E0,
    0x45EEE2B6, 0xA3AAABEA, 0xDB6C4F15, 0xFACB4FD0,
    0xC742F442, 0xEF6ABBB5, 0x654F3B1D, 0x41CD2105,
    0xD81E799E, 0x86854DC7, 0xE44B476A, 0x3D816250,
    0xCF62A1F2, 0x5B8D2646, 0xFC8883A0, 0xC1C7B6A3,
    0x7F1524C3, 0x69CB7492, 0x47848A0B, 0x5692B285,
    0x095BBF00, 0xAD19489D, 0x1462B174, 0x23820E00,
    0x58428D2A, 0x0C55F5EA, 0x1DADF43E, 0x233F7061,
    0x3372F092, 0x8D937E41, 0xD65FECF1, 0x6C223BDB,
    0x7CDE3759, 0xCBEE7460, 0x4085F2A7, 0xCE77326E,
    0xA6078084, 0x19F8509E, 0xE8EFD855, 0x61D99735,
    0xA969A7AA, 0xC50C06C2, 0x5A04ABFC, 0x800BCADC,
    0x9E447A2E, 0xC3453484, 0xFDD56705, 0x0E1E9EC9,
    0xDB73DBD3, 0x105588CD, 0x675FDA79, 0xE3674340,
    0xC5C43465, 0x713E38D8, 0x3D28F89E, 0xF16DFF20,
    0x153E21E7, 0x8FB03D4A, 0xE6E39F2B, 0xDB83ADF7
],
[
    0xE93D5A68, 0x948140F7, 0xF64C261C, 0x94692934,
    0x411520F7, 0x7602D4F7, 0xBCF46B2E, 0xD4A20068,
    0xD4082471, 0x3320F46A, 0x43B7D4B7, 0x500061AF,
```

```
0x1E39F62E, 0x97244546, 0x14214F74, 0xBF8B8840,
0x4D95FC1D, 0x96B591AF, 0x70F4DDD3, 0x66A02F45,
0xBFBC09EC, 0x03BD9785, 0x7FAC6DD0, 0x31CB8504,
0x96EB27B3, 0x55FD3941, 0xDA2547E6, 0xABCA0A9A,
0x28507825, 0x530429F4, 0x0A2C86DA, 0xE9B66DFB,
0x68DC1462, 0xD7486900, 0x680EC0A4, 0x27A18DEE,
0x4F3FFEA2, 0xE887AD8C, 0xB58CE006, 0x7AF4D6B6,
0xAACE1E7C, 0xD3375FEC, 0xCE78A399, 0x406B2A42,
0x20FE9E35, 0xD9F385B9, 0xEE39D7AB, 0x3B124E8B,
0x1DC9FAF7, 0x4B6D1856, 0x26A36631, 0xEAE397B2,
0x3A6EFA74, 0xDD5B4332, 0x6841E7F7, 0xCA7820FB,
0xFB0AF54E, 0xD8FEB397, 0x454056AC, 0xBA489527,
0x55533A3A, 0x20838D87, 0xFE6BA9B7, 0xD096954B,
0x55A867BC, 0xA1159A58, 0xCCA92963, 0x99E1DB33,
0xA62A4A56, 0x3F3125F9, 0x5EF47E1C, 0x9029317C,
0xFDF8E802, 0x04272F70, 0x80BB155C, 0x05282CE3,
0x95C11548, 0xE4C66D22, 0x48C1133F, 0xC70F86DC,
0x07F9C9EE, 0x41041F0F, 0x404779A4, 0x5D886E17,
0x325F51EB, 0xD59BC0D1, 0xF2BCC18F, 0x41113564,
0x257B7834, 0x602A9C60, 0xDFF8E8A3, 0x1F636C1B,
0x0E12B4C2, 0x02E1329E, 0xAF664FD1, 0xCAD18115,
0x6B2395E0, 0x333E92E1, 0x3B240B62, 0xEEBEB922,
0x85B2A20E, 0xE6BA0D99, 0xDE720C8C, 0x2DA2F728,
0xD0127845, 0x95B794FD, 0x647D0862, 0xE7CCF5F0,
0x5449A36F, 0x877D48FA, 0xC39DFD27, 0xF33E8D1E,
0x0A476341, 0x992EFF74, 0x3A6F6EAB, 0xF4F8FD37,
0xA812DC60, 0xA1EBDDF8, 0x991BE14C, 0xDB6E6B0D,
0xC67B5510, 0x6D672C37, 0x2765D43B, 0xDCD0E804,
0xF1290DC7, 0xCC00FFA3, 0xB5390F92, 0x690FED0B,
0x667B9FFB, 0xCEDB7D9C, 0xA091CF0B, 0xD9155EA3,
0xBB132F88, 0x515BAD24, 0x7B9479BF, 0x763BD6EB,
0x37392EB3, 0xCC115979, 0x8026E297, 0xF42E312D,
0x6842ADA7, 0xC66A2B3B, 0x12754CCC, 0x782EF11C,
0x6A124237, 0xB79251E7, 0x06A1BBE6, 0x4BFB6350,
0x1A6B1018, 0x11CAEDFA, 0x3D25BDD8, 0xE2E1C3C9,
0x44421659, 0x0A121386, 0xD90CEC6E, 0xD5ABEA2A,
0x64AF674E, 0xDA86A85F, 0xBEBFE988, 0x64E4C3FE,
0x9DBC8057, 0xF0F7C086, 0x60787BF8, 0x6003604D,
0xD1FD8346, 0xF6381FB0, 0x7745AE04, 0xD736FCCC,
0x83426B33, 0xF01EAB71, 0xB0804187, 0x3C005E5F,
0x77A057BE, 0xBDE8AE24, 0x55464299, 0xBF582E61,
0x4E58F48F, 0xF2DDFDA2, 0xF474EF38, 0x8789BDC2,
0x5366F9C3, 0xC8B38E74, 0xB475F255, 0x46FCD9B9,
0x7AEB2661, 0x8B1DDF84, 0x846A0E79, 0x915F95E2,
0x466E598E, 0x20B45770, 0x8CD55591, 0xC902DE4C,
0xB90BACE1, 0xBB8205D0, 0x11A86248, 0x7574A99E,
0xB77F19B6, 0xE0A9DC09, 0x662D09A1, 0xC4324633,
0xE85A1F02, 0x09F0BE8C, 0x4A99A025, 0x1D6EFE10,
0x1AB93D1D, 0x0BA5A4DF, 0xA186F20F, 0x2868F169,
0xDCB7DA83, 0x573906FE, 0xA1E2CE9B, 0x4FCD7F52,
0x50115E01, 0xA70683FA, 0xA002B5C4, 0x0DE6D027,
0x9AF88C27, 0x773F8641, 0xC3604C06, 0x61A806B5,
0xF0177A28, 0xC0F586E0, 0x006058AA, 0x30DC7D62,
0x11E69ED7, 0x2338EA63, 0x53C2DD94, 0xC2C21634,
0xBBCBEE56, 0x90BCB6DE, 0xEBFC7DA1, 0xCE591D76,
0x6F05E409, 0x4B7C0188, 0x39720A3D, 0x7C927C24,
0x86E3725F, 0x724D9DB9, 0x1AC15BB4, 0xD39EB8FC,
0xED545578, 0x08FCA5B5, 0xD83D7CD3, 0x4DAD0FC4,
0x1E50EF5E, 0xB161E6F8, 0xA28514D9, 0x6C51133C,
```

```
    0x6FD5C7E7, 0x56E14EC4, 0x362ABFCE, 0xDDC6C837,
    0xD79A3234, 0x92638212, 0x670EFA8E, 0x406000E0
],
[
    0x3A39CE37, 0xD3FAF5CF, 0xABC27737, 0x5AC52D1B,
    0x5CB0679E, 0x4FA33742, 0xD3822740, 0x99BC9BBE,
    0xD5118E9D, 0xBF0F7315, 0xD62D1C7E, 0xC700C47B,
    0xB78C1B6B, 0x21A19045, 0xB26EB1BE, 0x6A366EB4,
    0x5748AB2F, 0xBC946E79, 0xC6A376D2, 0x6549C2C8,
    0x530FF8EE, 0x468DDE7D, 0xD5730A1D, 0x4CD04DC6,
    0x2939BBDB, 0xA9BA4650, 0xAC9526E8, 0xBE5EE304,
    0xA1FAD5F0, 0x6A2D519A, 0x63EF8CE2, 0x9A86EE22,
    0xC089C2B8, 0x43242EF6, 0xA51E03AA, 0x9CF2D0A4,
    0x83C061BA, 0x9BE96A4D, 0x8FE51550, 0xBA645BD6,
    0x2826A2F9, 0xA73A3AE1, 0x4BA99586, 0xEF5562E9,
    0xC72FEFD3, 0xF752F7DA, 0x3F046F69, 0x77FA0A59,
    0x80E4A915, 0x87B08601, 0x9B09E6AD, 0x3B3EE593,
    0xE990FD5A, 0x9E34D797, 0x2CF0B7D9, 0x022B8B51,
    0x96D5AC3A, 0x017DA67D, 0xD1CF3ED6, 0x7C7D2D28,
    0x1F9F25CF, 0xADF2B89B, 0x5AD6B472, 0x5A88F54C,
    0xE029AC71, 0xE019A5E6, 0x47B0ACFD, 0xED93FA9B,
    0xE8D3C48D, 0x283B57CC, 0xF8D56629, 0x79132E28,
    0x785F0191, 0xED756055, 0xF7960E44, 0xE3D35E8C,
    0x15056DD4, 0x88F46DBA, 0x03A16125, 0x0564F0BD,
    0xC3EB9E15, 0x3C9057A2, 0x97271AEC, 0xA93A072A,
    0x1B3F6D9B, 0x1E6321F5, 0xF59C66FB, 0x26DCF319,
    0x7533D928, 0xB155FDF5, 0x03563482, 0x8ABA3CBB,
    0x28517711, 0xC20AD9F8, 0xABCC5167, 0xCCAD925F,
    0x4DE81751, 0x3830DC8E, 0x379D5862, 0x9320F991,
    0xEA7A90C2, 0xFB3E7BCE, 0x5121CE64, 0x774FBE32,
    0xA8B6E37E, 0xC3293D46, 0x48DE5369, 0x6413E680,
    0xA2AE0810, 0xDD6DB224, 0x69852DFD, 0x09072166,
    0xB39A460A, 0x6445C0DD, 0x586CDECF, 0x1C20C8AE,
    0x5BBEF7DD, 0x1B588D40, 0xCCD2017F, 0x6BB4E3BB,
    0xDDA26A7E, 0x3A59FF45, 0x3E350A44, 0xBCB4CDD5,
    0x72EACEA8, 0xFA6484BB, 0x8D6612AE, 0xBF3C6F47,
    0xD29BE463, 0x542F5D9E, 0xAEC2771B, 0xF64E6370,
    0x740E0D8D, 0xE75B1357, 0xF8721671, 0xAF537D5D,
    0x4040CB08, 0x4EB4E2CC, 0x34D2466A, 0x0115AF84,
    0xE1B00428, 0x95983A1D, 0x06B89FB4, 0xCE6EA048,
    0x6F3F3B82, 0x3520AB82, 0x011A1D4B, 0x277227F8,
    0x611560B1, 0xE7933FDC, 0xBB3A792B, 0x344525BD,
    0xA08839E1, 0x51CE794B, 0x2F32C9B7, 0xA01FBAC9,
    0xE01CC87E, 0xBCC7D1F6, 0xCF0111C3, 0xA1E8AAC7,
    0x1A908749, 0xD44FBD9A, 0xD0DADECB, 0xD50ADA38,
    0x0339C32A, 0xC6913667, 0x8DF9317C, 0xE0B12B4F,
    0xF79E59B7, 0x43F5BB3A, 0xF2D519FF, 0x27D9459C,
    0xBF97222C, 0x15E6FC2A, 0x0F91FC71, 0x9B941525,
    0xFAE59361, 0xCEB69CEB, 0xC2A86459, 0x12BAA8D1,
    0xB6C1075E, 0xE3056A0C, 0x10D25065, 0xCB03A442,
    0xE0EC6E0E, 0x1698DB3B, 0x4C98A0BE, 0x3278E964,
    0x9F1F9532, 0xE0D392DF, 0xD3A0342B, 0x8971F21E,
    0x1B0A7441, 0x4BA3348C, 0xC5BE7120, 0xC37632D8,
    0xDF359F8D, 0x9B992F2E, 0xE60B6F47, 0x0FE3F11D,
    0xE54CDA54, 0x1EDAD891, 0xCE6279CF, 0xCD3E7E6F,
    0x1618B166, 0xFD2C1D05, 0x848FD2C5, 0xF6FB2299,
    0xF523F357, 0xA6327623, 0x93A83531, 0x56CCCD02,
    0xACF08162, 0x5A75EBB5, 0x6E163697, 0x88D273CC,
    0xDE966292, 0x81B949D0, 0x4C50901B, 0x71C65614,
```

```
          0xE6C6C7BD, 0x327A140A, 0x45E1D006, 0xC3F27B9A,
          0xC9AA53FD, 0x62A80F00, 0xBB25BFE2, 0x35BDD2F6,
          0x71126905, 0xB2040222, 0xB6CBCF7C, 0xCD769C2B,
          0x53113EC0, 0x1640E3D3, 0x38ABBD60, 0x2547ADF0,
          0xBA38209C, 0xF746CE76, 0x77AFA1C5, 0x20756060,
          0x85CBFE4E, 0x8AE88DD8, 0x7AAAF9B0, 0x4CF9AA7E,
          0x1948C25C, 0x02FB8A8C, 0x01C36AE4, 0xD6EBE1F9,
          0x90D4F869, 0xA65CDEA0, 0x3F09252D, 0xC208E69F,
          0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6
      ]
]

#key given by user

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

p_new = p.copy()

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

#blowfish 16  encrytion rounds

def blowfish_encrypt(data):
    L = data>>32
    R = data & 0xffffffff
    for i in range(0,16):
        L = p[i]^L
        L1 = func(L)
        R = R^func(L1)
        L,R = swap(L,R)
    L,R = swap(L,R)
    L = L^p[17]
    R = R^p[16]
    encrypted = (L<<32) ^ R
    return encrypted


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

#blowfish 16  decrytion rounds

def blowfish_decrypt(data):
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
        L = p[i]^L
        L1 = func(L)
```

```
        R = R^func(L1)
        L,R = swap(L,R)
    L,R = swap(L,R)
    L = L^p[0]
    R = R^p[1]
    decrypted_data1 = (L<<32) ^ R
    return decrypted_data1

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

#function for cbc mode of operation

def cbcencrypt(plaintext):
    ciphertext = []
    for i in range(0,5):
        if i == 0:
            plaintext[i] = plaintext[i]^IV
        else:
            plaintext[i] = plaintext[i]^ciphertext[i-1]
        ciphertext.append(blowfish_encrypt(plaintext[i]))
    return ciphertext

def cbcdecrypt(ciphertext):
    plaintext = []
    for i in range(0,5):
        ct = blowfish_decrypt(ciphertext[i])
        if i == 0:
            plaintext.append(ct^IV)
        else:
            plaintext.append(ct^ciphertext[i-1])
    return plaintext

#function for ofb mode of operation

def ofbencrypt(plaintext):
    ciphertext = []
    register = IV
    uwu = IV
    for i in range(0,4):
      if i>0:
            l = register & (2**48 - 1)
            r = uwu
            register = (l<<16)^r
      uwu = blowfish_encrypt(register)
      uwu = uwu>>48
      r = plaintext & (2**48 - 1)
      l = plaintext>>48
      ciphertext.append(uwu^l)
      plaintext = r<<16
```

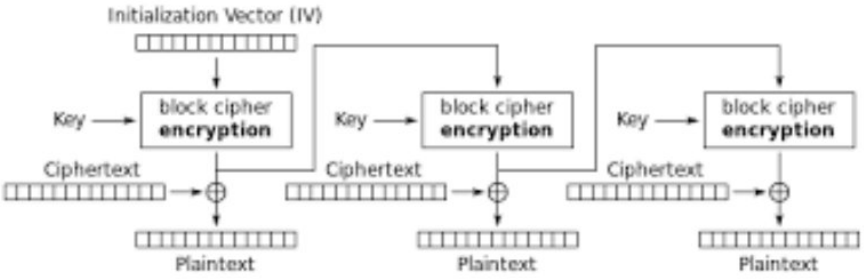
```
    return ciphertext


def ofbdecrypt(ciphertext):
    plaintext = 0
    register = IV
    uwu = IV
    for i in range(0,4):
        if i>0:
            l = register & (2**48 - 1)
            r = uwu
            register = (l<<16)^r
        uwu = blowfish_encrypt(register)
        uwu = uwu>>48
        add = (ciphertext[i]^uwu)
        plaintext = plaintext<<16
        plaintext+=add
    return plaintext

#main function for input and output

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

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

Conclusion :

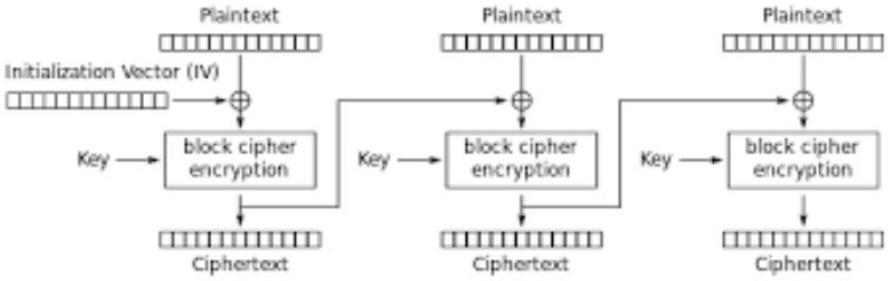
Advantages and Disadvantages of Blowfish Algorithm:

1.)Blowfish is a fast block cipher except when changing keys. Each new key requires a pre-processing equivalent to 4KB of text.
2.)It is faster and much better than DES Encryption.
3.)Blowfish uses a 64-bit block size which makes it vulnerable to birthday attacks.
4.)A reduced round variant of blowfish is known to be suceptible to known plain text attacks(2nd order differential attacks – 4 rounds).

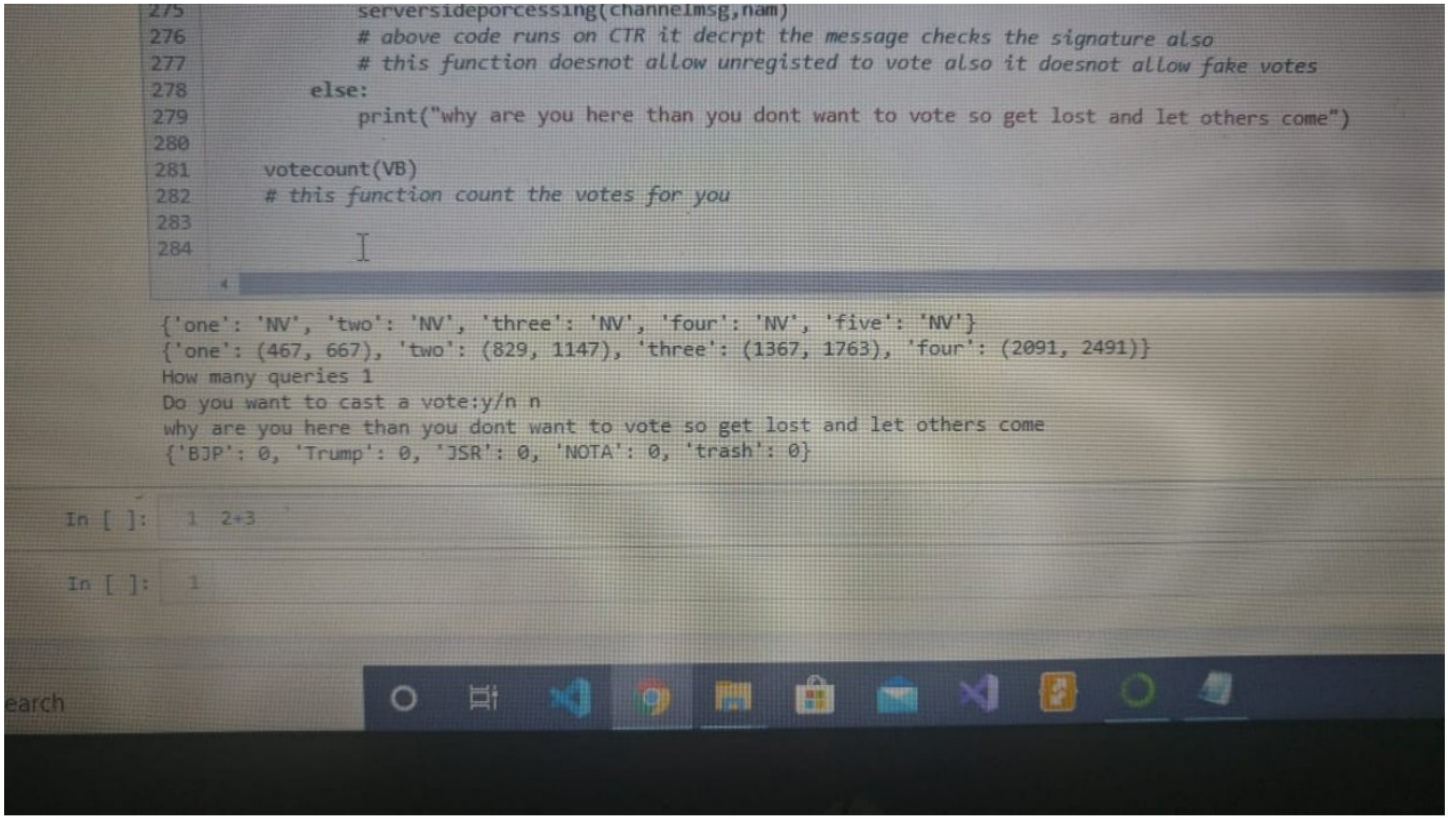
Screenshots and images:

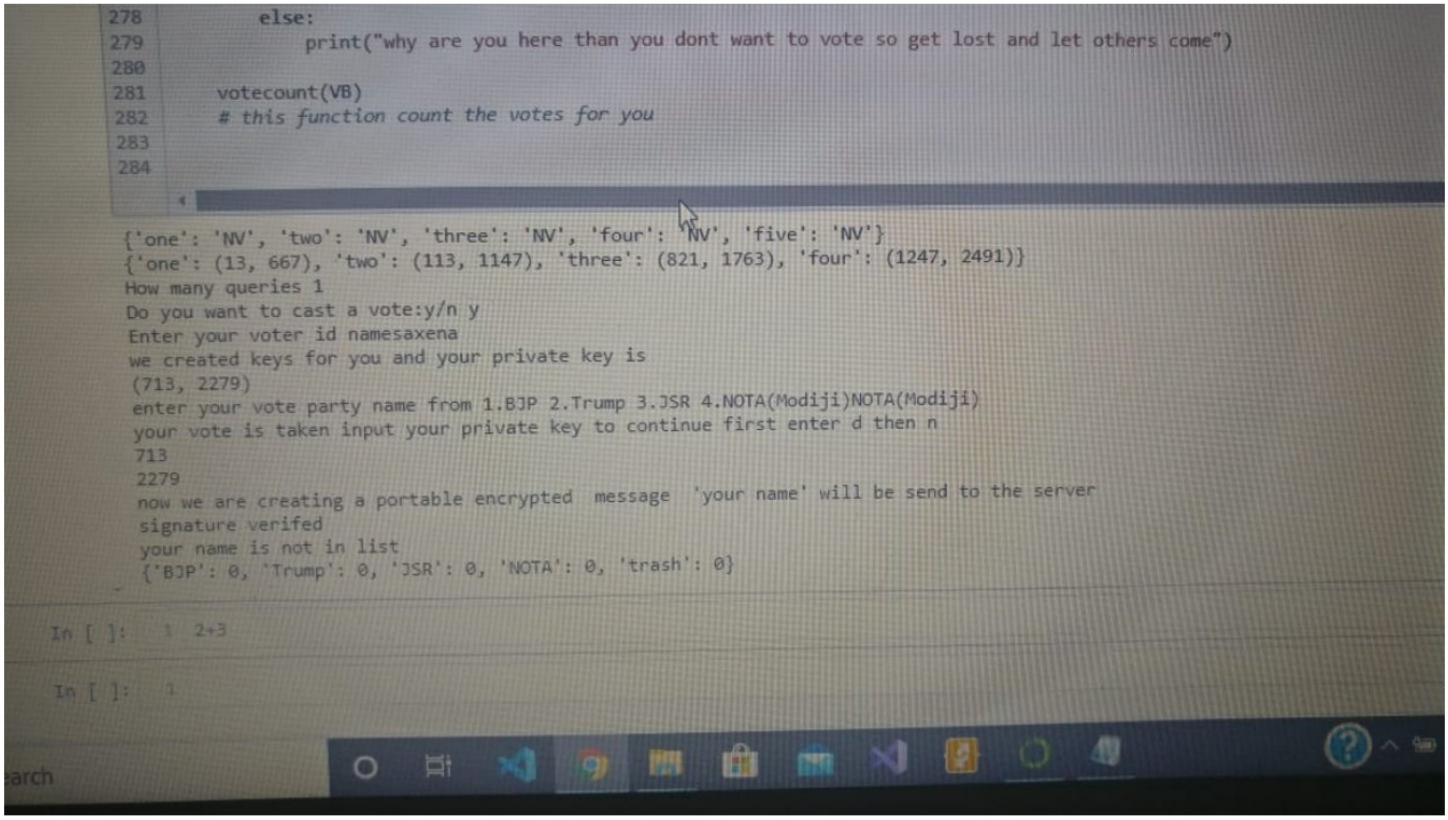


Output Feedback (OFB) mode decryption



Cipher Block Chaining (CBC) mode encryption


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
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (467, 667), 'two': (829, 1147), 'three': (1367, 1763), 'four': (2091, 2491)}
How many queries 1
Do you want to cast a vote:y/n n
why are you here than you dont want to vote so get lost and let others come
{'BJP': 0, 'Trump': 0, 'JSR': 0, 'NOTA': 0, 'trash': 0}
In [ ]: 2+3
In [ ]:
earch

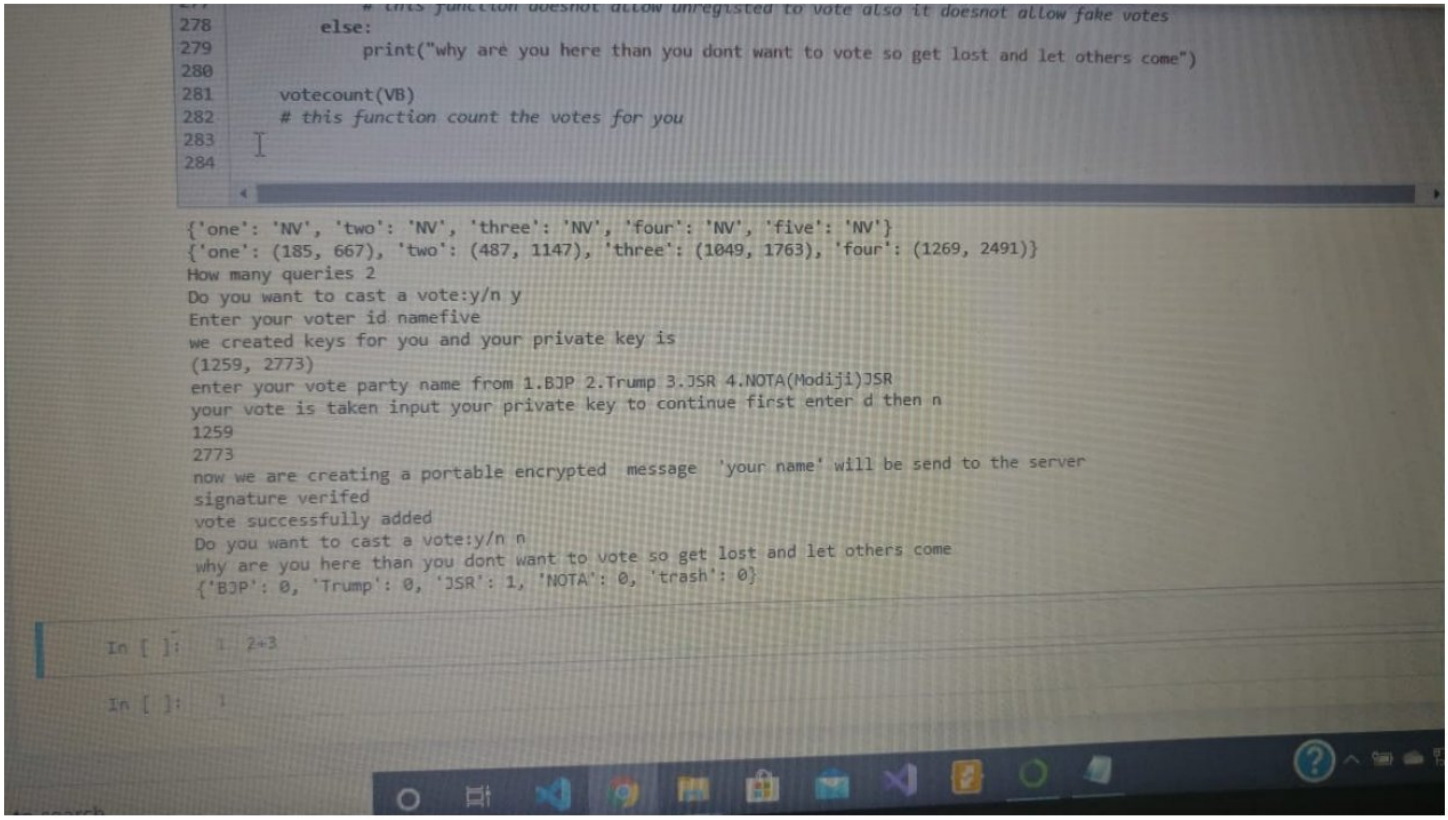

278 else:
279 print("why are you here than you dont want to vote so get lost and let others come")
280
281 votecount(VB)
282 # this function count the votes for you
283
284
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (13, 667), 'two': (113, 1147), 'three': (821, 1763), 'four': (1247, 2491)}
How many queries 1
Do you want to cast a vote:y/n y
Enter your voter id namesaxena
we created keys for you and your private key is
(713, 2279)
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)NOTA(Modiji)
your vote is taken input your private key to continue first enter d then n
713
2279
now we are creating a portable encrypted message 'your name' will be send to the server
signature verifed
your name is not in list
{'BJP': 0, 'Trump': 0, 'JSR': 0, 'NOTA': 0, 'trash': 0}
In [ ]: 1 2+3
In [ ]: 1

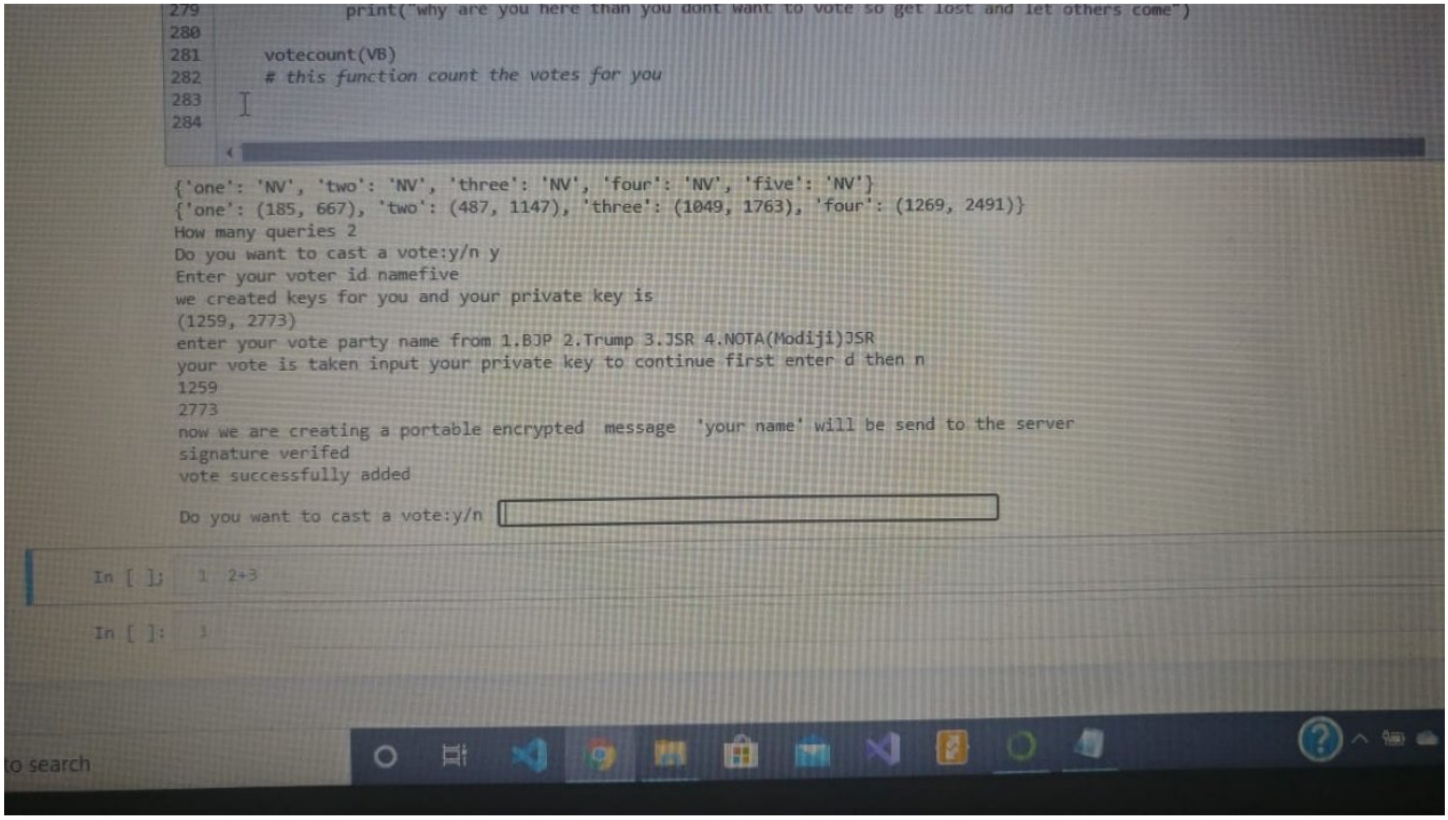
```
281     votecount(VB)
282     # this function count the votes for you
283
284
```

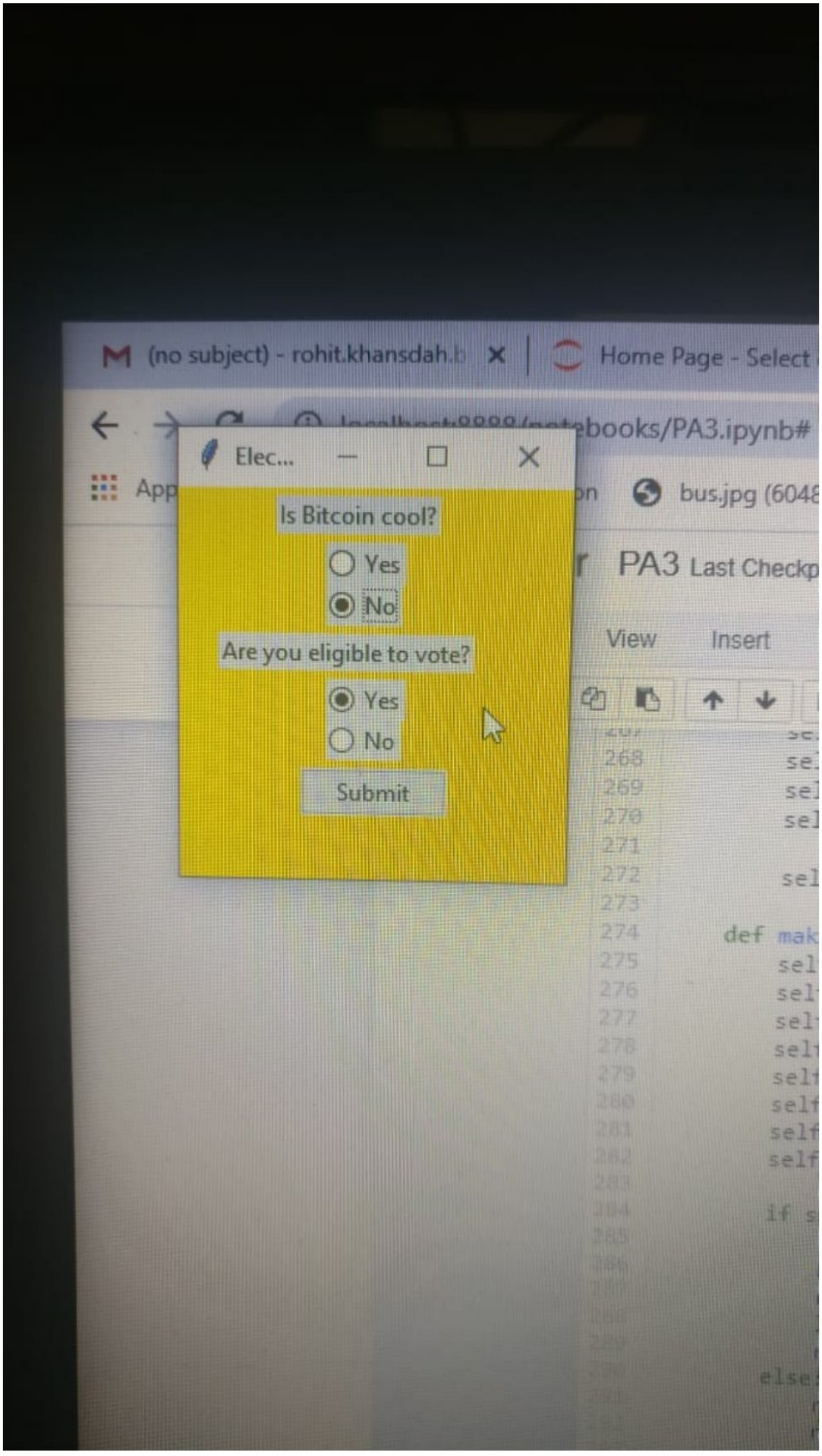
```
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (369, 667), 'two': (901, 1147), 'three': (109, 1763), 'four': (775, 2491)}
How many queries 1
Do you want to cast a vote:y/n y
Enter your voter id namefive
we created keys for you and your private key is
(989, 1577)
enter 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
989
1577
now we are creating a portable encrypted  message  'your name' will be send to the server
signature verifed
vote successfully added
{'BJP': 0, 'Trump': 1, 'JSR': 0, 'NOTA': 0, 'trash': 0}
```

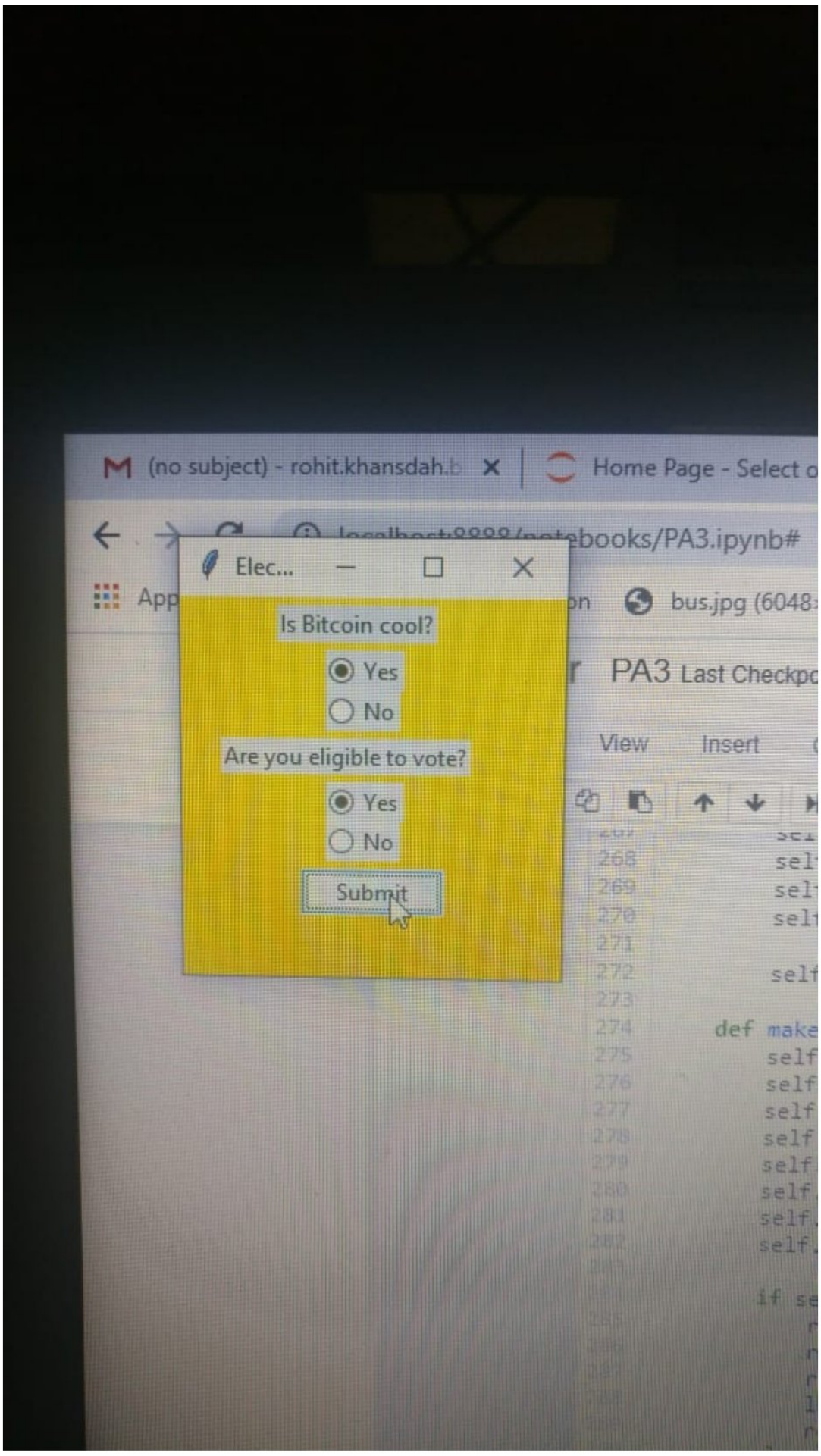
```
In [ ]:  1  2+3
```

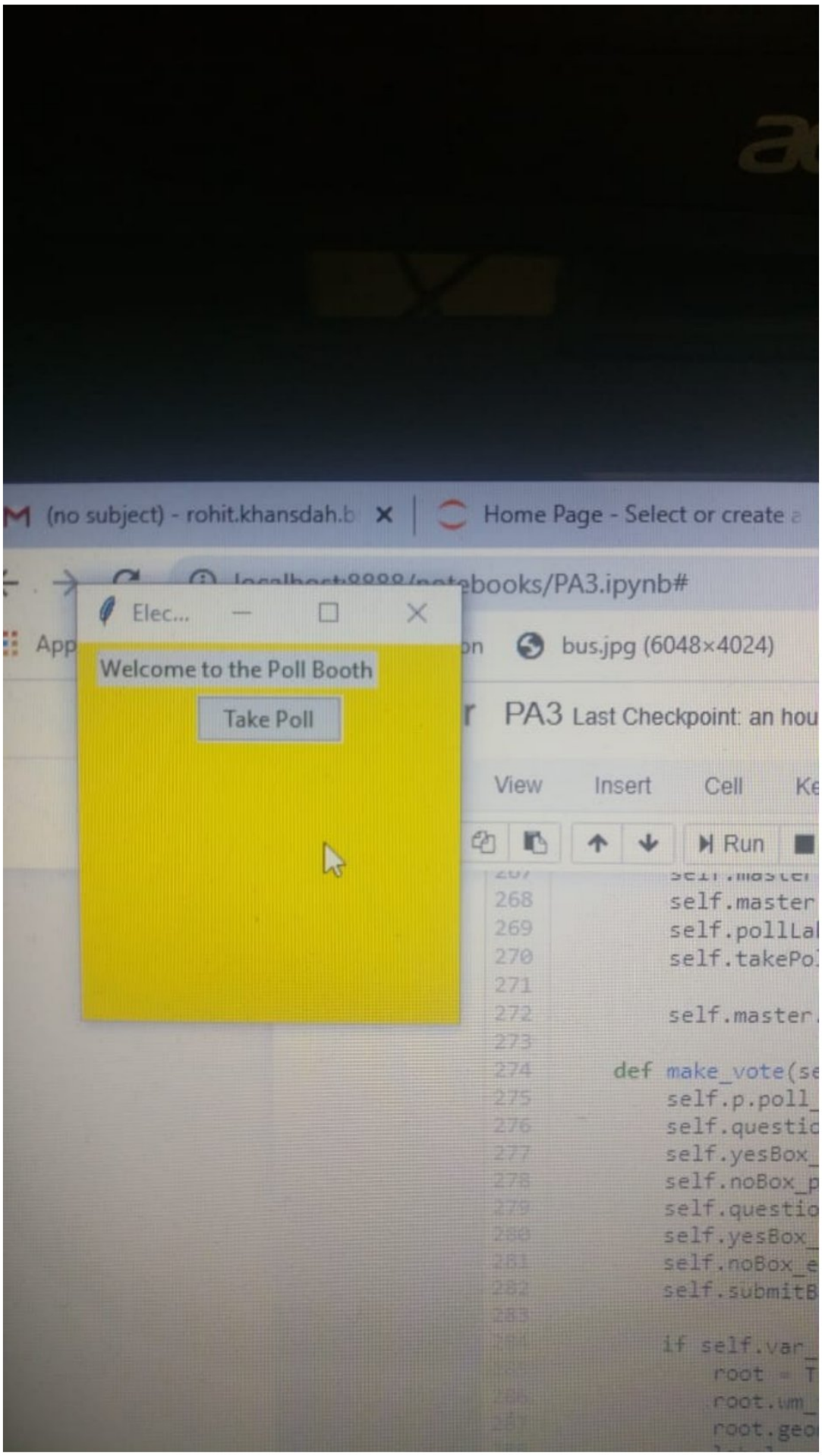
```
In [ ]:  1
```


278 else:
279 print("why are you here than you dont want to vote so get lost and let others come")
280
281 votecount(VB)
282 # this function count the votes for you
283
284
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (185, 667), 'two': (487, 1147), 'three': (1049, 1763), 'four': (1269, 2491)}
How many queries 2
Do you want to cast a vote:y/n y
Enter your voter id namefive
we created keys for you and your private key is
(1259, 2773)
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
1259
2773
now we are creating a portable encrypted message 'your name' will be send to the server
signature verifed
vote successfully added
Do you want to cast a vote:y/n n
why are you here than you dont want to vote so get lost and let others come
{'BJP': 0, 'Trump': 0, 'JSR': 1, 'NOTA': 0, 'trash': 0}
In [ ]: 1 2+3
In [ ]: 1


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
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (185, 667), 'two': (487, 1147), 'three': (1049, 1763), 'four': (1269, 2491)}
How many queries 2
Do you want to cast a vote:y/n y
Enter your voter id namefive
we created keys for you and your private key is
(1259, 2773)
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
1259
2773
now we are creating a portable encrypted message 'your name' will be send to the server
signature verifed
vote successfully added
Do you want to cast a vote:y/n
In [ ]: 1 2+3
In [ ]: 1
o search


(no subject) - rohit.khansdah.b
Home Page - Select
ebooks/PA3.ipynb#
App
bus.jpg (6048
Elec...
Is Bitcoin cool?
Yes
No
Are you eligible to vote?
Yes
No
Submit
PA3 Last Checkp
View
Insert
def mak
if s
else:


(no subject) - rohit.khansdah
Home Page - Select o
ebooks/PA3.ipynb#
App
bus.jpg (6048
Elec...
Is Bitcoin cool?
Yes
No
Are you eligible to vote?
Yes
No
Submit
PA3 Last Checkpo
View
Insert
def make
self
self
self
self
self
self
self
self
if se


(no subject) - rohit.khansdah.b
Home Page - Select or create
bus.jpg (6048×4024)
PA3 Last Checkpoint: an hou
View
Insert
Cell
Run
Elec...
Welcome to the Poll Booth
Take Poll
268 self.master
269 self.pollLa
270 self.takePo
271
272 self.master.
273
274 def make_vote(se
275 self.p.poll_
276 self.questio
277 self.yesBox_
278 self.noBox_p
279 self.questio
280 self.yesBox_
281 self.noBox_e
282 self.submitB
283
if self.var_
root = T
root.um_
root.geo

PA3(Practical assignment 3)

Objective:

Implement the following 2 protocols :

[A] Simplistic Voting Protocol #1

(1) Each voter signs his vote with his private key.
(2) Each voter encrypts his signed vote with the CTF's public key.
(3) Each voter sends his vote to the CTF.
(4) The CTF decrypts the votes, checks the signatures, tabulates the votes, and makes the results public.
Poll start and end time has to be specified.
This protocol satisfies properties one and two: Only authorized voters can vote and no one can vote more than once—the CTF would record votes received in step (3).

[B]Voting with Blind Signatures

We need to somehow dissociate the vote from the voter, while still maintaining authentication. The blind signature based protocol does just that:
(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 (ah, democracy!) 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:

Verifiable Voting Systems:

SHA:

The initial version of the SHA-256 algorithm was created by the US National Security Agency in the spring of 2002.
A few months later, the national metrological University published the newly-announced encryption Protocol in the FIPS PUB 180-2 secure data processing standard adopted at the Federal level.
In the winter of 2004 it was replenished with the second version of the algorithm.
Over the next 3 years, the NSA issued a second-generation Sha patent under Royalty-free license.

This is what gave rise to the use of technology in civilian areas.
This Protocol works with information broken down into pieces of 512 bits (or 64 bytes in other words).
It produces its cryptographic "mixing" and then issues a 256-bit hash code.
The algorithm includes a relatively simple round, which is repeated 64 times.
In addition, SHA-256 has quite good technical parameters:

block size indicator (byte): 64.
maximum allowed message length (bytes): 33.
characteristics of the message digest size (bytes): 32.
the standard word size (bytes): 4.
internal position length parameter (bytes): 32.
the number of iterations in one cycle: 64.
the speed achieved by the Protocol (MiB/s): approximately 140.
The Sha-256 algorithm is based on the Merkle-Damgard construction method, according to which the initial index is divided into blocks immediately after the change is made, and those, in turn, into 16 words.

SHA-256 is used in several different parts of the Bitcoin network:

Mining uses SHA-256 as the proof-of-work algorithm.
SHA-256 is used in the creation of bitcoin addresses to improve security and privacy.

Blind Signature:

Blind signature is a kind of digital signature in which the message is blinded before it is signed. T
herefore, the signer will not learn the message content. Then the signed message will be unblinded.
At this moment, it is similar to a normal digital signature, and it can be publicly checked against the original message.
Blind signature can be implemented using a number of public-key encryption schemes.

Here, we only introduce the simplest one, which is based on RSA encryption.
The signer has a public key (n,e) and a secret key d.
Suppose a party A wants to have a message signed using the blind signature.
She should execute the protocol with the signer S as follows:

1.)A first randomly chooses a value k, which satisfies $0 \leq k \leq n - 1$ and gcd(n,k) = 1.

2.)For the message m, A computes $m* = mke(\text{mod } n)$ and sends $m*$ to S.

3.)When S receives $m*$, S computes and sends back to A.

4.)A computes $s = s*/k(\text{mod } n)$. Now s is S's signature on the message m.

Implementation details :

Implementation in RSA:

Recap on plain RSA signatures:
Euler's totient. Two numbers are "relative primes" if their only common factor is 1.
The Euler's totient of a number n, written phi(n), is the number of relative primes to n which are less than n.

If n=57, all 56 of the numbers less than n are relatively prime to n, because n is prime.
So phi(57)=56. Fact 1: if n is the composite pq, where p and q are prime, then phi(n) = (p-1)(q-1).
For example, suppose n=35. We can work out manually that there are 2
4 primes relative to 35 which are below 35, namely
{1,2,3,4,6,8,9,11,12,13,16,17,18,19,22,23,24,26,27,29,31,32,33,34}.
We can use the fact mentioned to calculate this more directly. Since 35 = 5*7, and 5 and 7 are primes, phi(35)=4*6=24, as expected.
Modulo arithmetic. In "mod n" arithmetic, all numbers are reduced to their remainder on division by n. For example, we are used to working in "mod 256", where (for example) 250 + 10 = 4 (mod 256), and 100 * 4 = 144 (mod 256), and 1002 = 16, 1003=64, 1004=0 (mod 256). Fact 2: aphi(n) = 1 (mod n).

Generating the public and private keys. Exercise. Which of the following key pairs are valid?
K=(3,99), K-1=(27,99)
K=(7,187), K-1=(23,187)
K=(23,187), K-1=(7,187)
K=(7,143), K-1=(23,143)
Can you invert the key (7,147)?
Pick two large prime numbers, p and q. Let n=pq. Typically, n is a 1024 bit number.

Pick e relatively prime to (p-1)(q-1). Now find d such that ed=1 mod (p-1)(q-1).
You can use Euclid's algorithm to find this d.
The pair of numbers (e, n) is the public key.
The pair of numbers (d, n) is the private key. The two primes p,q are no longer needed, and can be discarded, but should never be revealed.

Message format-> Divide the message into blocks, each block corresponding to a number less than n. For example, for binary data, the blocks will be (log2 n) bits.
Signing-> The signature of message m is s = md mod n.
Signature verification-> To recover the message from the signature s, put m' = se mod n.

Code walkthrough :

python code :

part A:

```
# Cryptomath Module for both protocols to be implemented
import random

def gcd(a, b):
    # Returns the GCD of positive integers a and b using the Euclidean Algorithm.
    x, y = a, b
    while y != 0:
        r = x % y
        x = y
        y = r
    return x

def extendedGCD(a,b):
    # Returns integers u, v such that au + bv = gcd(a,b).
    x, y = a, b
    u1, v1 = 1, 0
    u2, v2 = 0, 1
    while y != 0:
```

```
        r = x % y
        q = (x - r) // y
        u, v = u1 - q*u2, v1 - q*v2
        x = y
        y = r
        u1, v1 = u2, v2
        u2, v2 = u, v
    return (u1, v1)

def findModInverse(a, m):
    # Returns the inverse of a modulo m, if it exists.
    if gcd(a,m) != 1:
        return None
    u, v = extendedGCD(a,m)
    return u % m

def RabinMiller(n):
    # Applies the probabilistic Rabin-Miller test for primality.
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    d = n - 1
    s = 0
    while(d % 2 == 0):
        s += 1
        d = d // 2
    # At this point n - 1 = 2^s*d with d odd.
    # Try fifty times to prove that n is composite.
    for i in range(50):
        a = random.randint(2, n - 1)
        if gcd(a, n) != 1:
            return False
        b = pow(a, d, n)
        if b == 1 or b == n - 1:
            continue
        isWitness = True
        r = 1
        while(r < s and isWitness):
            b = pow(b, 2, n)
            if b == n - 1:
                isWitness = False
            r += 1
        if isWitness:
            return False
    return True


def isPrime(n):
    # Determines whether a positive integer n is composite or probably prime.
    if n < 2:
        return False
    smallPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
                   59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,
                   127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181,
                   191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
                   257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317,
```

```
                    331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397,
                    401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
                    467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
                    563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619,
                    631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701,
                    709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787,
                    797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
                    877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953,
                    967, 971, 977, 983, 991, 997]
    # See if n is a small prime.
    if n in smallPrimes:
        return True
    # See if n is divisible by a small prime.
    for p in smallPrimes:
        if n % p == 0:
            return False
    # Apply Fermat test for compositeness.
    for base in [2,3,5,7,11]:
        if pow(base, n - 1, n) != 1:
            return False
    # Apply Rabin-Miller test.
    return RabinMiller(n)


def findPrime(bits=1024, tries=10000):
    # Find a prime with the given number of bits.
    x = 2**(bits - 1)
    y = 2*x
    for i in range(tries):
        n = random.randint(x, y)
        if n % 2 == 0:
            n += 1
        if isPrime(n):
            return n
    return None

def base_b_digits(x, b):
    # Builds a list of the base-b digits of x.
    digits = []
    n = x
    while(n > 0):
        r = n % b
        digits.append(r)
        n = (n - r) // b
    return digits

def isSquare(a, p):
    # Determines whether a is a square modulo p.
    # Assumes that p is an odd prime and a is coprime to p.
    return pow(a, (p - 1) // 2, p) == 1

def modularSqrt(a, p):
    # Returns a square root of a modulo p, if one exists.
    # Assumes that p is a prime congruent to 3 mod 4.
    if isSquare(a, p):
        return pow(a, (p + 1) // 4, p)
    return None
########################################## END OF CRYPTOMATH PART
##############################################
```

```
#Class part of program
import random, hashlib #Initialize Library

class Signer:
    #Creating the class to sign the vote
    def __init__(self):
        self.publicKey, self.privateKey = (self.generateInformation())

    def generateInformation(self):
        # Generates public and private keys and saves them to a file.(RSA)
        p = findPrime()
        q = findPrime()
        phi = (p - 1)*(q - 1)
        n = p*q


        foundEncryptionKey = False
        while not foundEncryptionKey:
            e = random.randint(2, phi - 1)
            if gcd(e, phi) == 1:
                foundEncryptionKey = True

        d = findModInverse(e, phi)

        publicInfo = {"n" : n, "e": e} #Public key of voter
        privateInfo = {"n" : n, "d": d} #Private key of voter

        return[(publicInfo),(privateInfo)]
        ###### This is MY Private and Public Key,Not CTFs Private/Public Key

    def getPublicKey(self):
        return self.publicKey

    def signMessage(self, message, eligible):
        #Function to sign message if voter eligible.
        if eligible == "y":
            return pow(message, self.privateKey['d'], self.publicKey['n'])
        else:
            return None

    def verifyVoter(self, eligible):
        #Function to verify voter needs to be built using information sent to CTF
and then received from it.
        pass


class Voter:
    def __init__(self, n, eligible):
        self.eligible = eligible

    def getEligibility(self):
        #Always Eligibible by the CTF atm.
        return self.eligible

################################################################################
#######

import websocket
import hashlib
```

```
from tkinter import *
from tkinter.ttk import *
import random
# To start machine in case of a poll
class poll:
    def __init__(self, ws):
        #Getting all information required during the poll
        # seLf = CTF
        self.ws = ws
        self.signer = Signer()
        self.publicKey = self.signer.getPublicKey()
        #CTF's public info
        self.n = self.publicKey['n']
        self.e = self.publicKey['e']

    def poll_response(self, poll_answer, eligble_answer):
       #Poll response sent from UI as eligible_answer
       #Yes and no as Poll response for testing the code
       if (poll_answer == 0): poll_answer = 2;
       if (eligble_answer == 0): eligble_answer = "n";
       if (eligble_answer == 1): eligble_answer = "y";

       message = poll_answer
       #Voter class created with eligible answer and public key info.
       voter = Voter(self.n, eligble_answer)

       message_hash = hashlib.sha256(str(message).encode('utf-8'))
       message_hash = message_hash.hexdigest()
       message_hash = int(message_hash,16)
       #Signs the message.
       signedMessage = self.signer.signMessage(message_hash,voter.getEligibility())

       #Sent the signed message to the CTF.
       self.ws.send("Signed message: " + str(signedMessage))



class poll_machine:
    #UI part seen by voter
    def __init__(self):
        self.ws = websocket.WebSocketApp("ws://localhost:8000",
                                on_message = self.on_message,
                                on_error = self.on_error,
                                on_close = self.on_close)
        self.p = poll(self.ws)
        self.master = Tk()
        self.master.configure(background='yellow')
        self.var_poll = IntVar()
        self.var_answer = IntVar()

        self.question_poll = Label(self.master, text="Is Bitcoin cool?")
        self.yesBox_poll = Radiobutton(self.master, text="Yes",
variable=self.var_poll, value=1)
        self.noBox_poll = Radiobutton(self.master, text="No",
variable=self.var_poll, value=0)
        self.question_eligible = Label(self.master, text="Are you eligible to
vote?")
        self.yesBox_eligible = Radiobutton(self.master, text="Yes",
```

```
variable=self.var_answer, value=1)
        self.noBox_eligible = Radiobutton(self.master, text="No",
variable=self.var_answer, value=0)
        self.submitButton = Button(self.master, text='Submit',
command=self.make_vote)

        self.pollLabel = Label(self.master, text="Welcome to the Poll Booth")
        self.takePollButton = Button(self.master, text='Take Poll',
command=self.reset_poll)
        self.pollLabel.grid(row=0, sticky=W, padx=10, pady=4)
        self.takePollButton.grid(row=1, sticky=W, padx=62)

    def on_message(self,ws, message):
        pass

    def on_error(self,ws, error):
        print ("error")

    def on_close(self,ws):
        print ("### closed ###")

    def on_open(self):
        self.master.title("Election Poll Demo")
        self.master.geometry('200x200')
        self.pollLabel.grid(row=0, sticky=W, padx=10, pady=4)
        self.takePollButton.grid(row=1, sticky=W, padx=62)

        self.master.mainloop()

    def make_vote(self):
        self.p.poll_response(self.var_poll.get(),self.var_answer.get())
        self.question_poll.grid_remove()
        self.yesBox_poll.grid_remove()
        self.noBox_poll.grid_remove()
        self.question_eligible.grid_remove()
        self.yesBox_eligible.grid_remove()
        self.noBox_eligible.grid_remove()
        self.submitButton.grid_remove()

        if self.var_answer.get() == 0:
            root = Tk()
            root.wm_title("Unsuccessful Vote")
            root.geometry('200x100')
            label = Label(root, text="Please try again!").grid(row=0, sticky=W)
            root.configure(background='red')
        else:
            root = Tk()
            root.wm_title("Successful Vote")
            root.geometry('200x100')
            label = Label(root, text="Thanks for voting!").grid(row=0, sticky=W)
            root.configure(background='green')

        self.pollLabel.grid(row=0, sticky=W, padx=10, pady=4)
        self.takePollButton.grid(row=1, sticky=W, padx=62)


    def reset_poll(self):

        self.question_poll.grid(row=0, sticky=W, padx=50, pady=4)
```

```
        self.yesBox_poll.grid(row=1, sticky=W, padx=75)
        self.noBox_poll.grid(row=2, sticky=W, padx=75)
        self.question_eligible.grid(row=3, sticky=W, padx=20, pady=4)
        self.yesBox_eligible.grid(row=4, sticky=W, padx=75)
        self.noBox_eligible.grid(row=5, sticky=W, padx=75)
        self.submitButton.grid(row=6, sticky=W, pady=4, padx=62)

        self.pollLabel.grid_remove()
        self.takePollButton.grid_remove()

    def main(self):
        #websocket.enableTrace(True)

        self.ws.on_open = self.on_open()
     #   self.ws.run_forever()

pm = poll_machine()
pm.main()

part B:

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
```

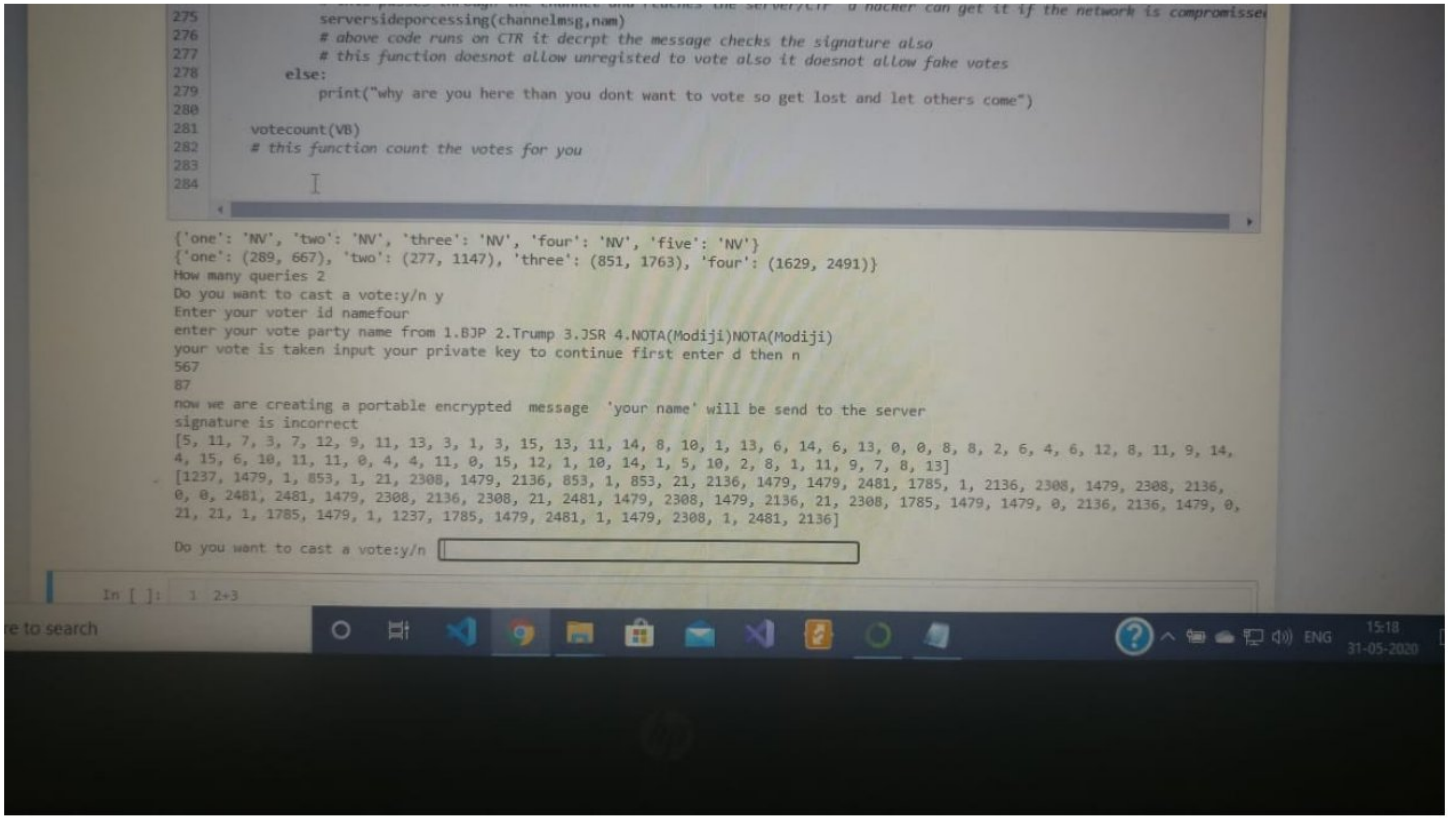
```
            # the voter selects the party name he want to vote this is our base
message
            vot =input("enter your vote party name from 1.BJP 2.Trump 3.JSR
4.NOTA(Modiji)")
            print("your vote is taken input your private key to continue first
enter d then n")
            privd=input()
            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
```

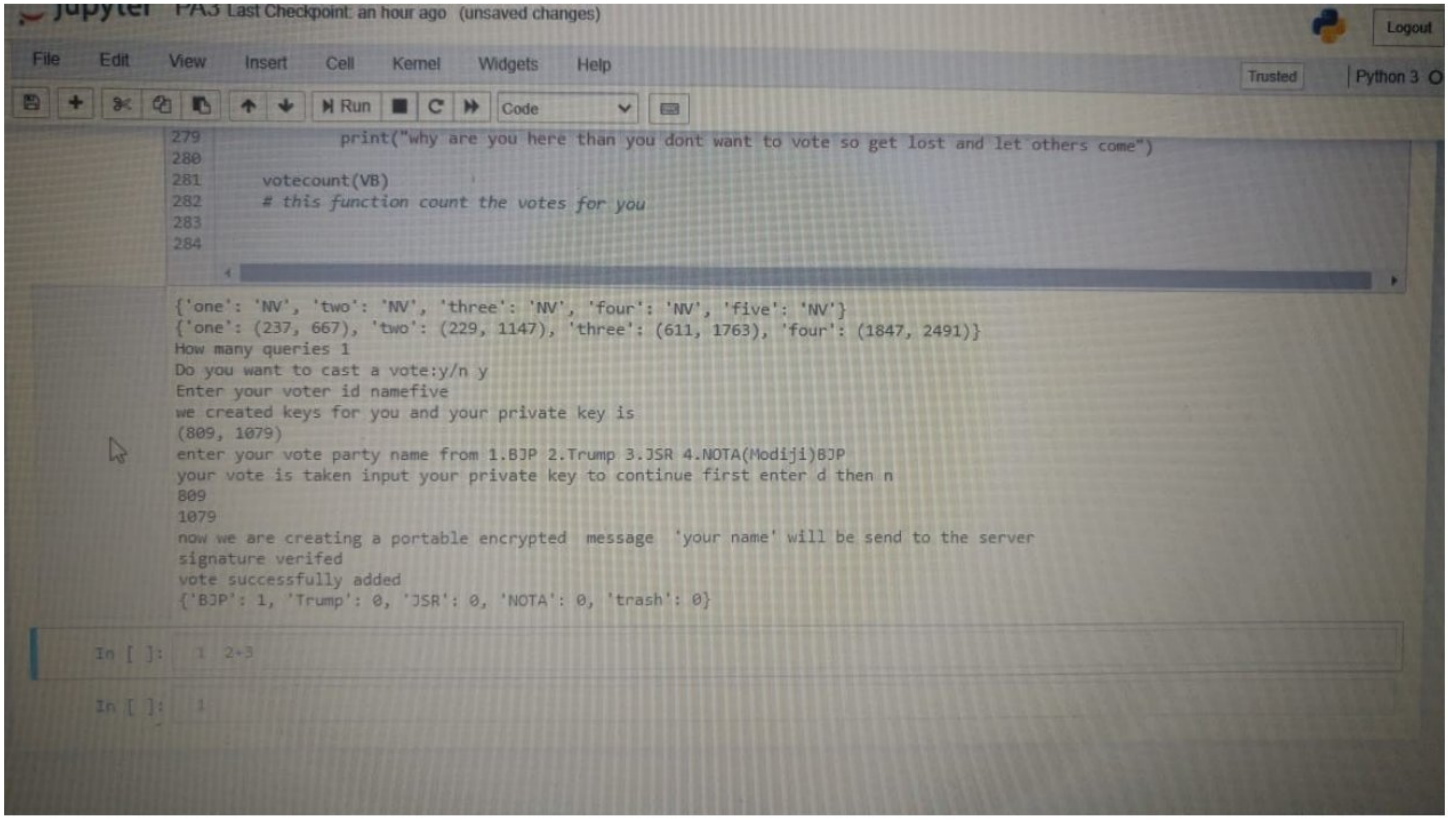
Conclusion:

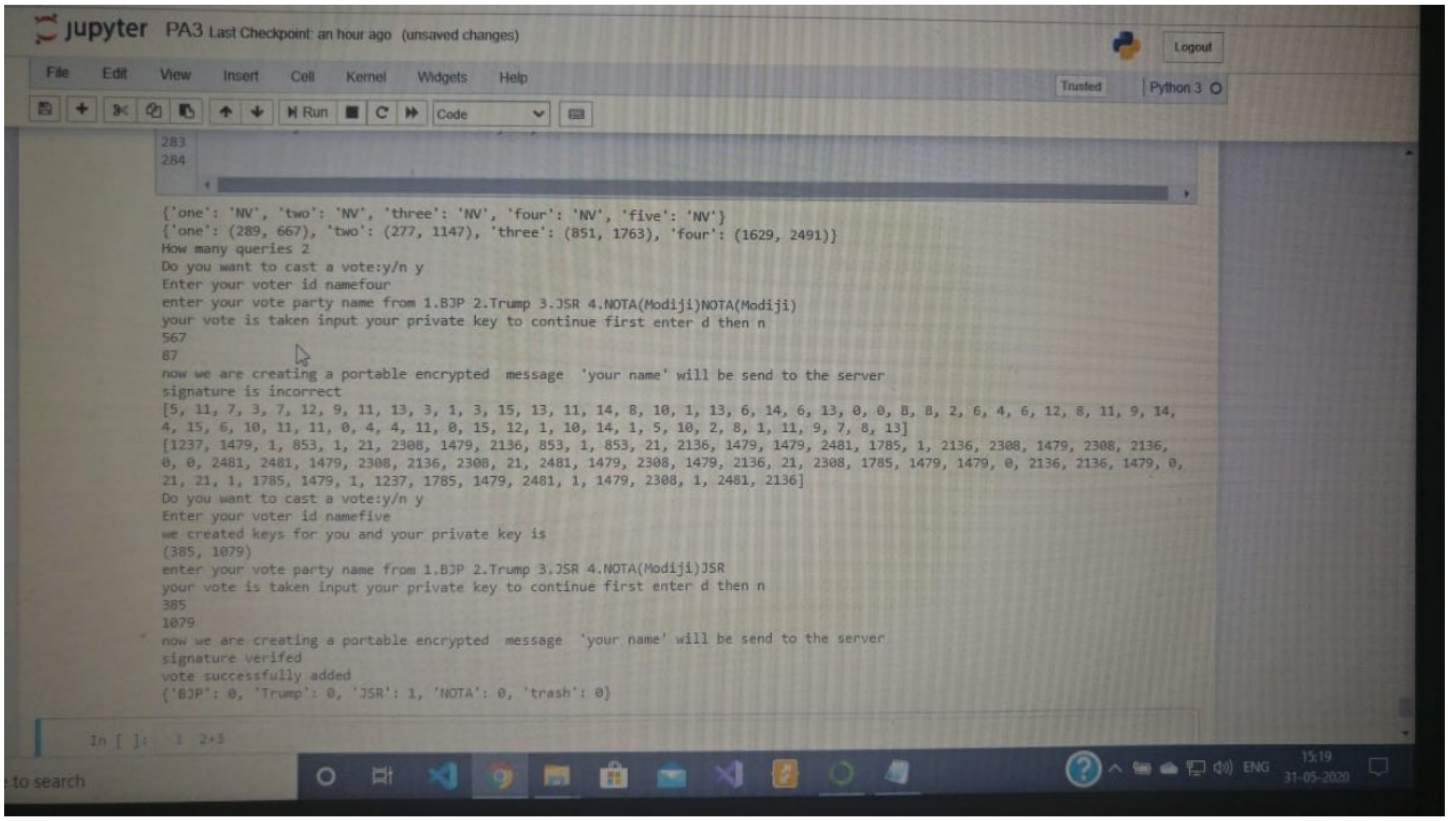
Dangers of RSA blind signing:

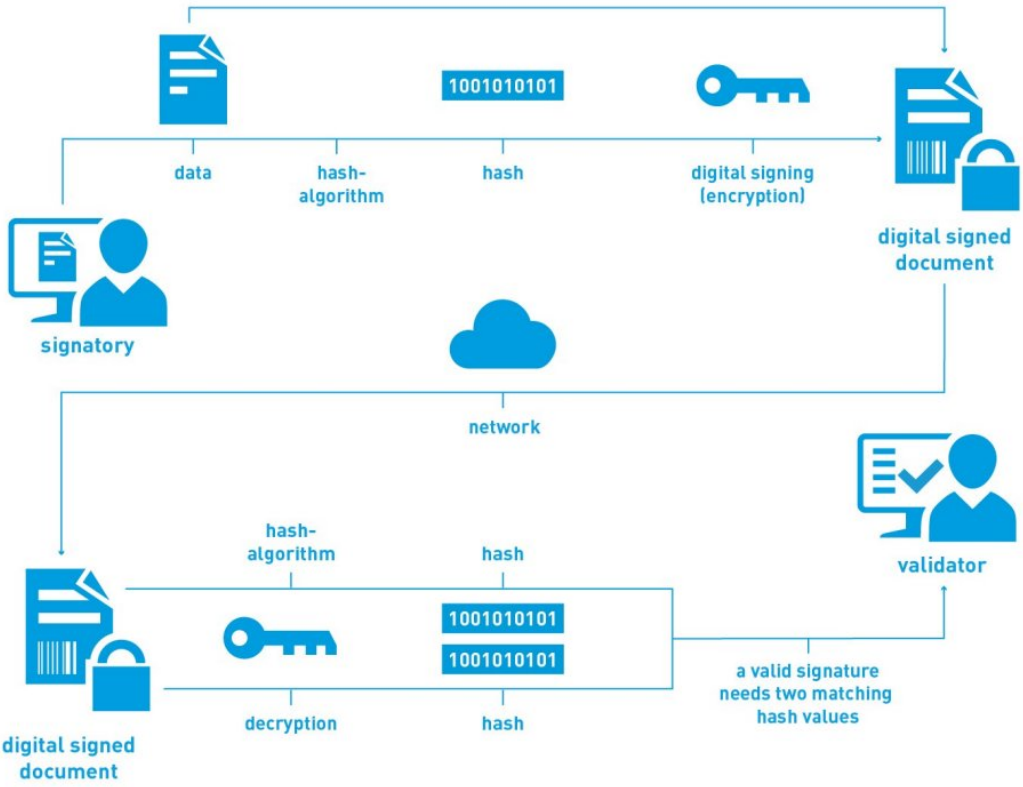
RSA is subject to the RSA blinding attack through which it is possible to be tricked into decrypting
a message by blind signing another message. Since the signing process is equivalent to
decrypting with the signer's secret key, an attacker can provide a blinded version of a message
m encrypted with the signer's public key, m' for them to sign.
The encrypted message would usually be some secret information which the attacker observed being sent
encrypted under the signer's public key which the attacker wants to learn more about.
When the attacker removes the blindness the signed version they will have the clear text

Screenshots and images :

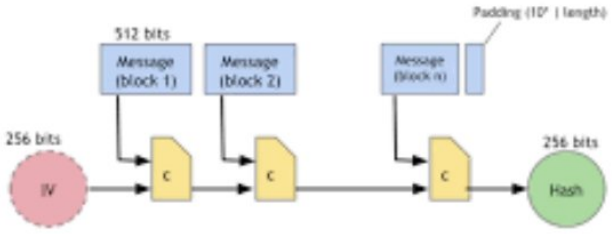

serversideporcessing(channelmsg,nam)
# above code runs on CTR it decrpt the message checks the signature also
# this function doesnot allow unregisted to vote also it doesnot allow fake votes
else:
print("why are you here than you dont want to vote so get lost and let others come")
votecount(VB)
# this function count the votes for you
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (289, 667), 'two': (277, 1147), 'three': (851, 1763), 'four': (1629, 2491)}
How many queries 2
Do you want to cast a vote:y/n y
Enter your voter id namefour
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)NOTA(Modiji)
your vote is taken input your private key to continue first enter d then n
567
87
now we are creating a portable encrypted message 'your name' will be send to the server
signature is incorrect
[5, 11, 7, 3, 7, 12, 9, 11, 13, 3, 1, 3, 15, 13, 11, 14, 8, 10, 1, 13, 6, 14, 6, 13, 0, 0, 8, 8, 2, 6, 4, 6, 12, 8, 11, 9, 14,
4, 15, 6, 10, 11, 11, 0, 4, 4, 11, 0, 15, 12, 1, 10, 14, 1, 5, 10, 2, 8, 1, 11, 9, 7, 8, 13]
[1237, 1479, 1, 853, 1, 21, 2308, 1479, 2136, 853, 1, 853, 21, 2136, 1479, 1479, 2481, 1785, 1, 2136, 2308, 1479, 2308, 2136,
0, 0, 2481, 2481, 1479, 2308, 2136, 2308, 21, 2481, 1479, 2308, 1479, 2136, 21, 2308, 1785, 1479, 1479, 0, 2136, 2136, 1479, 0,
21, 21, 1, 1785, 1479, 1, 1237, 1785, 1479, 2481, 1, 1479, 2308, 1, 2481, 2136]
Do you want to cast a vote:y/n
In [ ]: 1 2+3
to search
ENG
15:18
31-05-2020


PA3 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
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
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (237, 667), 'two': (229, 1147), 'three': (611, 1763), 'four': (1847, 2491)}
How many queries 1
Do you want to cast a vote:y/n y
Enter your voter id namefive
we created keys for you and your private key is
(809, 1079)
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
809
1079
now we are creating a portable encrypted message 'your name' will be send to the server
signature verifed
vote successfully added
{'BJP': 1, 'Trump': 0, 'JSR': 0, 'NOTA': 0, 'trash': 0}
In [ ]: 1 2+3
In [ ]: 1


jupyter PA3 Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
283
284
{'one': 'NV', 'two': 'NV', 'three': 'NV', 'four': 'NV', 'five': 'NV'}
{'one': (289, 667), 'two': (277, 1147), 'three': (851, 1763), 'four': (1629, 2491)}
How many queries 2
Do you want to cast a vote:y/n y
Enter your voter id namefour
enter your vote party name from 1.BJP 2.Trump 3.JSR 4.NOTA(Modiji)NOTA(Modiji)
your vote is taken input your private key to continue first enter d then n
567
87
now we are creating a portable encrypted message 'your name' will be send to the server
signature is incorrect
[5, 11, 7, 3, 7, 12, 9, 11, 13, 3, 1, 3, 15, 13, 11, 14, 8, 10, 1, 13, 6, 14, 6, 13, 0, 0, 8, 8, 2, 6, 4, 6, 12, 8, 11, 9, 14, 4, 15, 6, 10, 11, 11, 0, 4, 4, 11, 0, 15, 12, 1, 10, 14, 1, 5, 10, 2, 8, 1, 11, 9, 7, 8, 13]
[1237, 1479, 1, 853, 1, 21, 2308, 1479, 2136, 853, 1, 853, 21, 2136, 1479, 1479, 2481, 1785, 1, 2136, 2308, 1479, 2308, 2136, 0, 0, 2481, 2481, 1479, 2308, 2136, 2308, 21, 2481, 1479, 2308, 1479, 2136, 21, 2308, 1785, 1479, 1479, 0, 2136, 2136, 1479, 0, 21, 21, 1, 1785, 1479, 1, 1237, 1785, 1479, 2481, 1, 1479, 2308, 1, 2481, 2136]
Do you want to cast a vote:y/n y
Enter your voter id namefive
we created keys for you and your private key is
(385, 1079)
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
385
1079
now we are creating a portable encrypted message 'your name' will be send to the server
signature verifed
vote successfully added
{'BJP': 0, 'Trump': 0, 'JSR': 1, 'NOTA': 0, 'trash': 0}
In [ ]: 1 2+3
to search
ENG
15:19
31-05-2020


1001010101
data
hash-
algorithm
hash
digital signing
(encryption)
digital signed
document
signatory
network
validator
hash-
algorithm
hash
1001010101
1001010101
a valid signature
needs two matching
hash values
digital signed
document
decryption
hash

SHA-256 hash function



Theorem: If c is collision-free, then SHA-256 is collision-free.

All practical assignments of network security course(CSE-537)

Name : Rohit Kumar Hansdah

Roll Number : 17024012

Date of submission : 31/5/2020

Branch : Biomedical Engg.

Subject : Network Security

General Instructions:

Coding language used:

Python language for all the codes . Anaconda navigator is used with python version 3.7

All the code is tested on Jupyter Notebook IDE having 6.0.1 version
Operating System:

All the python code runs on Windows 10 operating system of 64 bit with 8 GB RAM of the laptop/CPU.