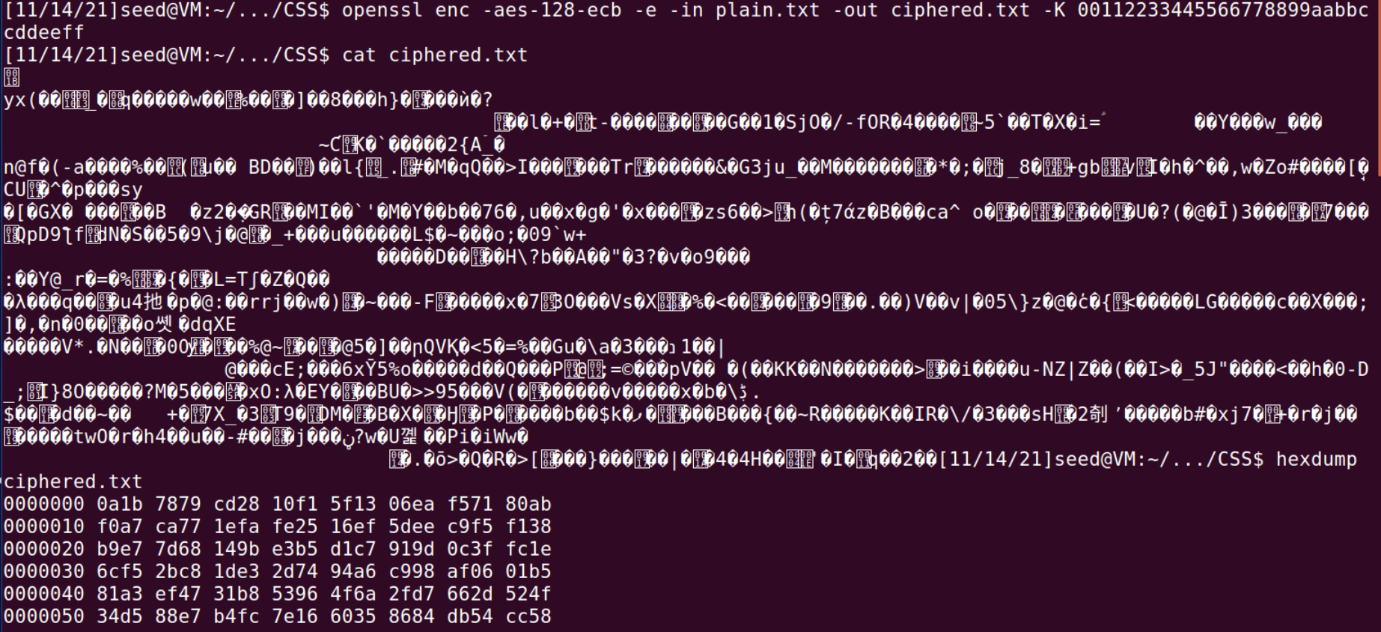
Experiment 3

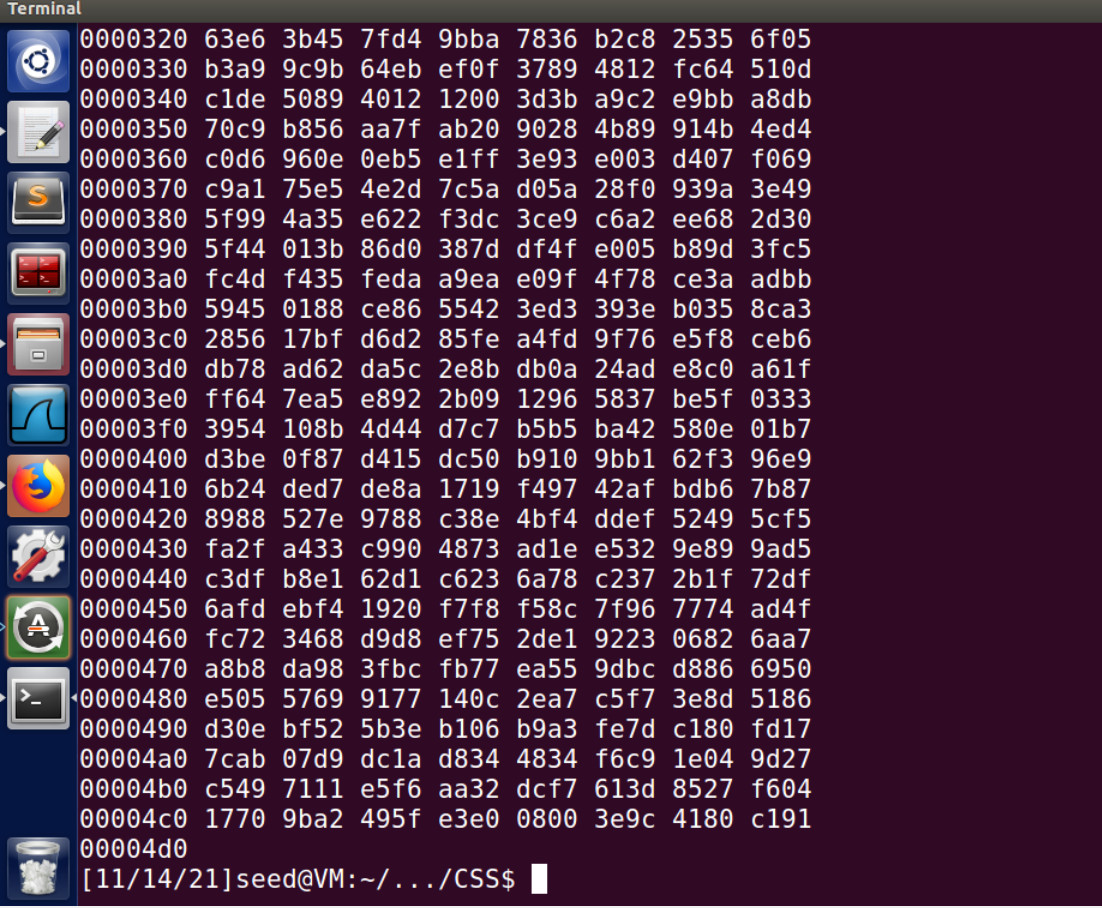
**Name: Abhishek Chopra UID: 2019130009**

**Batch A Subject: CSS**

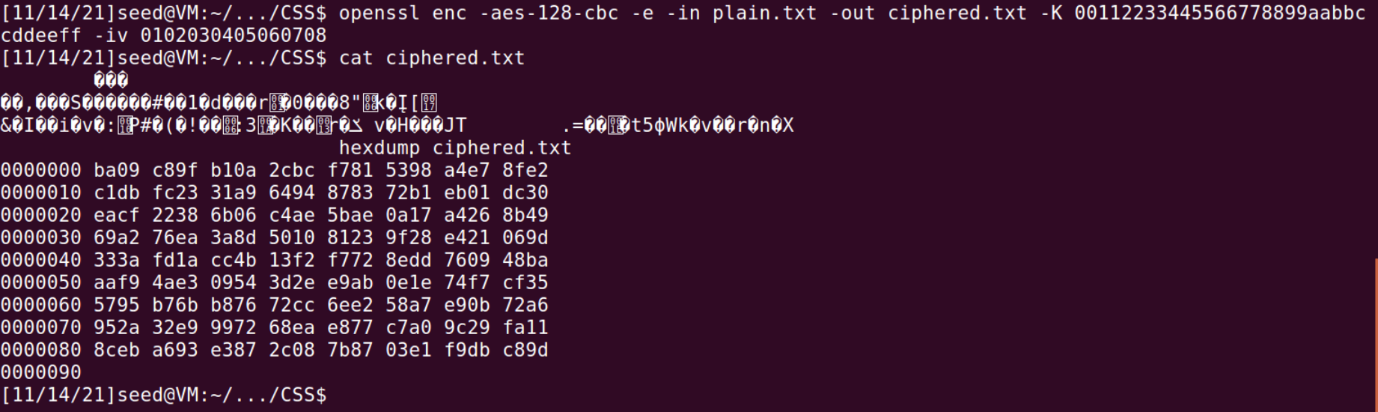
**Task 1: Encryption using different ciphers and modes**

-aes-128-ecb

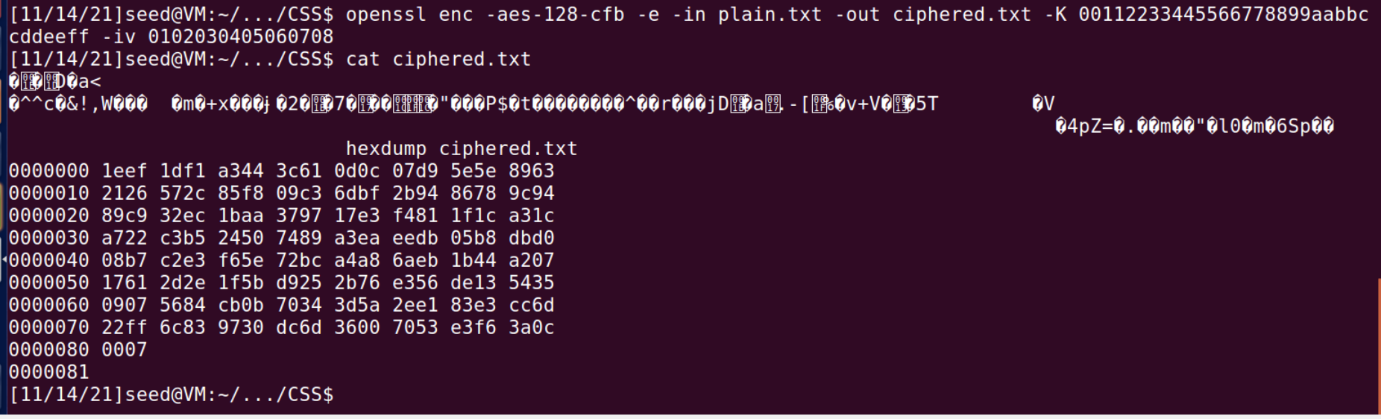
****

****

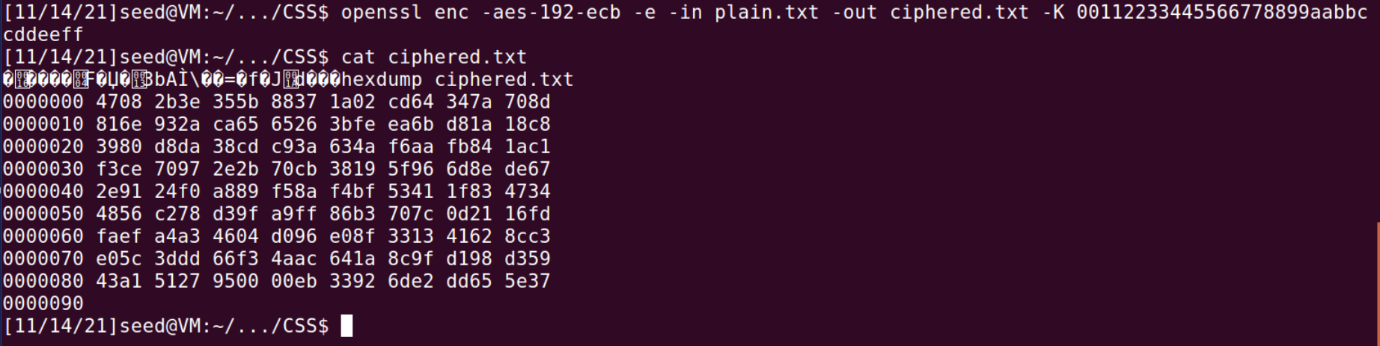
-aes-128-cbc

****

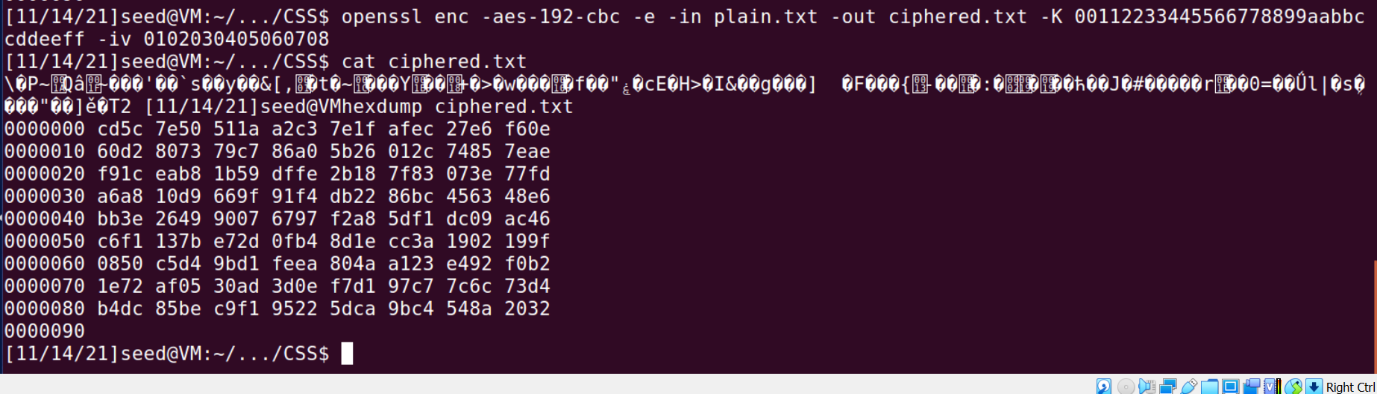
-aes-128-cfb

****

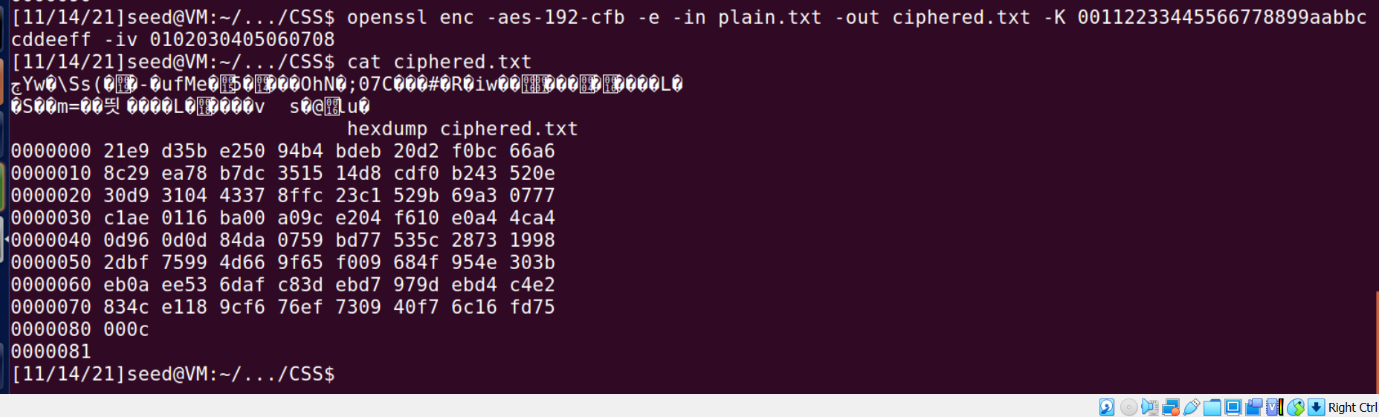
-aes-192-ecb

****

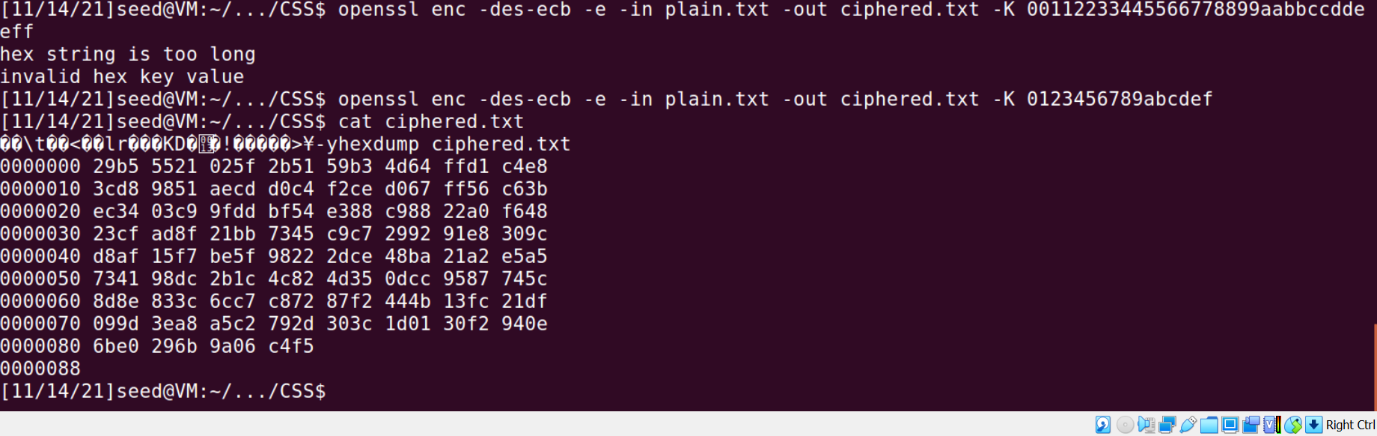
-aes-192-cbc

****

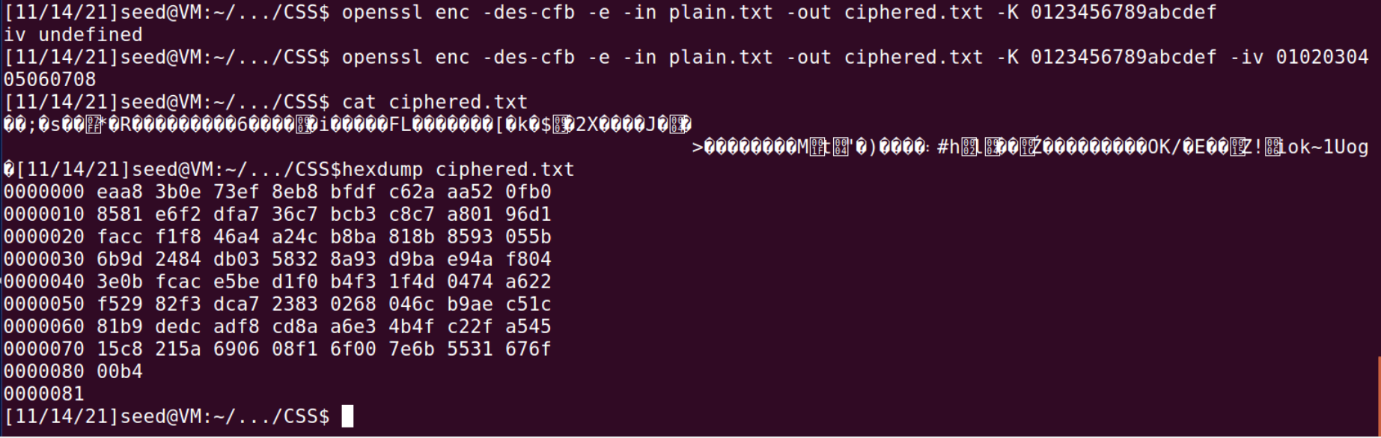
-aes-192-cfb

****

-des-ecb

****

-des-cfb

****

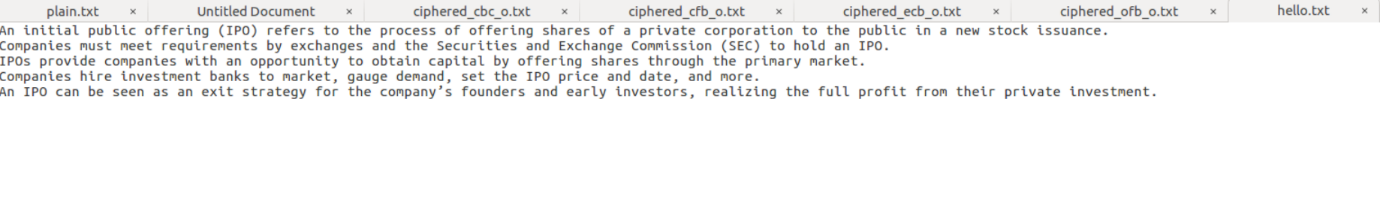
-des-cbc

****

**Task 2: Encryption Mode – ECB vs. CBC**

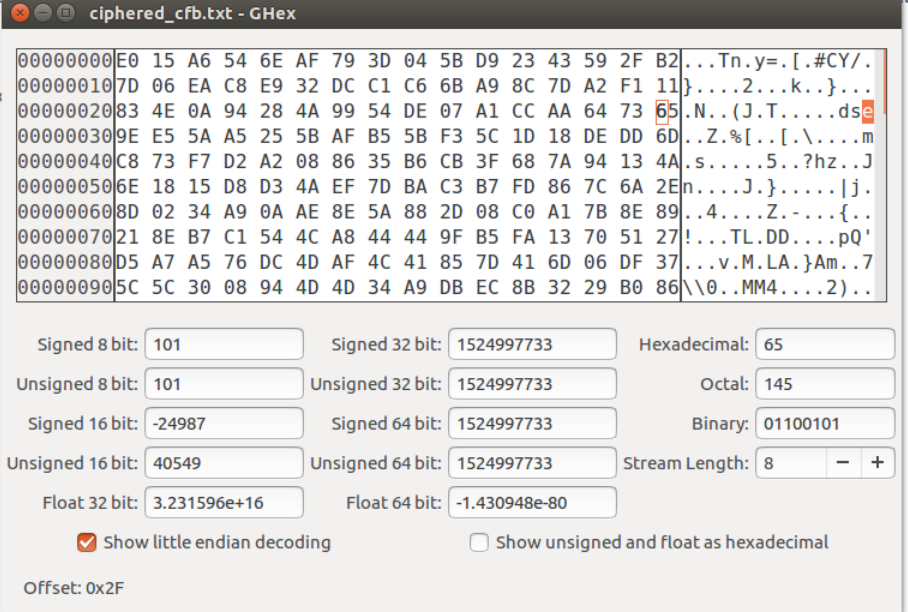
**Task 3: Encryption Mode – Corrupted Cipher Text**

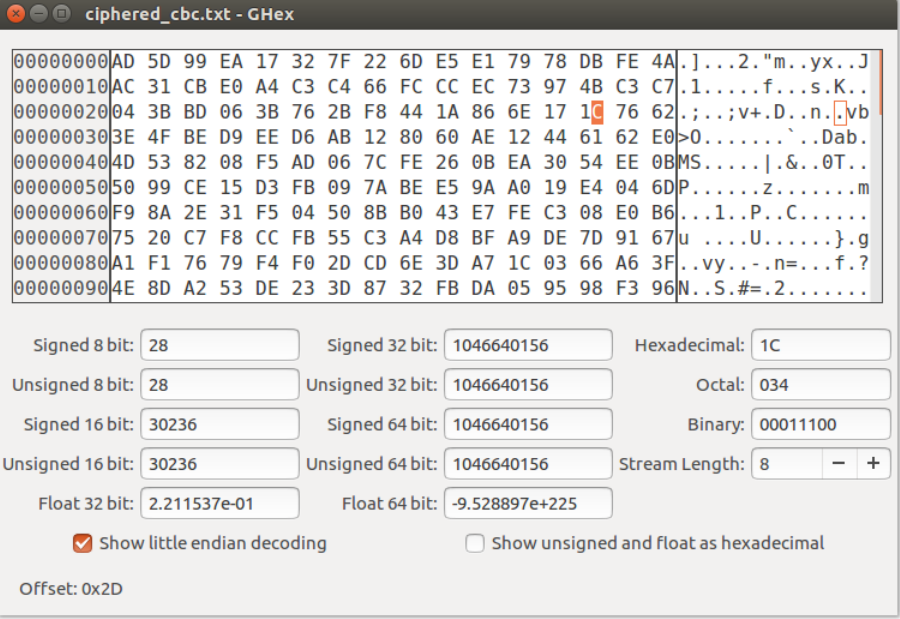
Original plain text:

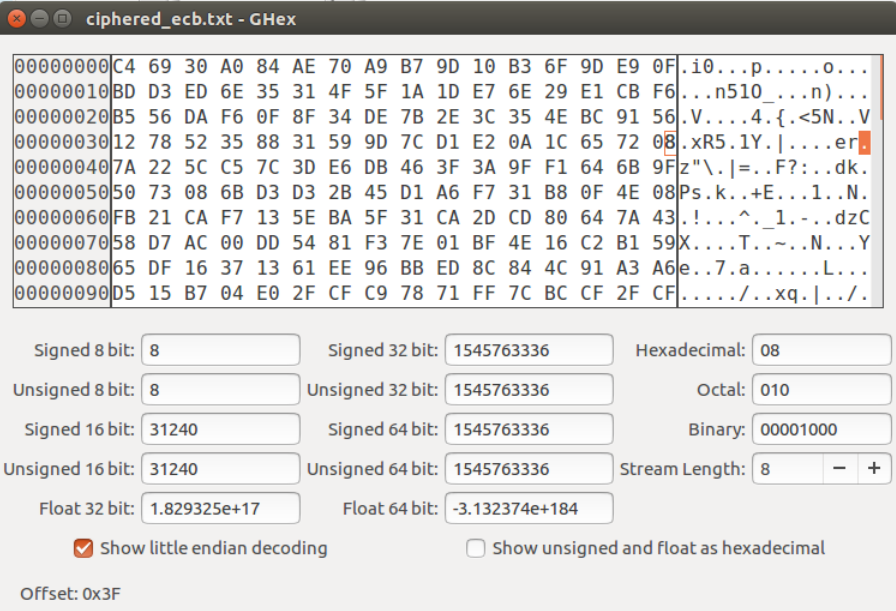
****

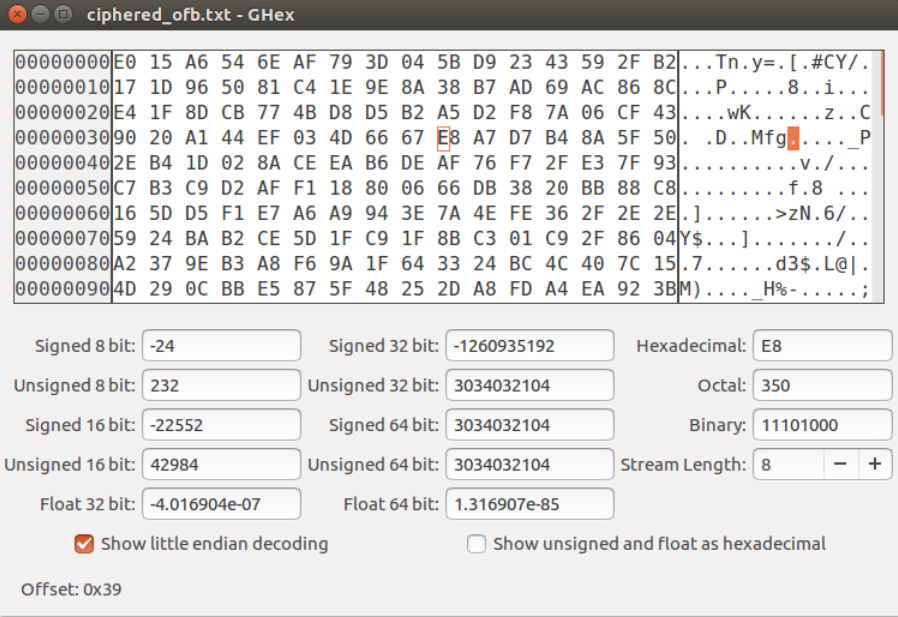
The above text is encrypted using aes-128 in 4 different modes: cbc, cfb, ofb and ecb.

After corrupting 30th byte of each cbc, cfb, ofb and ecb:

****

****

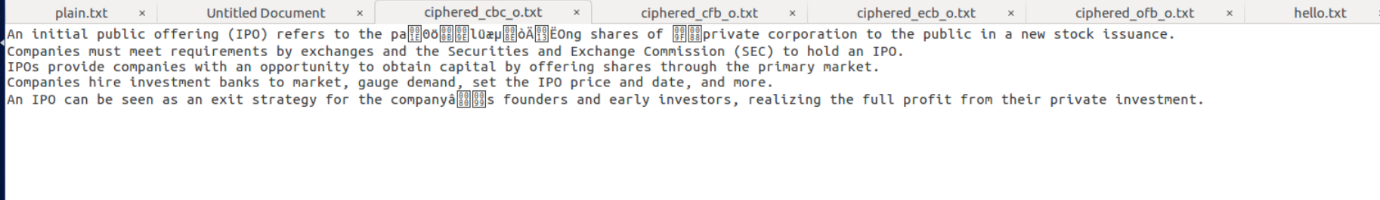
****

****

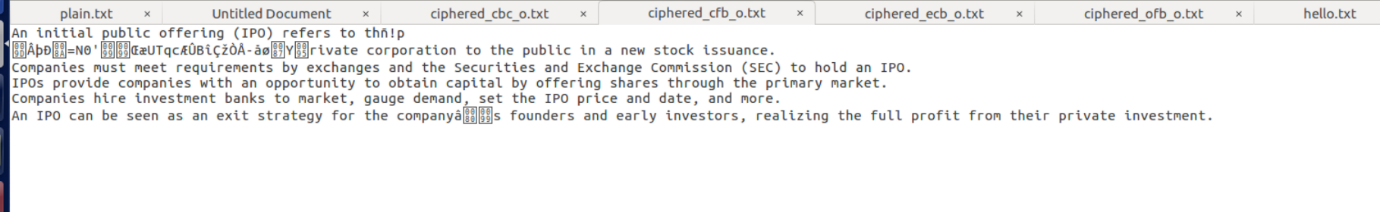
****

The decrypted text for the same is as follows:

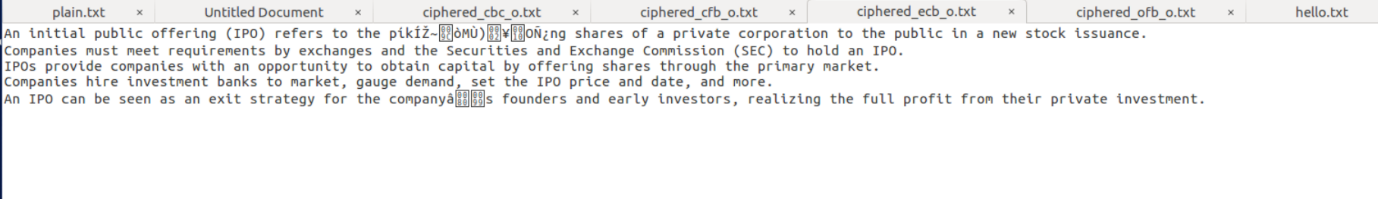
cbc:

****

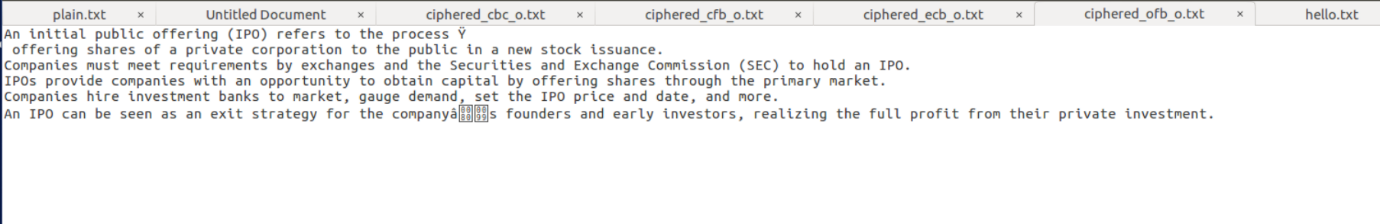
cfb:

****

ecb:

****

ofb:

****

**Inference:**

In the case of ecb mode encryption since we know that each plaintext block is encrypted separately similarly decrypted separately therefore only the block containing the corrupted byte gets corrupted there is no difference in the rest of the text. An advantage of this mode is that since there is no dependency upon other blocks, the encryption and decryption

can be carried out by many threads simultaneously.

As we know in the cbc mode the chaining between input and output takes place , the block of plain text is XOR ed with the encrypted block of the previous pass and thus the chain

continues. So I inferred and understood that if one bit of the actual plain block is corrupted then the entire chain will have corrupted bits leading to a totally corrupted text, but if say

only one bit of the ciphertext is damaged only two received plaintext blocks will be damaged hence making it possible to recover the original data.

The cfb mode is similar to the cbc mode but the only difference being that the ciphertext from the previous round needs to be encrypted and then added to the plaintext bits. Here the same encryption algorithm needs to be used for both encryption and decryption. I observed that after corrupting one ciphertext bit only the two consecutive plaintext blocks

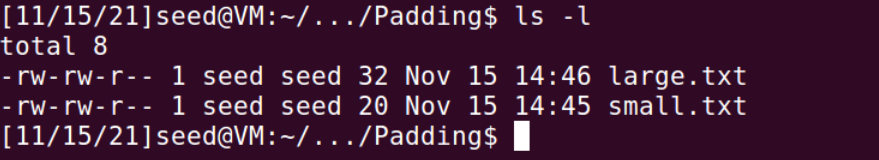
will be damaged.

In case of ofb mode the keystream bits are created that are used for the encryption of subsequent data blocks and due to this the working of this mode is similar to a typical stream cipher. In this case I observed that if one bit of a plaintext or ciphertext message is damaged, only one corresponding ciphertext or respectively plaintext bit is damaged as well.

**Task4: Padding**

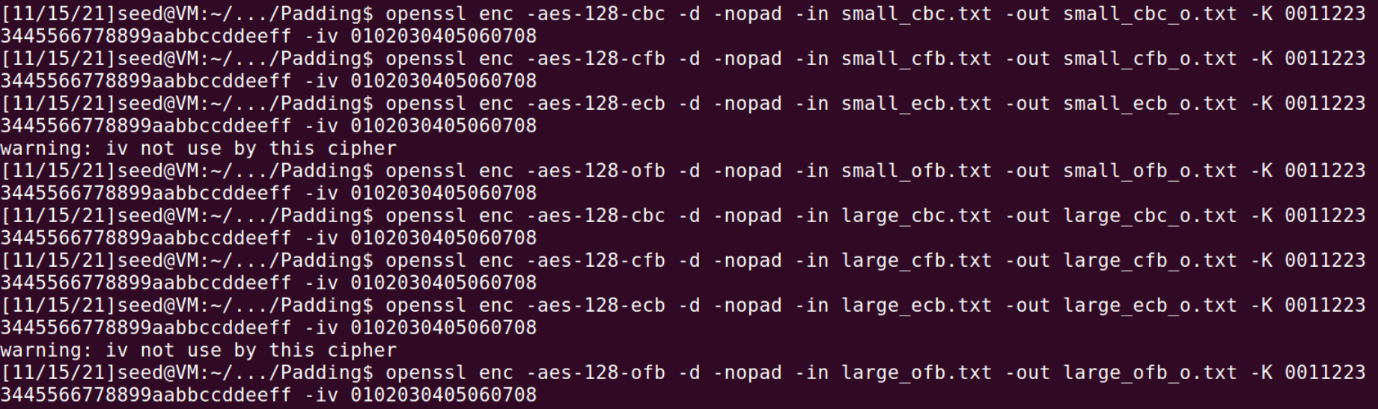
Two files namely large.txt and small.txt are made with each having 32 and 20 bytes respectively.

The files are encrypted using aes-128 in 4 modes namely: ecb, cbc, cfb, ofb

****

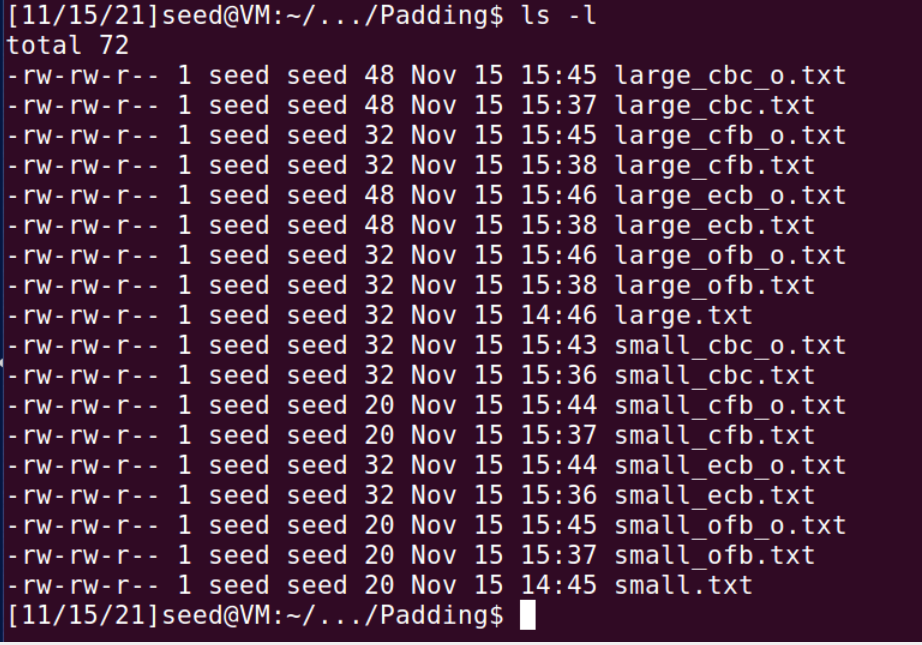
To confirm that openssl uses PKCS5 padding, decrypt the encrypted file with option–nopad.

This option turns off the standard block padding. Normally, the padding is included by default during encryption, so if I use the nopad option, I can see the padding in the decrypted file.

****

****

Finally we can see the decrypted and encrypted files as follows:

****

**Inference:**

The screenshot above shows that the size of CBC and ECB encrypted files with the nopad option is 12 bytes more for the 20 bytes file and 16 bytes larger for the 32 bytes file, however the size of OFB and CFB decrypted files is the same.

Result:

1. The experiment shows that padding is needed for ECB and CBC encryption modes. This can be because ECB and CBC are block ciphers and for a block cipher length of input must be an exact multiple of block length. If this is not the case then padding must be added to make it so. This padding is removed after decrypting.

2. In OFB and CFB, the padding is not required because they are stream ciphers and the

ciphertext is always the same length as plain text.

**Task 5: Programming using the Crypto Library**

**Code:**

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad

plaintText = b"This is a top secret."

cipherText = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"

myFile = open('words.txt', 'r')

lines = myFile.readlines()

words = [str.strip(line) for line in lines]

arr = []

for word in words:

if len(word)<16:

word=word.lower()

key=word.encode()+b' '\*(16-len(word))

getCipher=AES.new(key, AES.MODE\_CBC, iv=bytes.fromhex('0'\*32))

ciphertext=getCipher.encrypt(pad(plaintText, AES.block\_size))

match="Not Matched"

if bytes.hex(ciphertext)==cipherText:

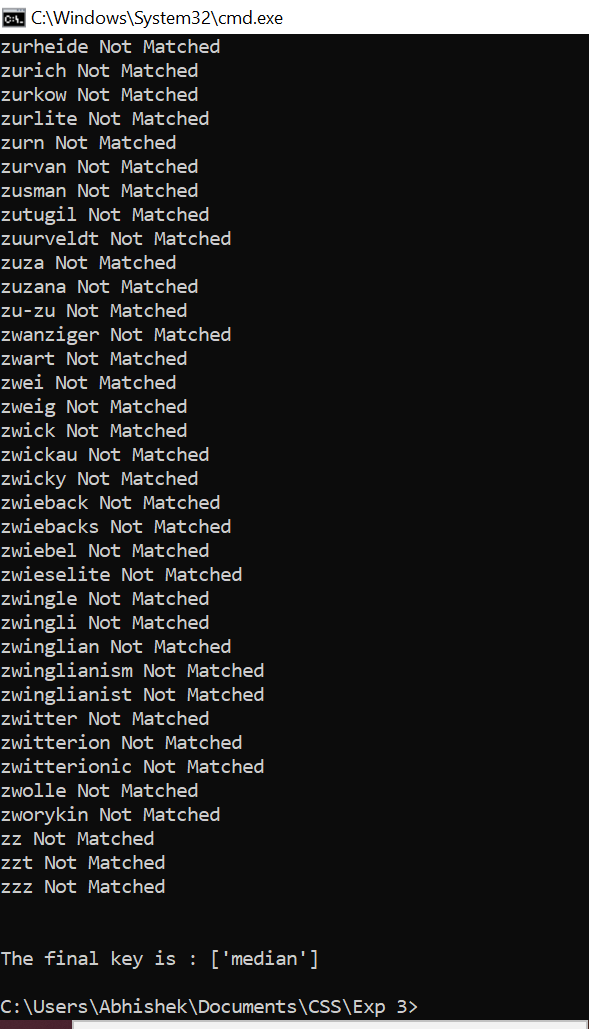
match="Matched"

arr.append(word)

print(word,match)

print("\n\nThe final key is :",arr)

**Output:**

****

The key used to encrypt is median.

**Conclusion:**

AES, DES are symmetric key algorithms using the same keys to encrypt and decrypt the data.

ECB mode of encryption is the weakest form of encryption in comparison to CBC, CFB and OFB.

I could conclude from the experiment that ECB and CBC use padding while encryption while the other two don’t. This proves that ECB and CBC are block ciphers while CFB and OFB are stream ciphers.

I learned how different modes react to a corrupted bit of a cipher text. The best decryption in such a case is provided by OFB where only the corrupted bit of cipher text is affected while encrypting.

I could conclude from this experiment that, If I have the plaintext, ciphertext and iv known, I can easily find the key using brute force method.

**Github Link:**

<https://github.com/AbhishekC20001/CSS-Lab-2019130009>