Benjamin Haedt

Lab 8 – SKE

CSCI 476 – Comp security

2 April 2023

Task 1

```
[04/02/23]seed@VM:~/.../08_ske$ ls
ciphertext.txt   pic_original.bmp  words.txt
freq_counter.py  sample_xor.py
[04/02/23]seed@VM:~/.../08_ske$ tr 'aet' 'XGE' < ciphertext.txt > out.txt
```

I ran the freq_counter.py on the ciphertext.txt file.

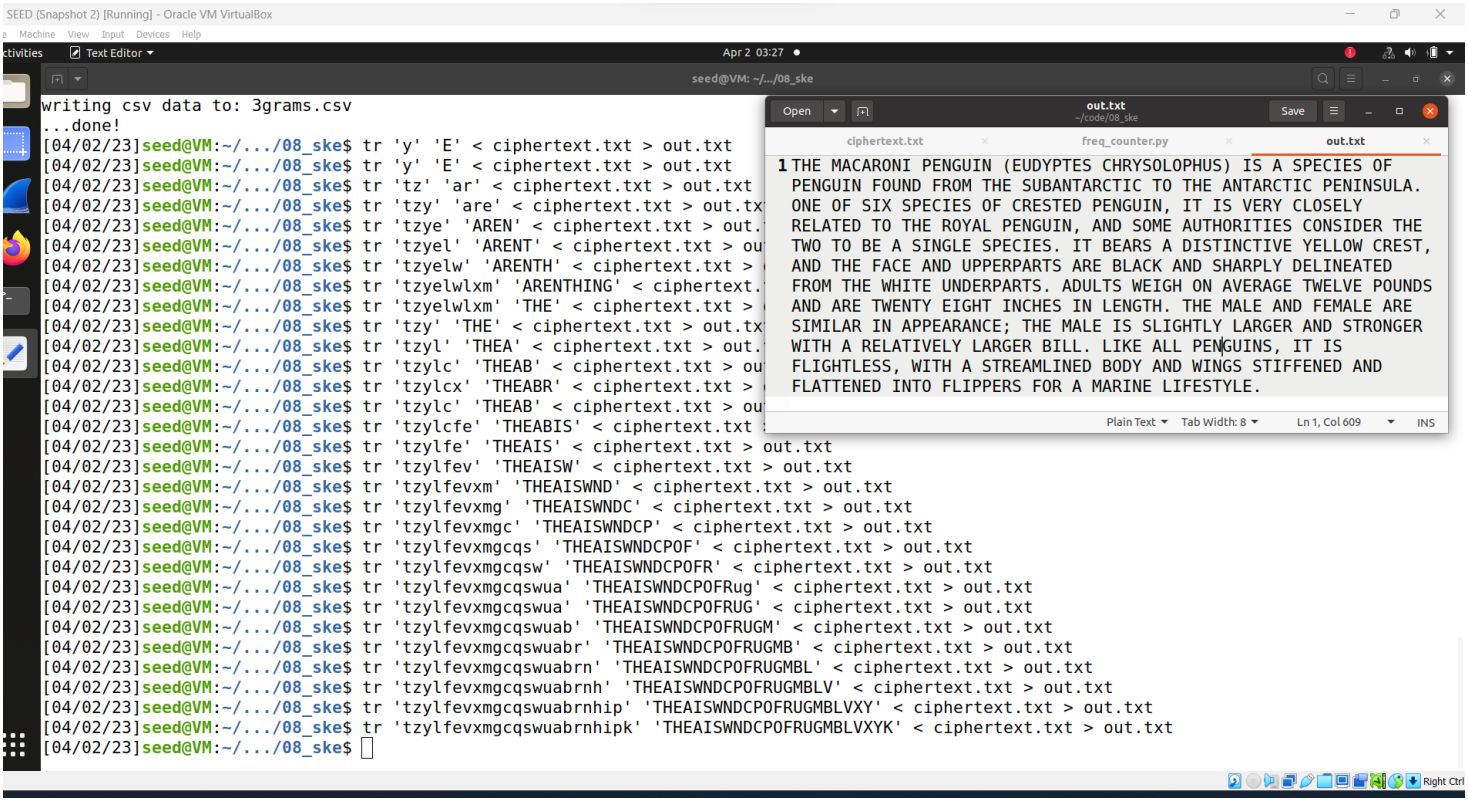
```
[04/02/23]seed@VM:~/.../08_ske$ sudo python3 freq_counter.py < ciphertext.txt

letter frequencies (all):
y : 81 (13.28%)
l : 52 (8.52%)
t : 50 (8.20%)
f : 50 (8.20%)
x : 46 (7.54%)
e : 42 (6.89%)
n : 37 (6.07%)
```

I used frequency analysis to make predictions about what each letter would be. I determined that the first letter, y, which occurred the most, would be e. Once I set y = e I guessed that the most common trigram tzy could be THE, so I did that. At each step I kind of guessed more and more, with each step I could see the next piece of the puzzle and continue making words and assumptions based on sentence structure and basic English language knowledge (that was some good writing though! Haha). One of these, I was able to guess fe = is, since it appeared in a certain spot in the sentence that made it seemed like that would be a good fit and work with the context. I can't go back through all the steps I did exactly how I did it now. I did record each step down below and also took a screenshot of the ciphered text all uncyphered. I did have to back track on a few letters early on, but eventually went in the right direction. I originally thought tzy was 'are', but nope. Weeeee! That was fun.

Y = e

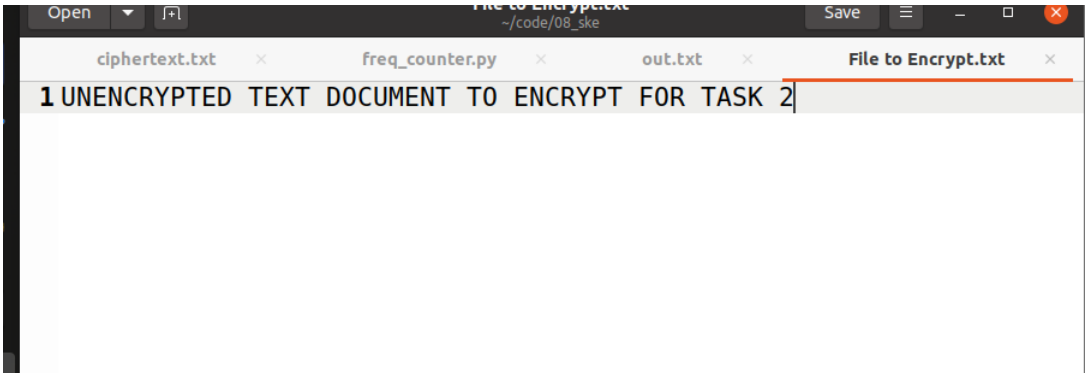
I guessed that the first word could be the, so that makes tzy = THE

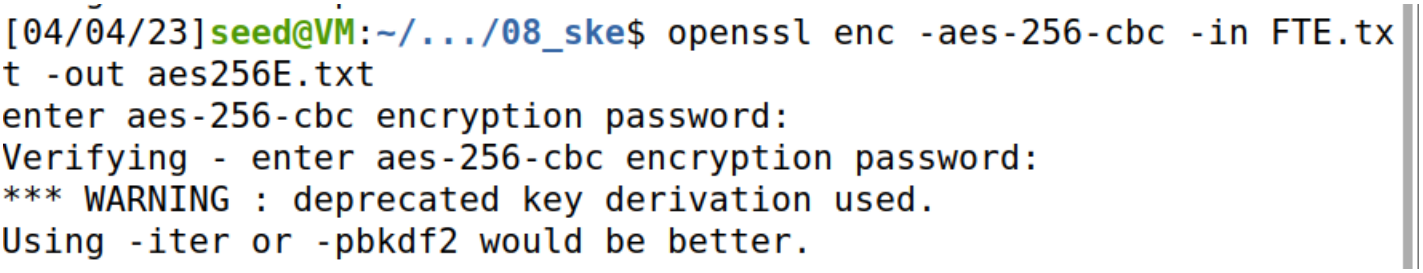
I guess l = a, fe = is, v = w, x = n, m = d, g= c, c = p, qs = of, w = r, u = u, a = g, b = m, r = b, n = l, h = v, I = x, p = y, k = k



Task 2

I made a text file to encrypt.

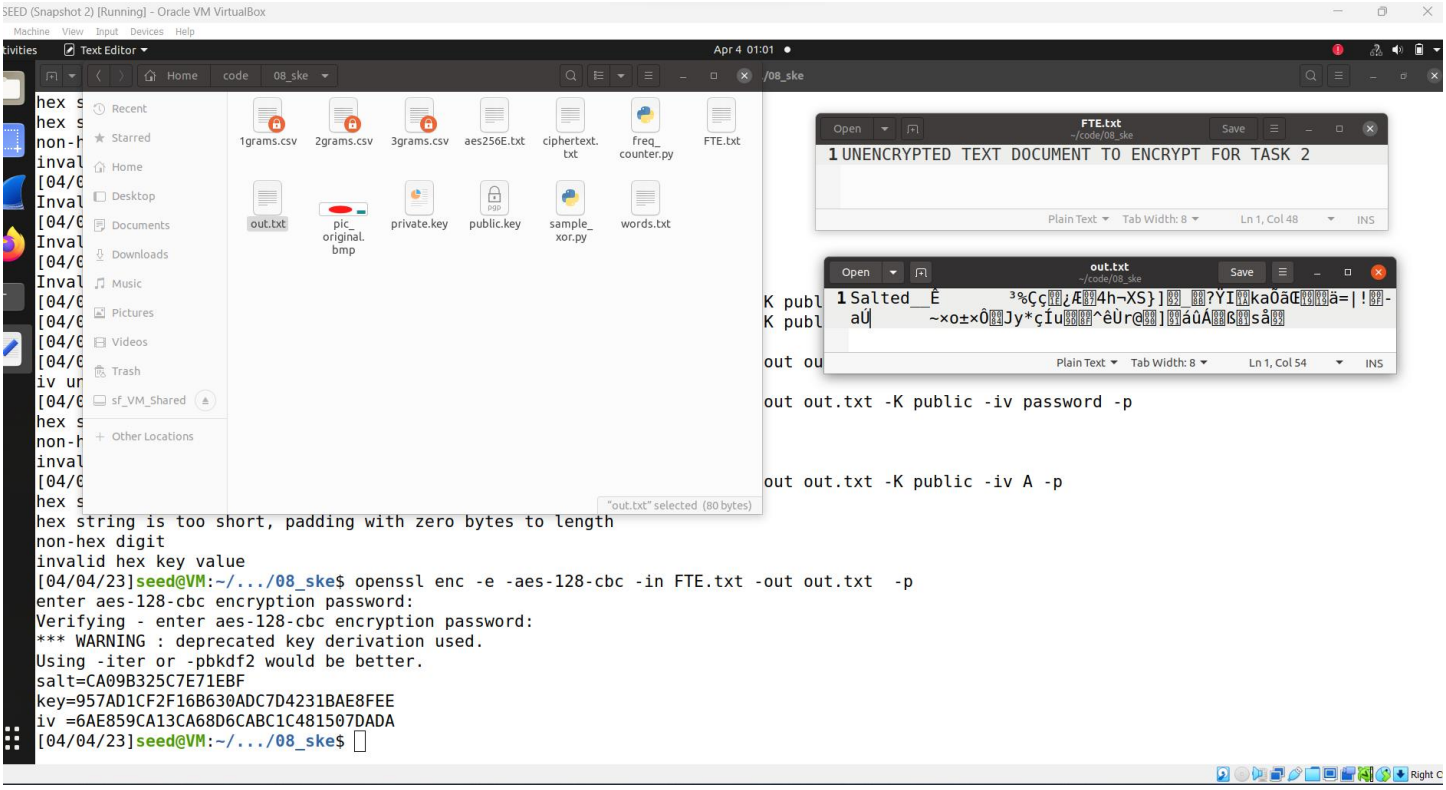




I encrypted FTE.txt using aes256 with basic options and used the password ‘password’ and my key.

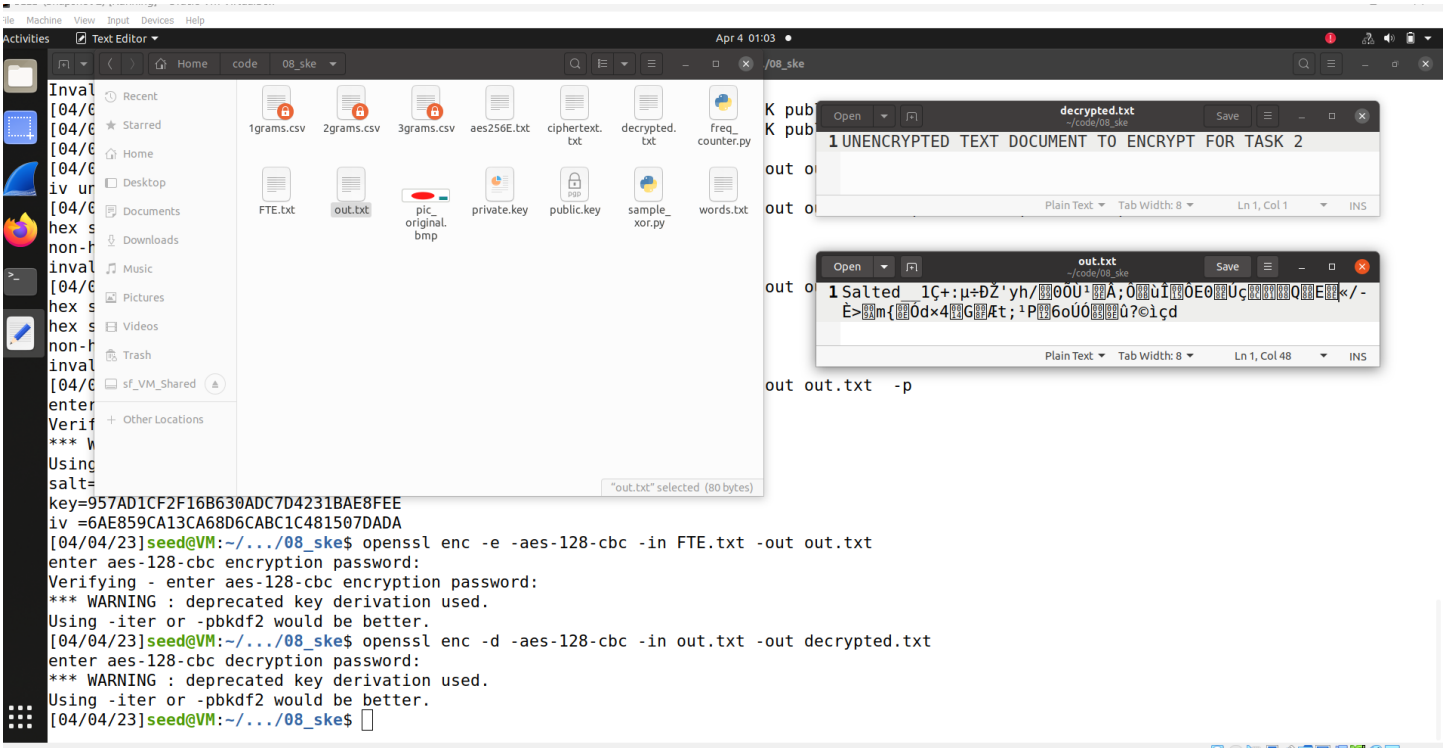
I think the requirements for this task is to create a private and public key, so I did so below and generated both a private and public key.

```
[04/04/23]seed@VM:~/.../08_ske$ openssl genrsa -aes256 -passout pas
s:password -out private.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.........+++++
.......................................+++++
e is 65537 (0x010001)
[04/04/23]seed@VM:~/.../08_ske$ openssl rsa -in private.key -passin
 pass:password -pubout -out public.key
writing RSA key
[04/04/23]seed@VM:~/.../08_ske$
```

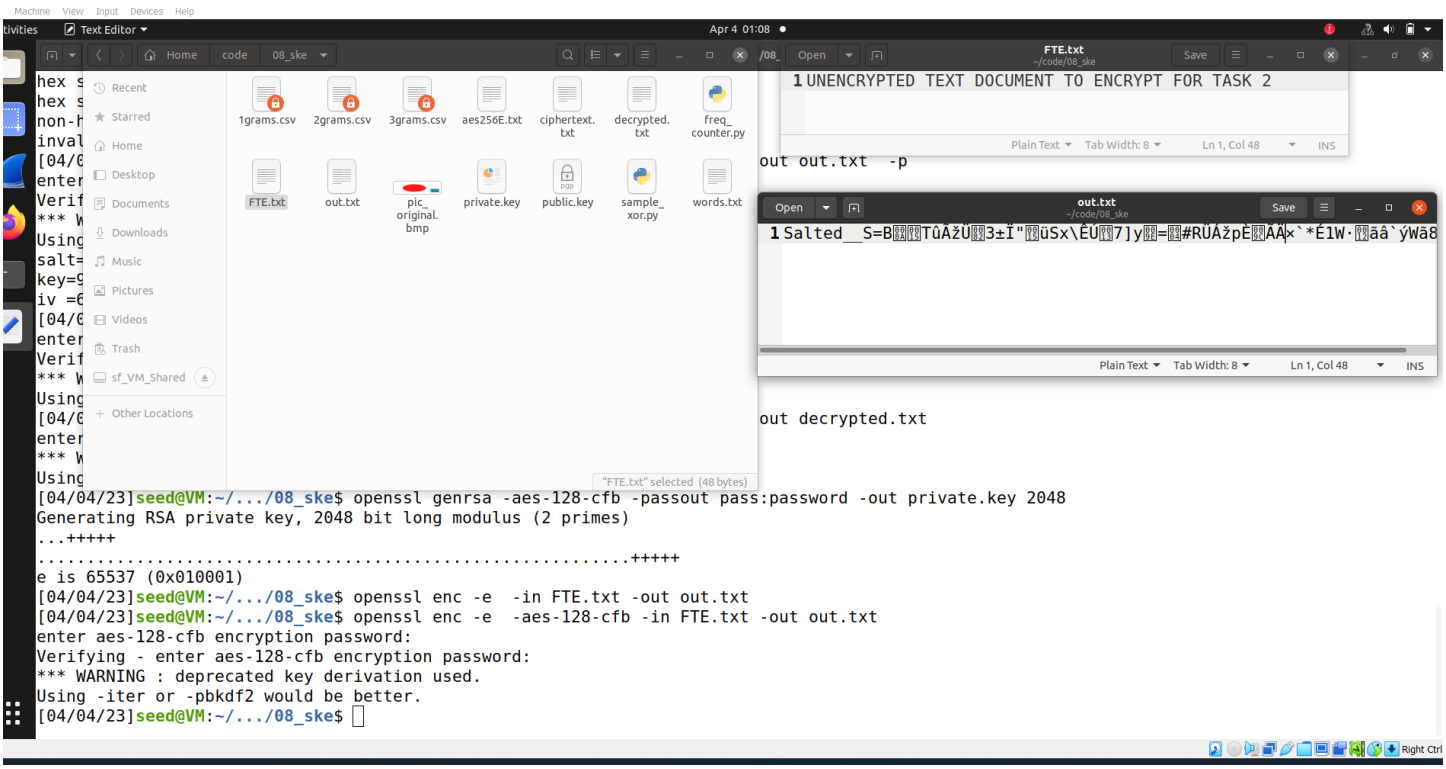


Using the public key, I encrypted the text file FTE.txt.

Using the private key I decrypted the text file.

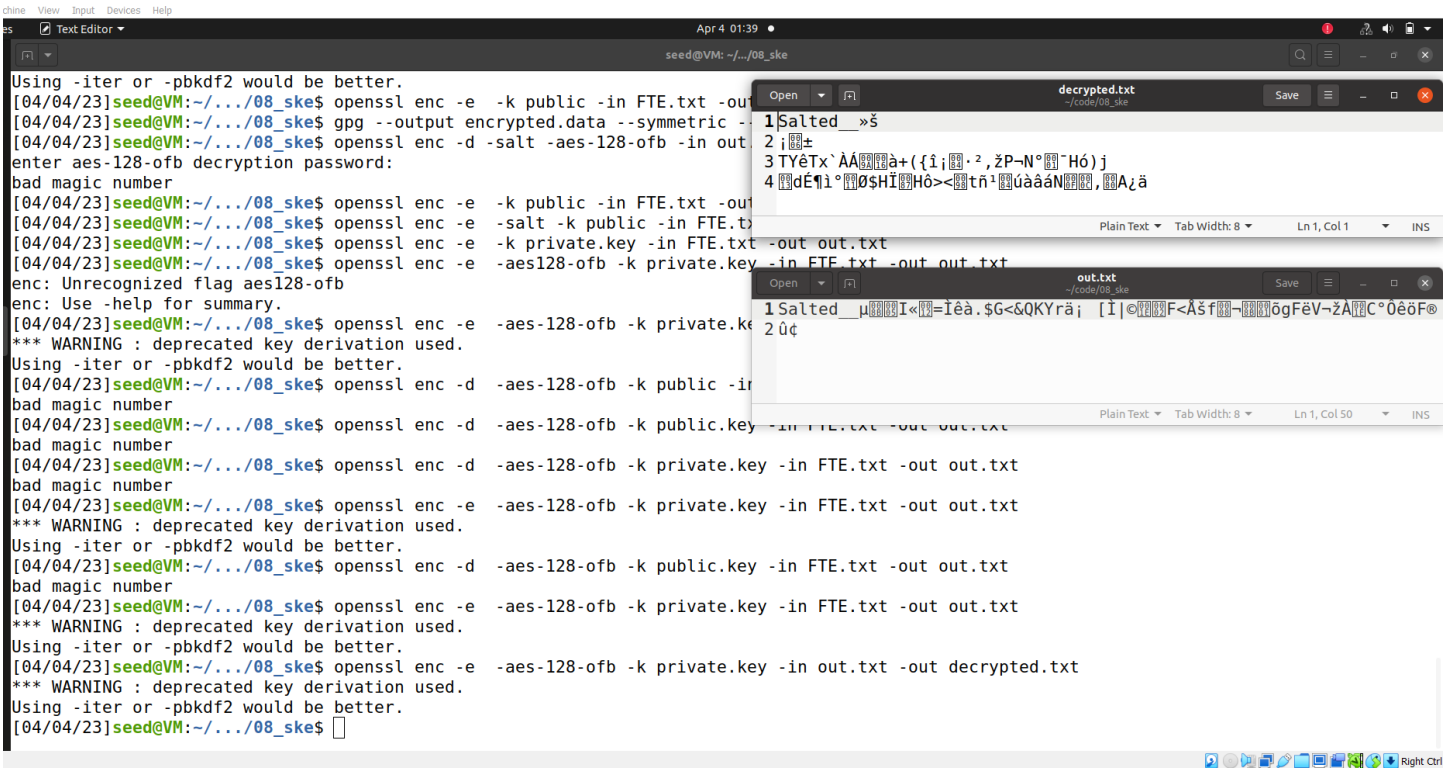


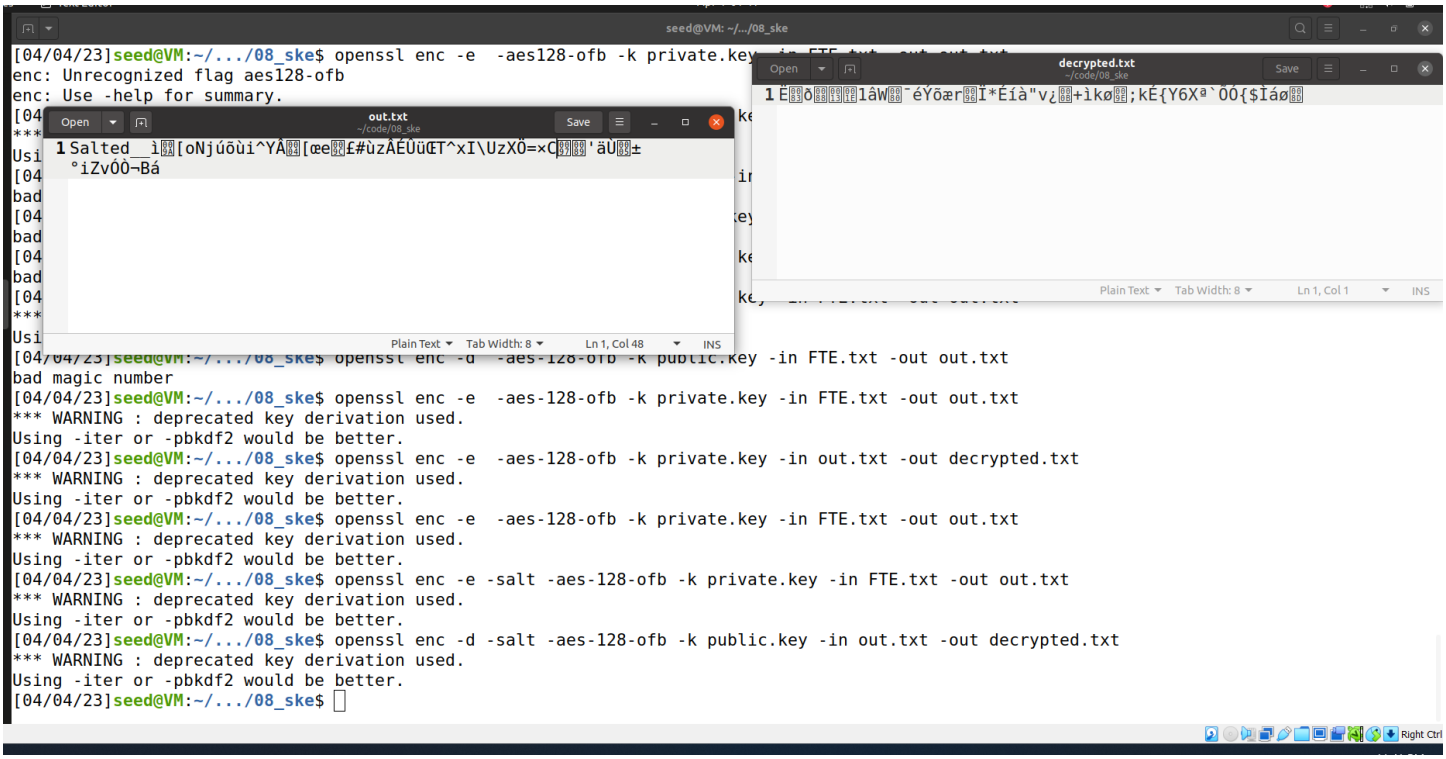
Next I created new keys using -aes-128-cfb, then encrypted a file. Then decrypted the encrypted txt file out.txt



Next I attempted to decrypt a file that I encrypted using the private key with the public key

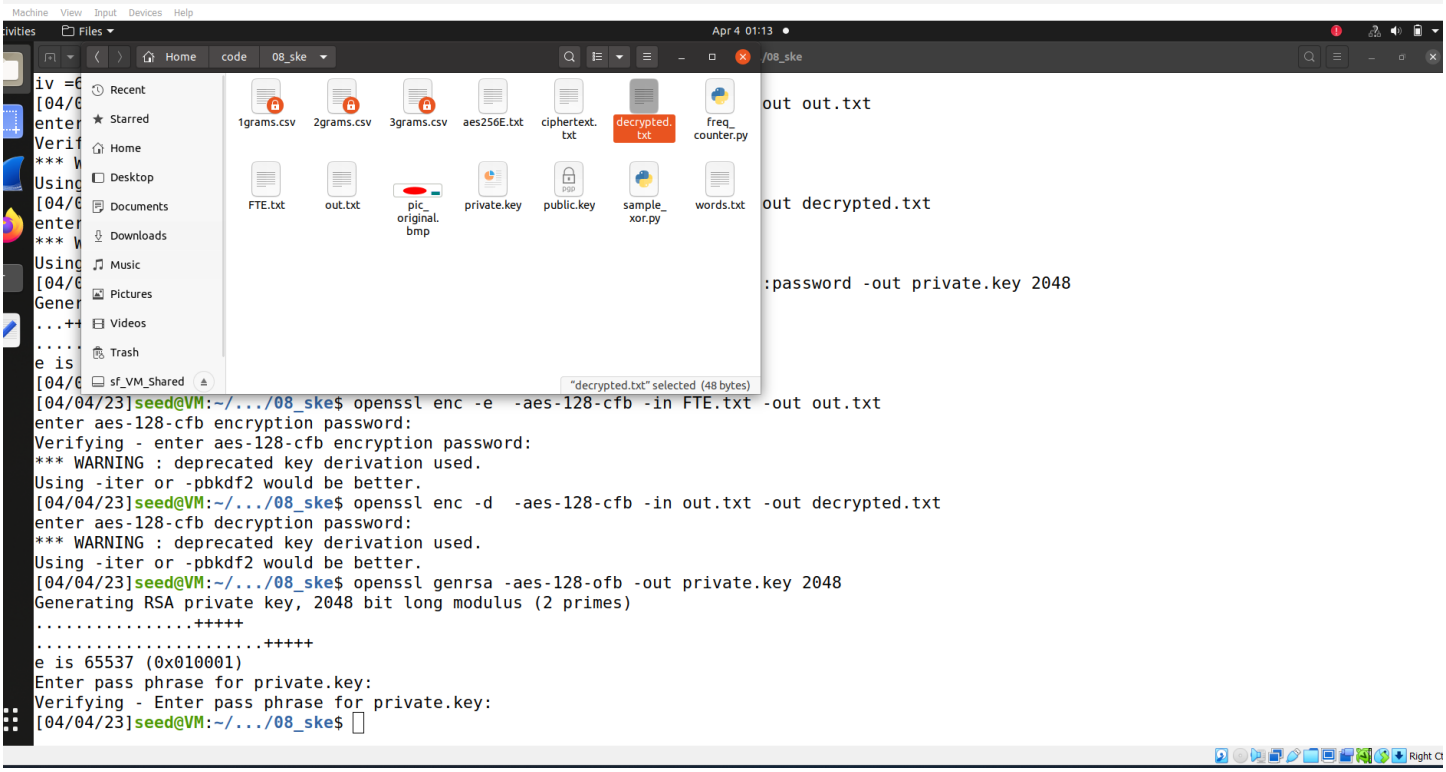
And it doesn’t decrypt the encrypted file as it shouldn’t.

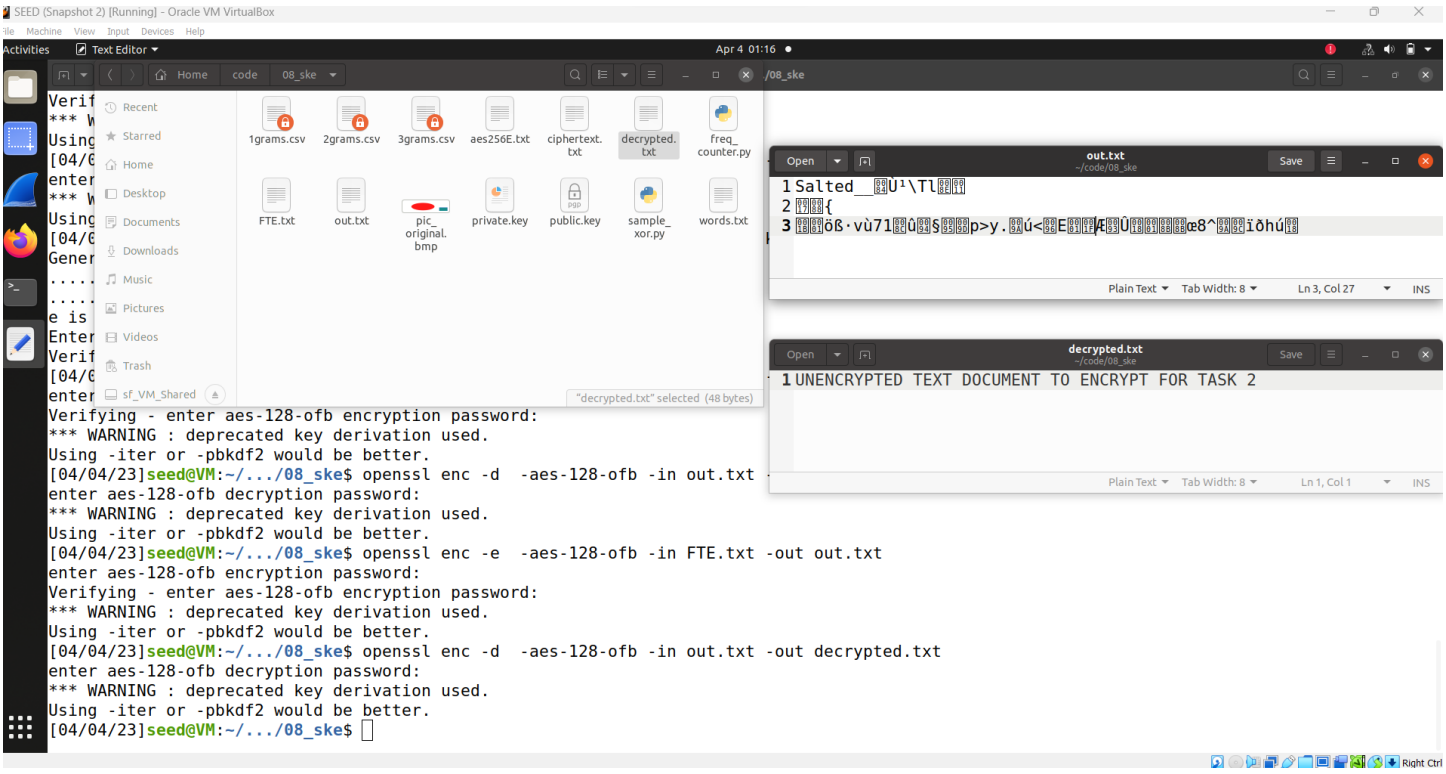




For the last encryption, I used -aes-128-ofb for my cipher type, I don't know why but it says I need to enter a password when I use a public key to encrypt a file, then that password is the same one used to decrypt the encrypted txt.

Then I encrypted FTE.txt with the public key and then decrypted using the private key.

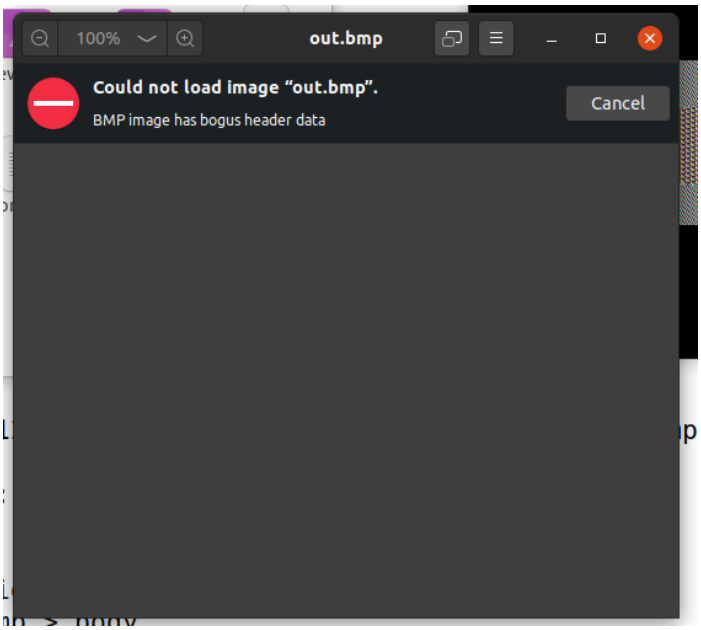


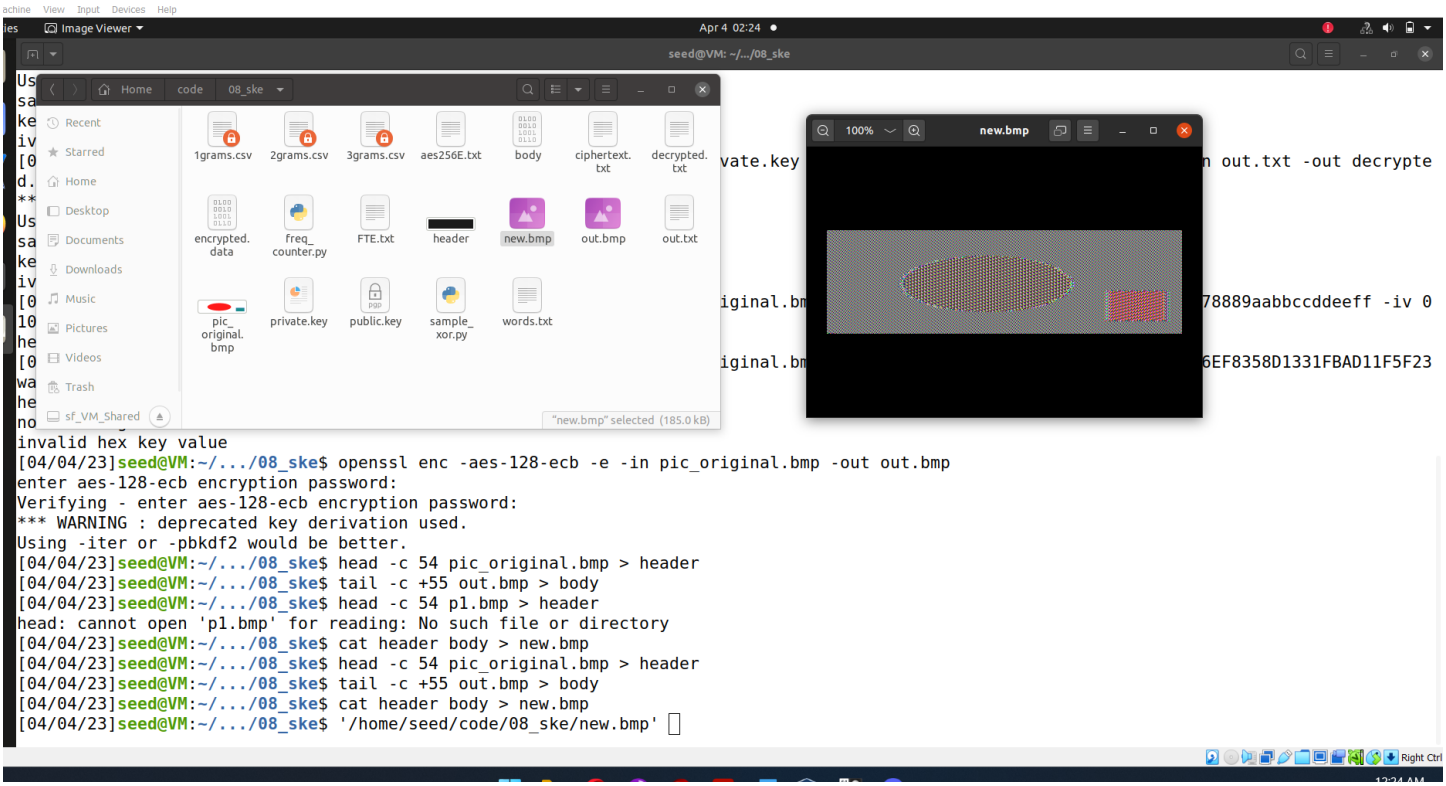


Hopefully this fulfills Task 2 requirements. I could continue using different ciphertexts and generating new keys, but this proves that I can use public and private keys to encrypt and decrypt, and also just use a simple password to encrypt a file.

Task 3

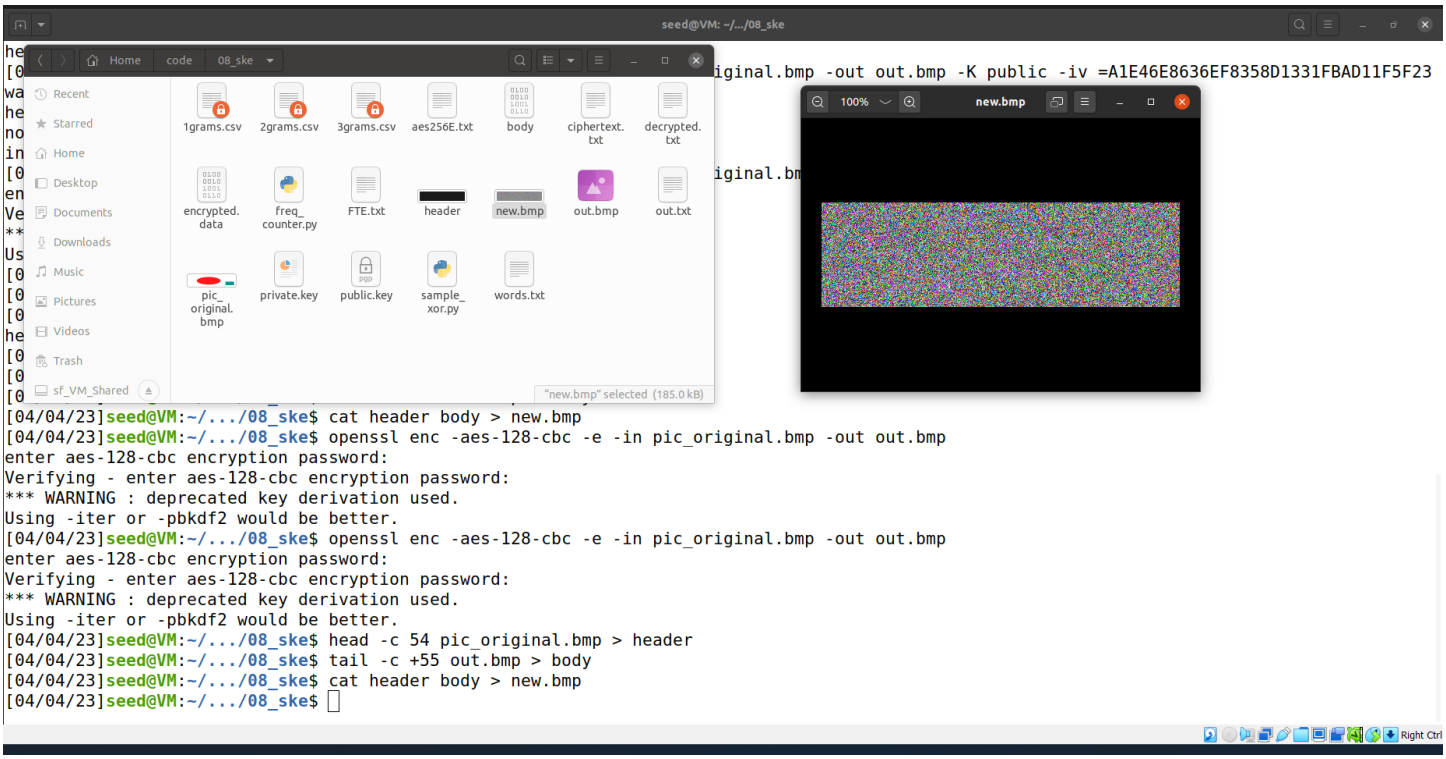
Encrypted the image with ecb, cannot open the output from encrypting it due to header information also being encrypted. So I followed the directions to get the right header information and then applied it to a image named out.bmp. It works.



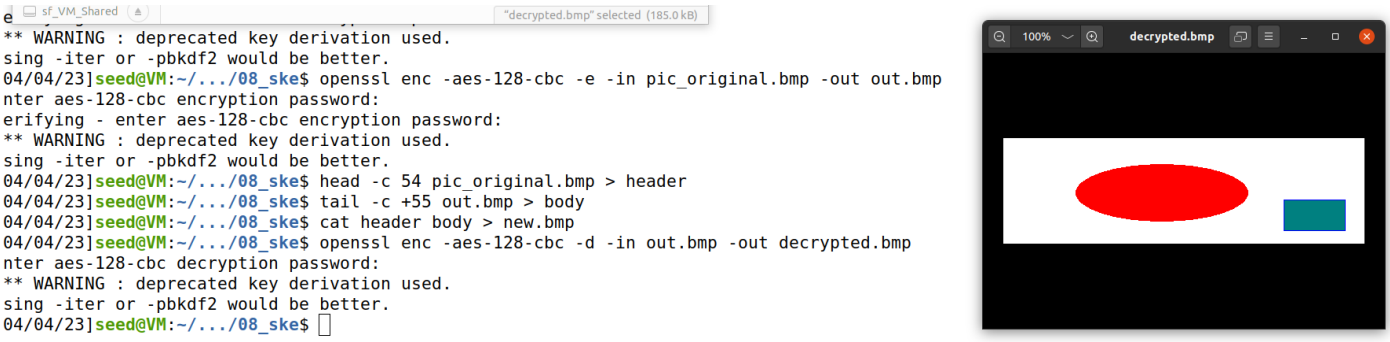


Yes, when we use ecb to encrypt the image, we can still clearly see what the image is, a oval and a rectangle.

Next I used cbc to encrypt the image and then applied the header information to that picture. This is clearly a better encryption option for images, since it completely distorts the image.



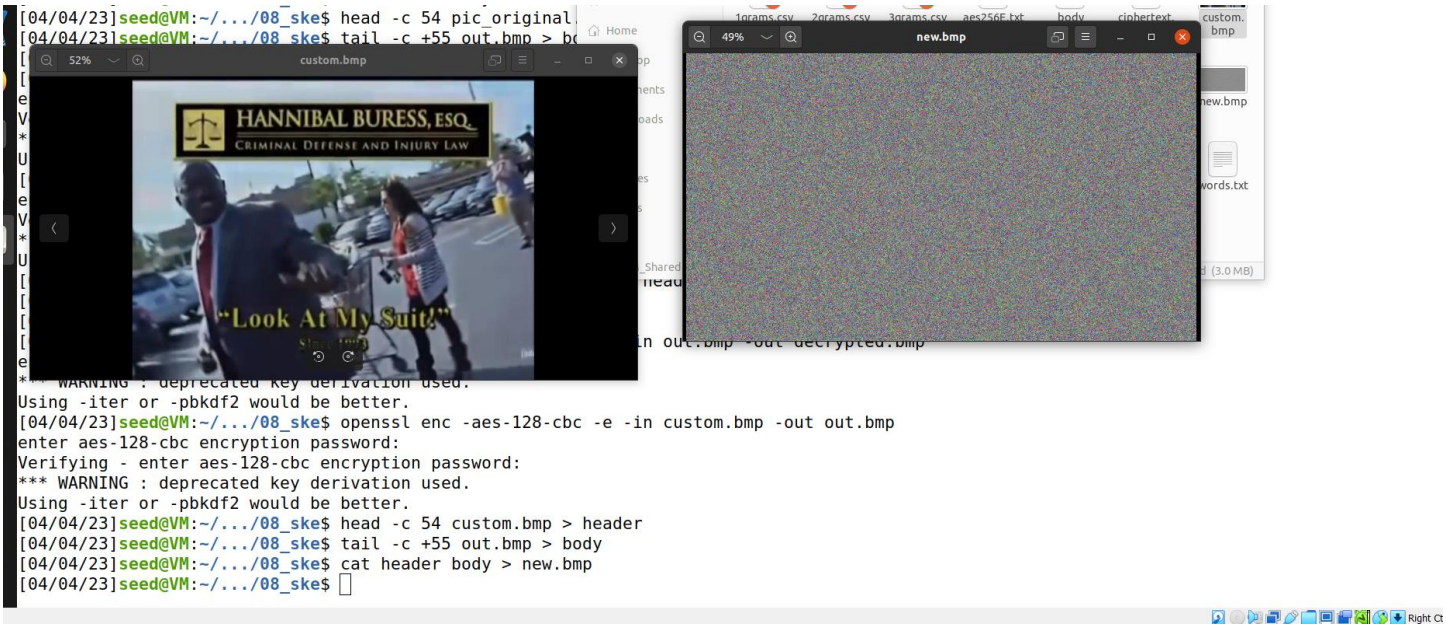
Also decrypted the image that was encrypted using cbc



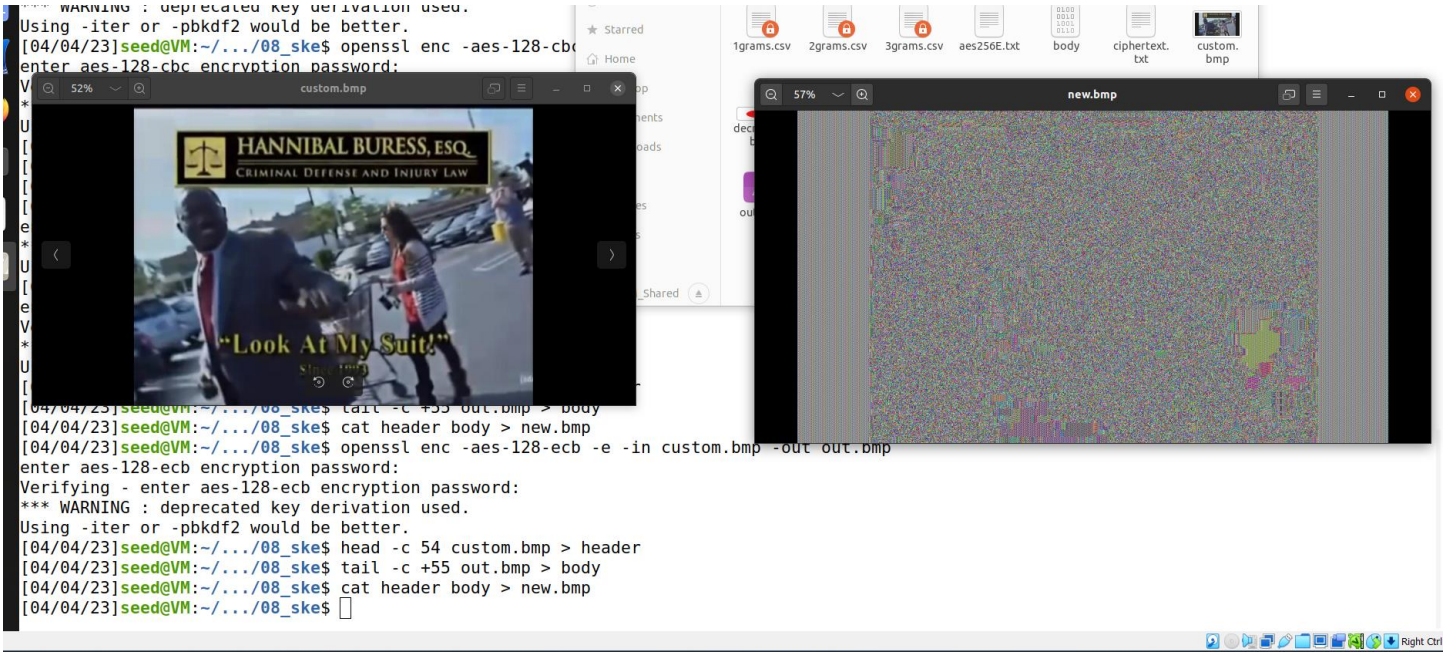
Task 3.2

For some reason, I have to use my own image and do the exact same thing I did for the previous image, so here we go…

Here is custom image using cbc to encrypt

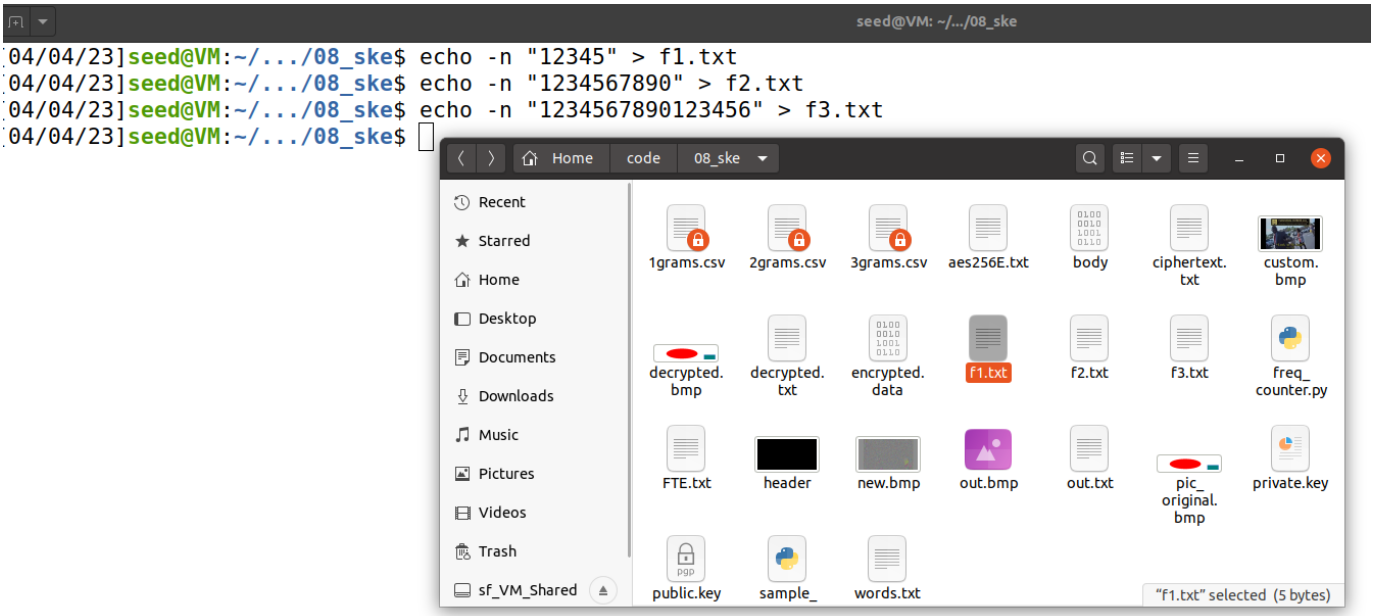


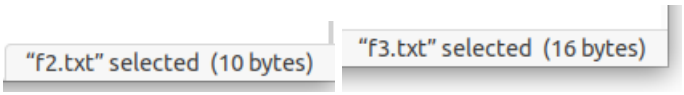
And here is the image using ecb

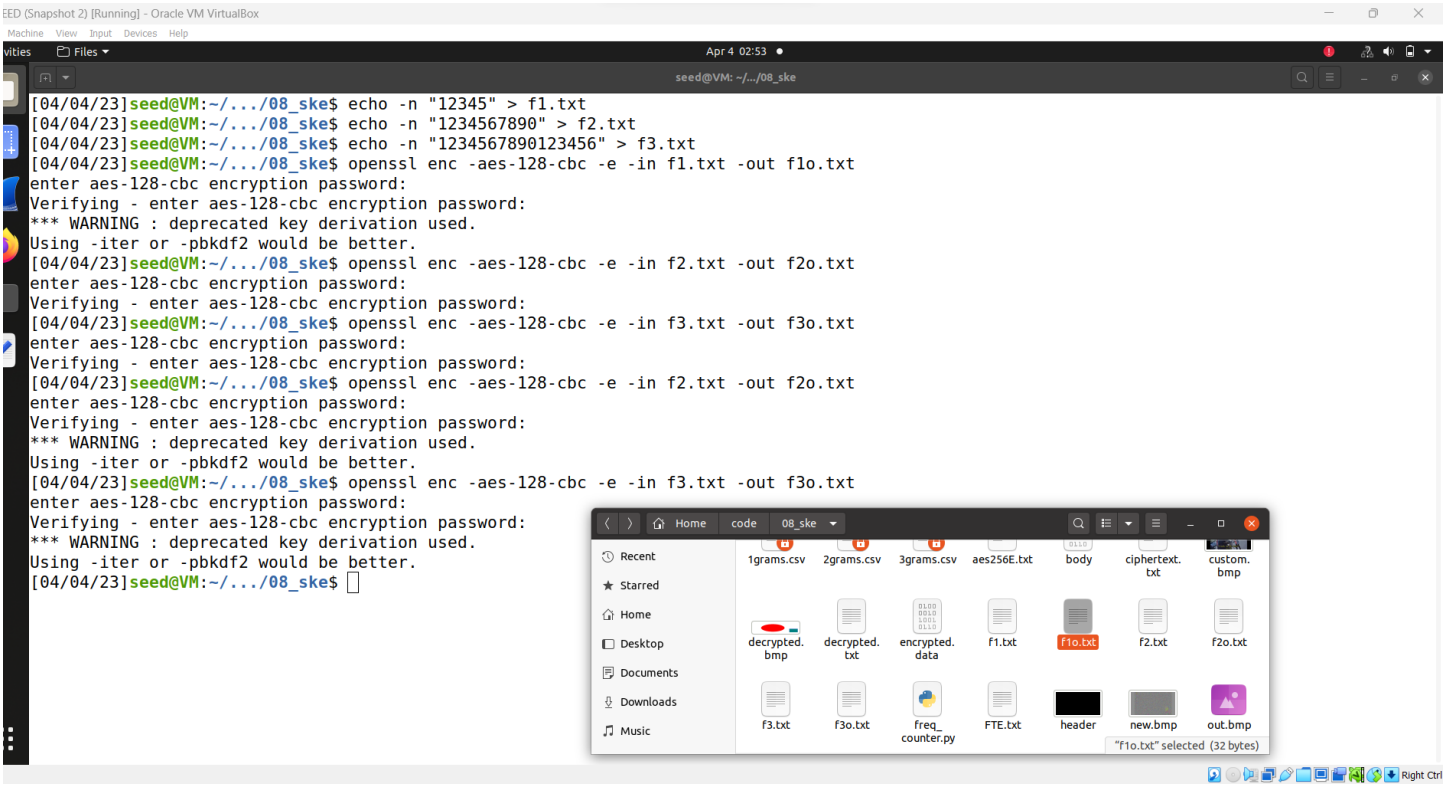


The ecb encryption on this file worked a lot better than the picture provided.

Task 4.1







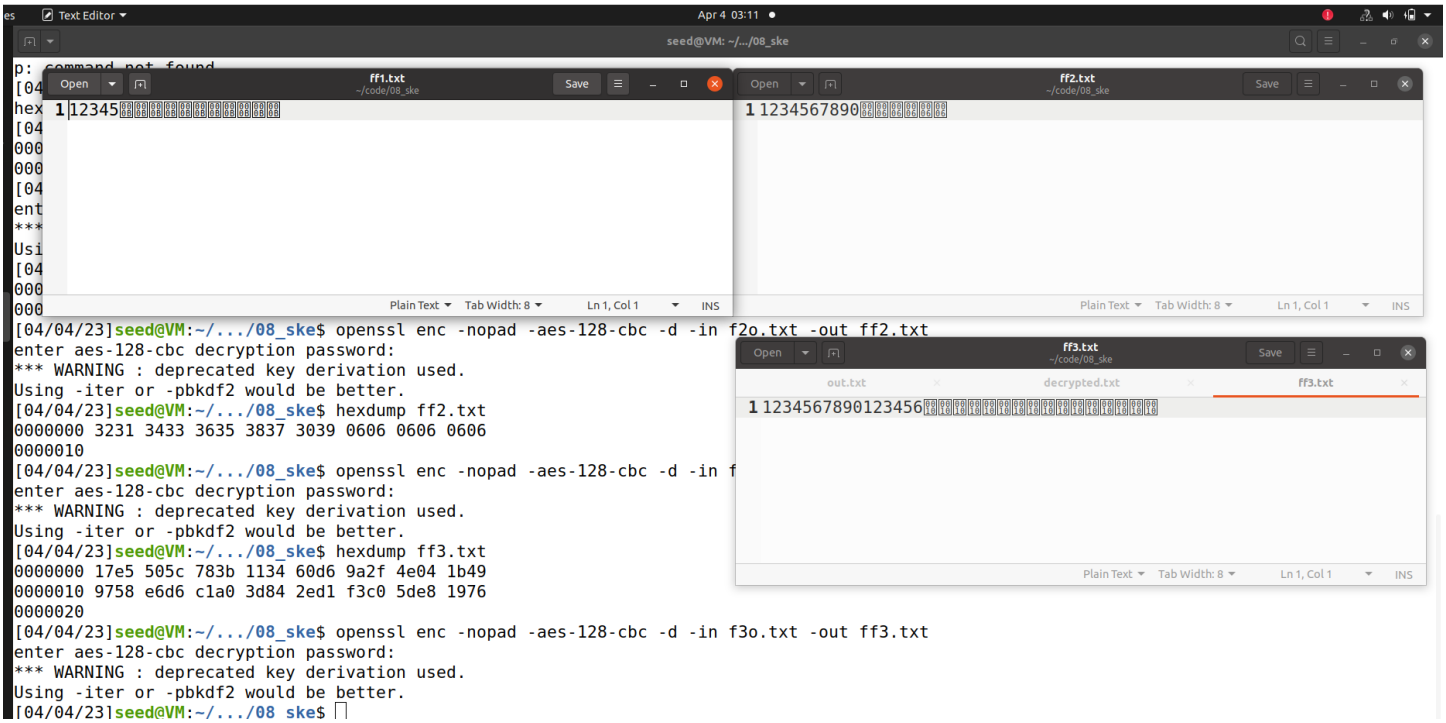
The size of the encrypted files increased. Since the first two where under 16 bytes, when encrypted they had enough room once encrypted and padded to increase the size to 16 bytes , but once we encrypted the 16 byte file, it needed more room to encrypt and pad so it now is 48 bytes long.

```
using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f1.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f1o.txt
00000000  53 61 6c 74 65 64 5f 5f  f3 e0 9d 85 75 2a ee a8  |Salted__....u*..|
00000010  51 70 65 5d a1 c3 dc 05  63 29 64 75 50 ec 0c 65  |Qpe]....c)duP..e|
00000020
[04/04/23]seed@VM:~/.../08_ske$ xxd f1.txt
00000000: 3132 3334 35                             12345
[04/04/23]seed@VM:~/.../08_ske$ xxd f1o.txt
```

```
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f2.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f2o.txt
00000000  53 61 6c 74 65 64 5f 5f  06 1d 92 10 07 1c d5 d3  |Salted__........|
00000010  da b8 75 f5 3d ab a6 e4  bf 21 0e 49 d8 88 e0 38  |..u.=....!.I...8|
00000020
[04/04/23]seed@VM:~/.../08_ske$ xxd f2.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
[04/04/23]seed@VM:~/.../08_ske$ xxd f2o.txt
00000000: 5361 6c74 6564 5f5f 061d 9210 071c d5d3  Salted__........
00000010: dab8 75f5 3dab a6e4 bf21 0e49 d888 e038  ..u.=....!.I...8
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f3.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[04/04/23]seed@VM:~/.../08_ske$ hexdump -C f3o.txt
00000000  53 61 6c 74 65 64 5f 5f  43 76 a6 30 a2 50 18 62  |Salted__Cv.0.P.b|
00000010  36 13 0d 3f 44 05 89 40  ac 4e 21 37 d2 0f 7f e4  |6..?D..@.N!7....|
00000020  78 f0 6e ed 77 ae 58 4a  cd 65 32 9f 21 b7 87 43  |x.n.w.XJ.e2.!..C|
00000030
[04/04/23]seed@VM:~/.../08_ske$ xxd f3.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
[04/04/23]seed@VM:~/.../08_ske$ xxd f3o.txt
00000000: 5361 6c74 6564 5f5f 4376 a630 a250 1862  Salted__Cv.0.P.b
00000010: 3613 0d3f 4405 8940 ac4e 2137 d20f 7fe4  6..?D..@.N!7....
00000020: 78f0 6eed 77ae 584a cd65 329f 21b7 8743  x.n.w.XJ.e2.!..C
[04/04/23]seed@VM:~/.../08_ske$
```

```
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cbc -d -in f1o.txt -out ff1.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cbc -d -in f2o.txt -out ff2.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff2.txt
0000000 3231 3433 3635 3837 3039 0606 0606 0606
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cbc -d -in f3o.txt -out ff3.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff3.txt
0000000 17e5 505c 783b 1134 60d6 9a2f 4e04 1b49
0000010 9758 e6d6 c1a0 3d84 2ed1 f3c0 5de8 1976
0000020
[04/04/23]seed@VM:~/   /08_ske$
```

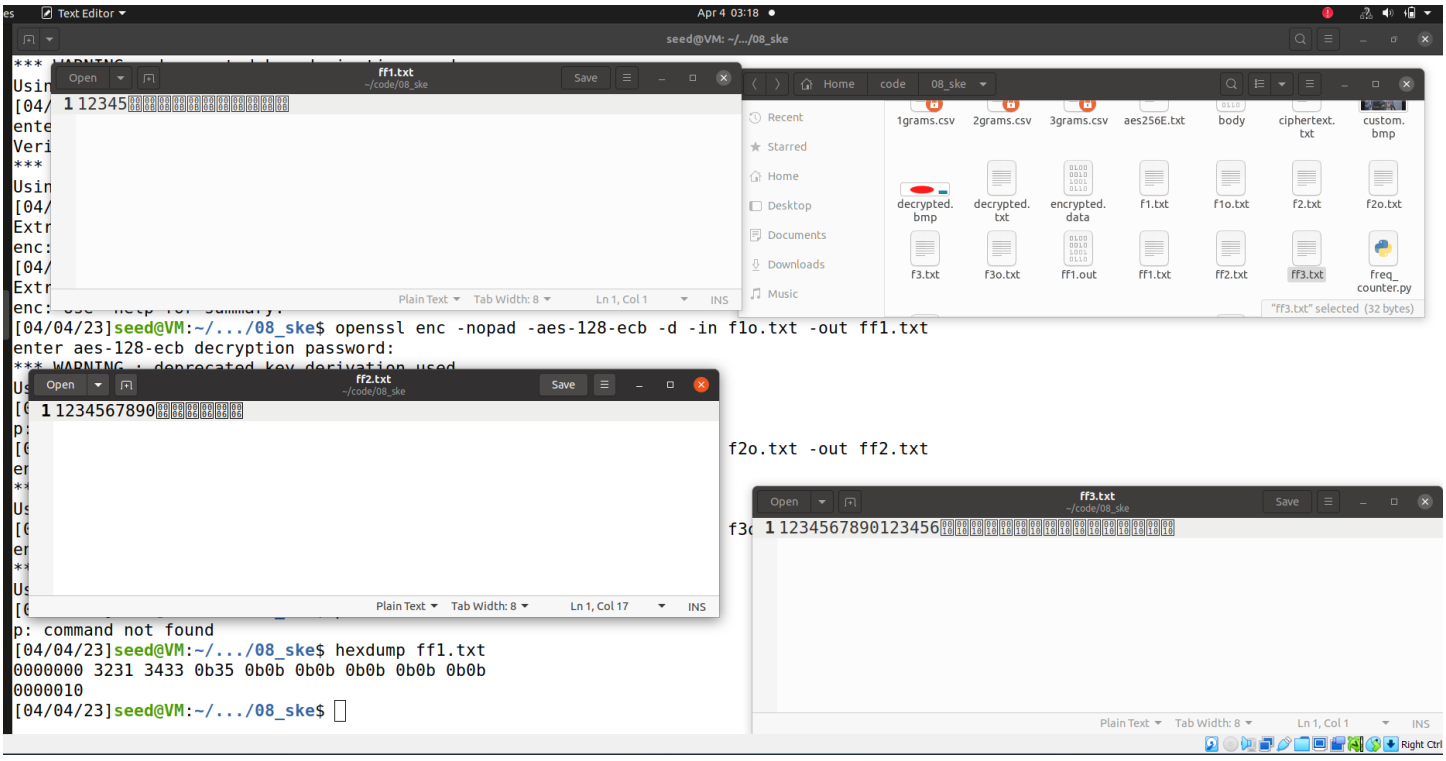
Here we can see the padded values that were added to each txt file as it was encrypted.



Task 4.2

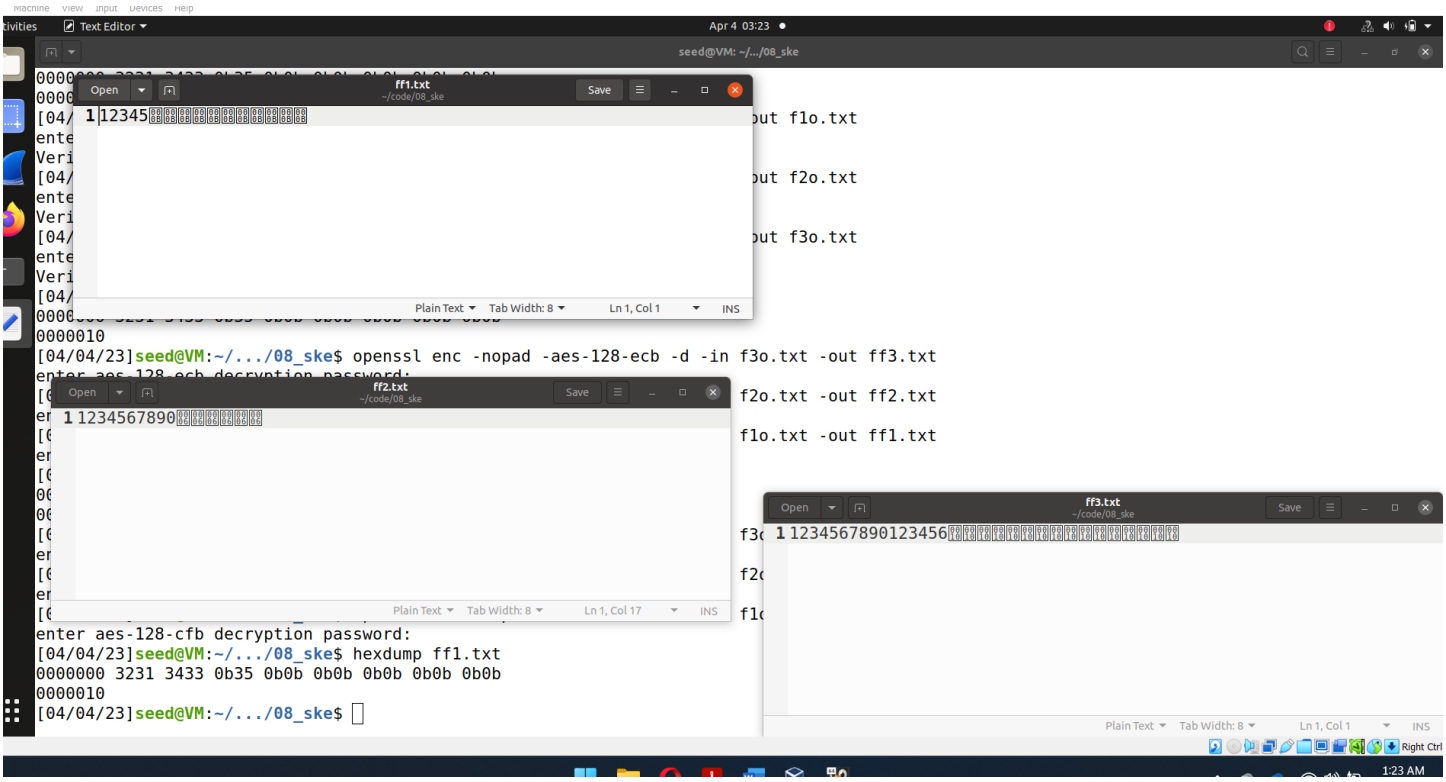
Using ecb to encrypt and decrypt. ECB uses padding.

```
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ecb -e -in f3.txt -out f3o.txt
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.
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ecb -d in f1o.txt -out ff1.txt
Extra arguments given.
enc: Use -help for summary.
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d in f1o.txt -out ff1.txt
Extra arguments given.
enc: Use -help for summary.
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f1o.txt -out ff1.txt
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ p
p: command not found
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f2o.txt -out ff2.txt
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f3o.txt -out ff3.txt
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[04/04/23]seed@VM:~/.../08_ske$ p
p: command not found
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
```

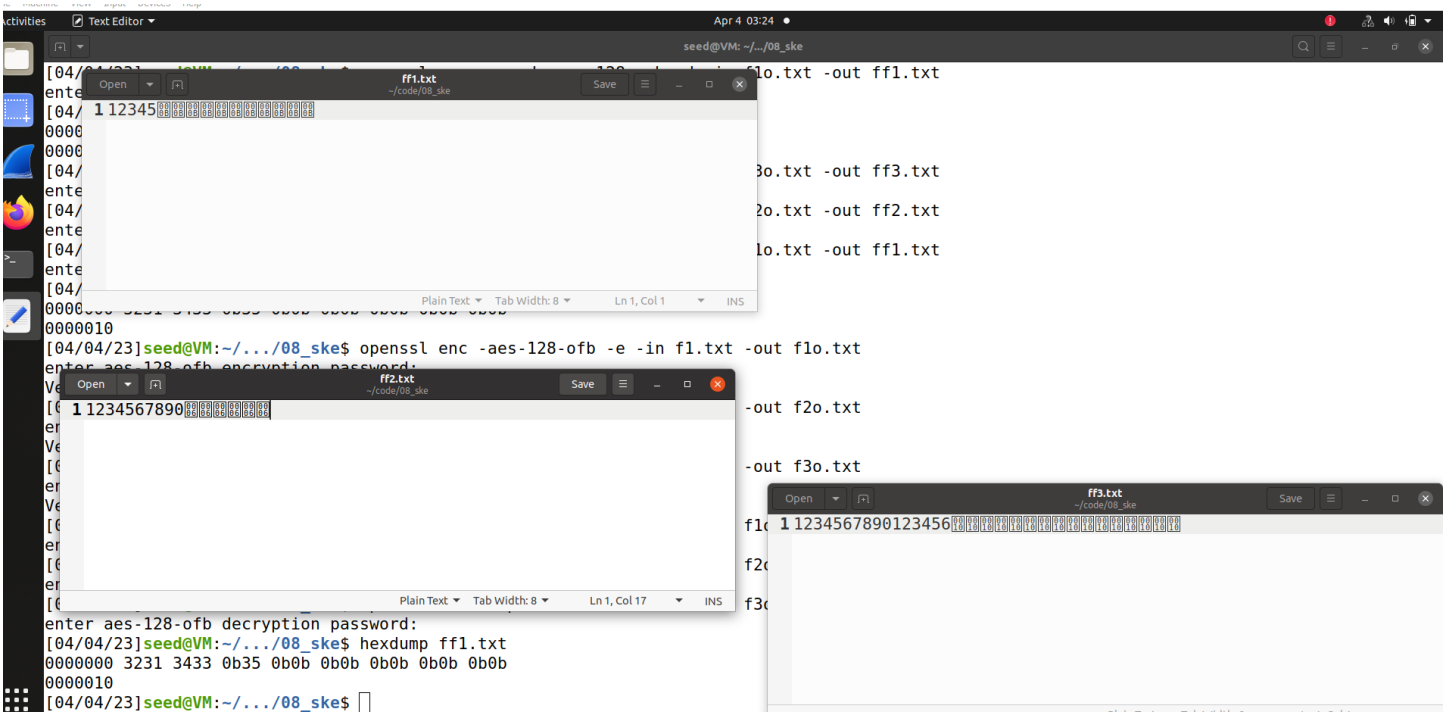


Using cfb, it does not use padding

```
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -e -in f1.txt -out f1o.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -e -in f2.txt -out f2o.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -e -in f3.txt -out f3o.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f3o.txt -out ff3.txt
enter aes-128-ecb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f2o.txt -out ff2.txt
enter aes-128-ecb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ecb -d -in f1o.txt -out ff1.txt
enter aes-128-ecb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cfb -d -in f3o.txt -out ff3.txt
enter aes-128-cfb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cfb -d -in f2o.txt -out ff2.txt
enter aes-128-cfb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-cfb -d -in f1o.txt -out ff1.txt
enter aes-128-cfb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
[04/04/23]seed@VM:~/.../08_ske$
```



Using OFB, it does not use padding.




```
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ofb -e -in f1.txt -out f1o.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ofb -e -in f2.txt -out f2o.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ofb -e -in f3.txt -out f3o.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ofb -d -in f1o.txt -out ff1.txt
enter aes-128-ofb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ofb -d -in f2o.txt -out ff2.txt
enter aes-128-ofb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -nopad -aes-128-ofb -d -in f3o.txt -out ff3.txt
enter aes-128-ofb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff1.txt
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
```

OFB doesn’t add padding

```
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ofb -e -in f3.txt -out f3o.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ofb -d -in f3o.txt -out ff3.txt
enter aes-128-ofb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010 1010 1010 1010 1010 1010 1010 1010 1010
0000020
[04/04/23]seed@VM:~/.../08_ske$ hexdump f3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010
[04/04/23]seed@VM:~/.../08_ske$
```

CFB does not add padding

```
0000010
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -d -in f3o.txt -out ff3.txt
enter aes-128-cfb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -e -in f3.txt -out f3o.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-cfb -d -in f3o.txt -out ff3.txt
enter aes-128-cfb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump f3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010 1010 1010 1010 1010 1010 1010 1010 1010
0000020
[04/04/23]seed@VM:~/.../08_ske$
```

ECB adds padding

```
0000020
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ecb -e -in f3.txt -out f3o.txt
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
[04/04/23]seed@VM:~/.../08_ske$ openssl enc -aes-128-ecb -d -in f3o.txt -out ff3.txt
enter aes-128-ecb decryption password:
[04/04/23]seed@VM:~/.../08_ske$ hexdump ff3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010 1010 1010 1010 1010 1010 1010 1010 1010
0000020
[04/04/23]seed@VM:~/.../08_ske$ hexdump f3.txt
0000000 3231 3433 3635 3837 3039 3231 3433 3635
0000010
[04/04/23]seed@VM:~/.../08_ske$
```

CFB and OFB do not use padding since it pseudo generates random XOR with the plain text to generate the ciphertext, since we can use as many bits of random streams as we want, when don't need padding for these.

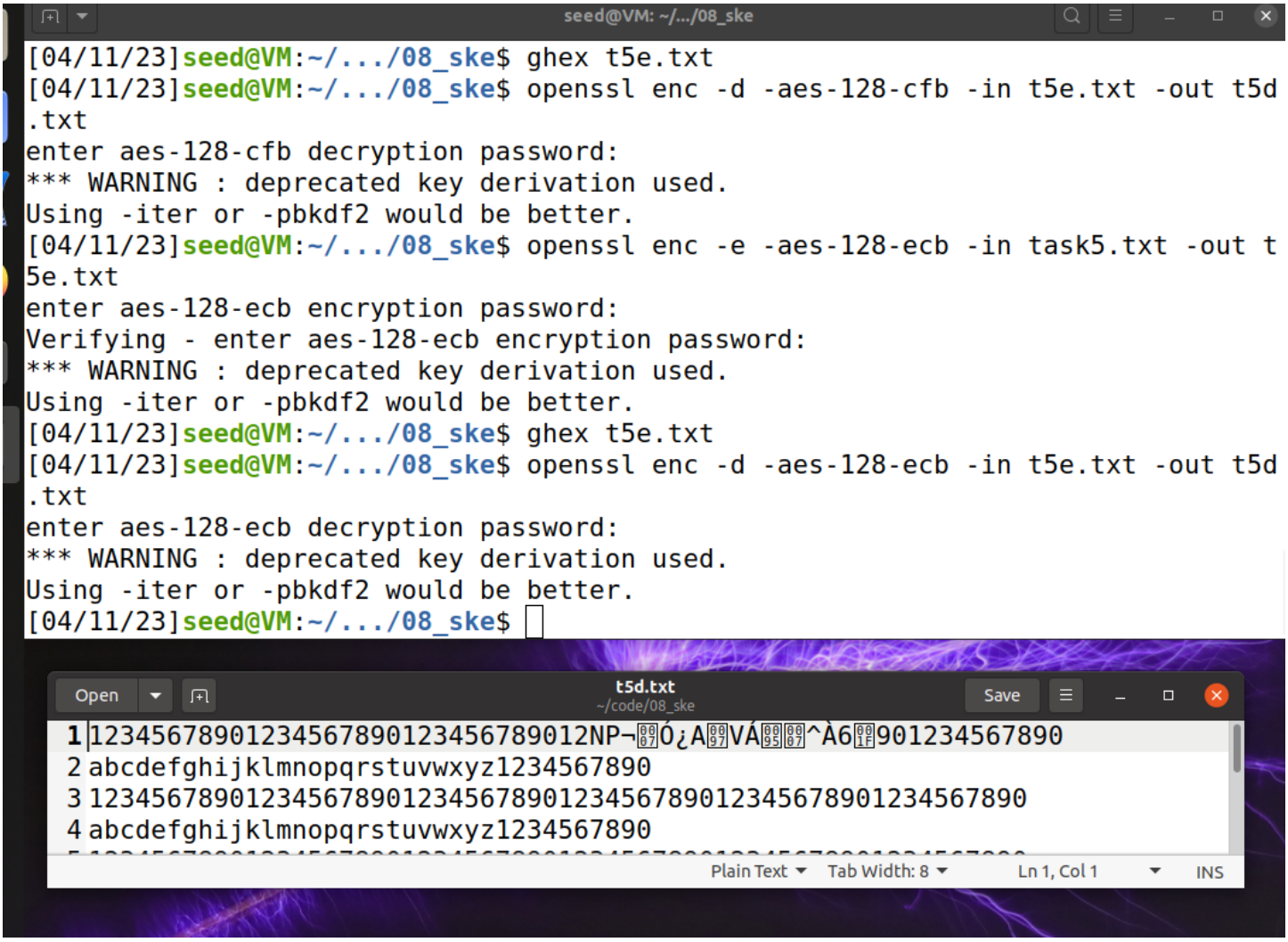
Task 5

Task 5.1

How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? I think we can recover all of the information from CFB and OFB since they do not use padding.

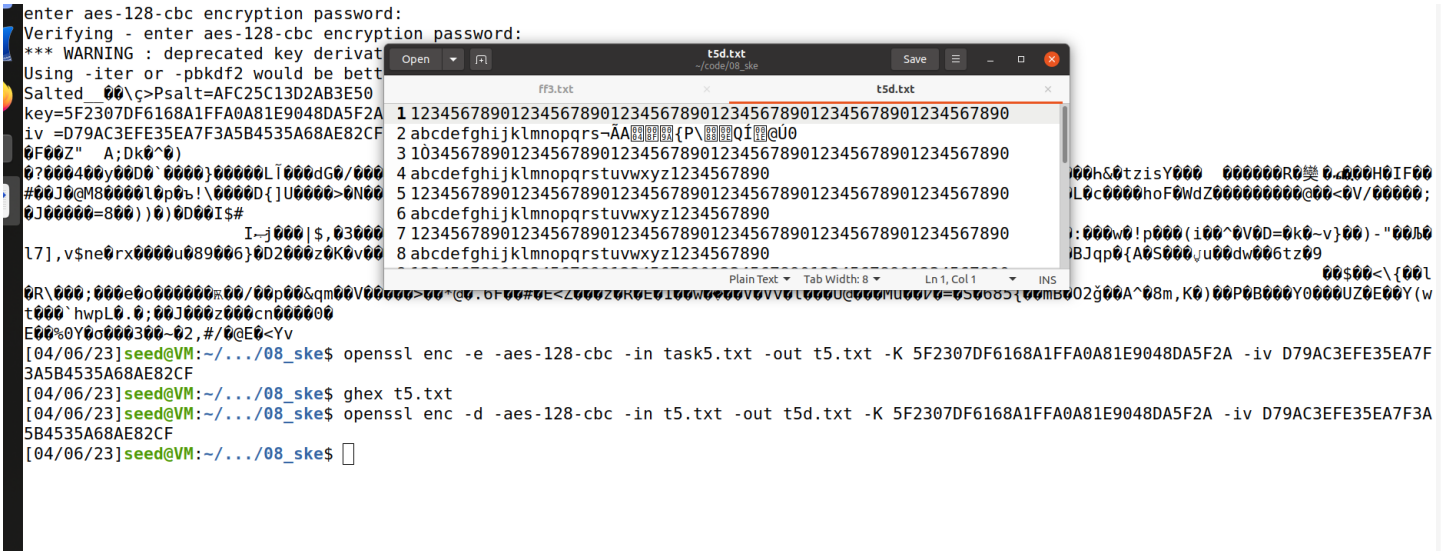
Task 5.2

Using ECB we can still most of the original data but the bit we changed couldn't decrypt properly

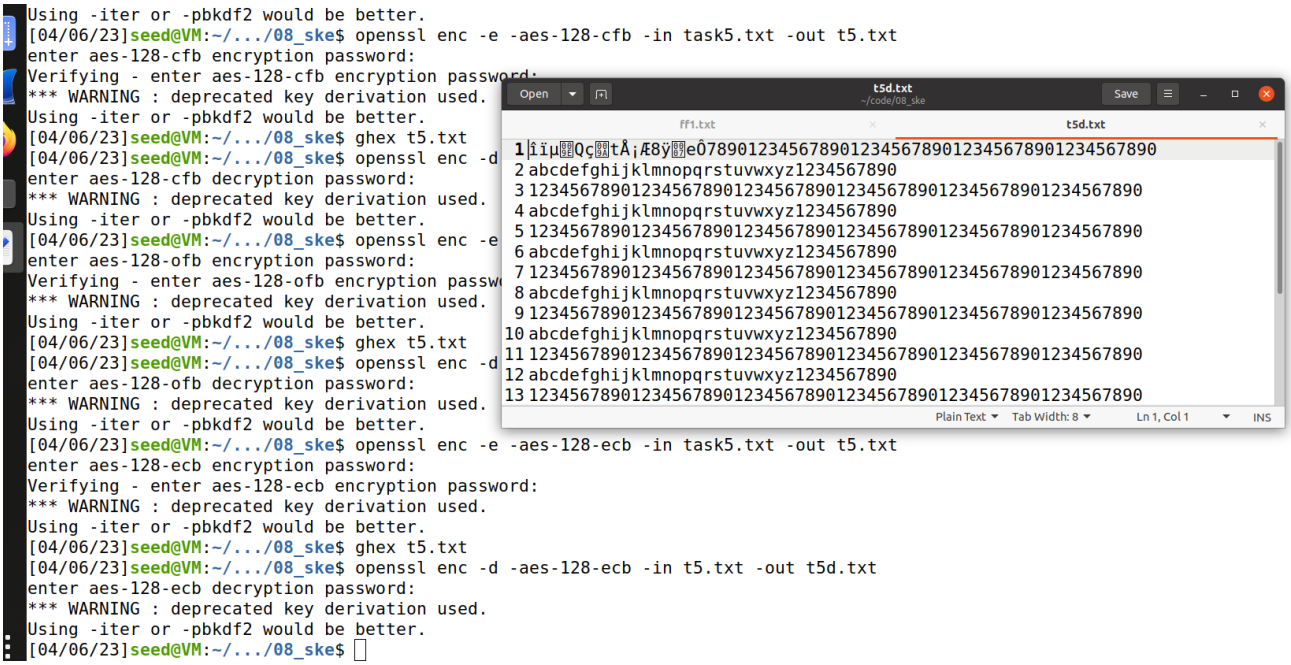


Task 5.3

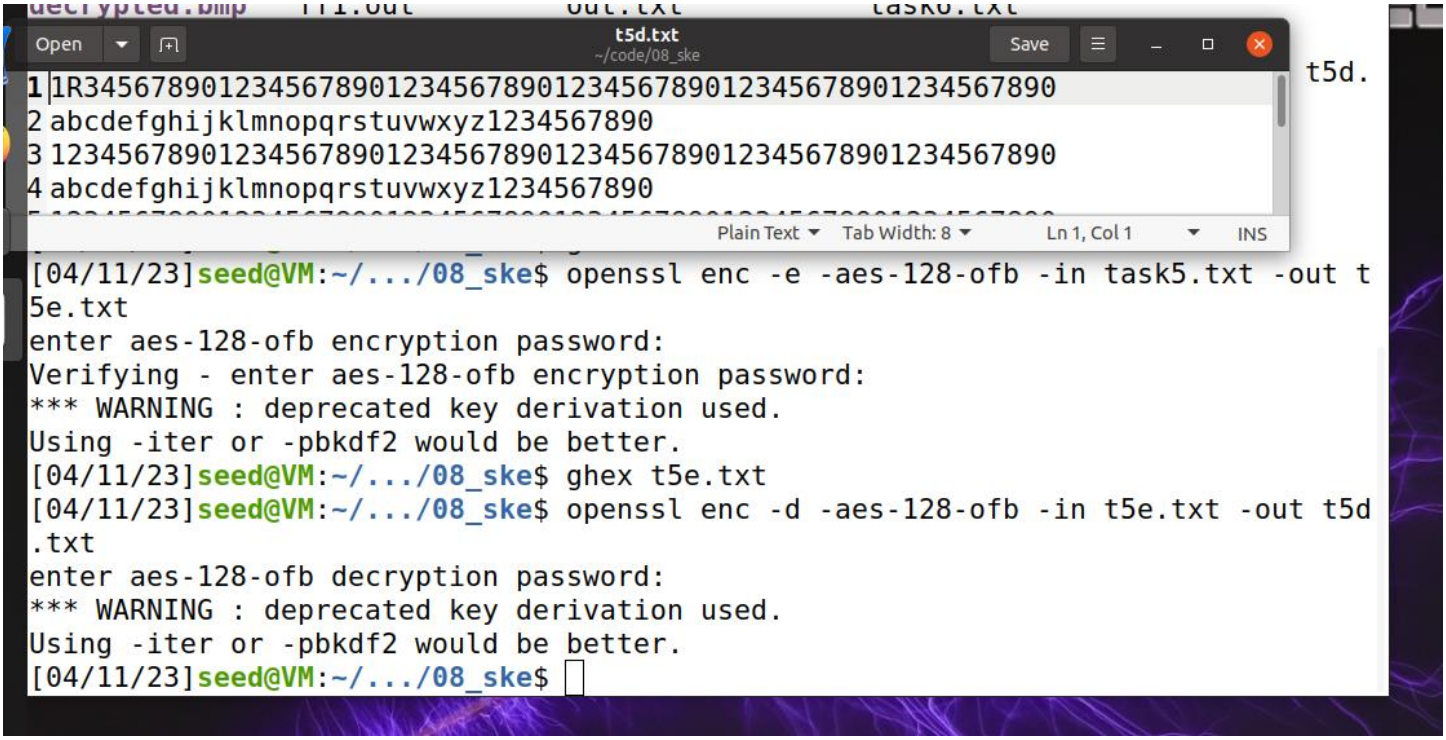
Using CBC, we can still most of the original data but the bit we changed couldn't decrypt properly.



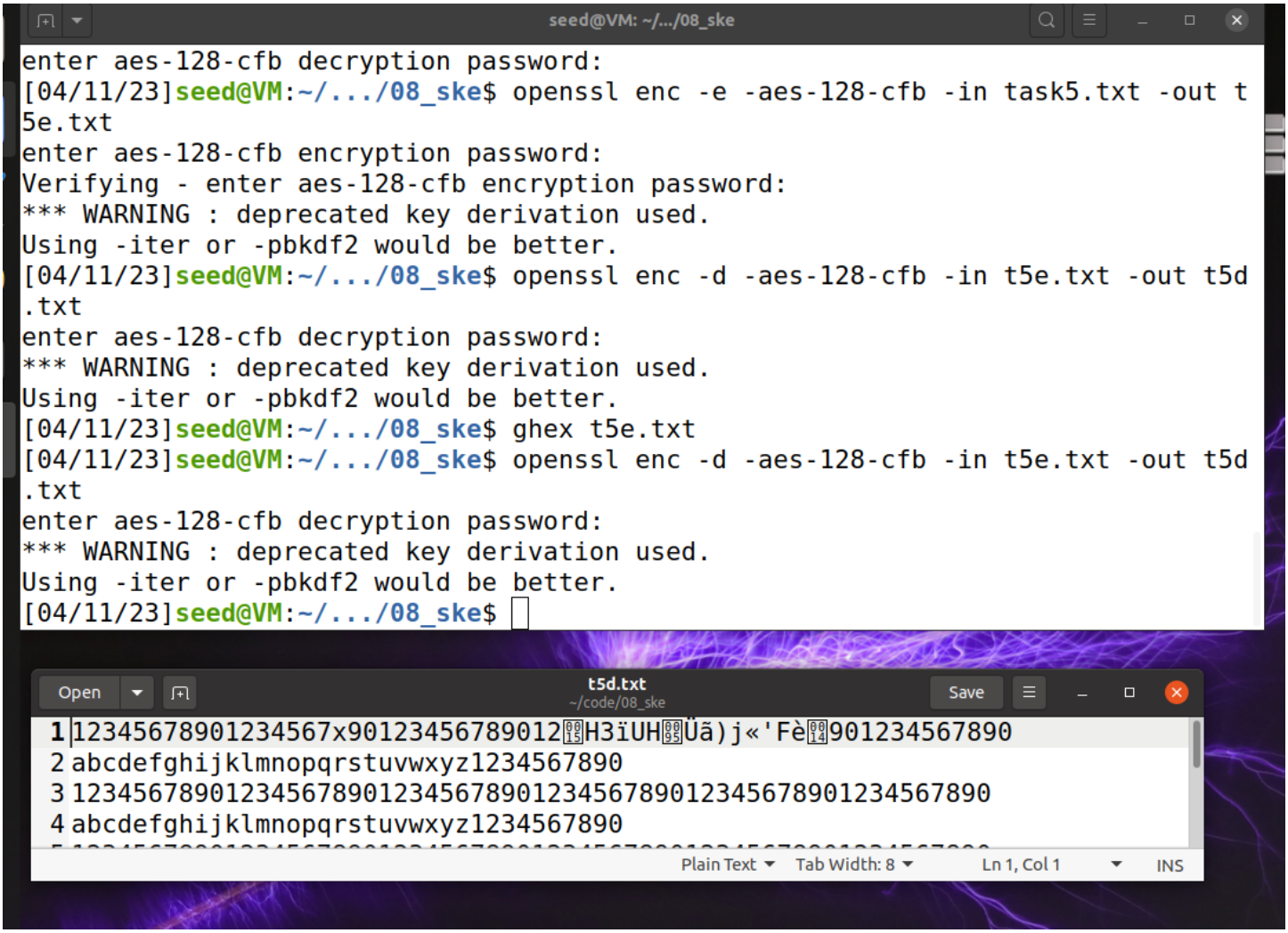
Using ECB we can still most of the original data but the bit we changed couldn't decrypt properly



Using OFB we can see that when we changed one byte in the encrypted .txt file and then unencrypted it, it only didn't decrypt that byte we changed.

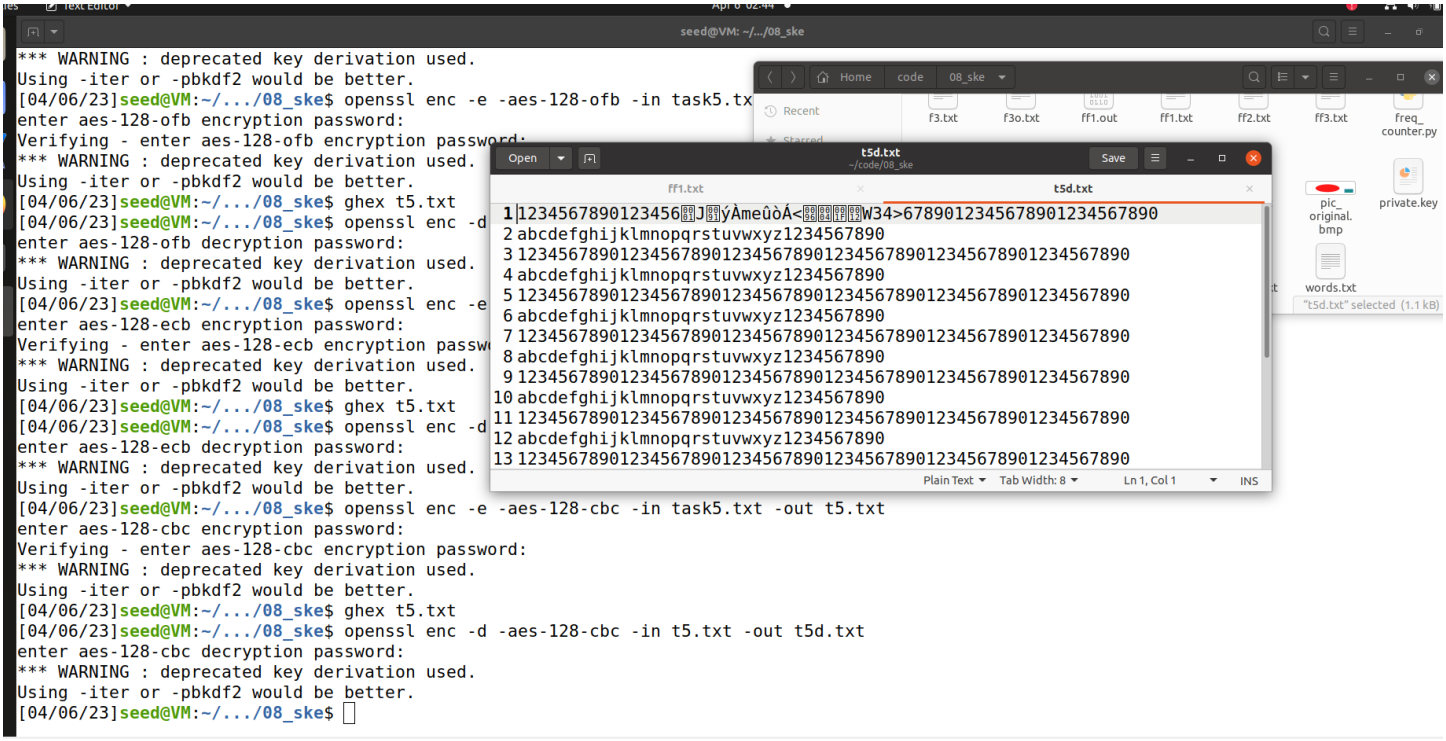


Using CFB we can see that changing one bit changed in the encrypted .txt, it changes multiple bytes, but we are still able to see most of the original data.



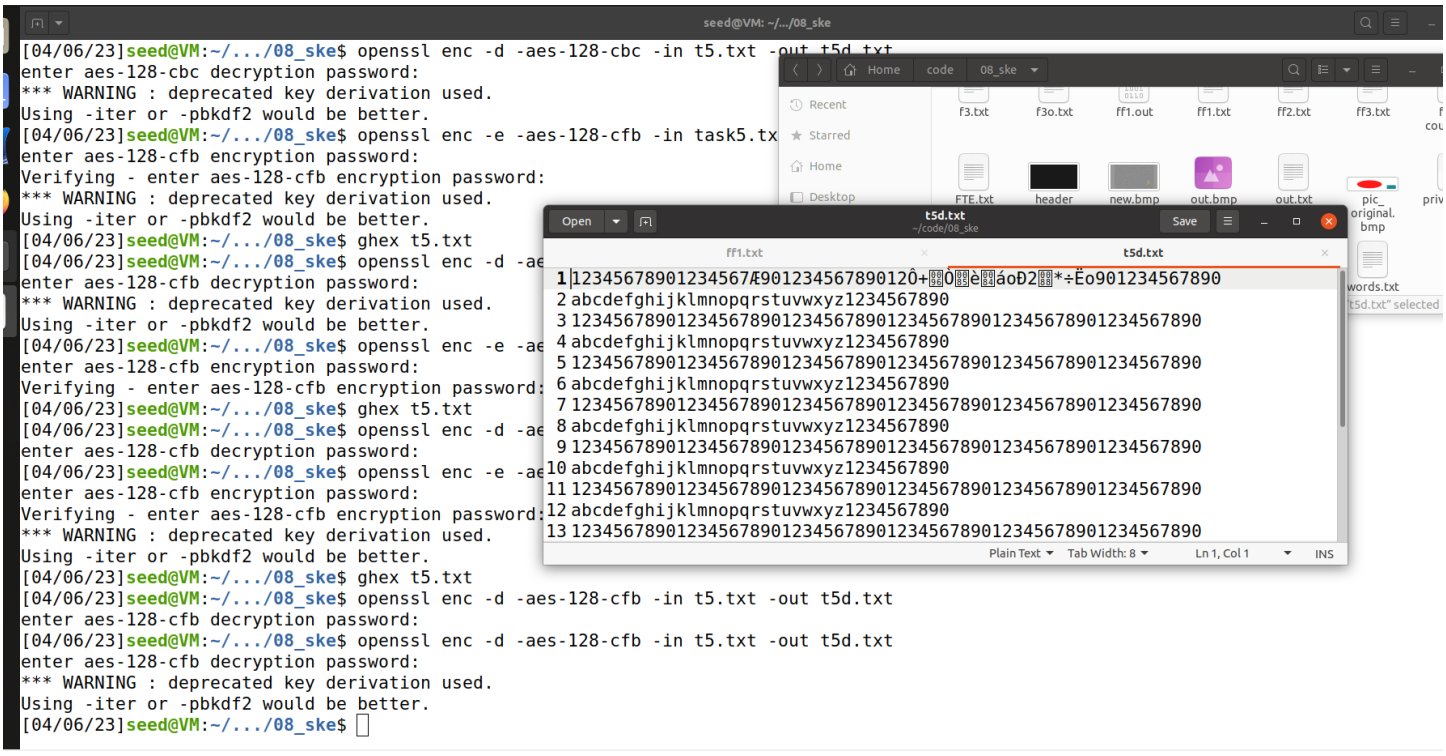
Task 5.3

Using CBC, the bit we changed didn’t decrypt but the ones we didn’t change decrypted just fine.



Task 5.4

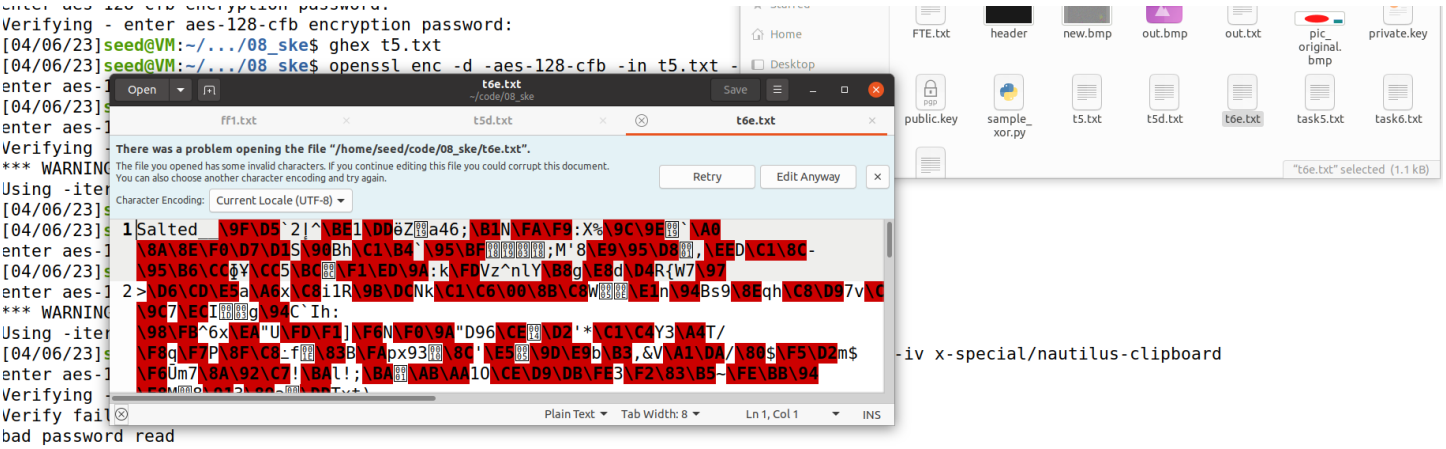
Using CFB we can see that changing one bit changed in the encrypted .txt, it changes multiple bytes, but we are still able to see most of the original data.



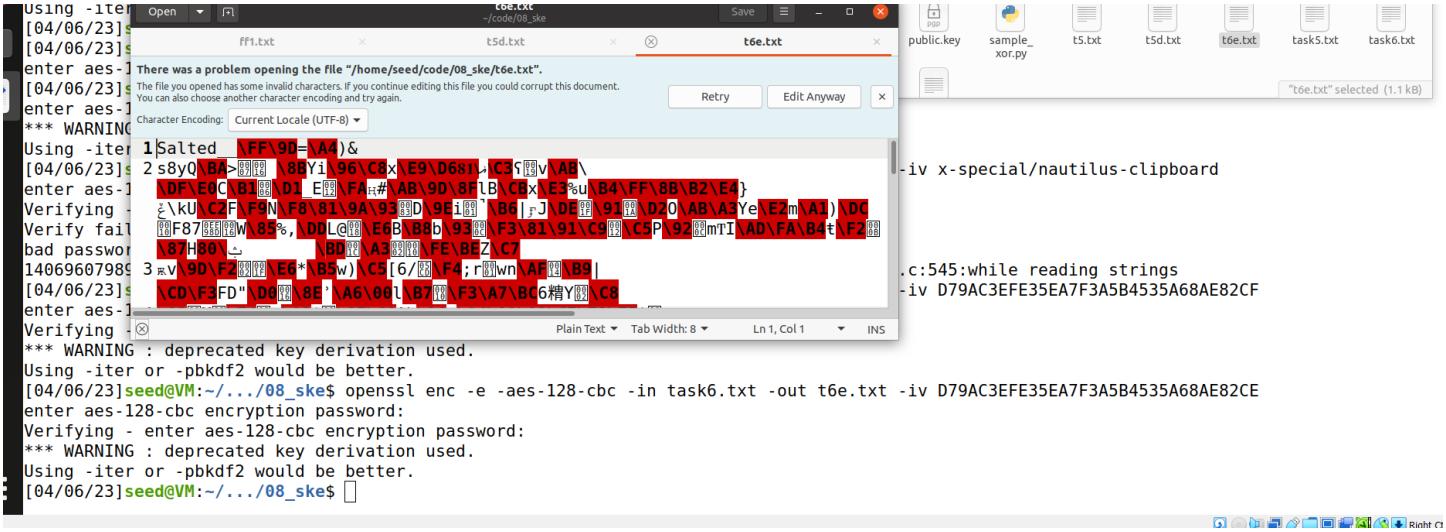
In conclusion for Task 5, using CFB and CBC, when we change one byte, it corrupts 2 blocks of data. When we use ECB and OFB, it only corrupts 1 blocks of the data.

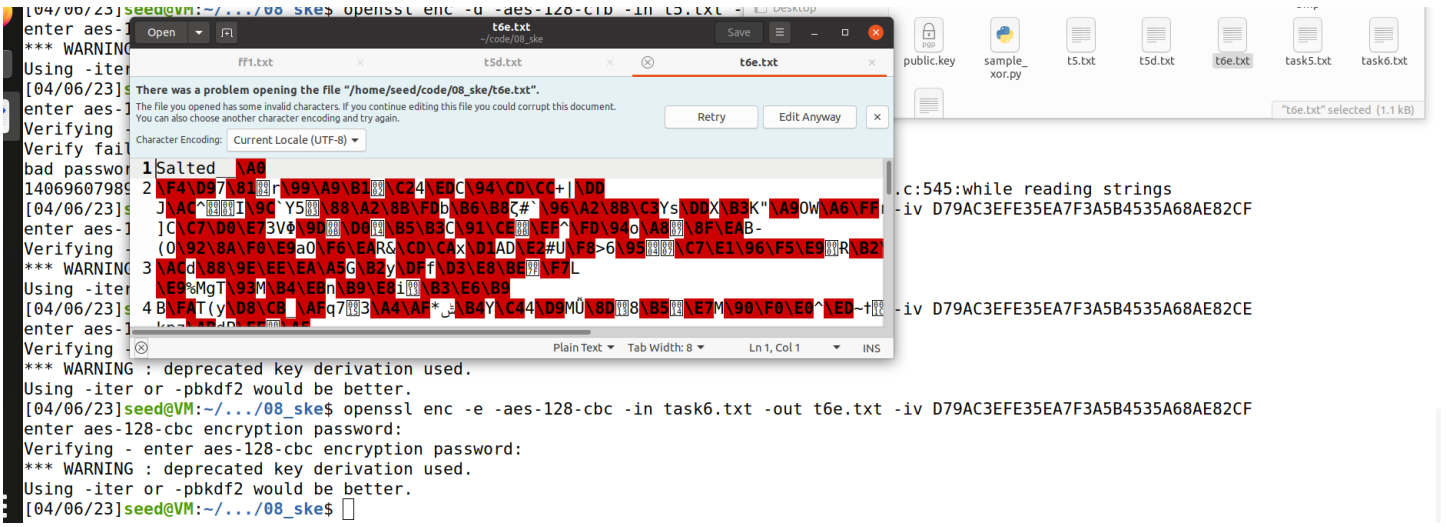
Task 6.

Task 6.1



Here is with a different iv, the encrypted text is different





Even with the same IV and password, the encrypted text was different.

Question: *Can they decrypt other (different) encrypted messages if the same IV is always used? Not in my experiments. The only time we could get the same encrypted file contents unencrypted the same is if we use both the same key and iv*