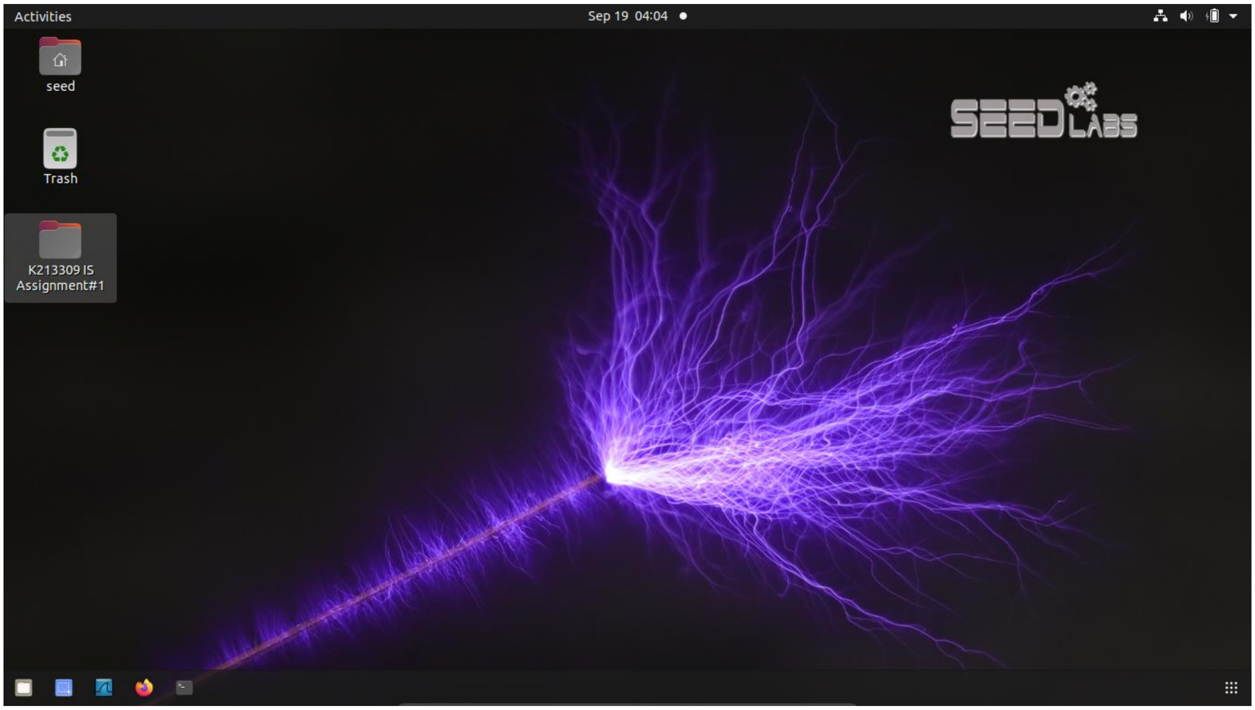
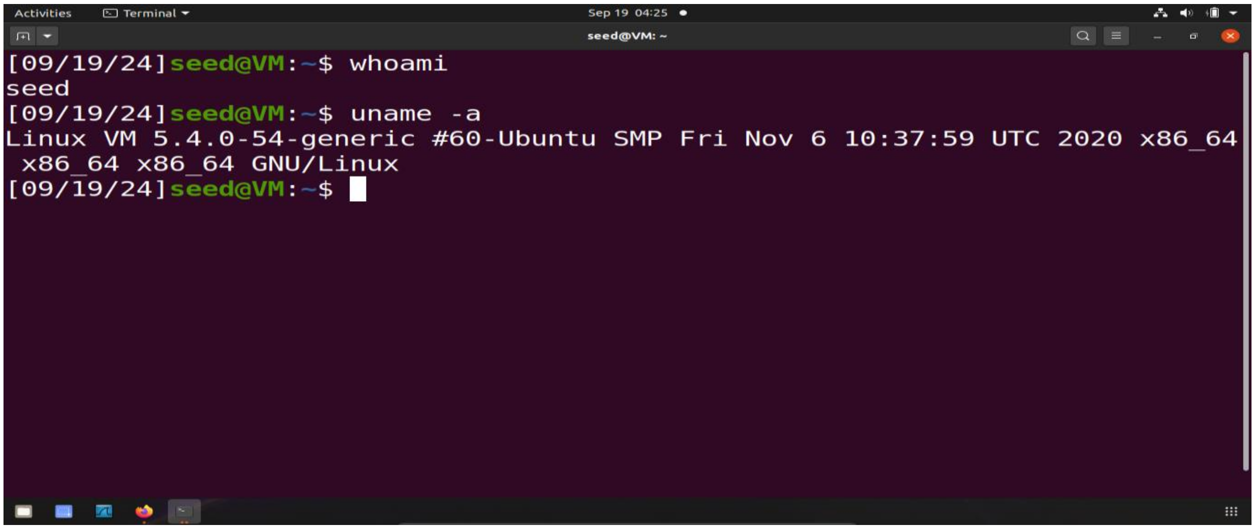
IS Assignment #1

K21-3309, BCS-7F

Setup

Since we need a dedicated VM for this lab, we first need to download the zip file provided by the SEED project which is a pre-built SEED Ubuntu 20.04 VirtualBox image. You can find this file here. After that, you can follow the steps here to properly setup your VM on virtual box. Once that is done, you will have a custom ubuntu for this lab.





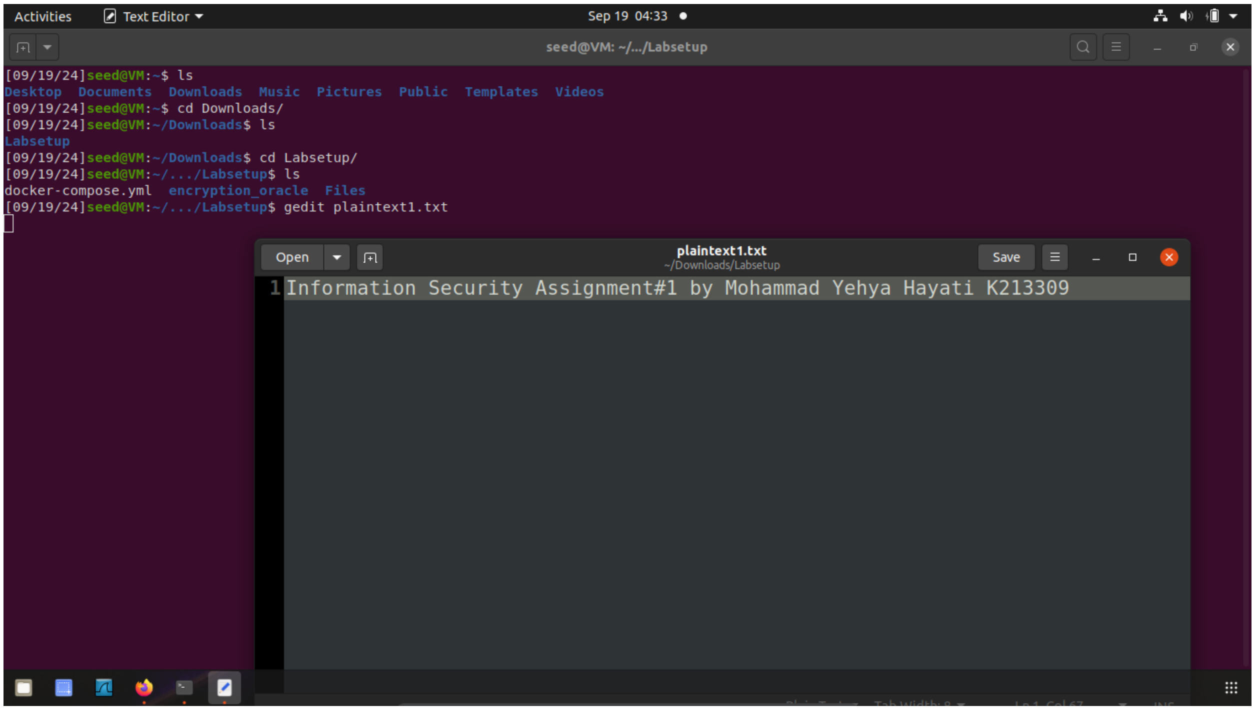
```
[09/19/24]seed@VM:~$ whoami
seed
[09/19/24]seed@VM:~$ uname -a
Linux VM 5.4.0-54-generic #60-Ubuntu SMP Fri Nov 6 10:37:59 UTC 2020 x86_64
 x86_64 x86_64 GNU/Linux
[09/19/24]seed@VM:~$ 
```

Next, we need to download the lab setup zip file to setup the environment for the lab and unzip it. That is the setup done.

Task #1

The first task is quite simple as all the commands and guidelines are given in the assignment document. Therefore we will follow the given commands.

First and foremost, we will create a plain text document.

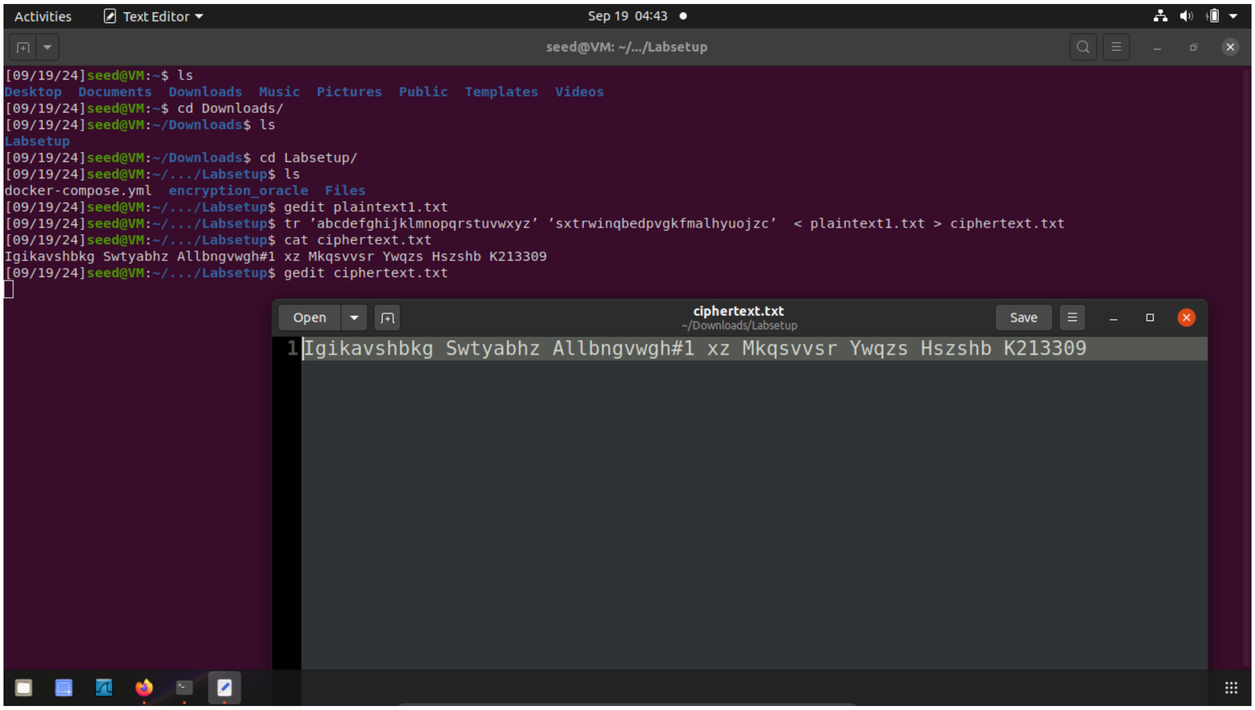


```
[09/19/24]seed@VM:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[09/19/24]seed@VM:~$ cd Downloads/
[09/19/24]seed@VM:~/Downloads$ ls
Labsetup
[09/19/24]seed@VM:~/Downloads$ cd Labsetup/
[09/19/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  encryption_oracle  Files
[09/19/24]seed@VM:~/.../Labsetup$ gedit plaintext1.txt
```

```
Information Security Assignment#1 by Mohammad Yehya Hayati K213309
```

Then, as given in the document, we will run the following command:

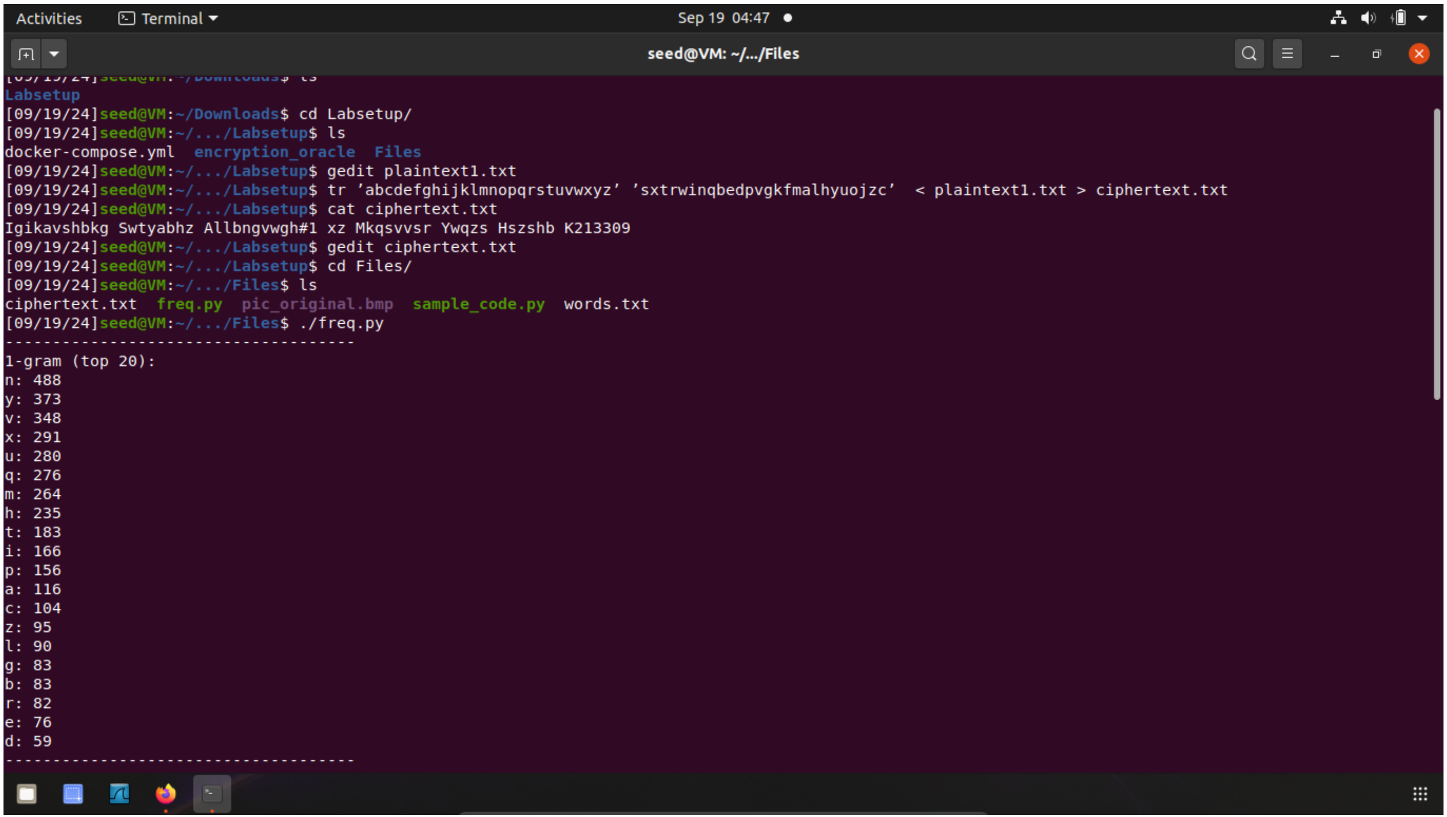
$ tr ’abcdefghijklmnopqrstuvwxyz’ ’sxtrwinqbedpvgkfmalhyuojzc’ < plaintext.txt > ciphertext.txt



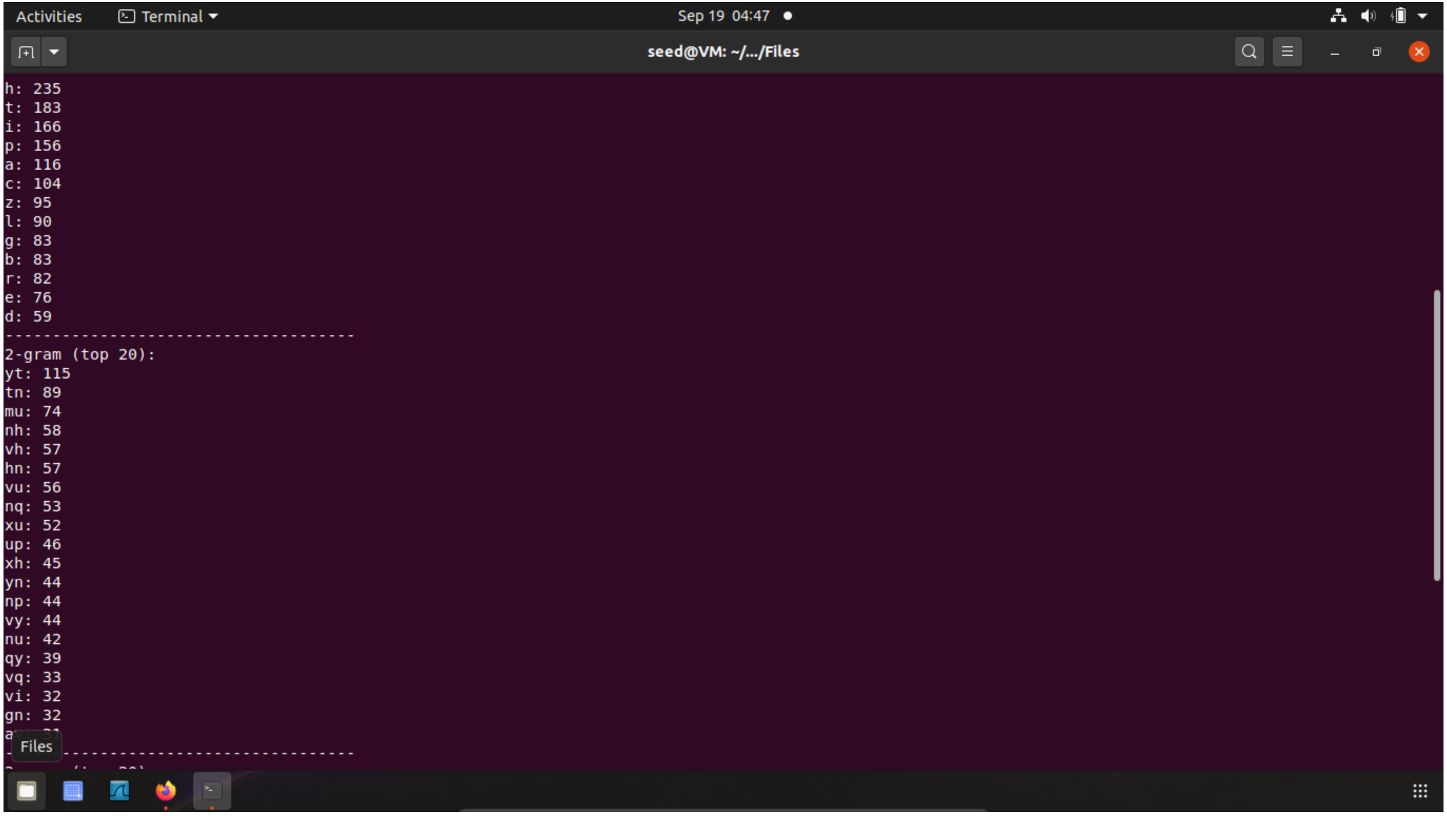
```
[09/19/24]seed@VM:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[09/19/24]seed@VM:~$ cd Downloads/
[09/19/24]seed@VM:~/Downloads$ ls
Labsetup
[09/19/24]seed@VM:~/Downloads$ cd Labsetup/
[09/19/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  encryption_oracle  Files
[09/19/24]seed@VM:~/.../Labsetup$ gedit plaintext1.txt
[09/19/24]seed@VM:~/.../Labsetup$ tr ’abcdefghijklmnopqrstuvwxyz’ ’sxtrwinqbedpvgkfmalhyuojzc’  < plaintext1.txt > ciphertext.txt
[09/19/24]seed@VM:~/.../Labsetup$ cat ciphertext.txt
Igikavshbkg Swtyabhz Allbngvwgh#1 xz Mkqsvvsr Ywqzs Hszshb K213309
[09/19/24]seed@VM:~/.../Labsetup$ gedit ciphertext.txt
```

```
Igikavshbkg Swtyabhz Allbngvwgh#1 xz Mkqsvvsr Ywqzs Hszshb K213309
```

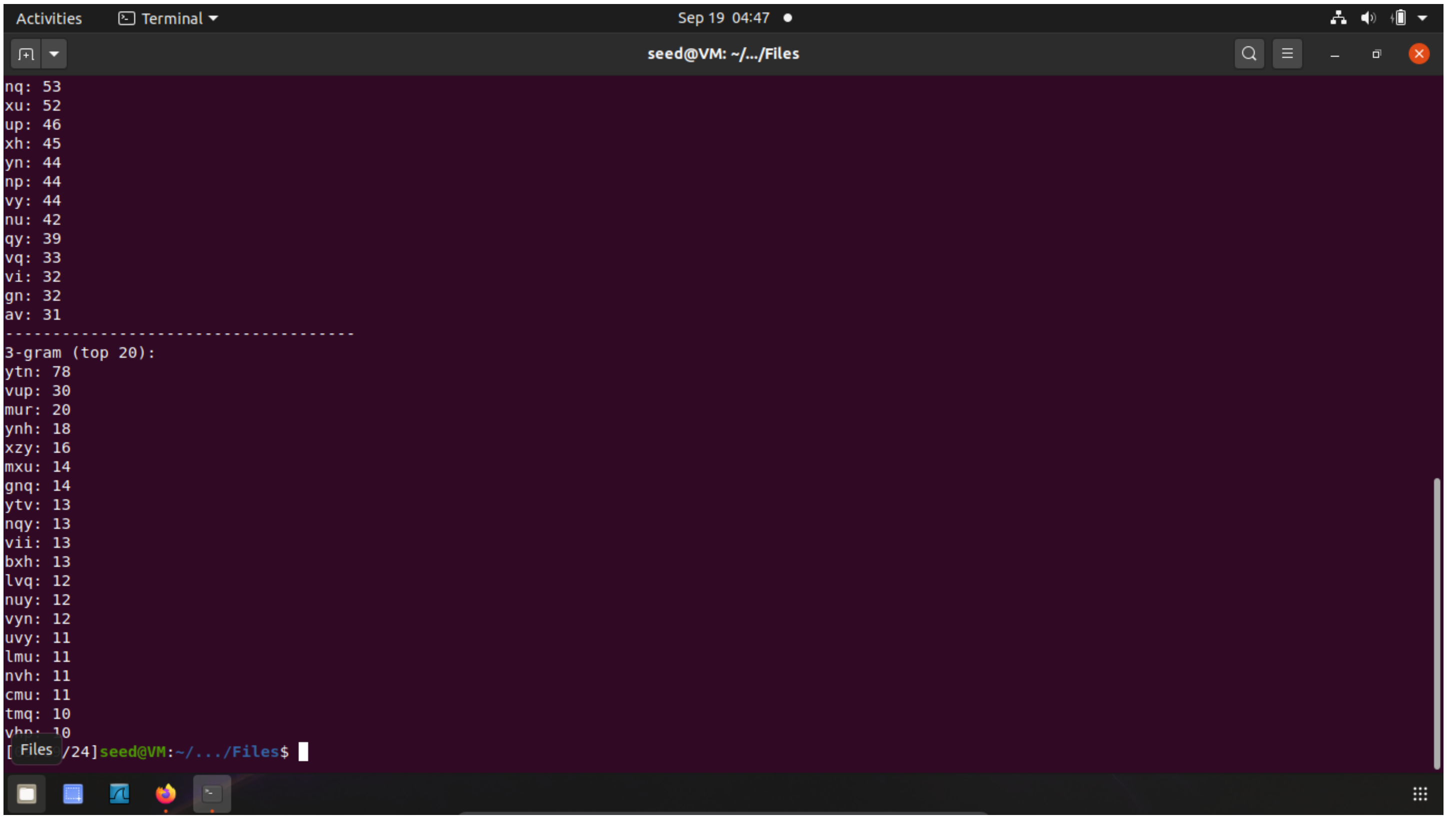
Next, we will use the freq.py file, which was given in the lap setup files, to generate the frequency of the words in the plain text. The frequency will be given in the form of all possible n-grams which is basically a ‘n-numbered’ word.



```
[09/19/24]seed@VM:~/Downloads$ ls
Labsetup
[09/19/24]seed@VM:~/Downloads$ cd Labsetup/
[09/19/24]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  encryption_oracle  Files
[09/19/24]seed@VM:~/.../Labsetup$ gedit plaintext1.txt
[09/19/24]seed@VM:~/.../Labsetup$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc'  < plaintext1.txt > ciphertext.txt
[09/19/24]seed@VM:~/.../Labsetup$ cat ciphertext.txt
Igikavshbkg Swtyabhz Allbngvwgh#1 xz Mkqsvvsr Ywqzs Hszshb K213309
[09/19/24]seed@VM:~/.../Labsetup$ gedit ciphertext.txt
[09/19/24]seed@VM:~/.../Labsetup$ cd Files/
[09/19/24]seed@VM:~/.../Files$ ls
ciphertext.txt  freq.py  pic_original.bmp  sample_code.py  words.txt
[09/19/24]seed@VM:~/.../Files$ ./freq.py
-------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
m: 264
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
-------------------------------------
```



```
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
-------------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
nh: 58
vh: 57
hn: 57
vu: 56
nq: 53
xu: 52
up: 46
xh: 45
yn: 44
np: 44
vy: 44
nu: 42
qy: 39
vq: 33
vi: 32
gn: 32
-------------------------------------
```



```
nq: 53
xu: 52
up: 46
xh: 45
yn: 44
np: 44
vy: 44
nu: 42
qy: 39
vq: 33
vi: 32
gn: 32
av: 31
-------------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10
[09/19/24]seed@VM:~/.../Files$
```

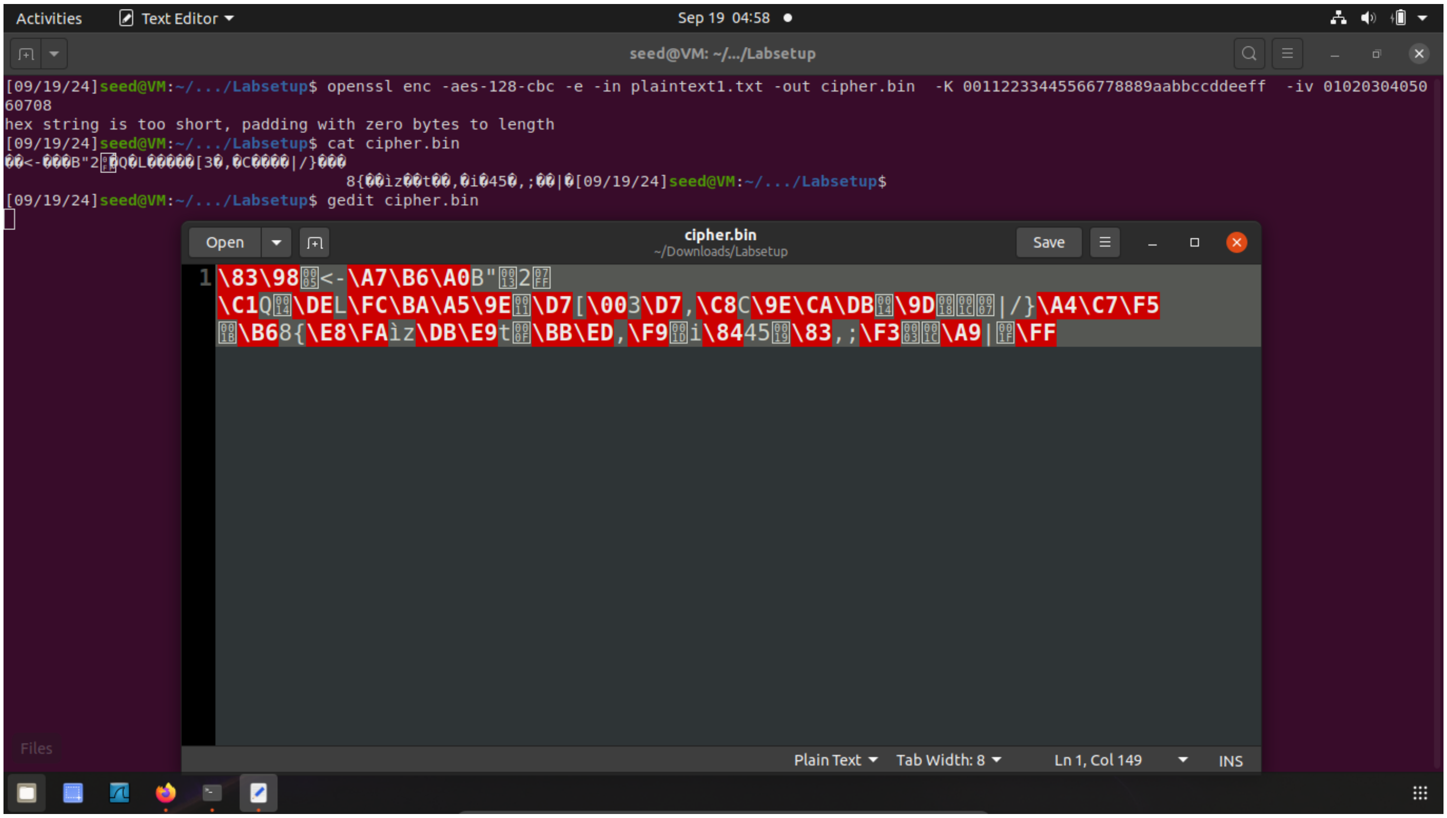
Task #2

For this task, we will again be using the resources in the lap setup file. This time, we will specifically be using the different encryption algorithms. The basic format of the command is:

$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

The first algorithm to try is -aes-128-cbc. So, the command with be:

$ openssl enc -aes-128-cbc -e -in plaintext1.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

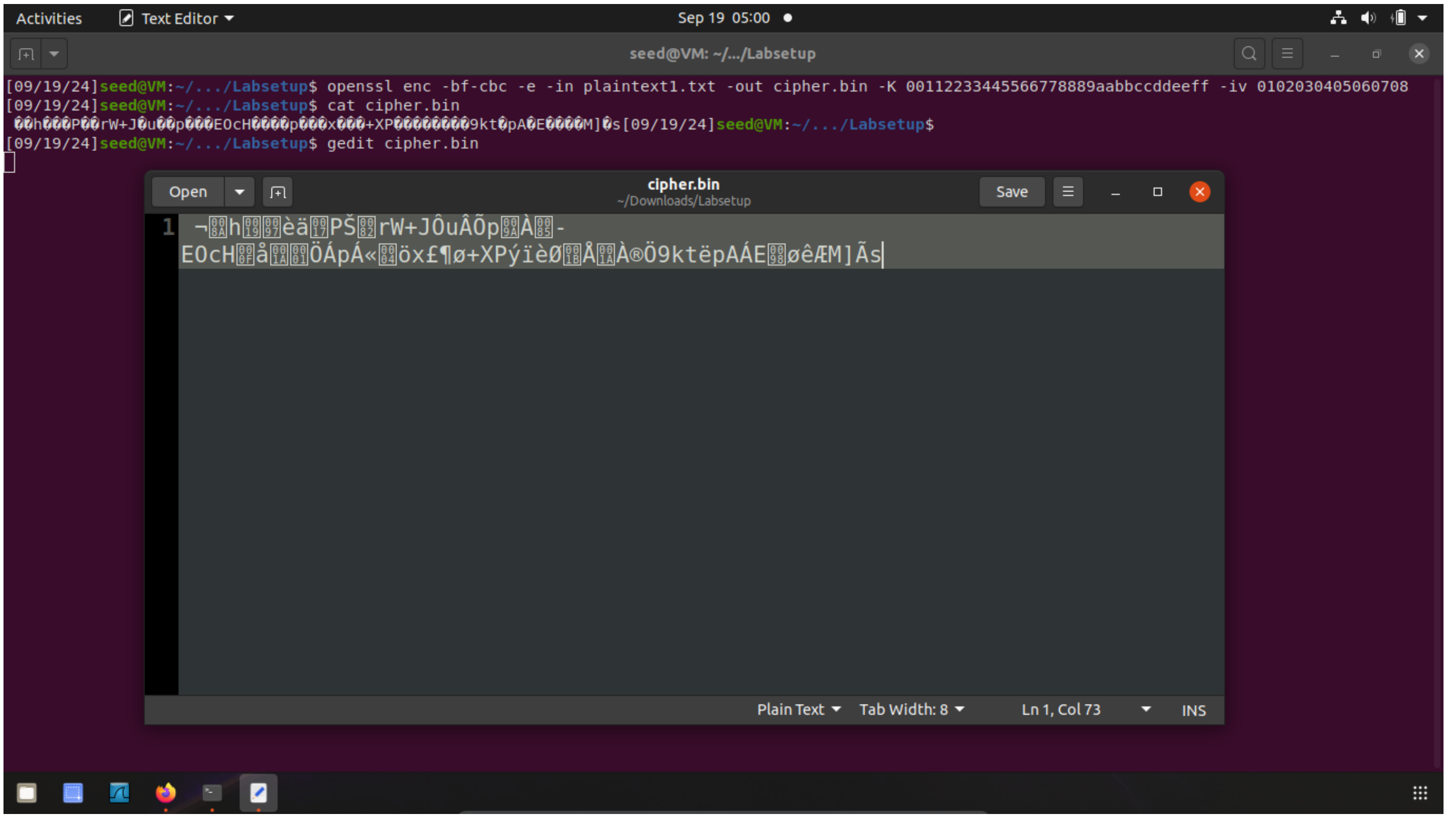


```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in plaintext1.txt -out cipher.bin  -K 00112233445566778889aabbccddeeff  -iv 01020304050
60708
hex string is too short, padding with zero bytes to length
[09/19/24]seed@VM:~/.../Labsetup$ cat cipher.bin
��<-���B"2��Q�L�����[3�,�C����|/}���
                                    8{��iz��t��,�i�45�,;��|�[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ gedit cipher.bin
```

```
\83\98�<-\A7\B6\A0B"�2�
\C1Q�\DEL\FC\BA\A5\9E�\D7[\003\D7,\C8C\9E\CA\DB�\9D���|/}\A4\C7\F5
�\B68{\E8\FAìz\DB\E9t�\BB\ED,\F9�i\8445�\83,;\F3��\A9|�\FF
```

Next we will try -bf-cbc. So the command will be:

$ openssl enc -bf-cbc -e -in plaintext1.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

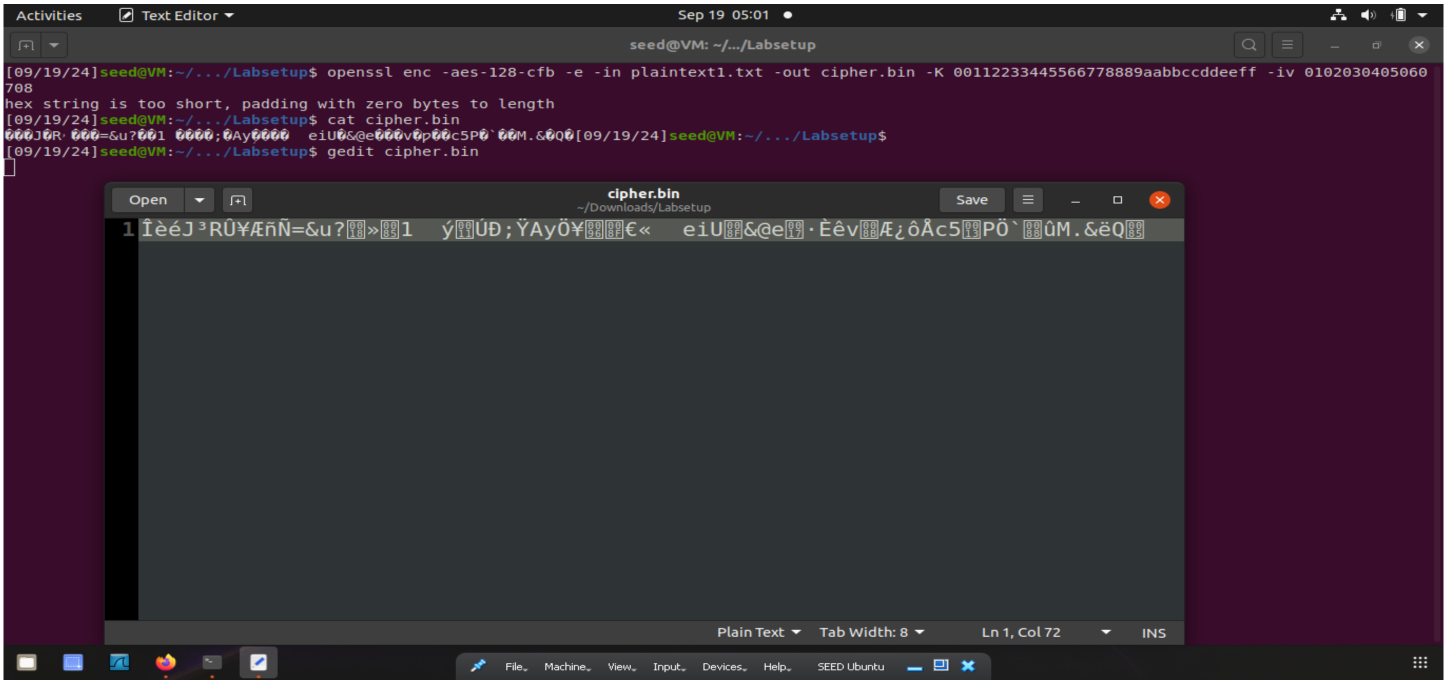


```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -bf-cbc -e -in plaintext1.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[09/19/24]seed@VM:~/.../Labsetup$ cat cipher.bin
 ��h���P��rW+J�u��p���EOcH����p���x���+XP���������9kt�pA�E����M]�s[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ gedit cipher.bin
```

```
1  ¬�h��èä�PŠ�rW+JÔuÂÕp�À�-
EOcH�å��ÖÁpÁ«�öx£¶ø+XPýïèØ�Å�À®Ö9ktëpAÁE�øêÆM]Ãs
```

Lastly, we will be using -aes-128-cfb. So the command will be:

$ openssl enc -aes-128-cfb -e -in plaintext1.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708



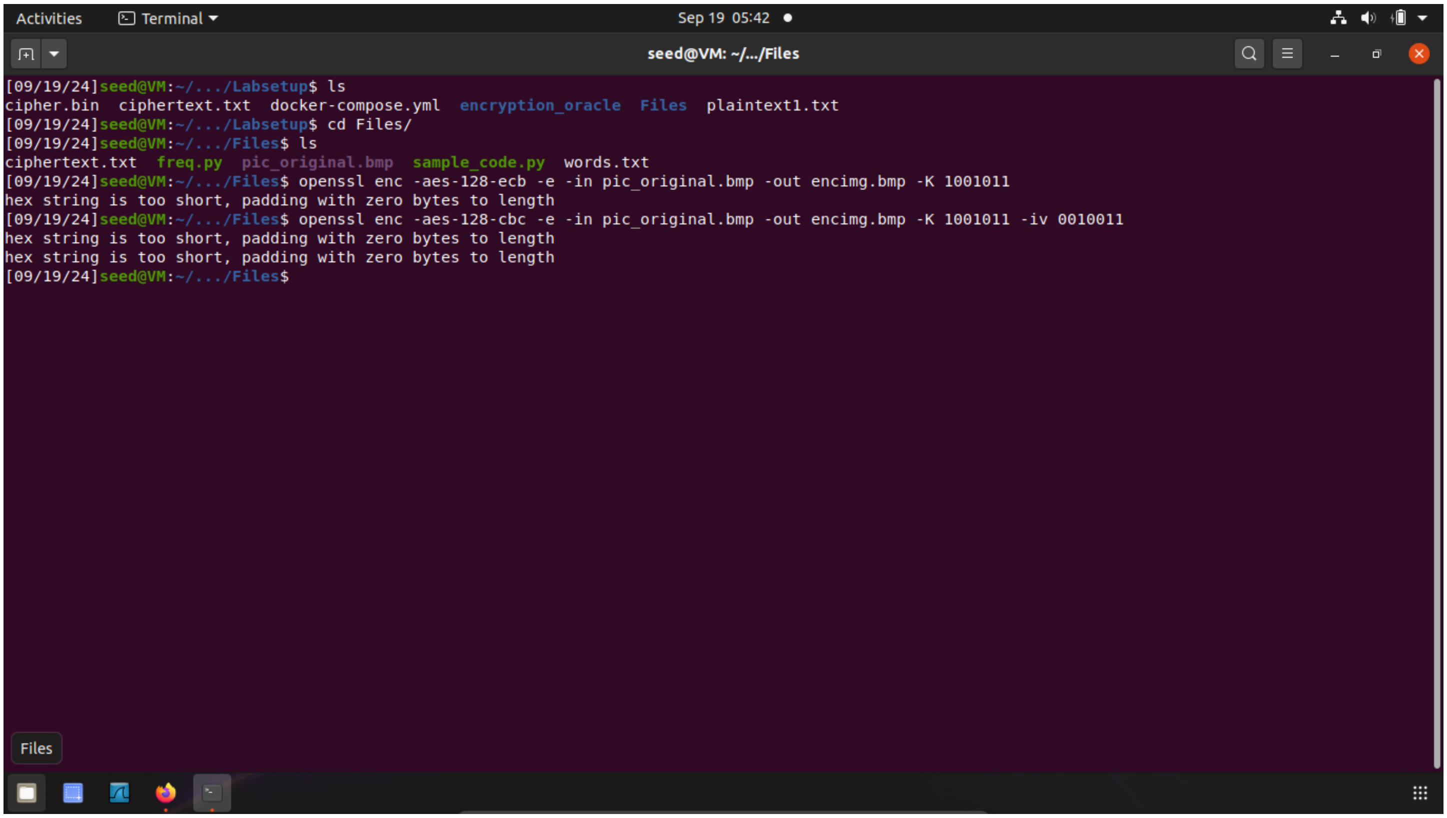
```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -e -in plaintext1.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060
708
hex string is too short, padding with zero bytes to length
[09/19/24]seed@VM:~/.../Labsetup$ cat cipher.bin
���J�R ���=&u?��1 ����;�Ay����  eiU�&@e���v�p��c5P�`��M.&�Q�[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ gedit cipher.bin
```

```
1 ÎèéJ³RÛ¥ÆñÑ=&u?�»�1   ý�ÚÐ;ŸAyÖ¥��€«   eiU�&@e�·Èêv�Æ¿ôÅc5�PÖ`�ûM.&ëQ�
```

Task #3

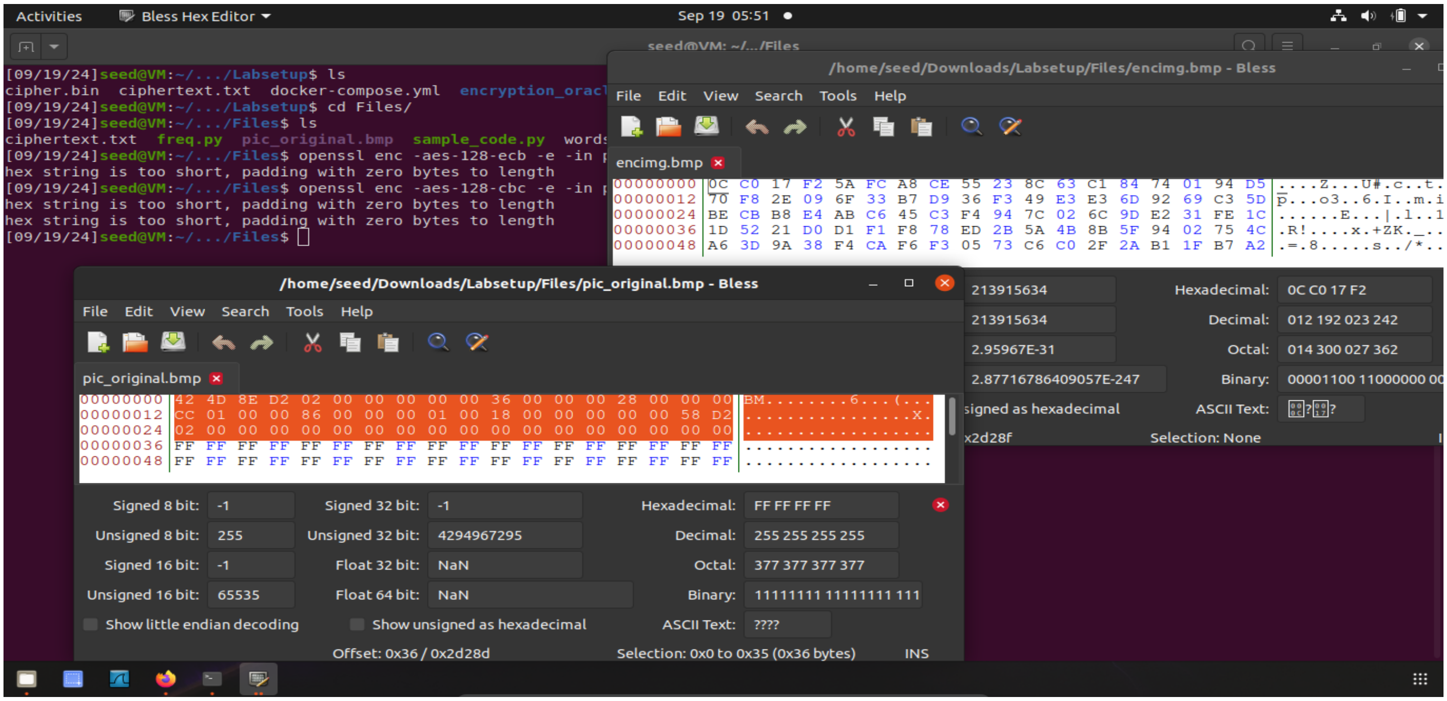
This task is similar to the previous, the only difference being that we will now be encrypting an image.

Firstly, we have to encrypt the image using ECB and CBC encryption modes. The command for this is:
$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out encimg.bmp -K 1001011 -iv 0010011
$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out encimg.bmp -K 1001011 -iv 0010011



```
[09/19/24]seed@VM:~/.../Labsetup$ ls
cipher.bin  ciphertext.txt  docker-compose.yml  encryption_oracle  Files  plaintext1.txt
[09/19/24]seed@VM:~/.../Labsetup$ cd Files/
[09/19/24]seed@VM:~/.../Files$ ls
ciphertext.txt  freq.py  pic_original.bmp  sample_code.py  words.txt
[09/19/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out encimg.bmp -K 1001011
hex string is too short, padding with zero bytes to length
[09/19/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out encimg.bmp -K 1001011 -iv 0010011
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
[09/19/24]seed@VM:~/.../Files$
```

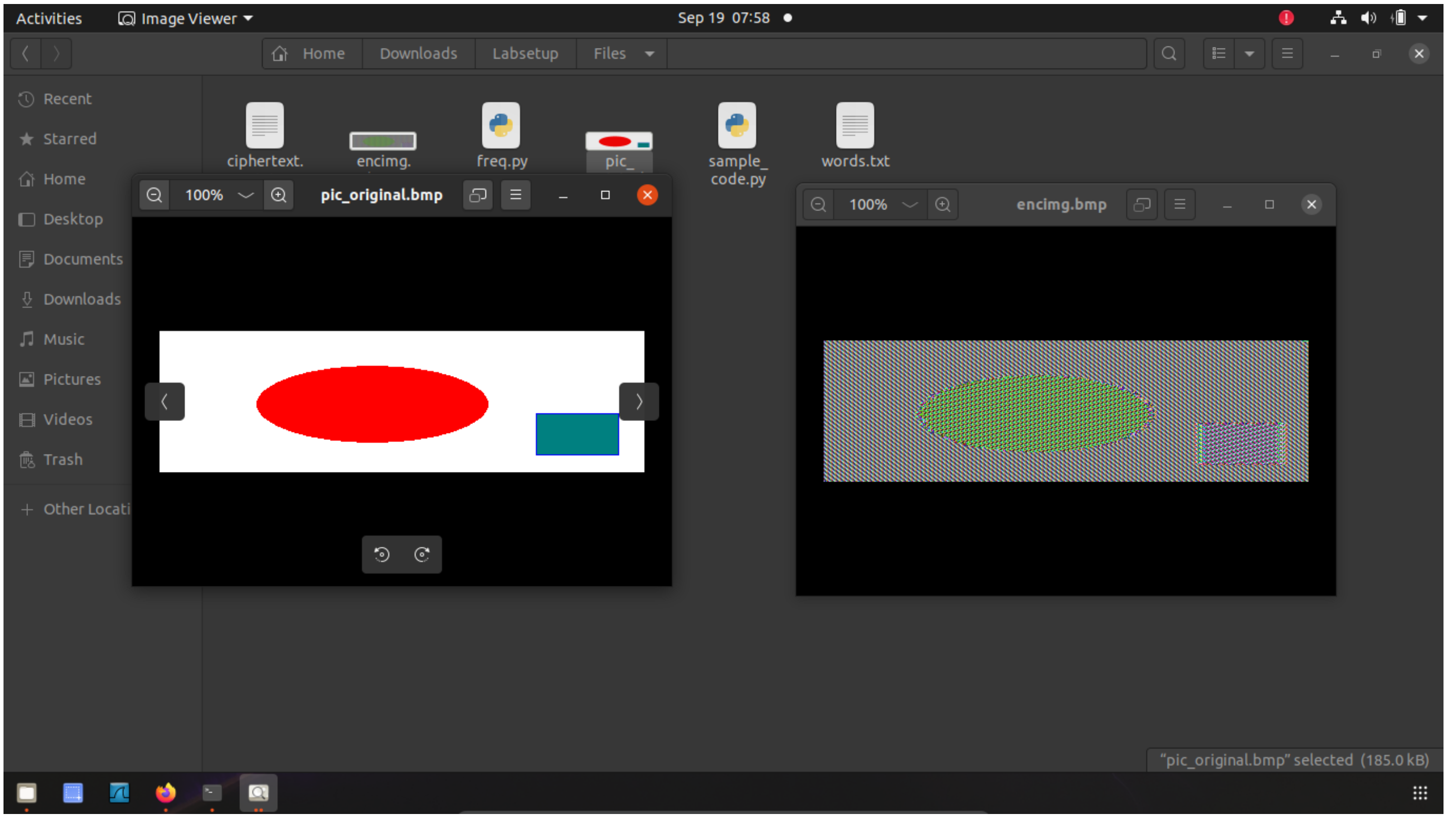
Now to retain the property of a bitmap (.bmp) image, we need the original 54 bytes of the image in the encrypted to actually view the encrypted image. For this we will use 'bless' (as mentioned in the assignment document) to edit the first 54 bytes of the encrypted image.



```
/home/seed/Downloads/Labsetup/Files/encimg.bmp - Bless

00000000 0C C0 17 F2 5A FC A8 CE 55 23 8C 63 C1 84 74 01 94 D5 ....Z...U#.c..t..
00000012 70 F8 2E 09 6F 33 B7 D9 36 F3 49 E3 E3 6D 92 69 C3 5D p...o3..6.I..m.i
00000024 BE CB B8 E4 AB C6 45 C3 F4 94 7C 02 6C 9D E2 31 FE 1C ......E...|.l..1
00000036 1D 52 21 D0 D1 F1 F8 78 ED 2B 5A 4B 8B 5F 94 02 75 4C .R!....x.+ZK._..
00000048 A6 3D 9A 38 F4 CA F6 F3 05 73 C6 C0 2F 2A B1 1F B7 A2 .=.8.....s../*..

213915634        Hexadecimal: 0C C0 17 F2
213915634        Decimal: 012 192 023 242
2.95967E-31      Octal: 014 300 027 362
2.87716786409057E-247   Binary: 00001100 11000000 00
signed as hexadecimal   ASCII Text:
x2d28f           Selection: None
```

```
/home/seed/Downloads/Labsetup/Files/pic_original.bmp - Bless

00000000 42 4D 8E D2 02 00 00 00 00 00 36 00 00 00 28 00 00 00 BM........6...(...
00000012 CC 01 00 00 86 00 00 00 01 00 18 00 00 00 00 00 58 D2 ................X.
00000024 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ..................
00000048 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ..................

Signed 8 bit: -1        Signed 32 bit: -1           Hexadecimal: FF FF FF FF
Unsigned 8 bit: 255     Unsigned 32 bit: 4294967295 Decimal: 255 255 255 255
Signed 16 bit: -1       Float 32 bit: NaN           Octal: 377 377 377 377
Unsigned 16 bit: 65535  Float 64 bit: NaN           Binary: 11111111 11111111 111
Show little endian decoding   Show unsigned as hexadecimal   ASCII Text: ????
Offset: 0x36 / 0x2d28d        Selection: 0x0 to 0x35 (0x36 bytes)   INS
```

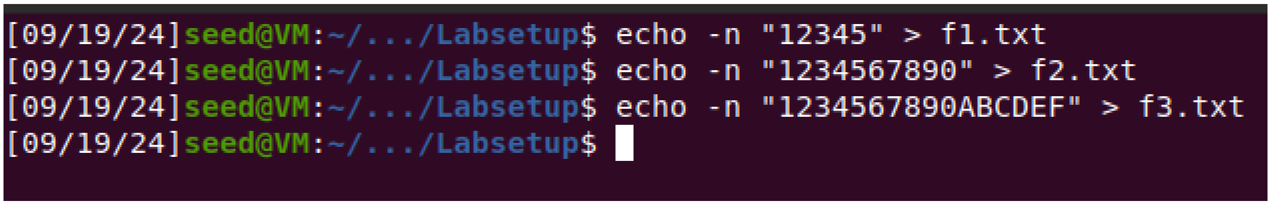
Now we can see the difference between the original image and the encrypted image.



Task #4

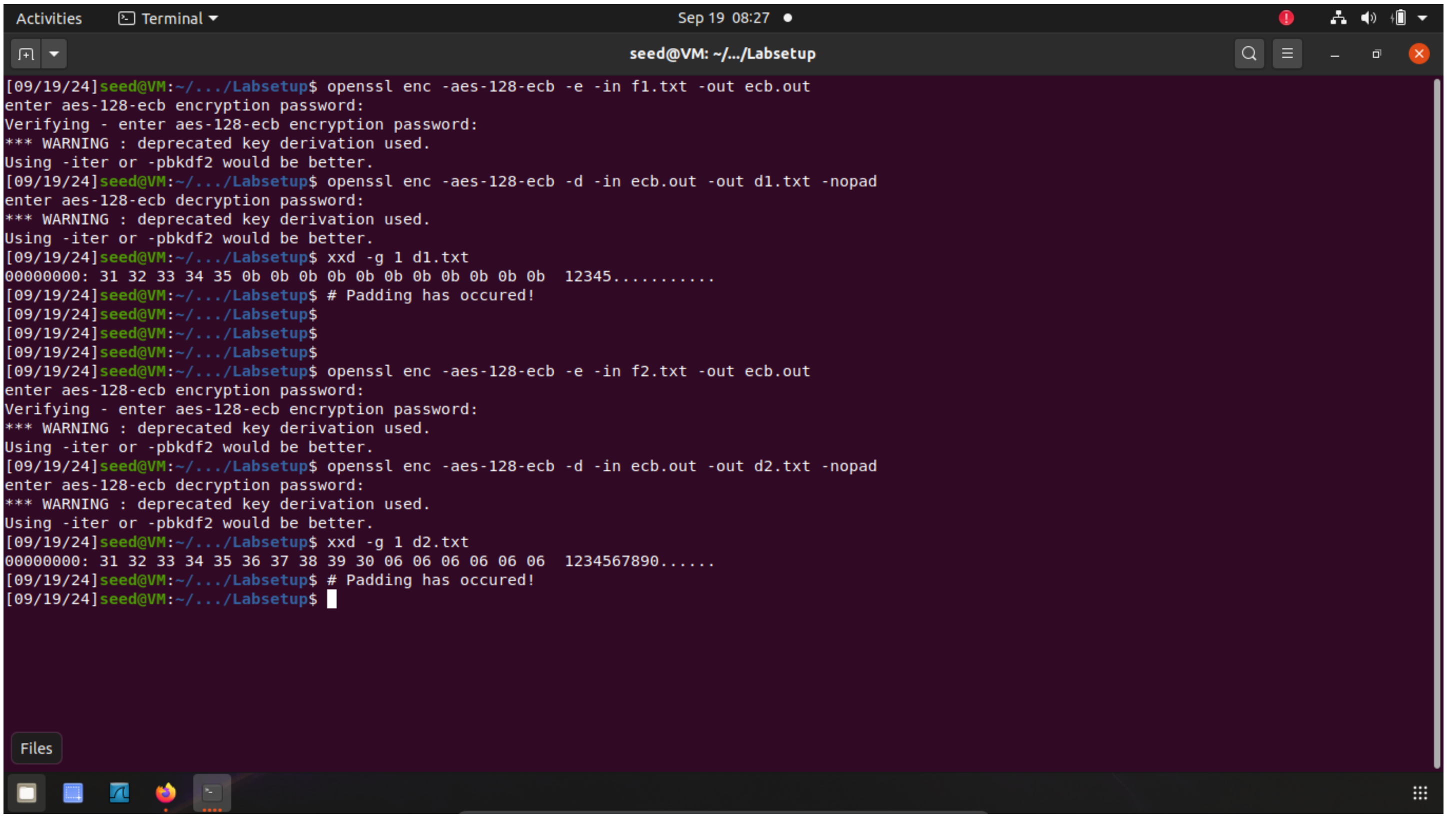
This task explores how different encryption algorithms use the concept of padding.

Firstly, we will generate the three files as mentioned in the document. Then, we will encrypt the files using the four given techniques: ECB, CBC, CFB, and OFB. After using each technique, we will check whether they use padding or not.



```
[09/19/24]seed@VM:~/.../Labsetup$ echo -n "12345" > f1.txt
[09/19/24]seed@VM:~/.../Labsetup$ echo -n "1234567890" > f2.txt
[09/19/24]seed@VM:~/.../Labsetup$ echo -n "1234567890ABCDEF" > f3.txt
[09/19/24]seed@VM:~/.../Labsetup$ 
```

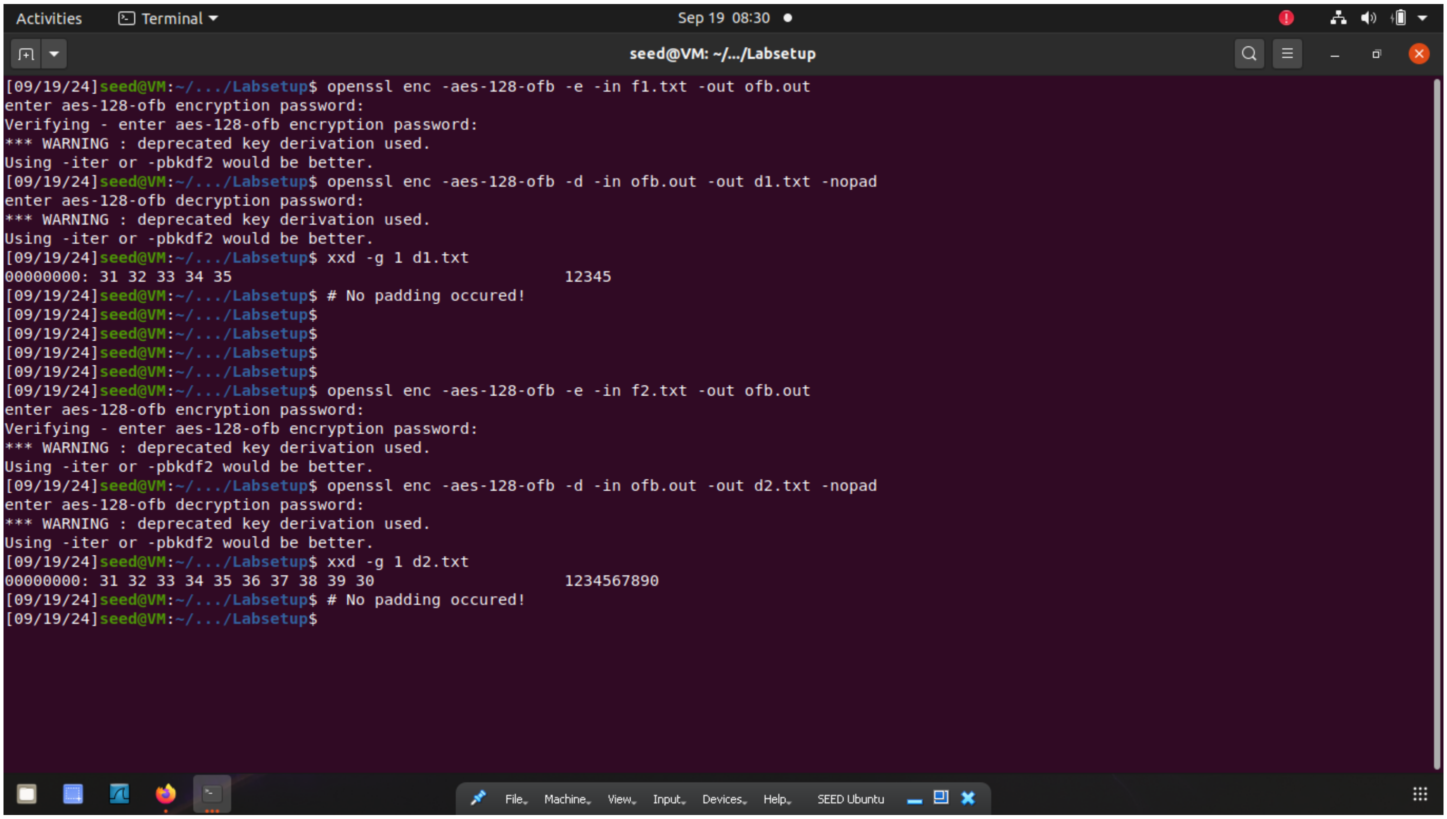
Here are the commands for ECB:



```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -e -in f1.txt -out ecb.out
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -d -in ecb.out -out d1.txt -nopad
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d1.txt
00000000: 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b  12345...........
[09/19/24]seed@VM:~/.../Labsetup$ # Padding has occured!
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -e -in f2.txt -out ecb.out
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -d -in ecb.out -out d2.txt -nopad
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d2.txt
00000000: 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06  1234567890......
[09/19/24]seed@VM:~/.../Labsetup$ # Padding has occured!
[09/19/24]seed@VM:~/.../Labsetup$ 
```

As you can see, ECB uses padding

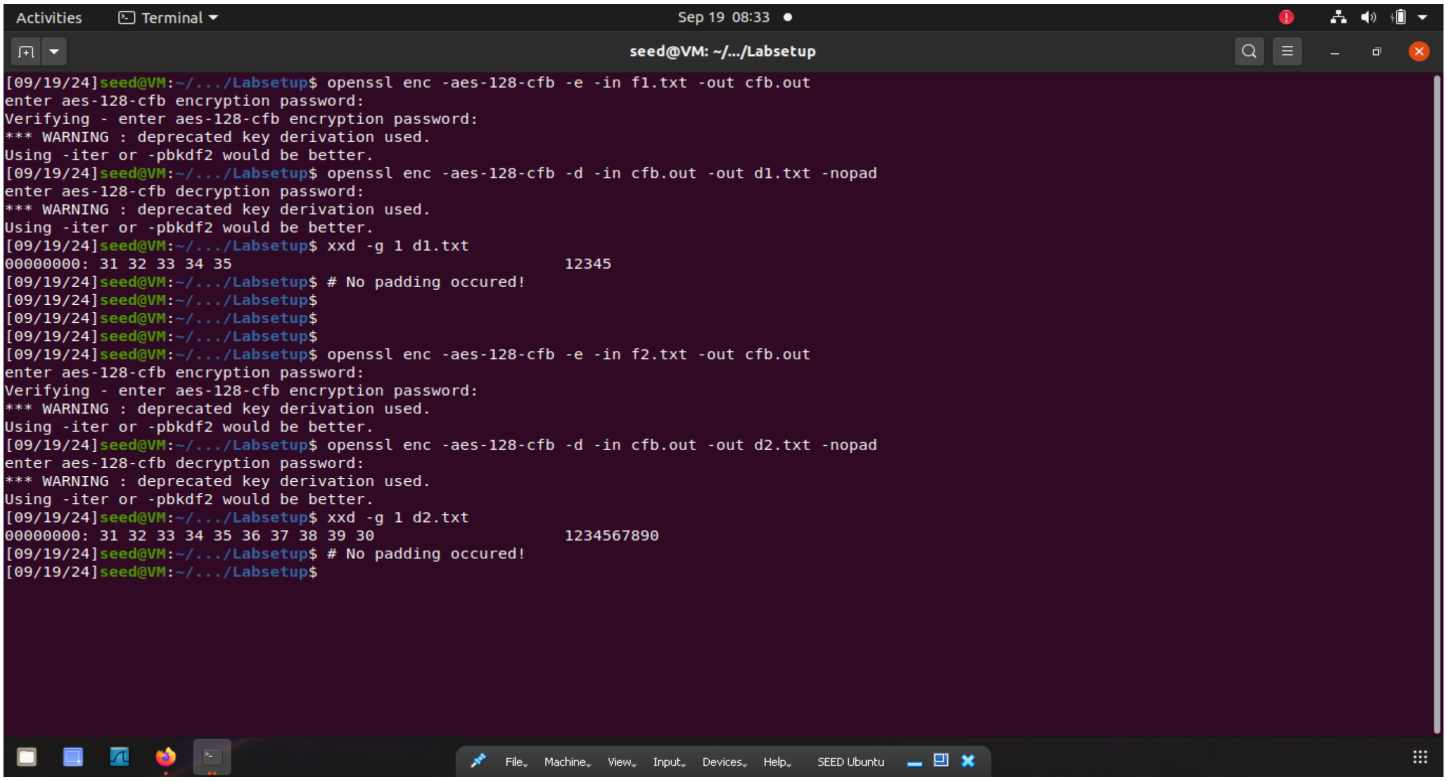
Here are the commands for OFB:



```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -e -in f1.txt -out ofb.out
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -d -in ofb.out -out d1.txt -nopad
enter aes-128-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d1.txt
00000000: 31 32 33 34 35                                   12345
[09/19/24]seed@VM:~/.../Labsetup$ # No padding occured!
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -e -in f2.txt -out ofb.out
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -d -in ofb.out -out d2.txt -nopad
enter aes-128-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d2.txt
00000000: 31 32 33 34 35 36 37 38 39 30                    1234567890
[09/19/24]seed@VM:~/.../Labsetup$ # No padding occured!
[09/19/24]seed@VM:~/.../Labsetup$
```

As you can see OFB does not use padding.

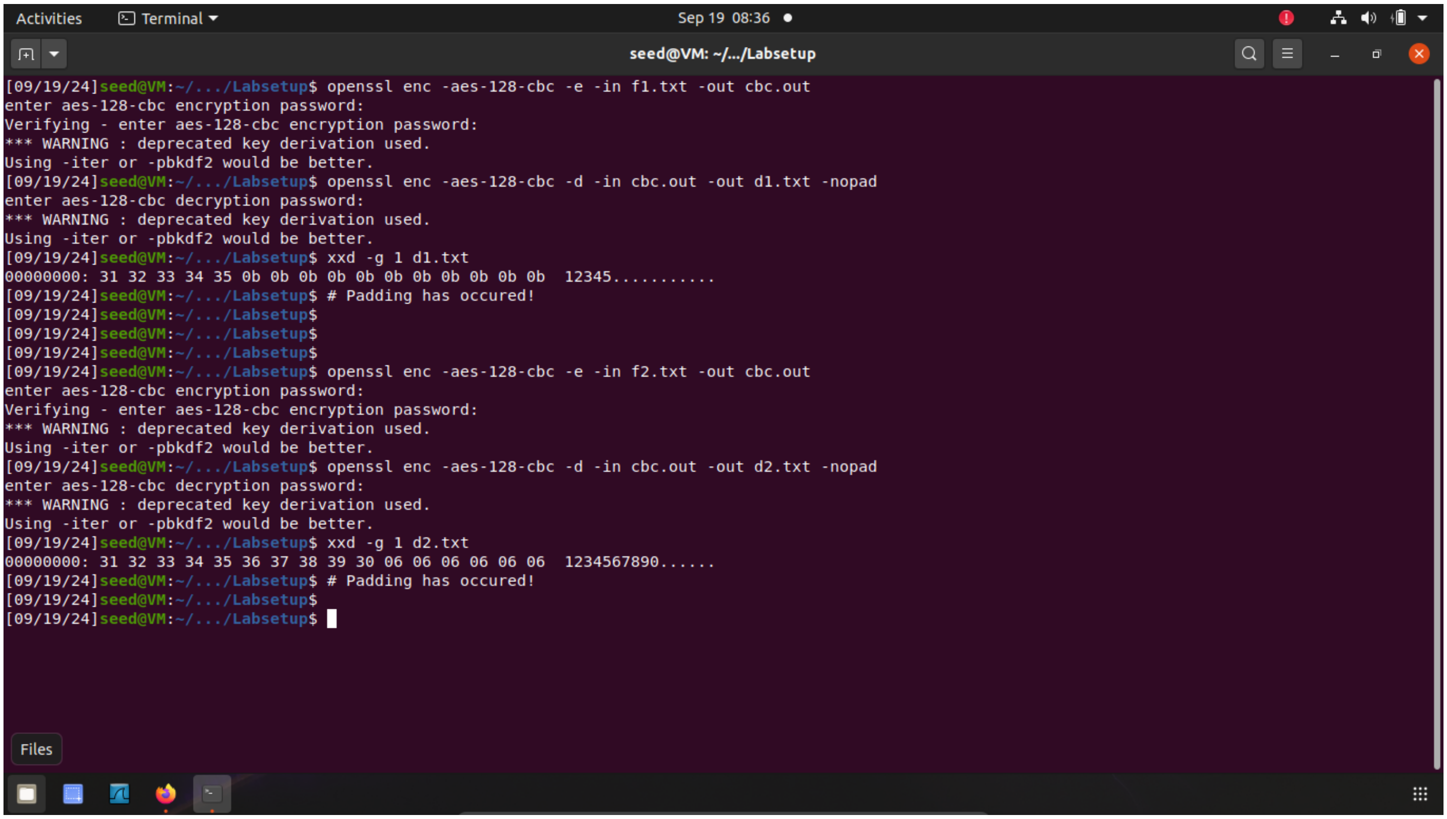
Here are the commands for CFB



```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -e -in f1.txt -out cfb.out
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -d -in cfb.out -out d1.txt -nopad
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d1.txt
00000000: 31 32 33 34 35                                   12345
[09/19/24]seed@VM:~/.../Labsetup$ # No padding occured!
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -e -in f2.txt -out cfb.out
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -d -in cfb.out -out d2.txt -nopad
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d2.txt
00000000: 31 32 33 34 35 36 37 38 39 30                    1234567890
[09/19/24]seed@VM:~/.../Labsetup$ # No padding occured!
[09/19/24]seed@VM:~/.../Labsetup$
```

As you can see CFB does not use padding.

Here are the commands for CBC



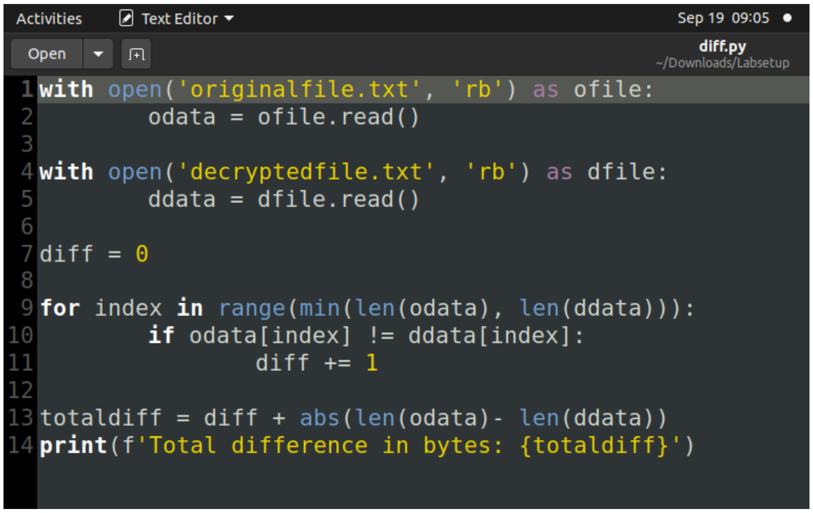
```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in f1.txt -out cbc.out
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -in cbc.out -out d1.txt -nopad
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d1.txt
00000000: 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b  12345...........
[09/19/24]seed@VM:~/.../Labsetup$ # Padding has occured!
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in f2.txt -out cbc.out
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -in cbc.out -out d2.txt -nopad
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/19/24]seed@VM:~/.../Labsetup$ xxd -g 1 d2.txt
00000000: 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06  1234567890......
[09/19/24]seed@VM:~/.../Labsetup$ # Padding has occured!
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$
```

As you can see CBC uses padding

Task #5

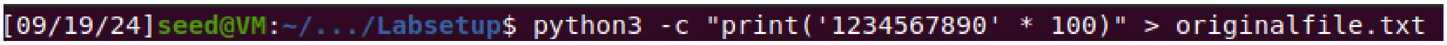
The aim of this task is to see the difference between original and decrypted texts if the encrypted text in between was corrupted. Each mode will behave differently since they use different techniques. OFB will only lose 1 byte as the keystream is generated independently of the cipher text. CFB will lose the equivalent of two chunks which depends on the block size, typically 16-32 bytes. ECB will lose 16 bytes because blocks are encrypted independently. CBC will lose 32 bytes due to the chaining of blocks.

Firstly, we need a function to find the difference in bytes between the two files:



```
with open('originalfile.txt', 'rb') as ofile:
        odata = ofile.read()

with open('decryptedfile.txt', 'rb') as dfile:
        ddata = dfile.read()

diff = 0

for index in range(min(len(odata), len(ddata))):
        if odata[index] != ddata[index]:
                diff += 1

totaldiff = diff + abs(len(odata)- len(ddata))
print(f'Total difference in bytes: {totaldiff}')
```

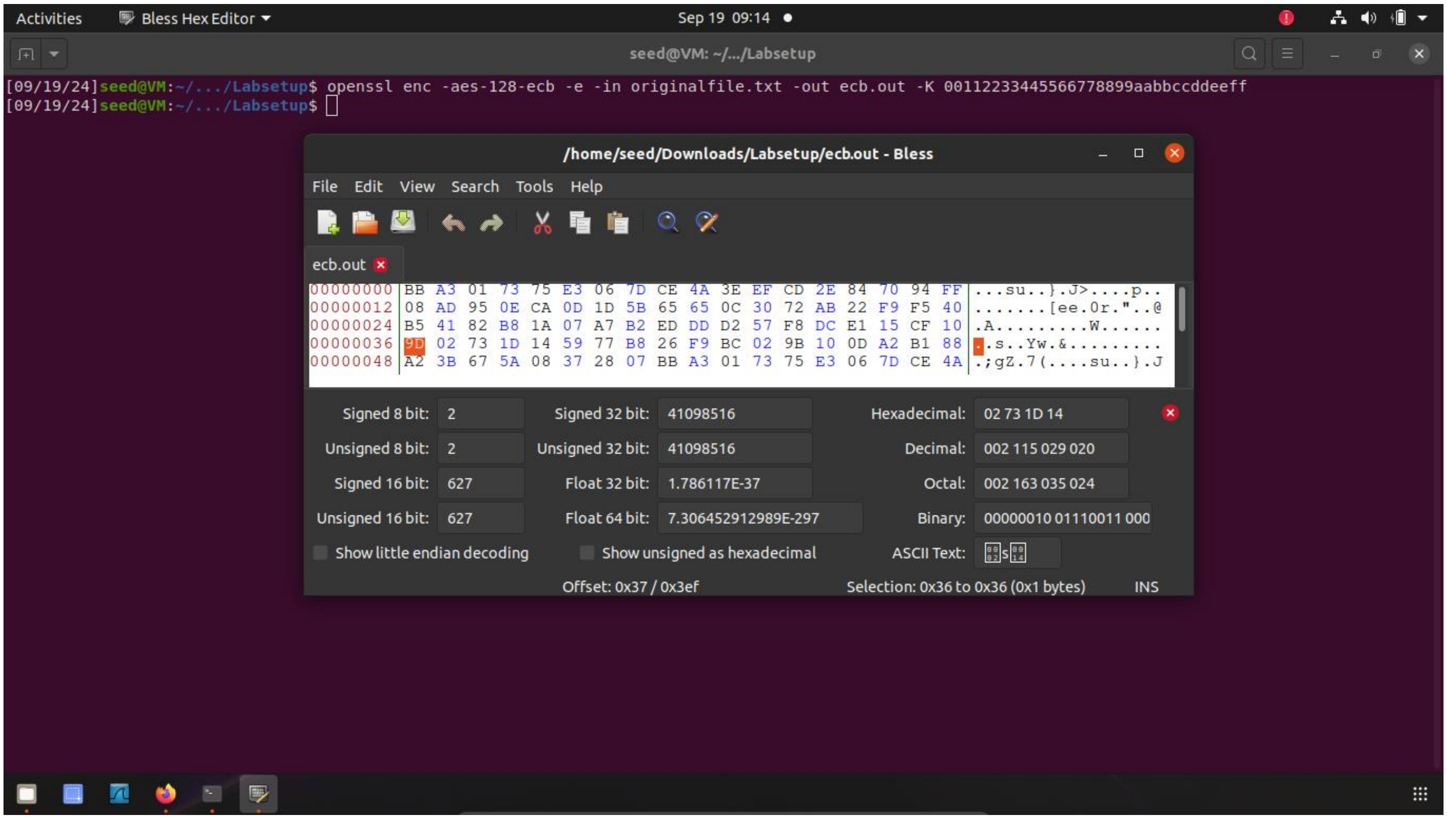
Also, we need a file of 1000 characters, so here is the command:



```
[09/19/24]seed@VM:~/.../Labsetup$ python3 -c "print('1234567890' * 100)" > originalfile.txt
```

Now we can perform the experiments. We need to first encrypt a file, and then purposely corrupt it by changing the 55th byte. We then decrypt the corrupted file and see the difference in bytes. Each mode will have a different result due to using different techniques. We will then compare the answers I gave above of each mode will the actual result.

For ECB



```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -e -in originalfile.txt -out ecb.out -K 00112233445566778899aabbccddeeff
[09/19/24]seed@VM:~/.../Labsetup$ 
```

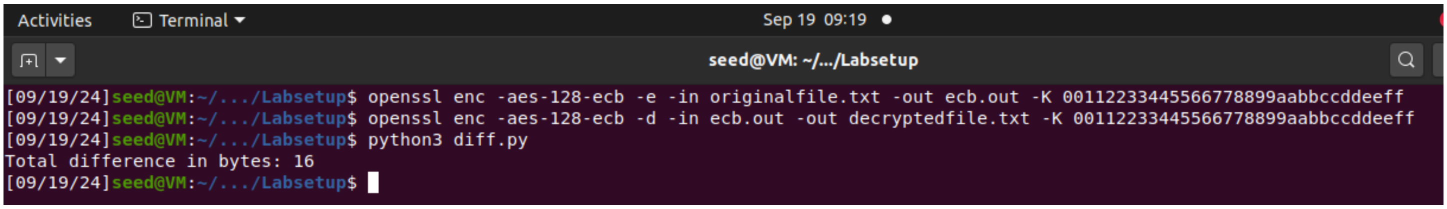
/home/seed/Downloads/Labsetup/ecb.out - Bless

```
00000000 BB A3 01 73 75 E3 06 7D CE 4A 3E EF CD 2E 84 70 94 FF ...su..}.J>....p..
00000012 08 AD 95 0E CA 0D 1D 5B 65 65 0C 30 72 AB 22 F9 F5 40 .......[ee.0r."..@
00000024 B5 41 82 B8 1A 07 A7 B2 ED DD D2 57 F8 DC E1 15 CF 10 .A.........W......
00000036 9D 02 73 1D 14 59 77 B8 26 F9 BC 02 9B 10 0D A2 B1 88 ..s..Yw.&.........
00000048 A2 3B 67 5A 08 37 28 07 BB A3 01 73 75 E3 06 7D CE 4A .;gZ.7(....su..}.J
```

Signed 8 bit: 2 | Signed 32 bit: 41098516 | Hexadecimal: 02 73 1D 14
Unsigned 8 bit: 2 | Unsigned 32 bit: 41098516 | Decimal: 002 115 029 020
Signed 16 bit: 627 | Float 32 bit: 1.786117E-37 | Octal: 002 163 035 024
Unsigned 16 bit: 627 | Float 64 bit: 7.306452912989E-297 | Binary: 00000010 01110011 000
□ Show little endian decoding | □ Show unsigned as hexadecimal | ASCII Text: s

Offset: 0x37 / 0x3ef | Selection: 0x36 to 0x36 (0x1 bytes) | INS

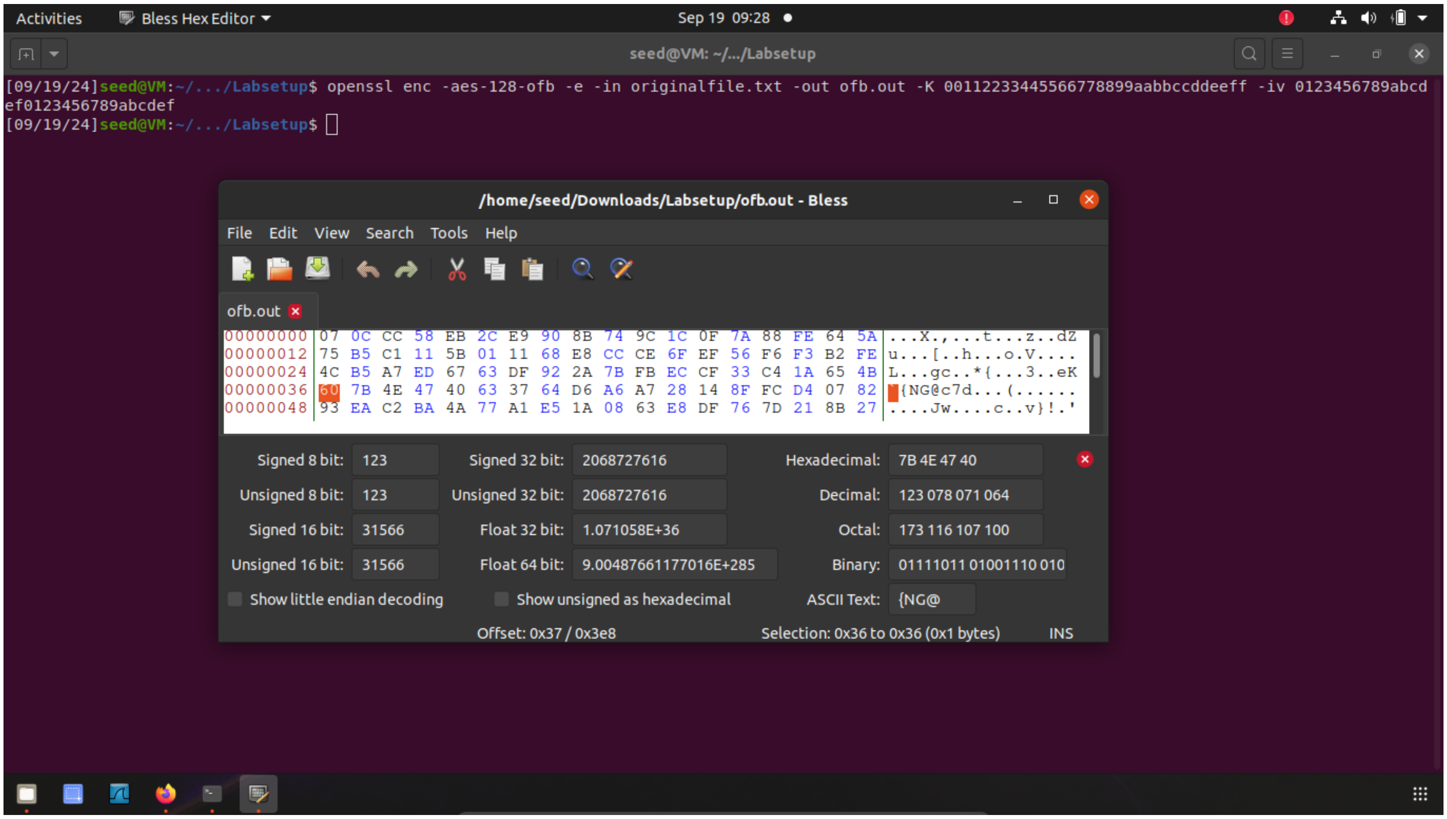
Let’s change to 00, and then decrypt to see the difference in bytes.



```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -e -in originalfile.txt -out ecb.out -K 00112233445566778899aabbccddeeff
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ecb -d -in ecb.out -out decryptedfile.txt -K 00112233445566778899aabbccddeeff
[09/19/24]seed@VM:~/.../Labsetup$ python3 diff.py
Total difference in bytes: 16
[09/19/24]seed@VM:~/.../Labsetup$ 
```

As you can see the difference is 16 bytes for ECB.

For OFB



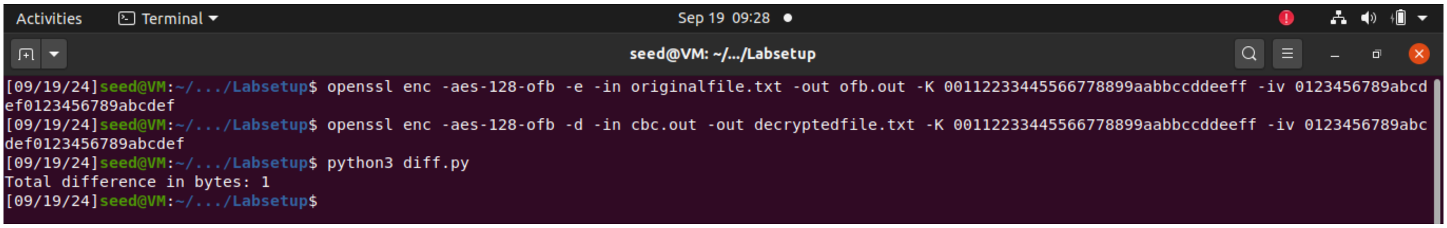
```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -e -in originalfile.txt -out ofb.out -K 00112233445566778899aabbccddeeff -iv 0123456789abcd
ef0123456789abcdef
[09/19/24]seed@VM:~/.../Labsetup$ 
```

/home/seed/Downloads/Labsetup/ofb.out - Bless

```
00000000 07 0C CC 58 EB 2C E9 90 8B 74 9C 1C 0F 7A 88 FE 64 5A ...X.,...t...z..dZ
00000012 75 B5 C1 11 5B 01 11 68 E8 CC CE 6F EF 56 F6 F3 B2 FE u...[..h...o.V....
00000024 4C B5 A7 ED 67 63 DF 92 2A 7B FB EC CF 33 C4 1A 65 4B L...gc..*{...3..eK
00000036 60 7B 4E 47 40 63 37 64 D6 A6 A7 28 14 8F FC D4 07 82 `{NG@c7d...(......
00000048 93 EA C2 BA 4A 77 A1 E5 1A 08 63 E8 DF 76 7D 21 8B 27 ....Jw....c..v}!.'
```

Signed 8 bit: 123
Signed 32 bit: 2068727616
Hexadecimal: 7B 4E 47 40
Unsigned 8 bit: 123
Unsigned 32 bit: 2068727616
Decimal: 123 078 071 064
Signed 16 bit: 31566
Float 32 bit: 1.071058E+36
Octal: 173 116 107 100
Unsigned 16 bit: 31566
Float 64 bit: 9.00487661177016E+285
Binary: 01111011 01001110 010
□ Show little endian decoding
□ Show unsigned as hexadecimal
ASCII Text: {NG@
Offset: 0x37 / 0x3e8
Selection: 0x36 to 0x36 (0x1 bytes)
INS

Let’s change to 00, and then decrypt to see the difference in bytes.



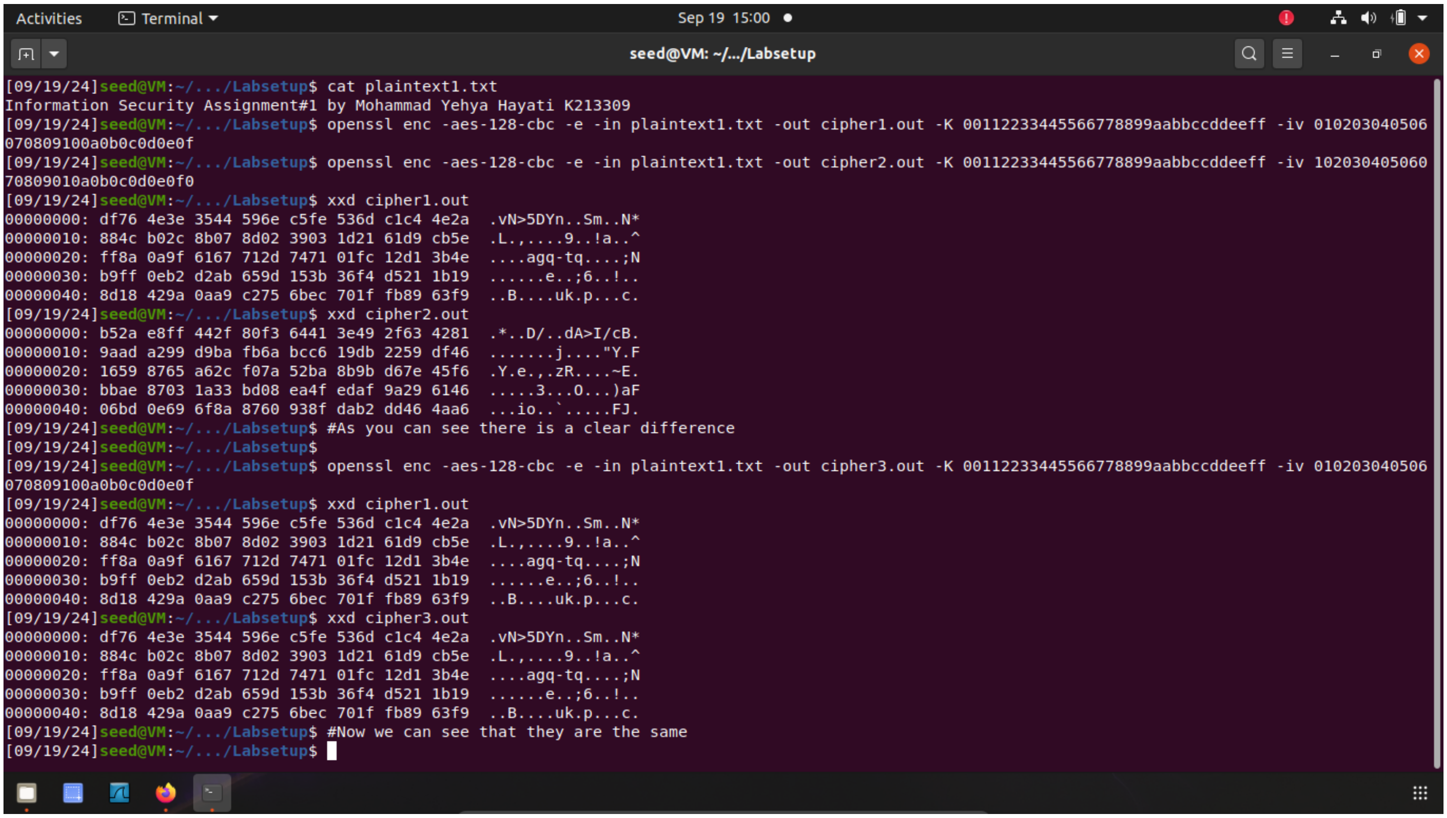
```
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -e -in originalfile.txt -out ofb.out -K 00112233445566778899aabbccddeeff -iv 0123456789abcd
ef0123456789abcdef
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-ofb -d -in cbc.out -out decryptedfile.txt -K 00112233445566778899aabbccddeeff -iv 0123456789abc
def0123456789abcdef
[09/19/24]seed@VM:~/.../Labsetup$ python3 diff.py
Total difference in bytes: 1
[09/19/24]seed@VM:~/.../Labsetup$
```

As you can see the difference is 1 byte for OFB.

Similar results can be seen for CFB and CBC.

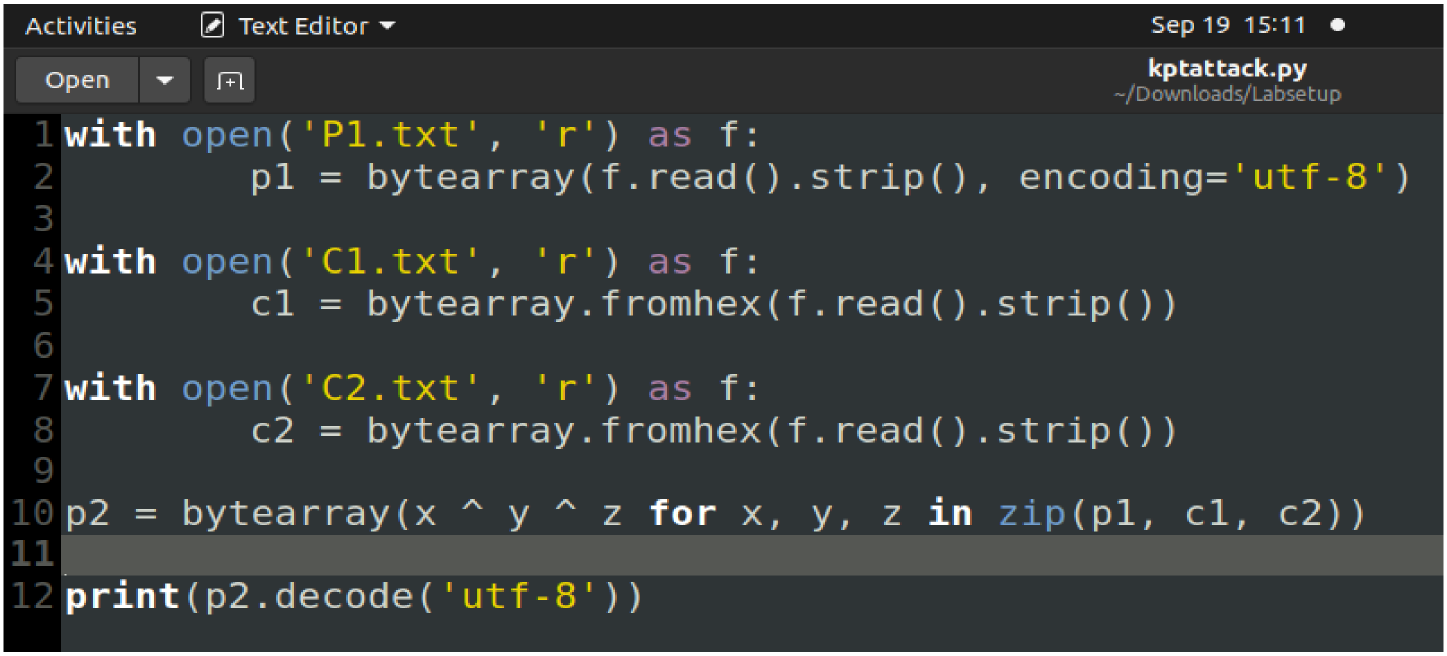
Task #6

The first part of this task demonstrates the purpose of an IV. We will encrypt a plaintext using the different IV and same IV.



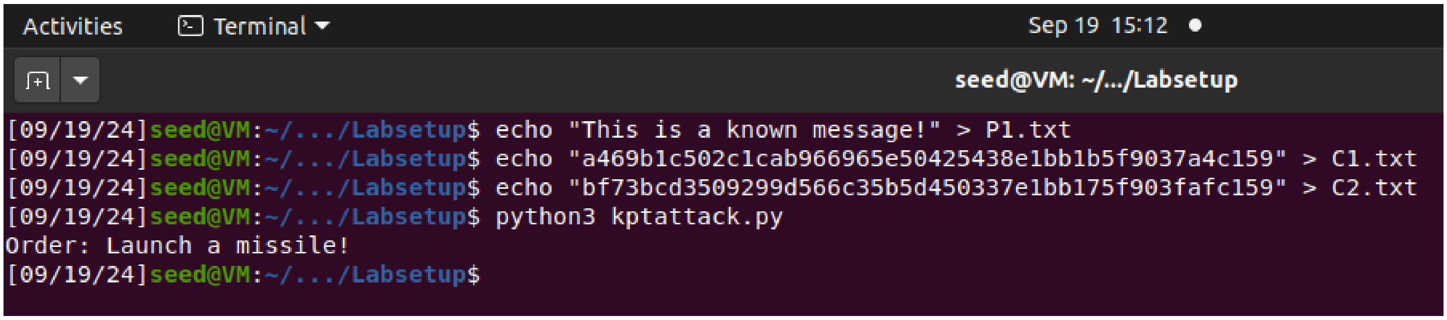
```
[09/19/24]seed@VM:~/.../Labsetup$ cat plaintext1.txt
Information Security Assignment#1 by Mohammad Yehya Hayati K213309
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in plaintext1.txt -out cipher1.out -K 00112233445566778899aabbccddeeff -iv 010203040506
070809100a0b0c0d0e0f
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in plaintext1.txt -out cipher2.out -K 00112233445566778899aabbccddeeff -iv 102030405060
70809010a0b0c0d0e0f0
[09/19/24]seed@VM:~/.../Labsetup$ xxd cipher1.out
00000000: df76 4e3e 3544 596e c5fe 536d c1c4 4e2a  .vN>5DYn..Sm..N*
00000010: 884c b02c 8b07 8d02 3903 1d21 61d9 cb5e  .L.,....9..!a..^
00000020: ff8a 0a9f 6167 712d 7471 01fc 12d1 3b4e  ....agq-tq....;N
00000030: b9ff 0eb2 d2ab 659d 153b 36f4 d521 1b19  ......e..;6..!..
00000040: 8d18 429a 0aa9 c275 6bec 701f fb89 63f9  ..B....uk.p...c.
[09/19/24]seed@VM:~/.../Labsetup$ xxd cipher2.out
00000000: b52a e8ff 442f 80f3 6441 3e49 2f63 4281  .*..D/..dA>I/cB.
00000010: 9aad a299 d9ba fb6a bcc6 19db 2259 df46  .......j...."Y.F
00000020: 1659 8765 a62c f07a 52ba 8b9b d67e 45f6  .Y.e.,.zR....~E.
00000030: bbae 8703 1a33 bd08 ea4f edaf 9a29 6146  .....3...O...)aF
00000040: 06bd 0e69 6f8a 8760 938f dab2 dd46 4aa6  ...io..`.....FJ.
[09/19/24]seed@VM:~/.../Labsetup$ #As you can see there is a clear difference
[09/19/24]seed@VM:~/.../Labsetup$
[09/19/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in plaintext1.txt -out cipher3.out -K 00112233445566778899aabbccddeeff -iv 010203040506
070809100a0b0c0d0e0f
[09/19/24]seed@VM:~/.../Labsetup$ xxd cipher1.out
00000000: df76 4e3e 3544 596e c5fe 536d c1c4 4e2a  .vN>5DYn..Sm..N*
00000010: 884c b02c 8b07 8d02 3903 1d21 61d9 cb5e  .L.,....9..!a..^
00000020: ff8a 0a9f 6167 712d 7471 01fc 12d1 3b4e  ....agq-tq....;N
00000030: b9ff 0eb2 d2ab 659d 153b 36f4 d521 1b19  ......e..;6..!..
00000040: 8d18 429a 0aa9 c275 6bec 701f fb89 63f9  ..B....uk.p...c.
[09/19/24]seed@VM:~/.../Labsetup$ xxd cipher3.out
00000000: df76 4e3e 3544 596e c5fe 536d c1c4 4e2a  .vN>5DYn..Sm..N*
00000010: 884c b02c 8b07 8d02 3903 1d21 61d9 cb5e  .L.,....9..!a..^
00000020: ff8a 0a9f 6167 712d 7471 01fc 12d1 3b4e  ....agq-tq....;N
00000030: b9ff 0eb2 d2ab 659d 153b 36f4 d521 1b19  ......e..;6..!..
00000040: 8d18 429a 0aa9 c275 6bec 701f fb89 63f9  ..B....uk.p...c.
[09/19/24]seed@VM:~/.../Labsetup$ #Now we can see that they are the same
[09/19/24]seed@VM:~/.../Labsetup$
```

In the second part of this task, we have to perform a *known-plaintext attack*. Using the hint given in the manual about XOR strings, we can perform this easily. First, we need to write a python script to perform the XOR operation.



```python
with open('P1.txt', 'r') as f:
        p1 = bytearray(f.read().strip(), encoding='utf-8')

with open('C1.txt', 'r') as f:
        c1 = bytearray.fromhex(f.read().strip())

with open('C2.txt', 'r') as f:
        c2 = bytearray.fromhex(f.read().strip())

p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, c1, c2))

print(p2.decode('utf-8'))
```

Next, we will use the python script to generate the message P2.

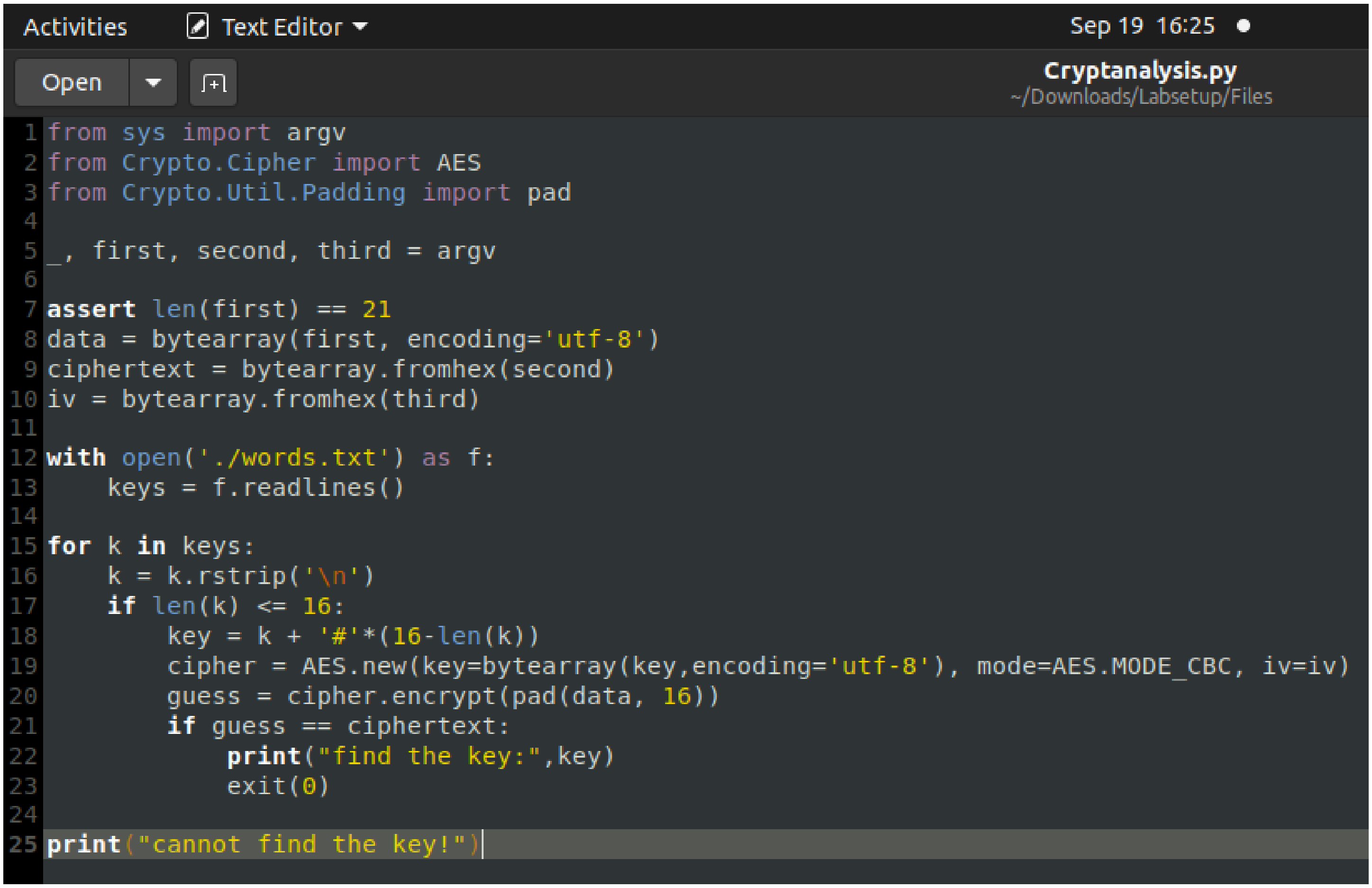


```
[09/19/24]seed@VM:~/.../Labsetup$ echo "This is a known message!" > P1.txt
[09/19/24]seed@VM:~/.../Labsetup$ echo "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159" > C1.txt
[09/19/24]seed@VM:~/.../Labsetup$ echo "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159" > C2.txt
[09/19/24]seed@VM:~/.../Labsetup$ python3 kptattack.py
Order: Launch a missile!
[09/19/24]seed@VM:~/.../Labsetup$
```

As you can see, we have successfully performed a *known-plaintext attack*.

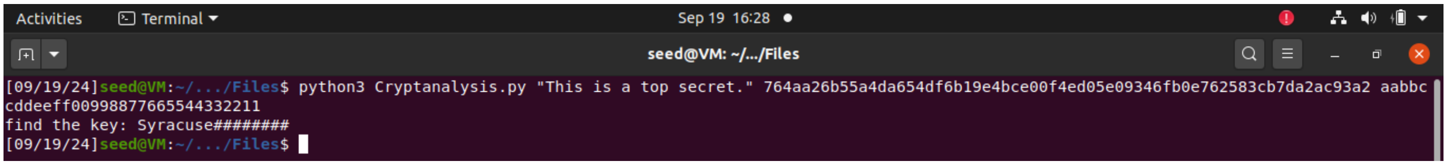
Task #7

In this task we need to perform a cryptanalysis to find out the encryption key. Firstly, we need a python script that can perform the analysis. This code is taken as a reference.



```python
from sys import argv
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

_, first, second, third = argv

assert len(first) == 21
data = bytearray(first, encoding='utf-8')
ciphertext = bytearray.fromhex(second)
iv = bytearray.fromhex(third)

with open('./words.txt') as f:
    keys = f.readlines()

for k in keys:
    k = k.rstrip('\n')
    if len(k) <= 16:
        key = k + '#'*(16-len(k))
        cipher = AES.new(key=bytearray(key,encoding='utf-8'), mode=AES.MODE_CBC, iv=iv)
        guess = cipher.encrypt(pad(data, 16))
        if guess == ciphertext:
            print("find the key:",key)
            exit(0)

print("cannot find the key!")
```

Now we run it.



```
[09/19/24]seed@VM:~/.../Files$ python3 Cryptanalysis.py "This is a top secret." 764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2 aabbc
cddeeff00998877665544332211
find the key: Syracuse########
[09/19/24]seed@VM:~/.../Files$ 
```

We now have the key.