**CRYPTO ENCRYPTION LAB**

**MANHATTAN UNIVERSITY**

**ASTHA PATEL**

**Task 1: Frequency Analysis**

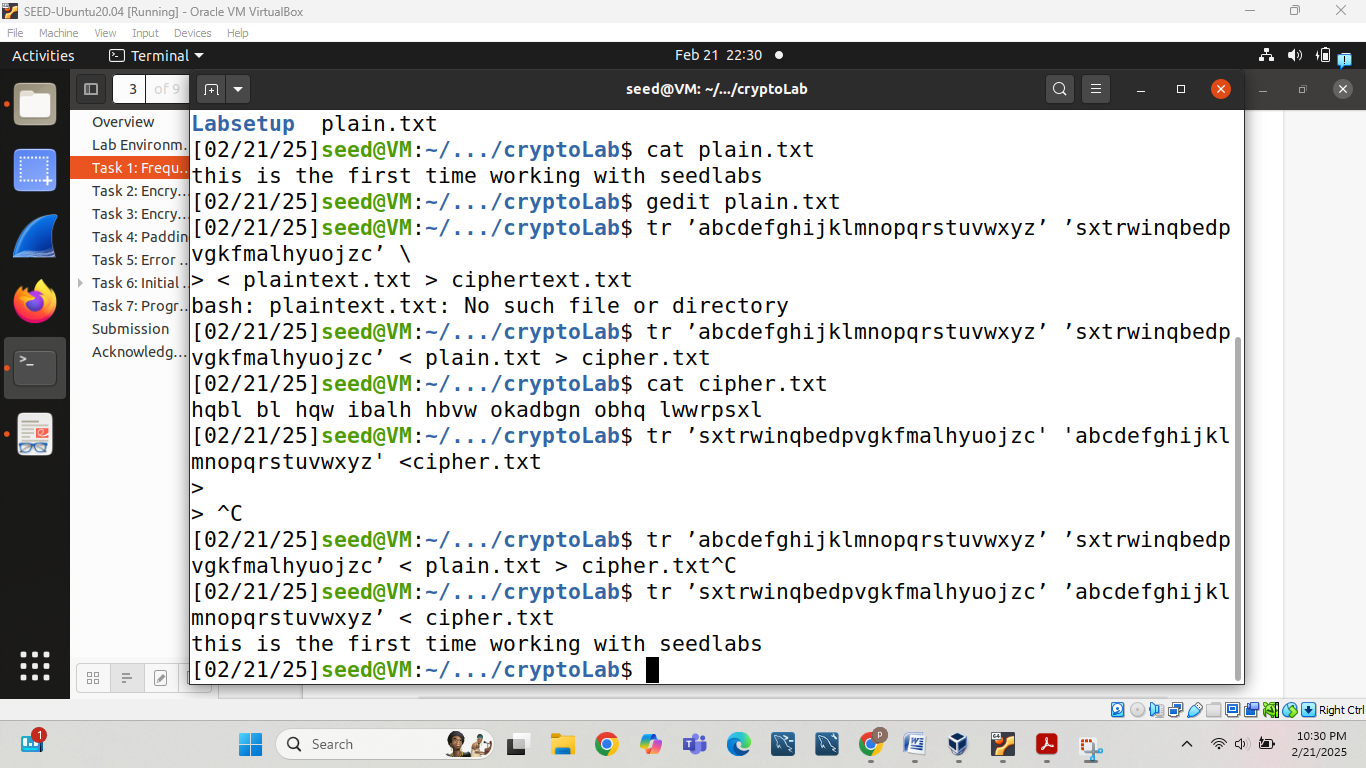
First we will create cryptoLab folder and insert Labsetup folder and also make one text file: plain.txt and then we will encrypt this text using following command:

**tr ’abcdefghijklmnopqrstuvwxyz’ ’sxtrwinqbedpvgkfmalhyuojzc’ < plain.txt > cipher.txt**



After then for decryption we will use Substitution Cipher where we write excetly opposite than encryption here we write key first and data later as follows:

**tr ’sxtrwinqbedpvgkfmalhyuojzc’ ’abcdefghijklmnopqrstuvwxyz’ < cipher.txt**

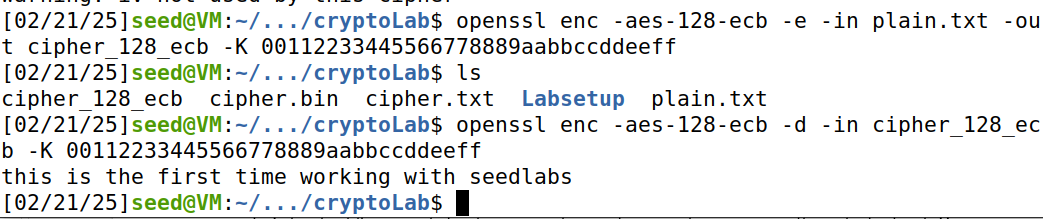


To highlight any letter we want to see in cipher text we will use following command:

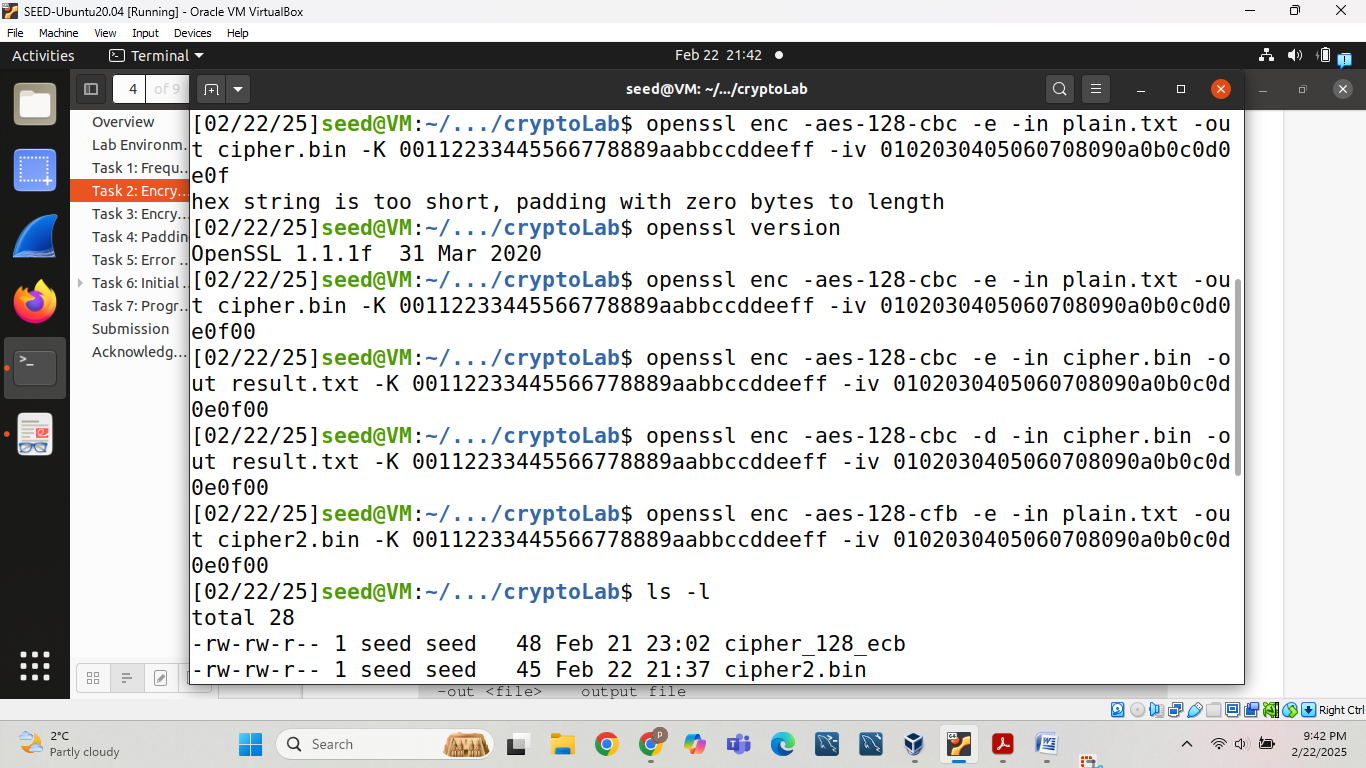
**grep a cipher.txt**

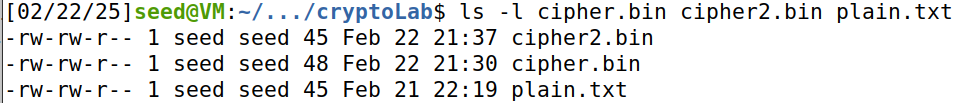
**Task 2: Encryption using Different Ciphers and Modes**

Here as we can see we can use ‘openssl enc –aes-128-ecb –d –in cipher\_128\_ecb –k 00112233445566778889aabbccddeeff’ to decrypt data as shown below:



Now, we will perform openssl command for different cipher type:





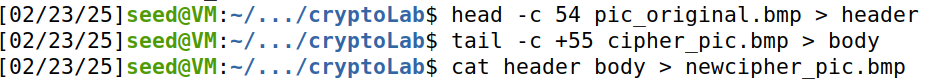
Here we can see their size for cipher is bigger than cipher2 and plain text.

**Task 3: Encryption Mode – ECB vs. CBC**

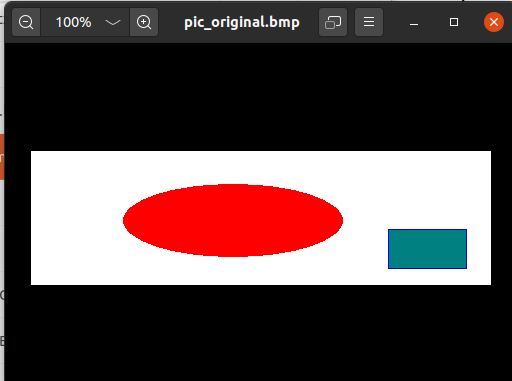
First we will copy our pic\_original.bmp from Labsetup folder and then perform following operation on picture first we will encrypt this pic using following command:

Screenshot 2025-02-23 163439.png

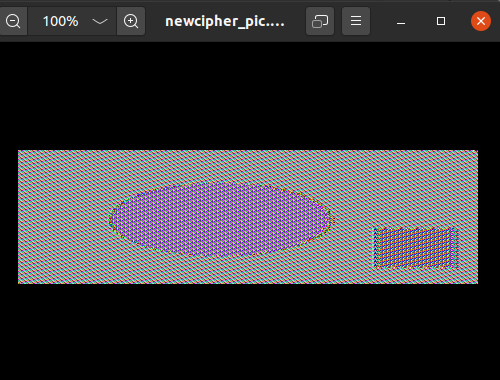
Then we will perform following operation on pic:



Original picture:

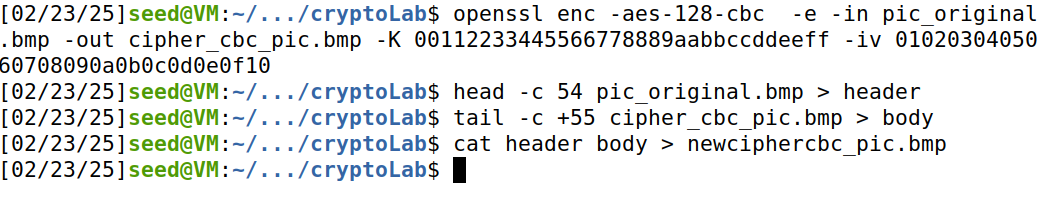


The output encrypted Image displayed as:

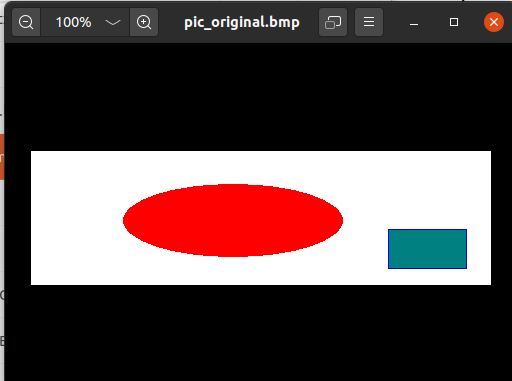


It seems similar to the original picture in some way. Because we break the file into blocks of size 128 bit, and the use AES algorithm to encrypt each block. If two blocks are the same in the original picture, they will remain identical in the encrypted one.

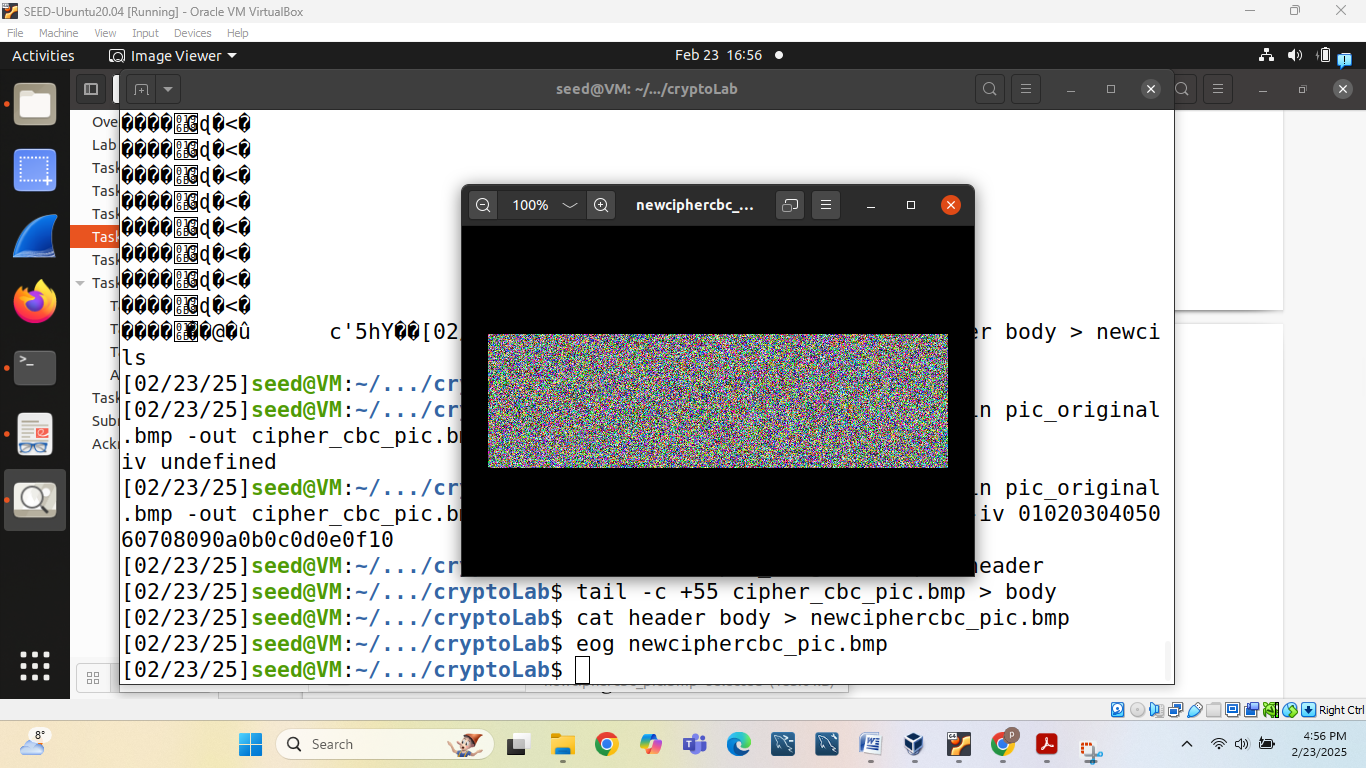
For CBC we are unable to identify original picture from encrypted one which will clear after seeing following details:



Original image:

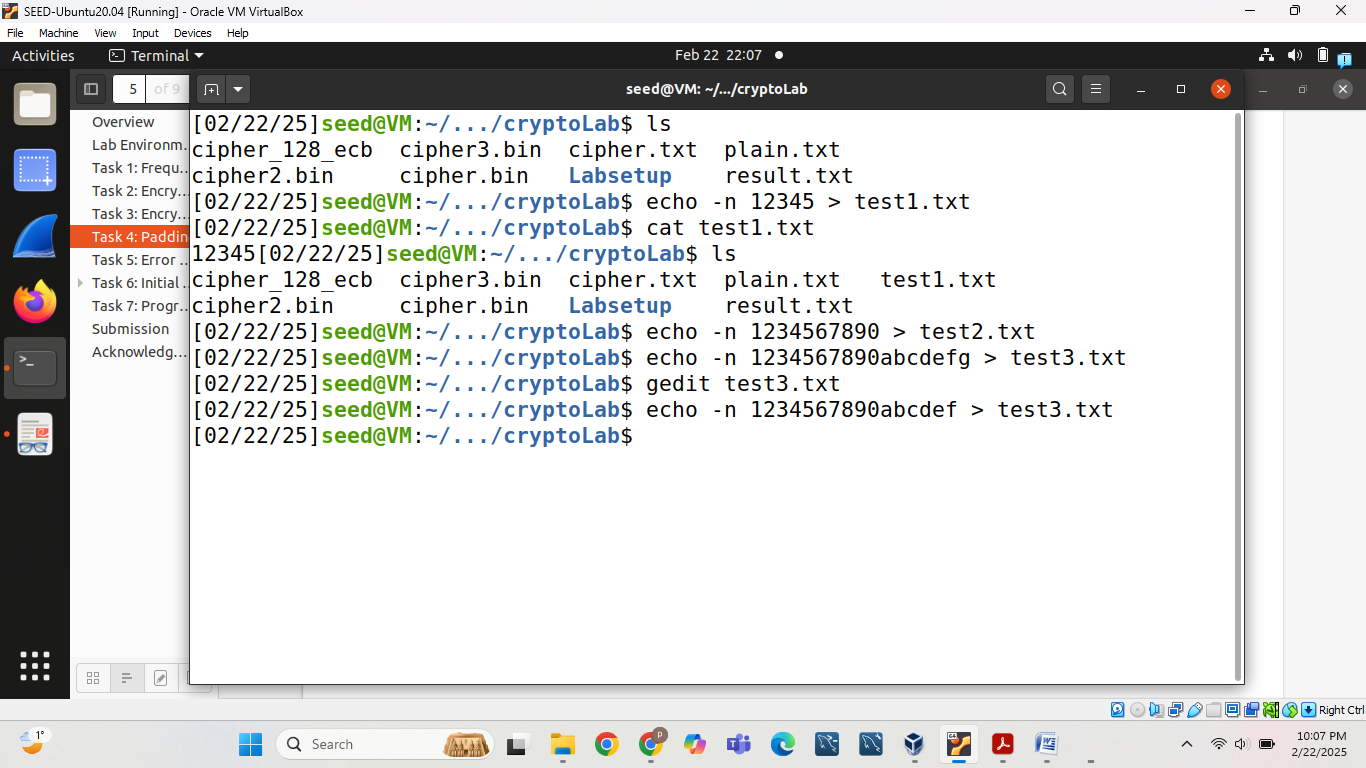


The output encrypted Image displayed as:



**Task 4: Padding**

To perform all the operation 1st we will create 3 files named test1.txt, test2.txt and test3.txt with the different byte sizes 5 bytes, 10 bytes and 16 bytes respectively.



We can use 4 different modes to encrypt a file which are follows:

ECB (Electronic Code Book)

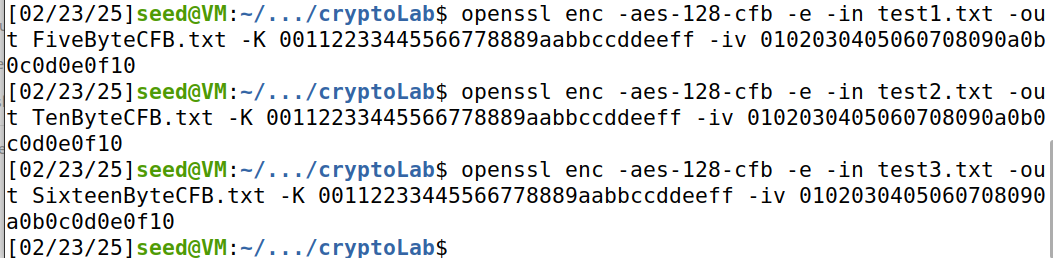
CBC (Cipher Block Chaining Mode)

CFB (Ciphertext feedback)

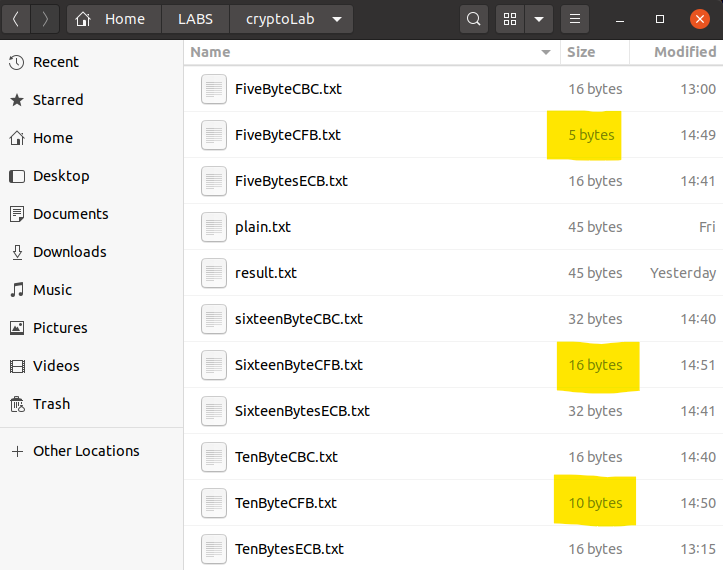
OFB (output feedback)

(1) First we will use **CFB mode** to encrypt a file where –e will use for encryption. Our command for encryption is follows:

openssl enc –aes-128-cfb –e –in test1.txt –out FiveByteCFB.txt –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

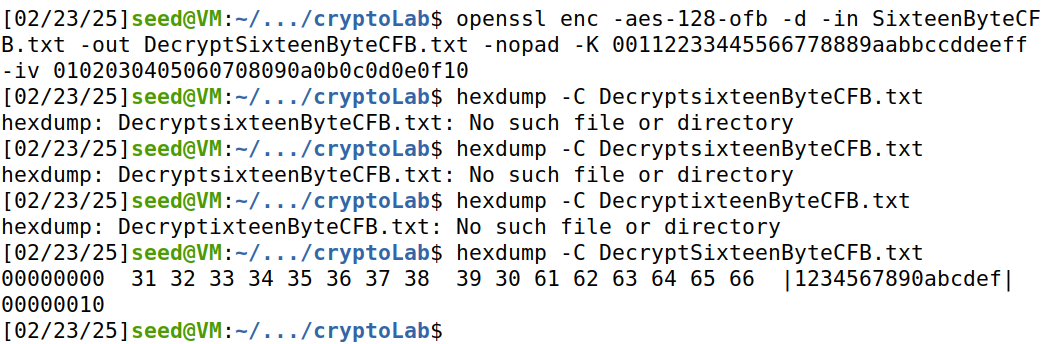


Size of the encrypted file is 5 bytes for 5 bytes data. Size of the encrypted file is 10 bytes for 10 bytes data and 16 bytes for 16 bytes data which is clear after watching following image:



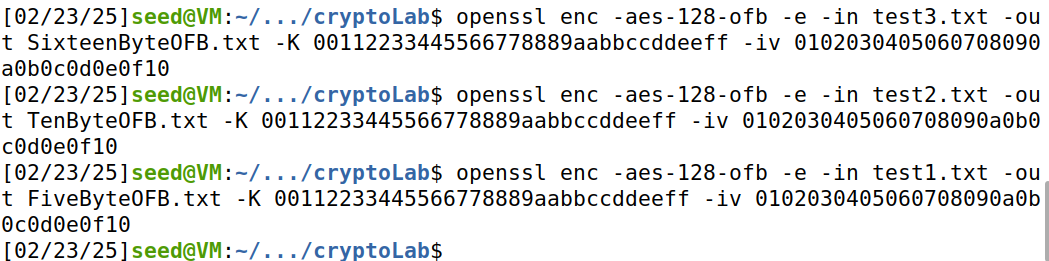
Now we will use **CFB mode** to decrypt a file where –d will use for decryption so our command for decryption follows:

openssl enc –aes-128-ofb –d –in SixteenByteCFB.txt –out DecryptSixteenBytesCFB.txt -nopad –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

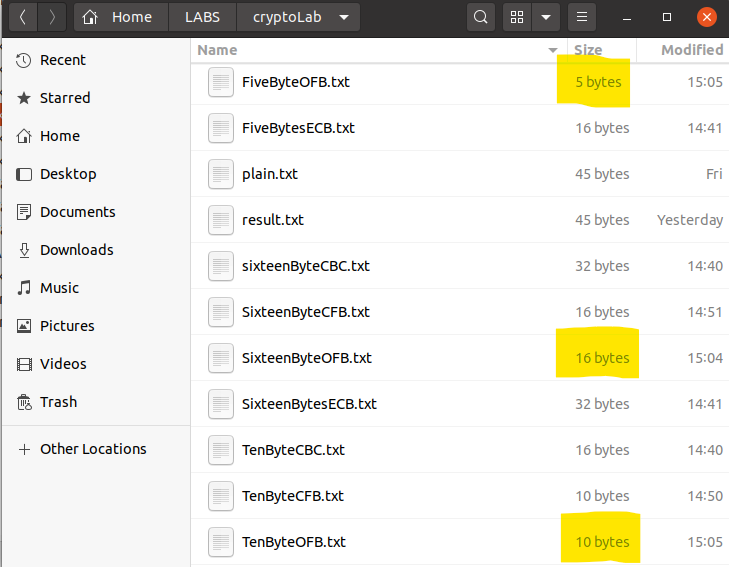


(2) First we will use **OFB mode** to encrypt a file where –e will use for encryption. Our command for encryption is follows:

openssl enc –aes-128-ofb –e –in test1.txt –out FiveByteOFB.txt –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

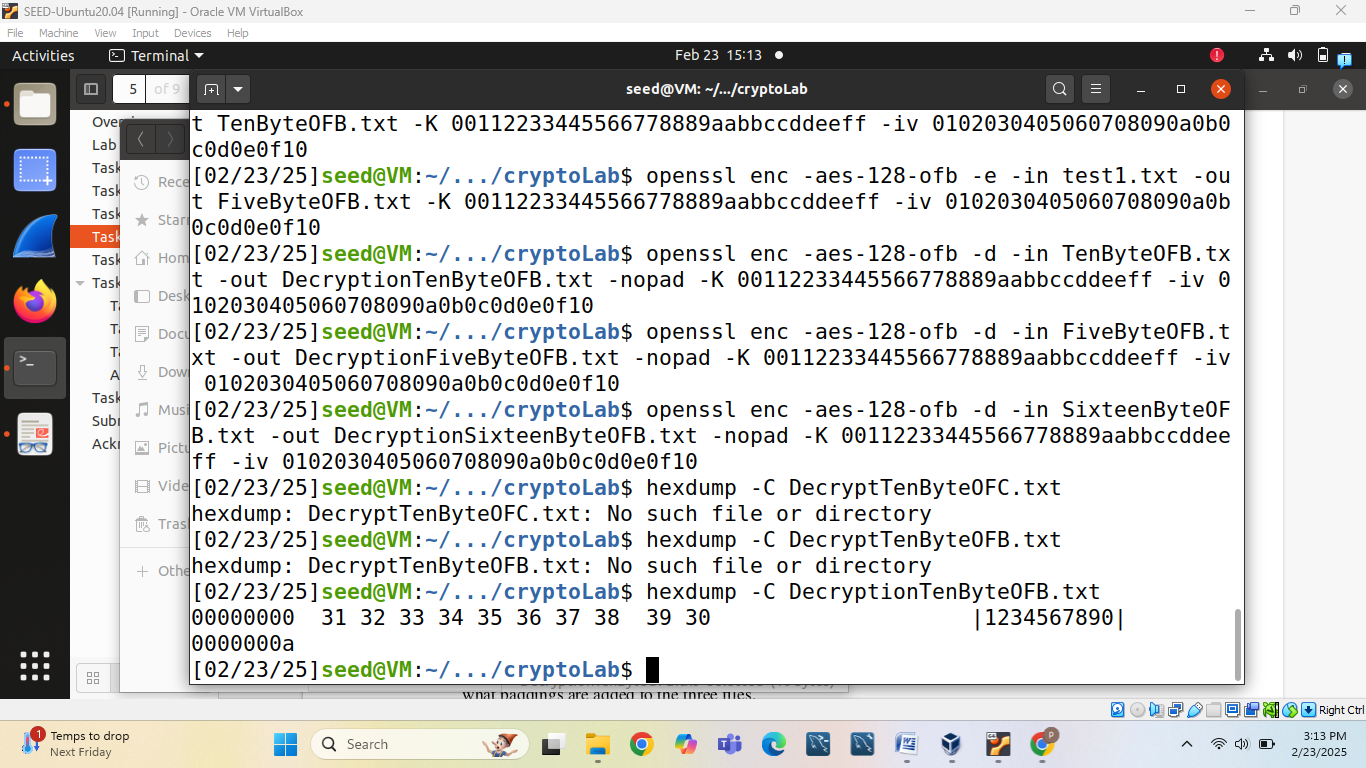


Size of the encrypted file is 5 bytes for 5 bytes data. Size of the encrypted file is 10 bytes for 10 bytes data and 16 bytes for 16 bytes data which is clear after watching following image:



Now we will use **OFB mode** to decrypt a file where–d will use for decryption so our command for decryption follows:

openssl enc –aes-128-ofb –d –in TenByteOFB.txt –out DecryptionTenBytesOFB.txt -nopad –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

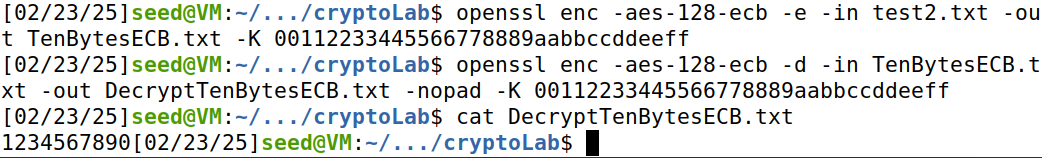


(3) First we will use **ECB mode** to encrypt and decrypt a file where –e will use for encryption and –d will use for decryption so our command for encryption is follows:

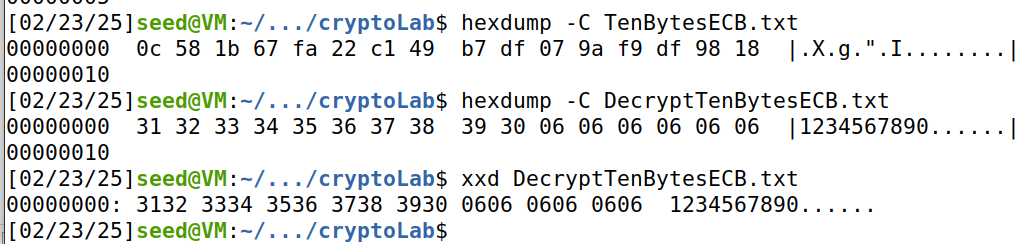
openssl enc –aes-128-ecb –e –in test2.txt –out TenBytesECB.txt –K 00112233445566778889aabbccddeeff

For decryption we will use following command:

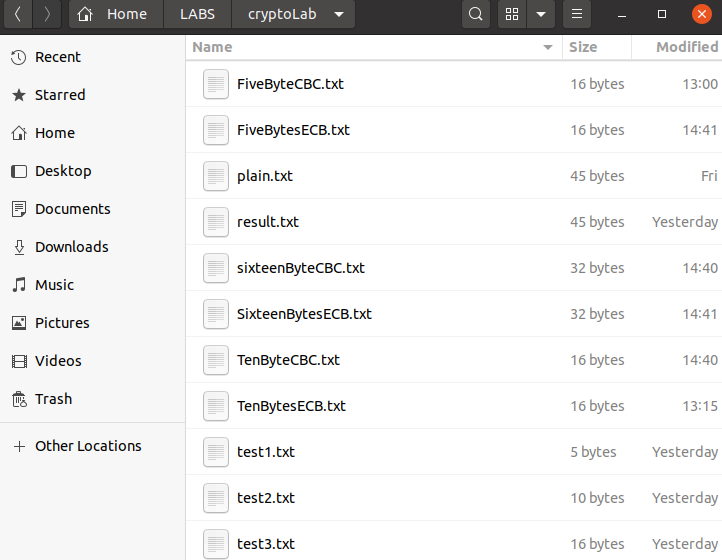
openssl enc –aes-128-cbc –d –in TenBytesECB.txt –out DecryptTenBytesECB.txt -nopad –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10



Following is the command to see the encrypted data:



Size of the encrypted file is 16 bytes for 10 byte data.



Size of the encrypted file is 16 bytes for 10 bytes data and 32 bytes for 16 bytes data.

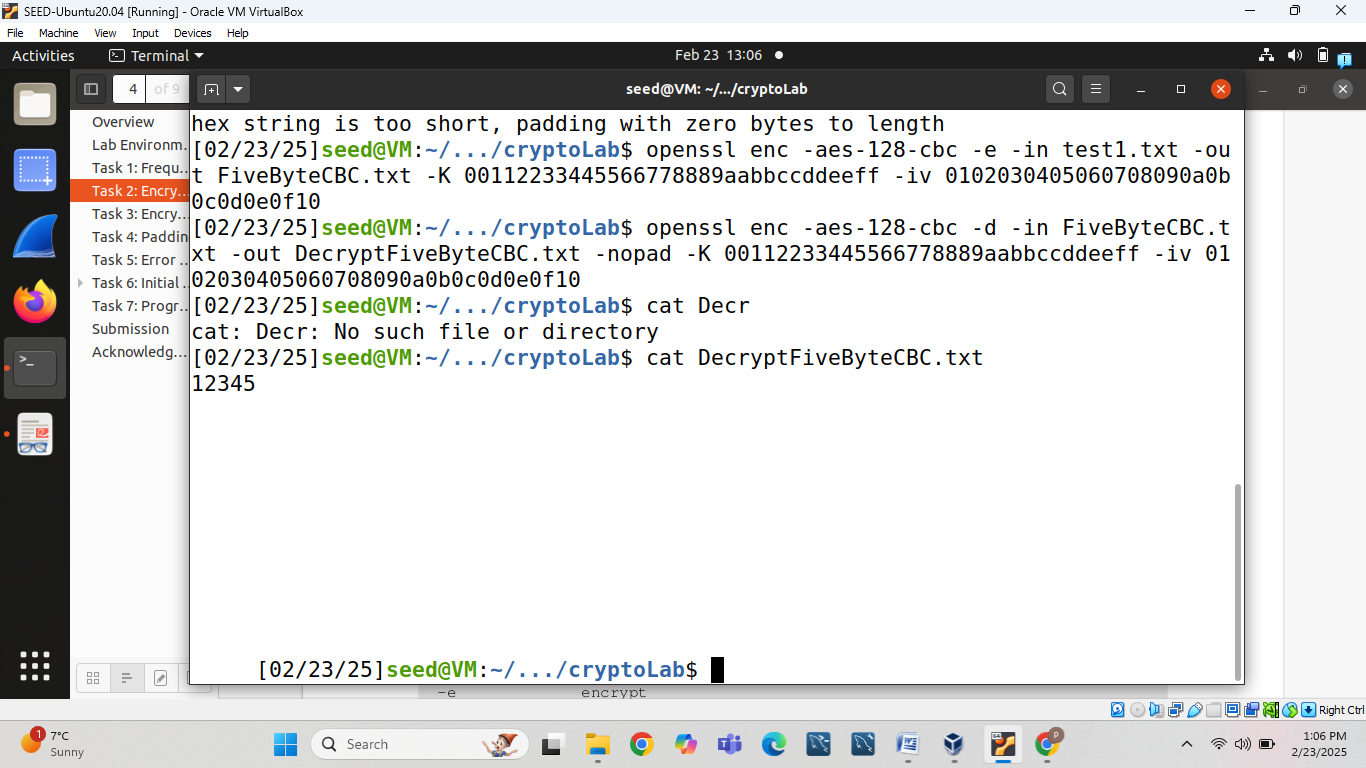
(4) First we will use **CBC mode** to encrypt and decrypt a file where –e will use for encryption and –d will use for decryption so our command for encryption is follows:

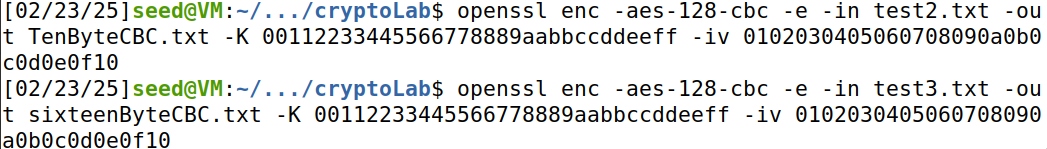
openssl enc –aes-128-cbc –e –in test1.txt –out FiveByteCBC.txt –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

For decryption we will use following command:

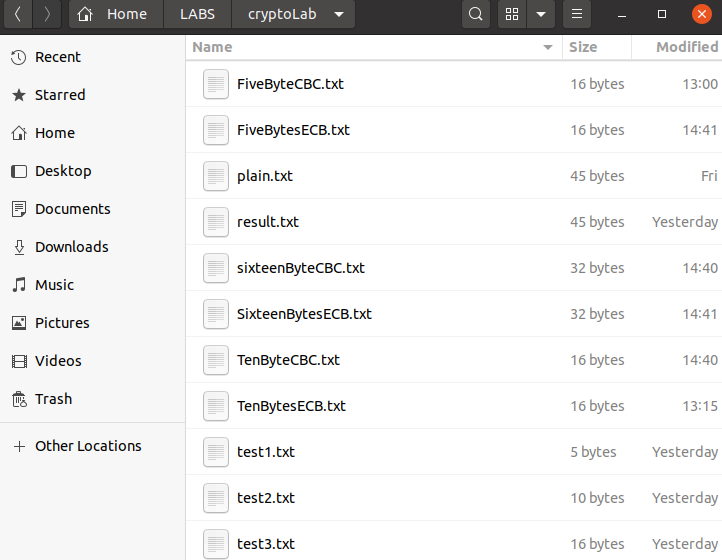
openssl enc –aes-128-cbc –d –in FiveByteCBC.txt –out DecryptFiveByteCBC.txt –K 00112233445566778889aabbccddeeff –iv 0102030405060708090a0b0c0d0e0f10

Size of the encrypted file is 16 bytes for 5 bytes data.





Size of the encrypted file is 16 bytes for 10 bytes data and 32 bytes for 16 bytes data which is clear after watching following image:



* CFB and OFB don't need padding. Because they take outputs of the previous block, which must be of the same size equal to cipher block size, as the inputs of its last cipher block encryption.

**Task 5: Error Propagation – Corrupted Cipher Text**

1. Create a text file that is at least 1000 bytes long.

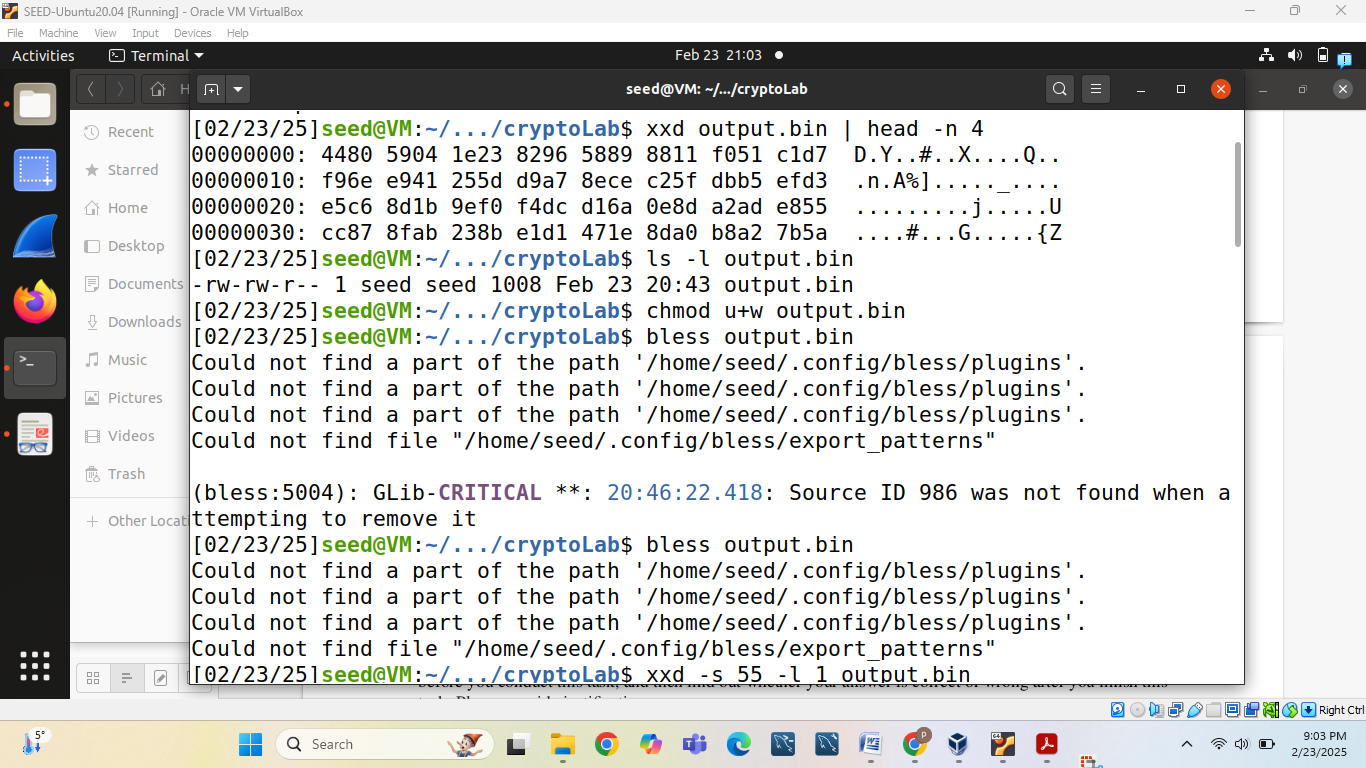
Screenshot 2025-02-23 210113.png

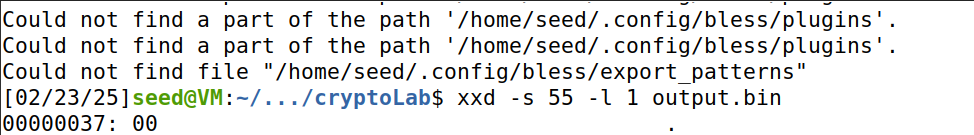
2. Encrypt the file using the AES-128 cipher.

Screenshot 2025-02-23 210206.png

3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this

corruption using the bless hex editor.





4. Decrypt the corrupted ciphertext file using the correct key and IV.

Screenshot 2025-02-23 210530.png

Now we will see what the difference between encrypted text and the text is after decryption:



After doing all these steps again for CBC, CFB and OFB we will get following information:

Difference between the original files and decrypted files:

| **Mode** | **Different bytes** |
| --- | --- |
| ECB | 16 |
| CBC | 17 |
| CFB | 17 |
| OFB | 1 |

**Justification:**

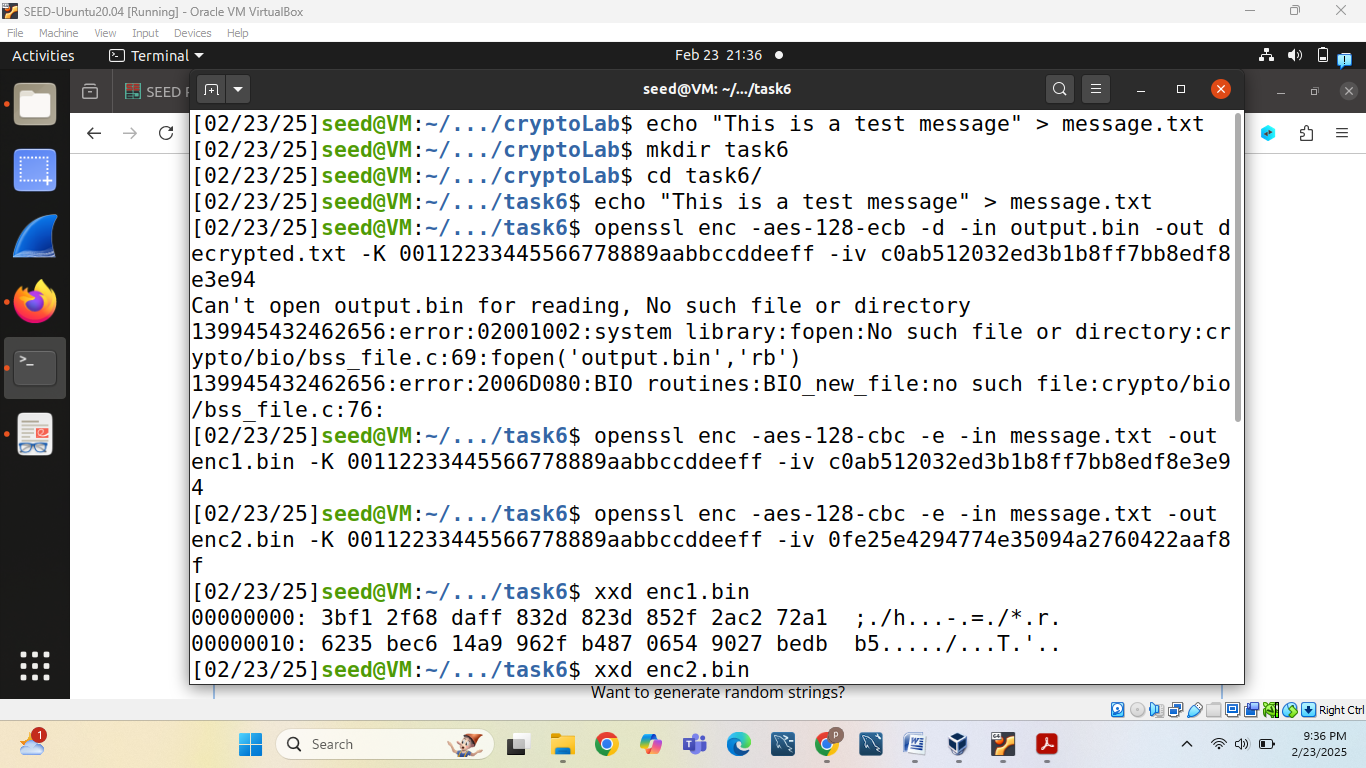
* **ECB:**   
  Since each 16-byte block is encrypted independently, any corruption within a block (here, the 55th byte which lies in one block) causes the entire block to decrypt incorrectly. Thus, the difference in the decrypted file is 16 bytes.
* **CBC:**   
  In CBC, decryption of a block uses the previous ciphertext block. Corruption in one block results in the complete loss of intelligibility for that block (16 bytes) and causes an error in the first byte of the next block due to the XOR with the corrupted ciphertext. This leads to a total of 17 bytes being affected.
* **CFB:**   
  CFB mode's error propagation is similar to CBC in that the corrupted bit affects not just the current block but also spills over into the next block's decryption, resulting in about 17 bytes of difference.
* **OFB:**   
  OFB mode, acting as a stream cipher, ensures that an error in the ciphertext only affects the corresponding bit (or byte) in the plaintext. Therefore, only the single byte where the error occurred is incorrect, with the rest of the file being recovered correctly.

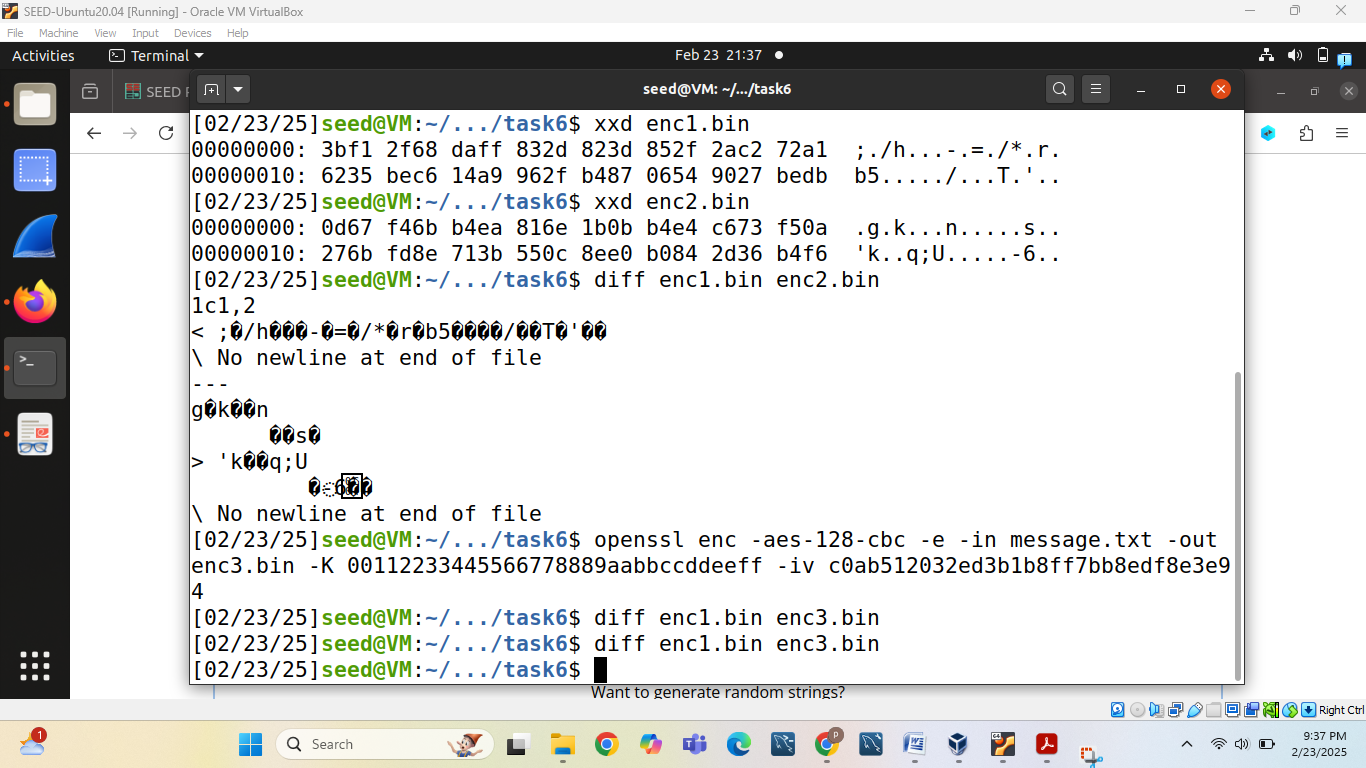
**Task 6: Initial Vector (IV) and Common Mistakes**

**Task 6.1: IV Experiment**

The ciphertexts are different, proving that different IVs produce different outputs.

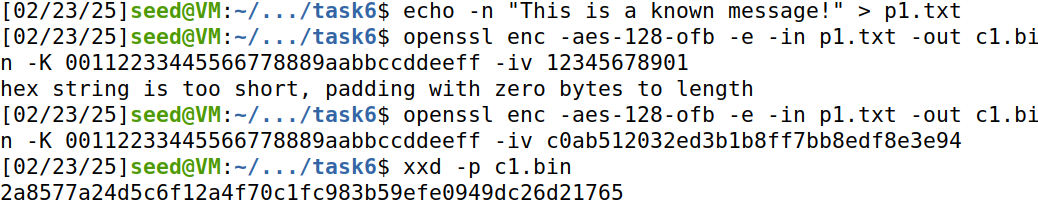
When plaintexts are the same, using the same IV leads to the same ciphertexts.[ The ciphertexts are identical, showing that reusing IVs leaks information.] which is clear after seeing following data:



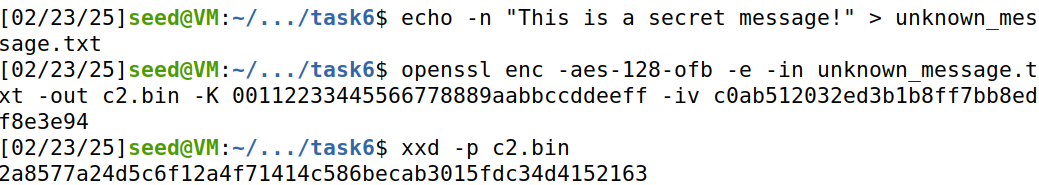


**Task 6.2: Common Mistake: Use the Same IV**

First we will encrypt known message as follows:



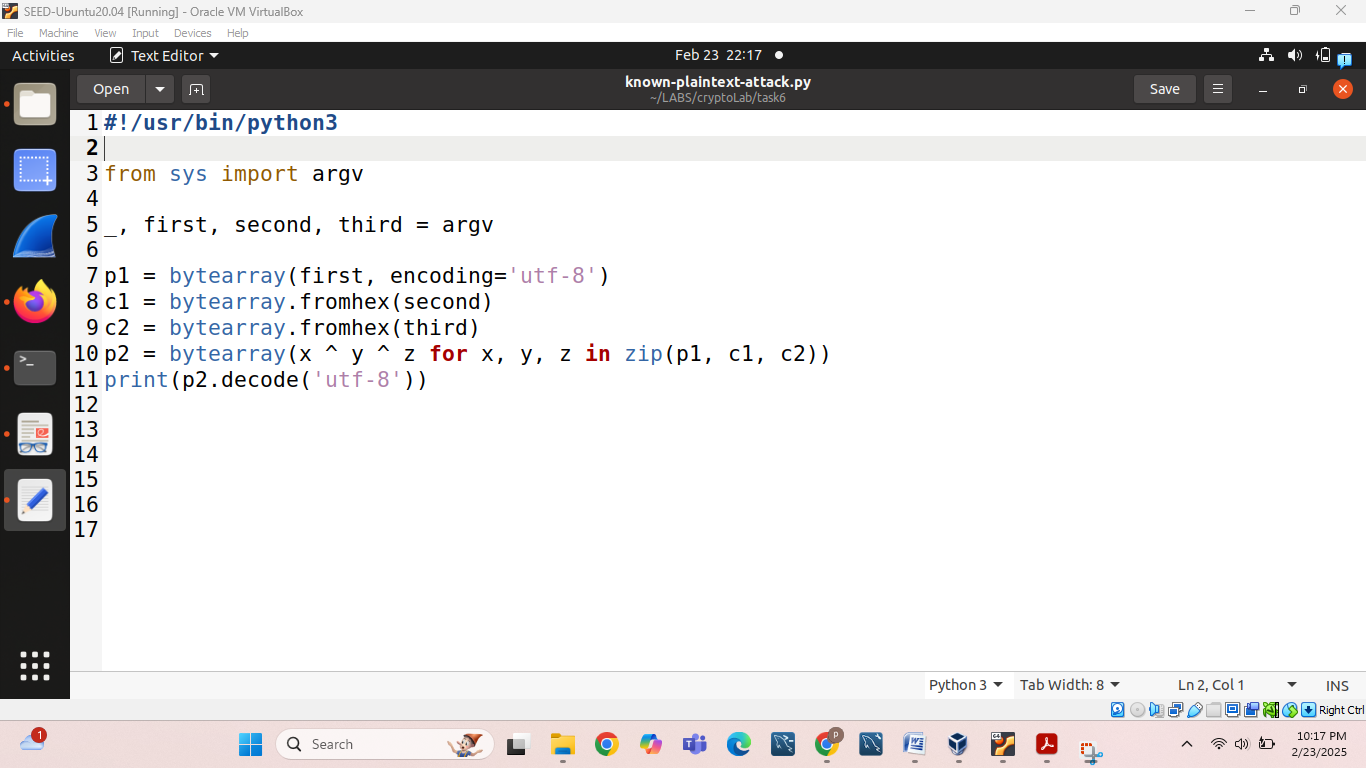
Then we will encrypt unknown message as follows:



Perform the known –plaintext attack:

Screenshot 2025-02-23 221636.png

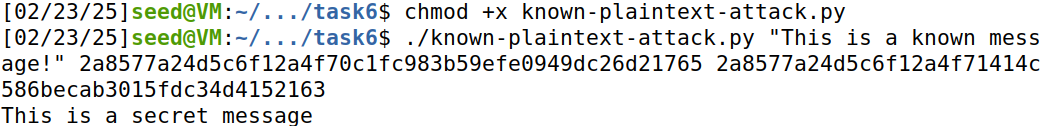
Here code for known-plaintext-attack.py is follows:



Now we will make it executable by following command:

chmod +x known-plaintext-attack.py

Run the script and get output by following command which shown in the figure:



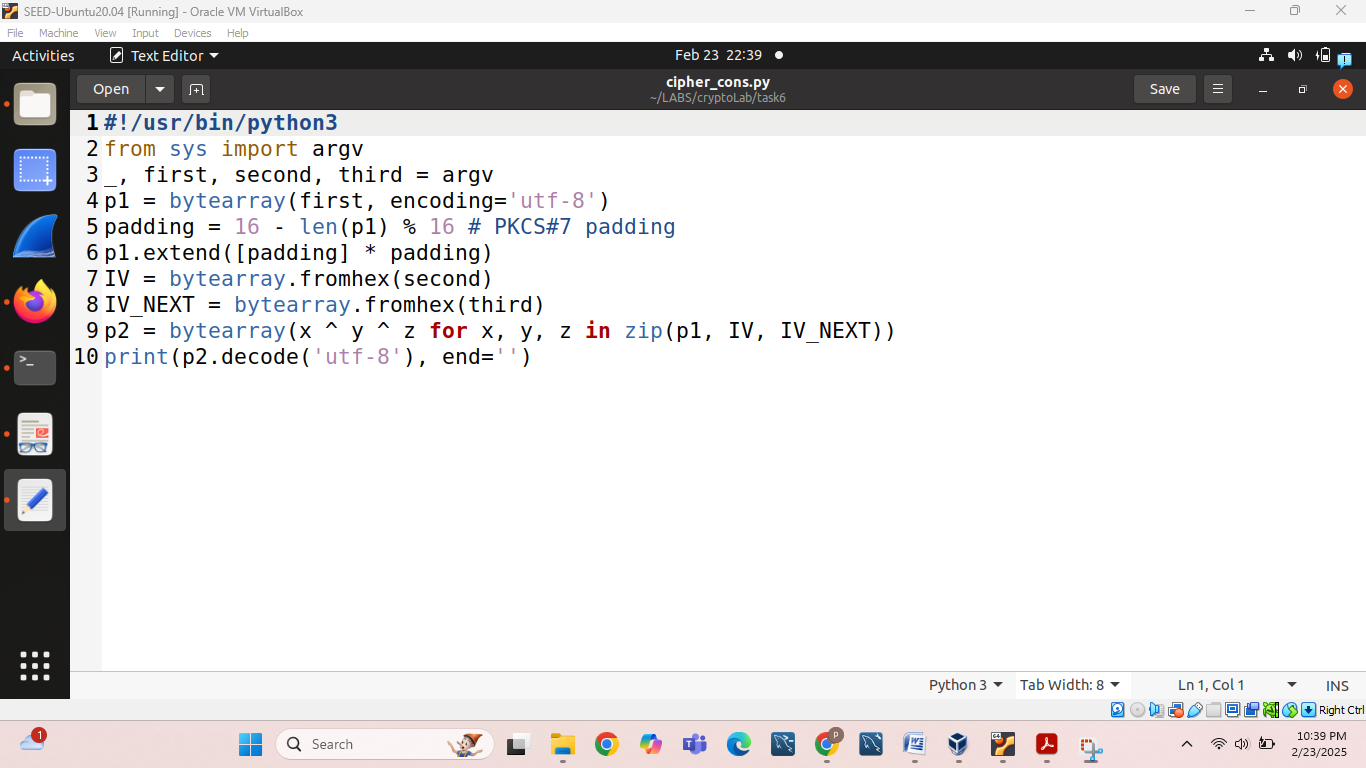
Since OFB mode generates the same key stream for the same IV, we can recover P2.

**Task 6.3: Common Mistake: Use a Predictable IV**

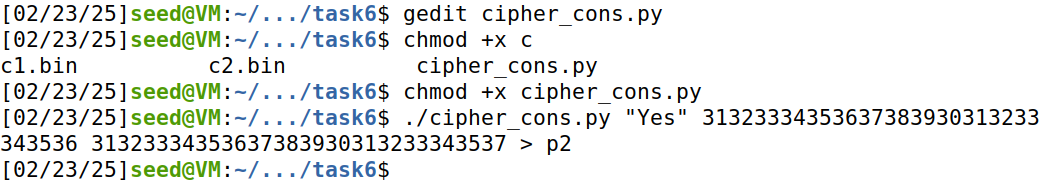
Construct the attack message by typing:

gedit cipher\_cons.py

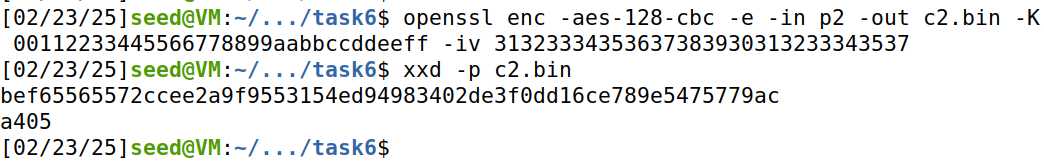
In which we will add following code:



Then we will encrypt p2 to get c2:



Now we will compare c1 and c2 and check either first 16 bytes match or not if match hypothesis will be correct: p1 was “Yes”

Here we can see 1st 16 bytes match so it matches hypothesis.

**Task 7: Programming using the Crypto Library**

First we will set plaintext.txt value as “This is a top secret.”

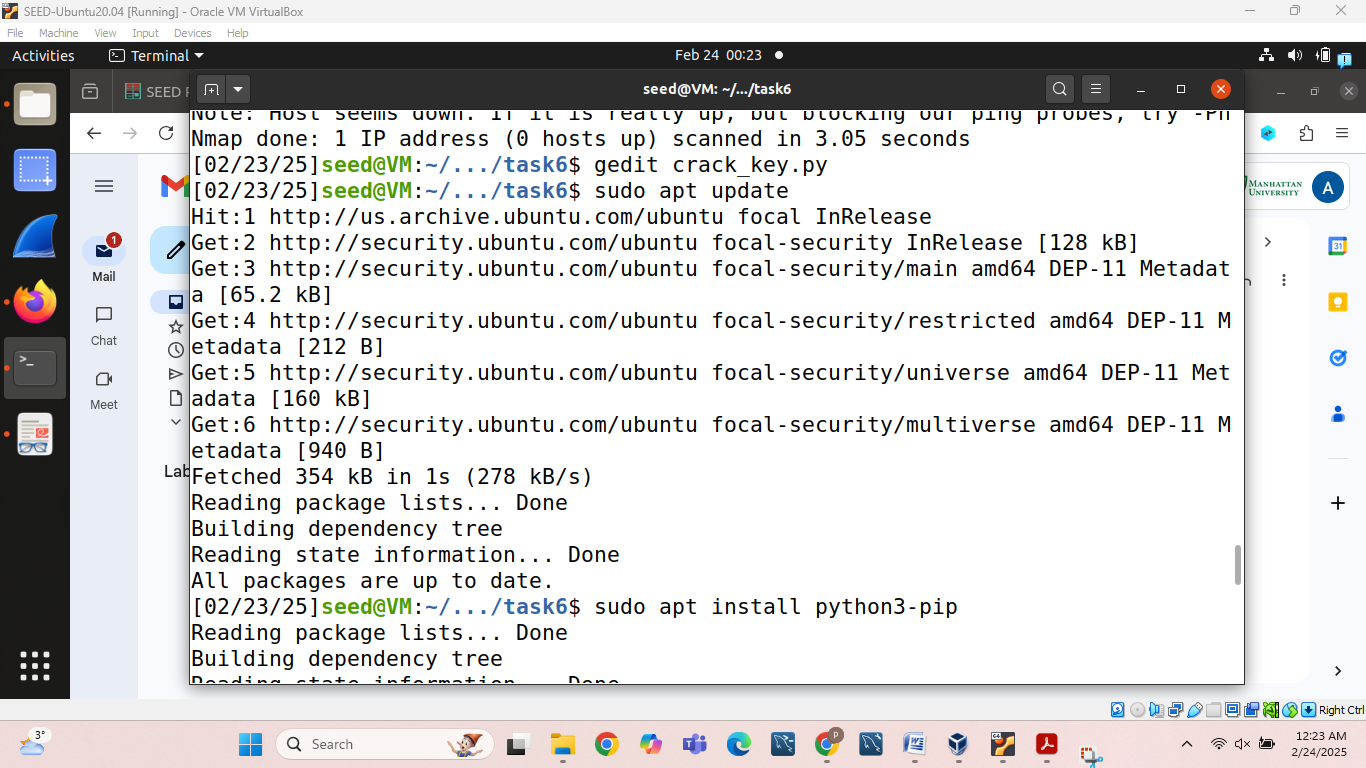
Screenshot 2025-02-24 002054.png

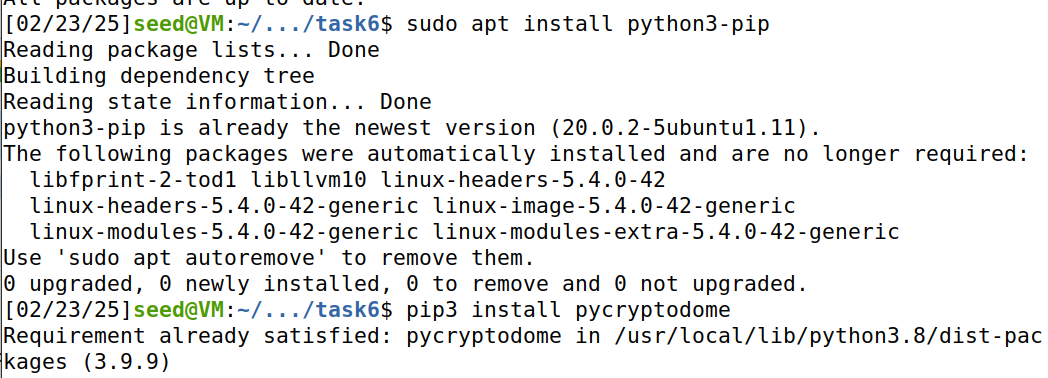
Then we will ensure that the OpenSSL development libraries are installed by doing following commands:

sudo apt update

sudo apt install python3-pip

pip3 install pycryptodome





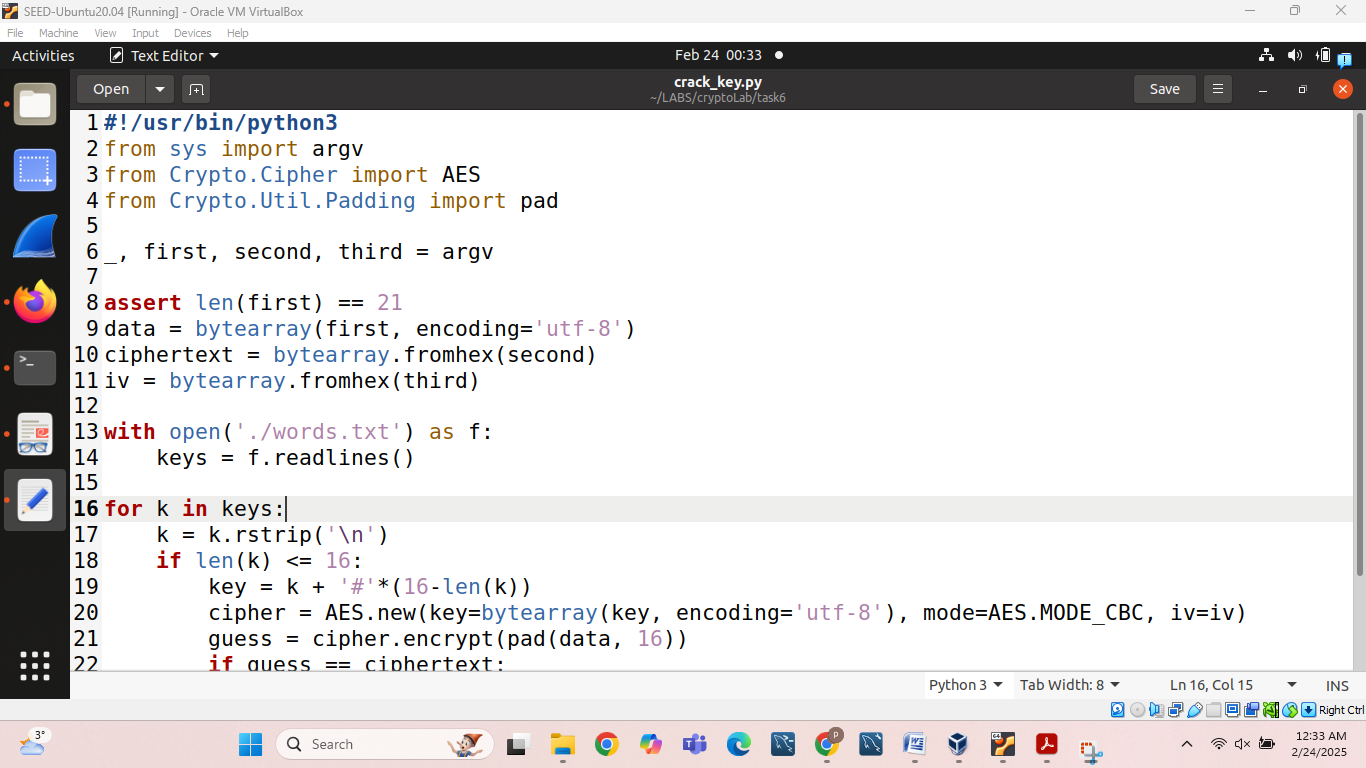
We will then extract words.txt file from either downloading labsetup zip folder and use from there or download fron github curl -O <https://raw.githubusercontent.com/dwyl/english-words/master/words.txt>

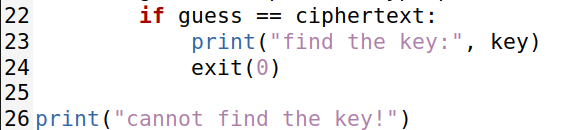
We will download and use words.txt file.

Then for the next step we will write down python code in order to perform this task by writing command:

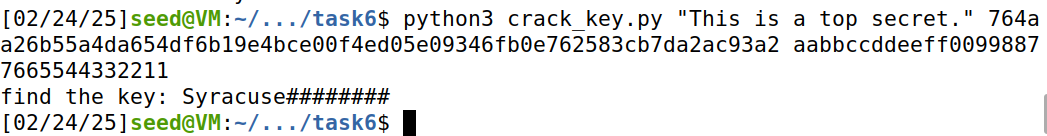
Gedit crack\_key.py

**Code is as follows:**





Then in order to run this code we type following command:



Finally, find the key:

Find the key: Syracuse########