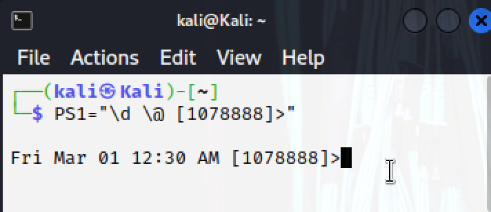
|  |
| --- |
| ADU logo |
| **Lab 1** |
| **ITE408: Information Security** |
| **Submission Date: 29th of September, 2024** |
| **Marks: 5 Marks** |
| **Fall 2024-25** |

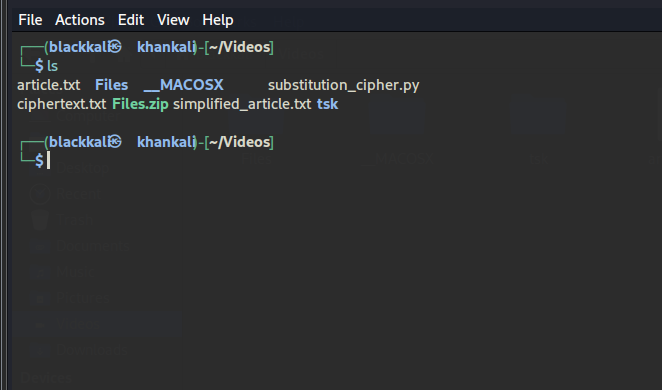
Please download the Files Folder to your VM(CSELAB\_VM) from the link provided on Blackboard, unzip it, and enter the Files Folder from the terminal. From the terminal and run the following command

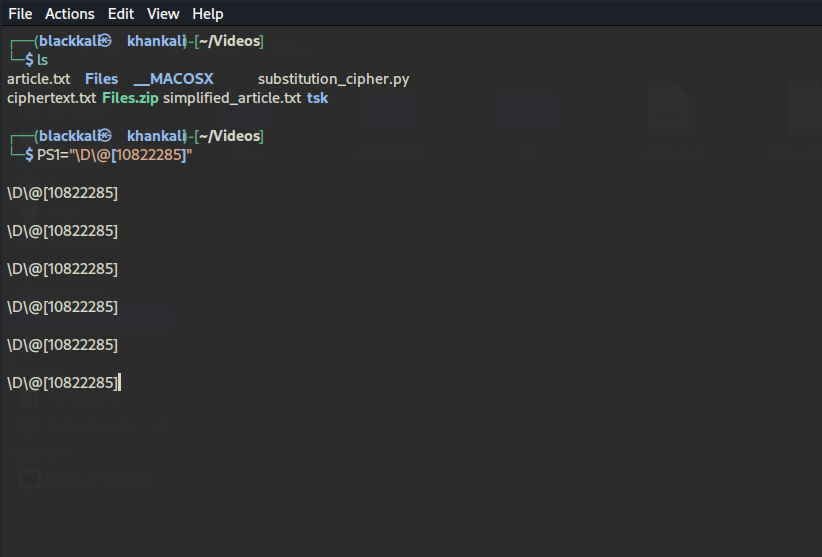
>>PS1=”\d \@ [Your\_Student-ID]”.

Use your own student ID. The prompt will change, as shown below.



All the screenshots from now onwards must show your studentID as shown above





# Task 1 Encryption using substitution cipher

It is well-known that monoalphabetic substitution cipher (also known as monoalphabetic cipher) is not secure, because it can be subjected to frequency analysis. In this lab, you are given a cipher-text that is encrypted using a monoalphabetic cipher; namely, each letter in the original text is replaced by another

letter, where the replacement does not vary (i.e., a letter is always replaced by the same letter during the encryption). Your job is to find out the original text using frequency analysis. It is known that the original text is an English article.

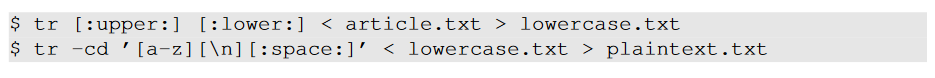
In the following, we describe how we encrypt the original article, and what simplification we have made.

* Step 1: Let us generate the encryption key, i.e., the substitution table. We will permute the alphabet from a to z using Python and use the permuted alphabet as the key. See the following program. The code for the Python is provided below. Create a python file inside your VM with the code provided below. Run the python file and generate the encryption key,





* Step 2: Now find a one-paragraph article from the Internet and save it in the text file named article.txt. Let us do some simplification to the original article, which we saved in article.txt. We will convert all upper cases to lower cases and then remove all the punctuations and numbers. We do keep the spaces between words so you can still see the boundaries of the words in the ciphertext. In real encryption using a monoalphabetic cipher, spaces will be removed. We keep the spaces to simplify the task. We can do this by using the following command

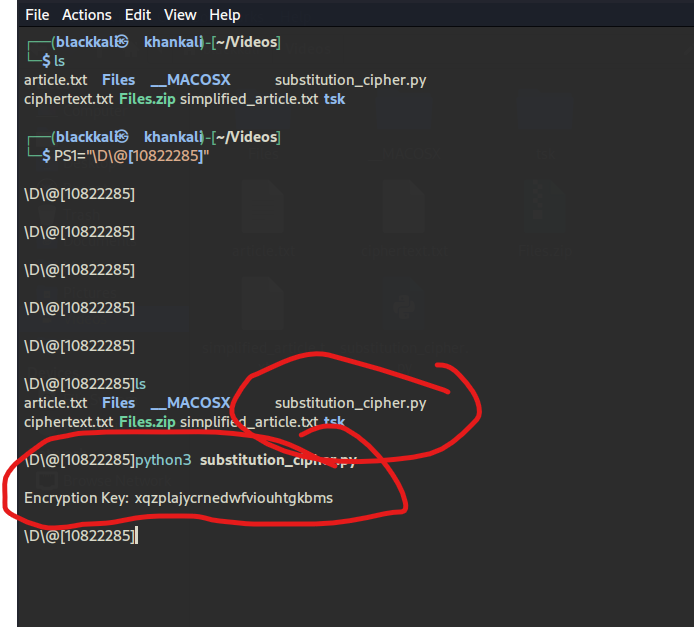


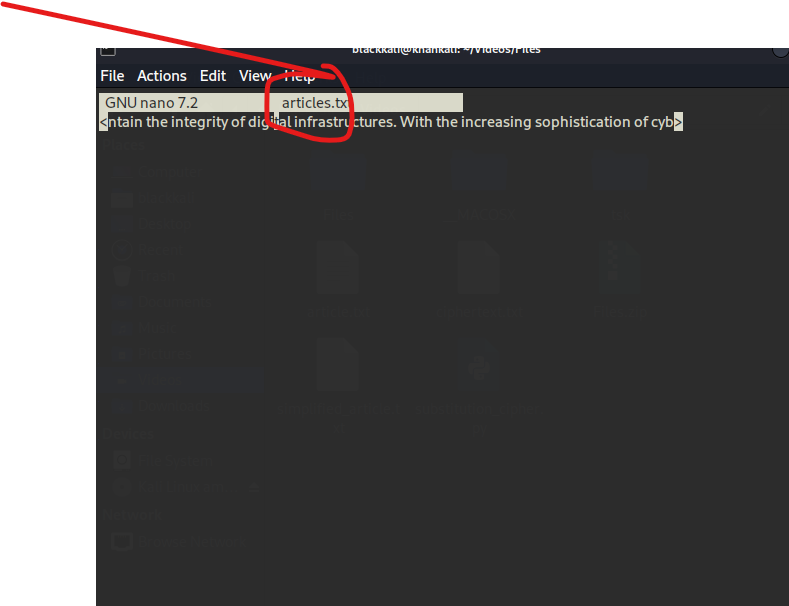
* Step 3: We will use the tr command to do the encryption. We only encrypt letters while leaving the space and return characters alone. To encrypt the letters, use the key generated in step 1.

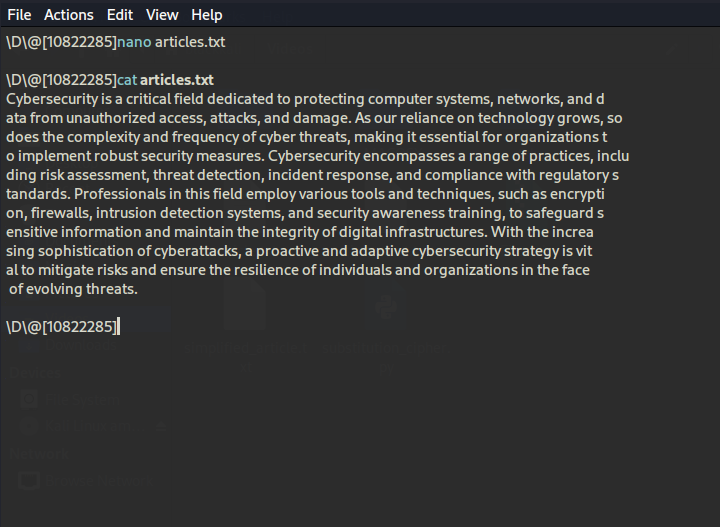


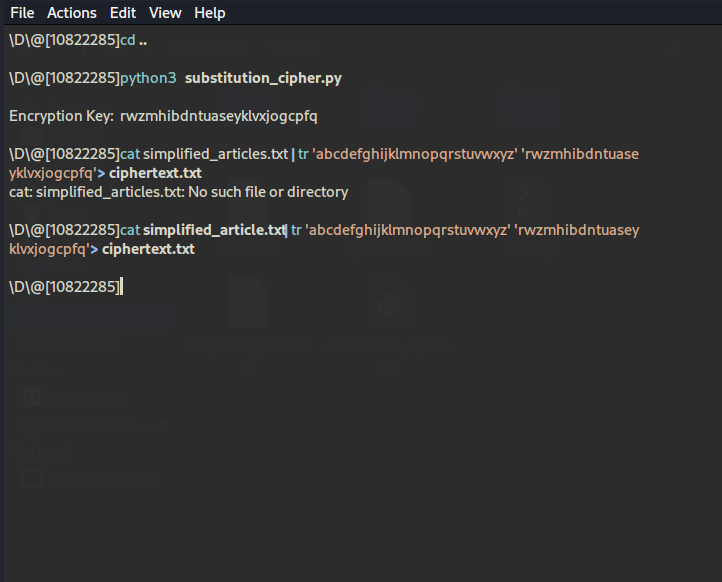
We have created a ciphertext using a different encryption key. Provide the screenshot of the ciphertext you have created using the above steps.

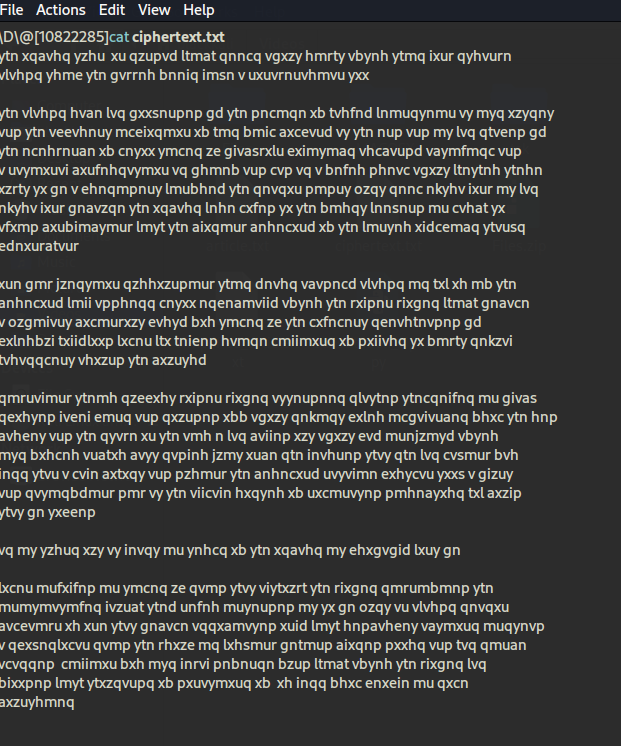
Generate the Encryption Key first:











# Task 2 Frequency Analysis

We have provided you with an encrypted article, i.e., ciphertext.txt, in the Files Folder, which you have downloaded from the blackboard. We have also provided a Python program (freq.py) inside the Files folder. It reads the ciphertext.txt file and produces the statistics for n-grams, including the single-letter frequencies, bigram frequencies (2-letter sequence), trigram frequencies (3-letter sequence), etc.



Using the frequency analysis, you can find out the plaintext for some of the characters quite easily. For those characters, you may want to change them back to its plaintext, as you may be able to get more clues. It is better to use capital letters for plaintext, so for the same letter, we know which is plaintext which is ciphertext. You can use the tr command to do this. For example, in the following, we replace

letters a, e, and t in in.txt with letters X, G, E, respectively; the results are saved in out.txt.

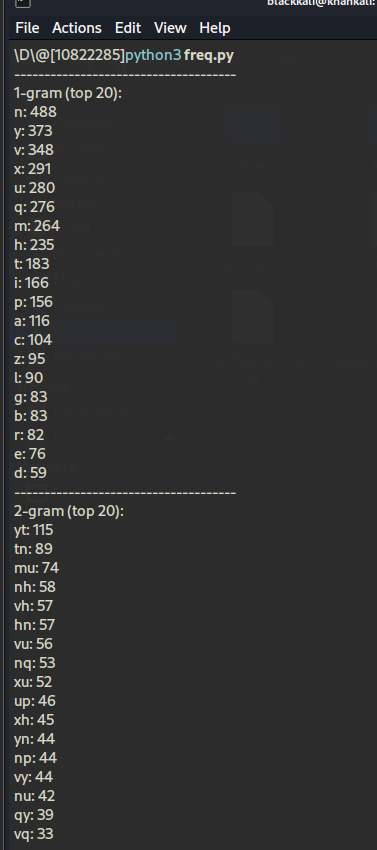


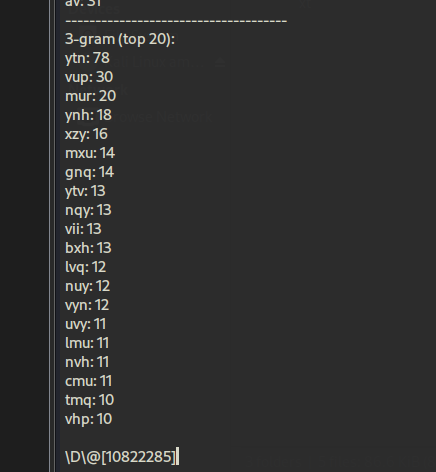
There are many online resources that you can use. We list some useful links in the following:

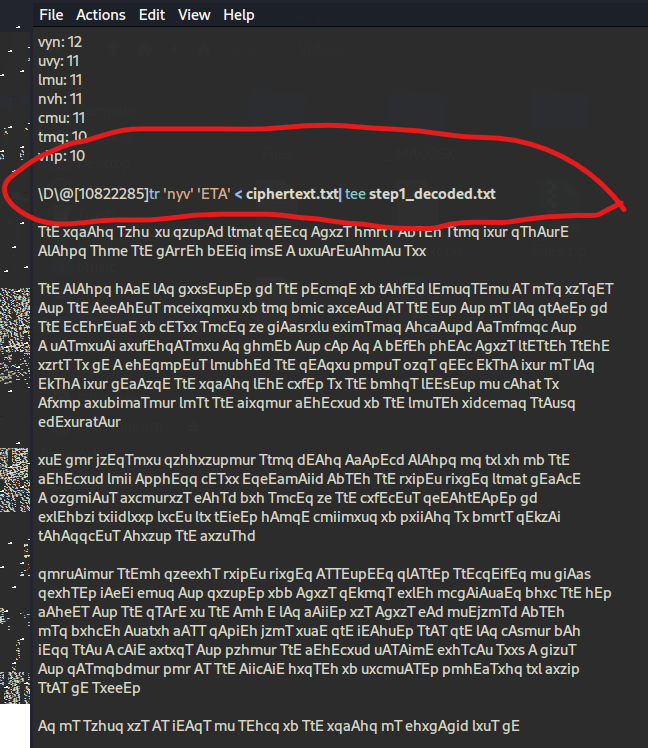
• https://en.wikipedia.org/wiki/Frequency\_analysis: This Wikipedia page provides frequencies for a typical English plaintext.

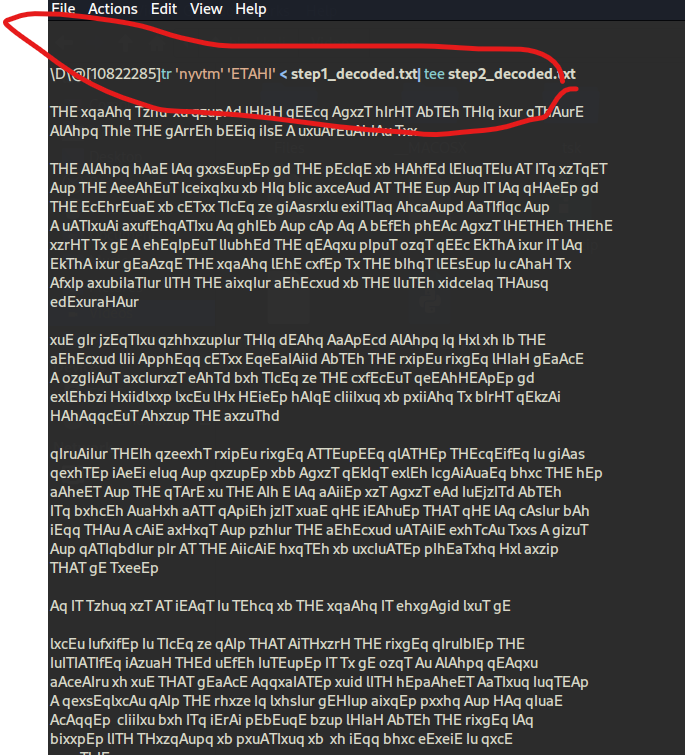
• https://en.wikipedia.org/wiki/Bigram: Bigram frequency.

• https://en.wikipedia.org/wiki/Trigram: Trigram frequency



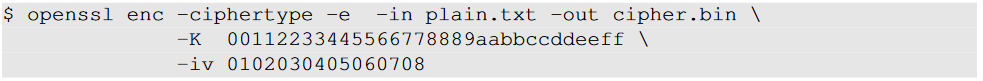




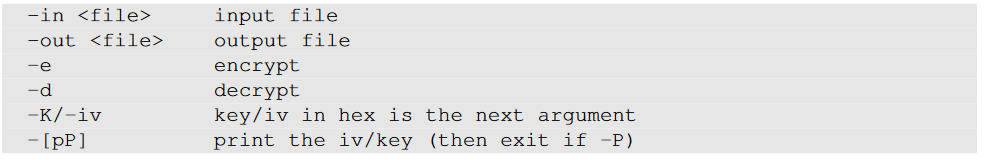


# Task 3

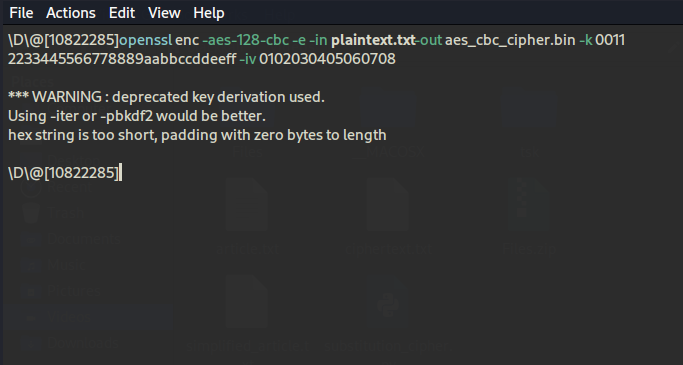
In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man enc.



Please replace the *ciphertype* with a specific cipher type, such as *-aes-128-cbc*, *-bf-cbc*,*-aes-128-cfb*, etc. In this task, you should try at least 3 different ciphers. You can find the meaning of the command-line options and all the supported cipher types by typing "*man enc*". We include some common options for the openssl enc command in the following:



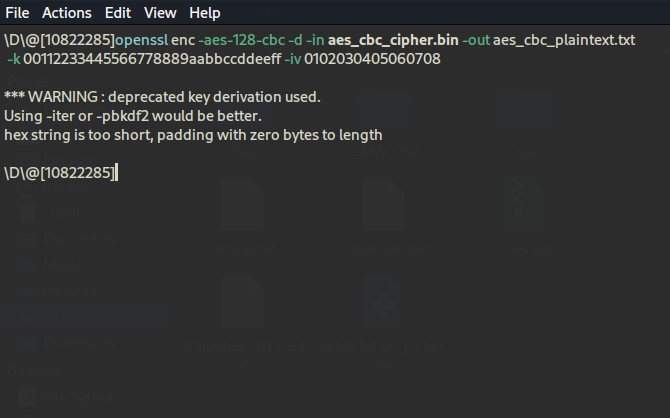
Show and explain the three different ciphers, you have used for encryption and decryption.



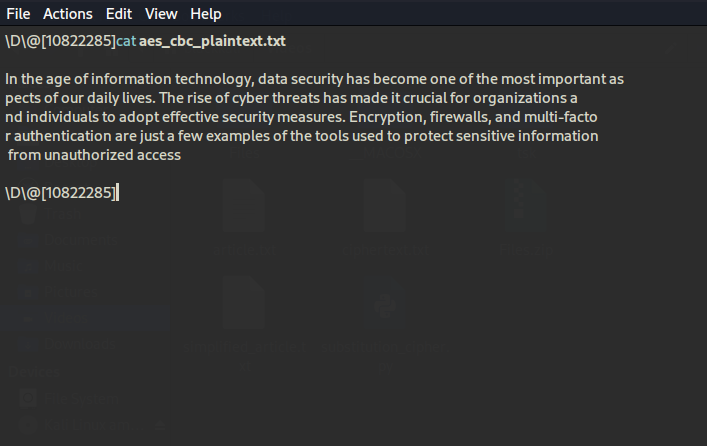
**Explanation:**

* -aes-128-cbc: Specifies the cipher type (AES with 128-bit key in CBC mode).
* -e: Indicates that we are encrypting the file.
* -in plain.txt: Input file that contains the plaintext.
* -out aes\_cbc\_cipher.bin: Output file for the encrypted data.
* -k: The encryption key in hexadecimal format (32 hexadecimal characters for AES-128).
* -iv: The initialization vector (IV) used for the CBC mode in hexadecimal.

To decrypt the file,

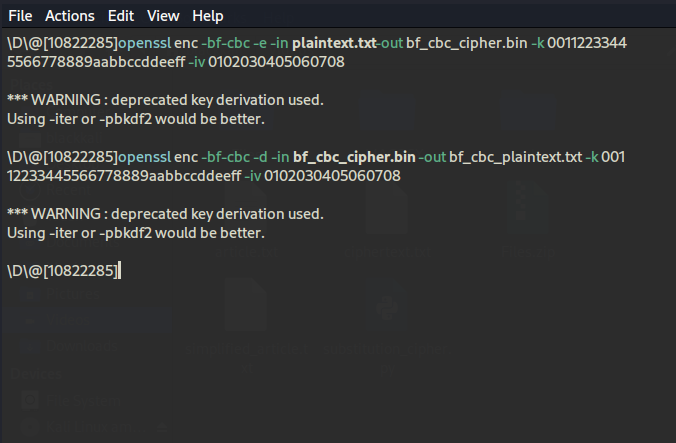


Verify the decryption by checking the content of aes\_cbc\_plaintext.txt



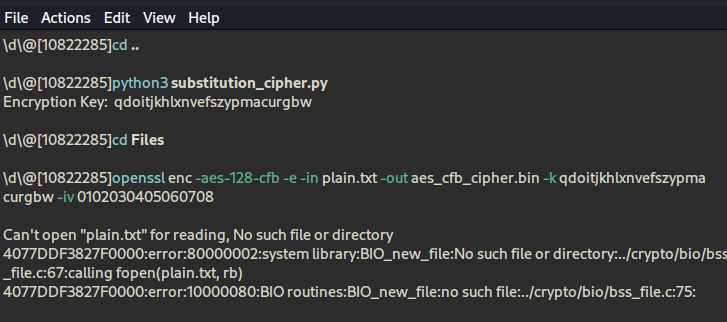
**Step 2: Encrypt using Blowfish-CBC**

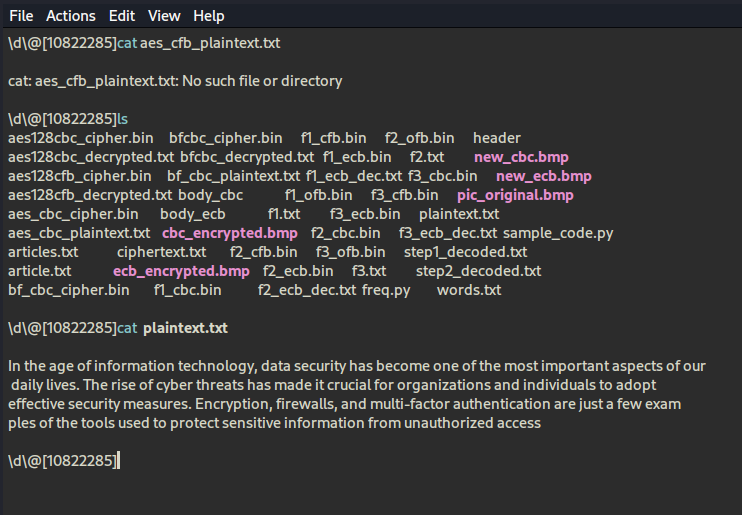
1. Encrypt the file using Blowfish in CBC mode with the following command:
2. And decrypt



Step 3: Encrypt using AES-128-CFB

Encrypt and Decrypt the file using AES-128 in CFB mode:





**AES-128-CBC:**

* **Type:** Symmetric key block cipher.
* **Block Size:** 128 bits.
* **Mode:** CBC (Cipher Block Chaining) mode, which requires an initialization vector (IV) for encryption.
* **Use Case:** Commonly used in secure communications, as it provides confidentiality and integrity.

**Blowfish-CBC:**

* **Type:** Symmetric key block cipher.
* **Block Size:** 64 bits.
* **Mode:** CBC mode.
* **Use Case:** Used in various encryption products and software due to its speed and effectiveness.

**AES-128-CFB:**

* **Type:** Symmetric key block cipher.
* **Block Size:** 128 bits.
* **Mode:** CFB (Cipher Feedback) mode, which allows encryption of data in smaller units than the block size.
* **Use Case:** Suitable for stream encryption and can encrypt data of any size.

# Task 4:

The file pic original.bmp is included in the Folder that you downloaded from blackboard. file, and it is a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. We can use the following commands to get the header from p1.bmp, the data from p2.bmp (from offset 55 to the end of the file), and then combine the header and data together into a new file.



1. Display the encrypted picture using a picture viewing program (we have installed an image viewer program called eog on our VM). If the tool is not installed on your system, install using the command below

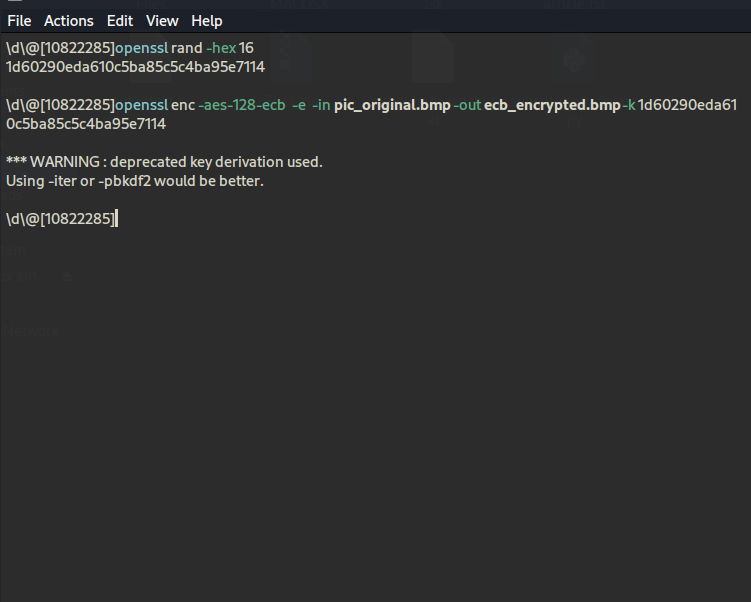
>> sudo apt install eog

1. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations

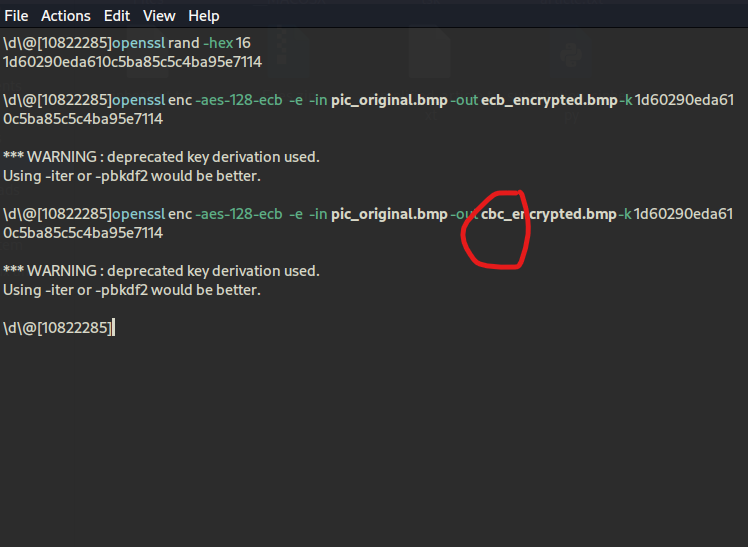
Step 1: Encrypt the Picture

**Encrypt Using ECB Mode**:

Run the following command to encrypt pic\_original.bmp using AES in ECB mode

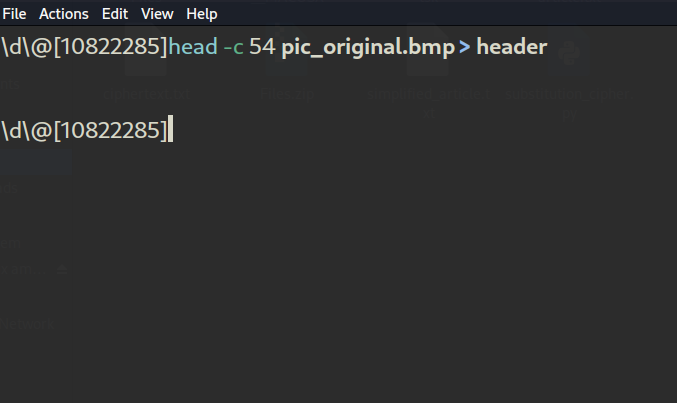


**Encrypt Using CBC Mode**: Next, encrypt the same picture using AES in CBC mode.

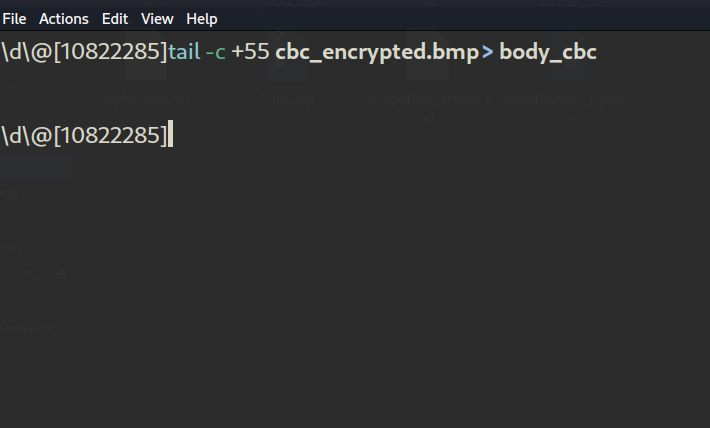


Step 2: Modify the Headers

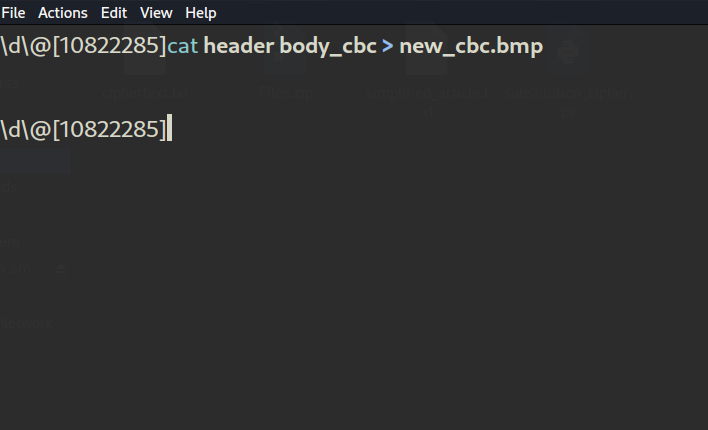
**Extract the Header from the Original BMP**:



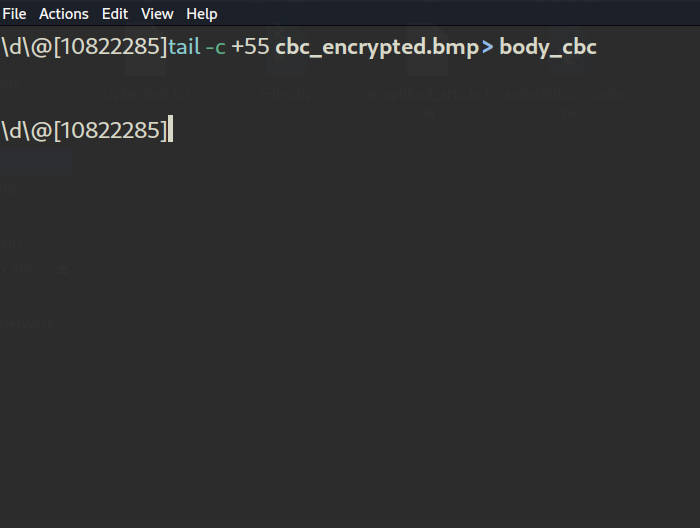
**Extract the Body from the Encrypted BMP (ECB)**:



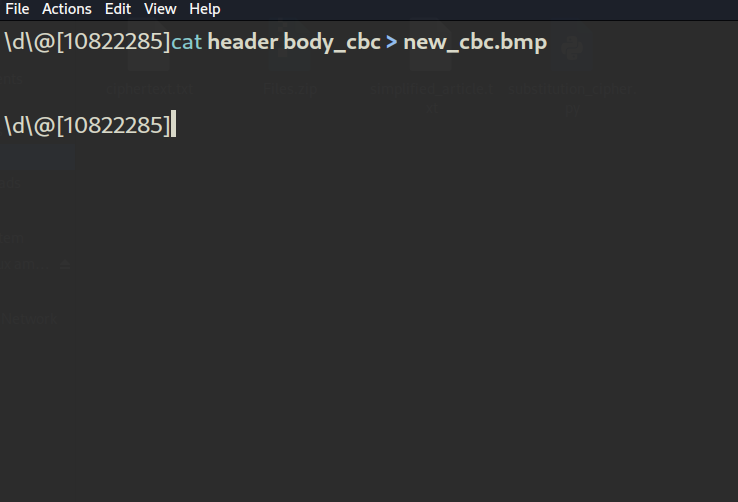
**Combine the Header and Encrypted Body (ECB)**:



Extract the Body from the Encrypted BMP (CBC)

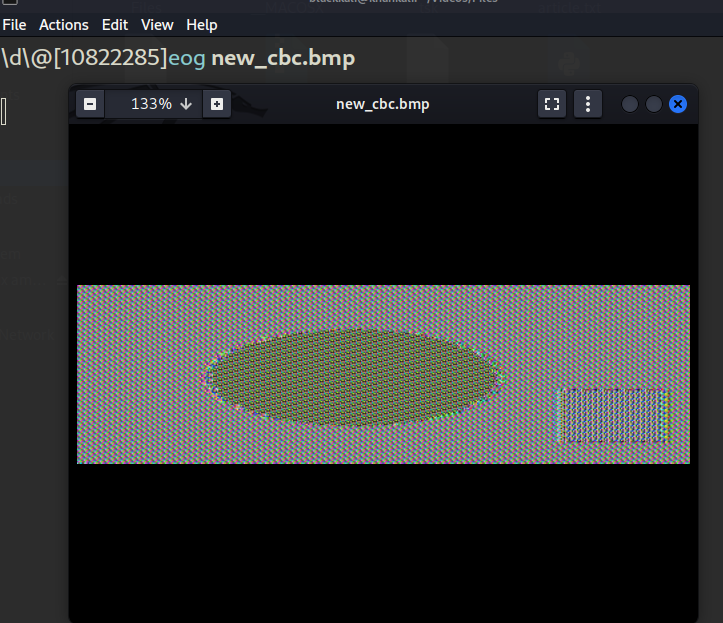


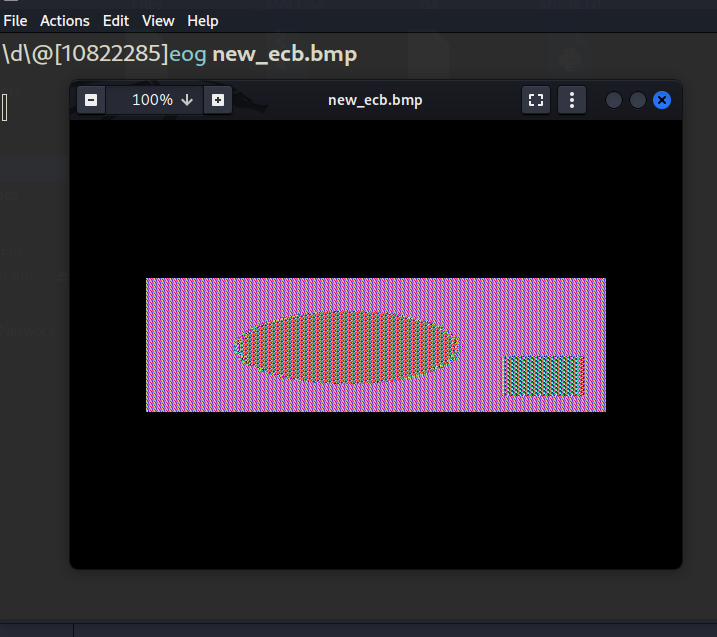
**Combine the Header and Encrypted Body (CBC)**:

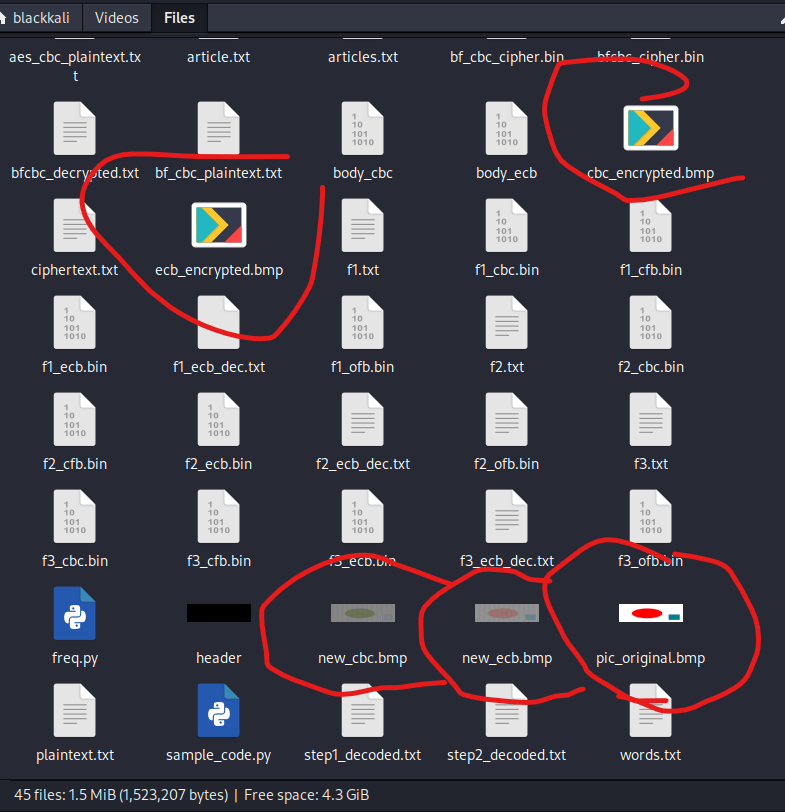


Step 3: View the Encrypted Pictures

**Display the cbc Encrypted Image**:







**ECB Mode**:

The ECB mode encrypts identical plaintext blocks into identical ciphertext blocks. This can lead to recognizable patterns in the encrypted image, such as sections that look the same. You may notice parts of the image structure being preserved, which can reveal some information about the original image.

**CBC Mode**:

The CBC mode introduces randomness because each block of plaintext is XORed with the previous ciphertext block before encryption. As a result, even identical blocks of plaintext will produce different ciphertexts. Therefore, the encrypted image will look more random compared to the ECB mode, making it less susceptible to analysis.

# Task 5:

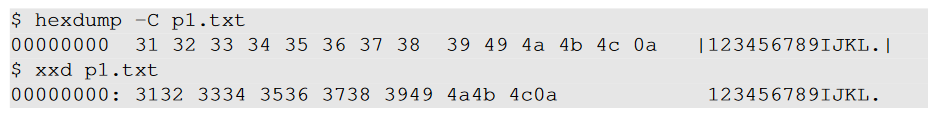
For block ciphers, when the size of a plaintext is not a multiple of the block size, padding may be required. The PKCS#5 padding scheme is widely used by many block ciphers. We will conduct the following experiments to understand how this type of padding works:

* Use ECB, CBC, CFB, and OFB modes to encrypt a file.
* Let us create three files, which contain 5 bytes, 10 bytes, and 16 bytes, respectively. We can use the following "echo -n" command to create such files. The following example creates a file f1.txt with length 5 (without the -n option, the length will be 6, because a newline character will be added by echo):



We then use "*openssl enc -aes-128-cbc -e*" to encrypt these three files using 128-bit AES with CBC mode. Please describe the size of the encrypted files. We would like to see what is added to the padding during the encryption. To achieve this goal, we will decrypt these files using "*openssl enc -aes-128-cbc -d*". Unfortunately, decryption by default will automatically remove the padding, making it impossible for us to see the padding.

However, the command does have an option called "*-nopad*", which disables the padding, i.e., during the decryption, the command will not remove the padded data. Therefore, by looking at the decrypted data, we can see what data are used in the padding. Please use this technique to figure out what paddings are added to the three files. It should be noted that padding data may not be printable, so you need to use a hex tool to display the content. The following example shows how to display a file in the hex format:

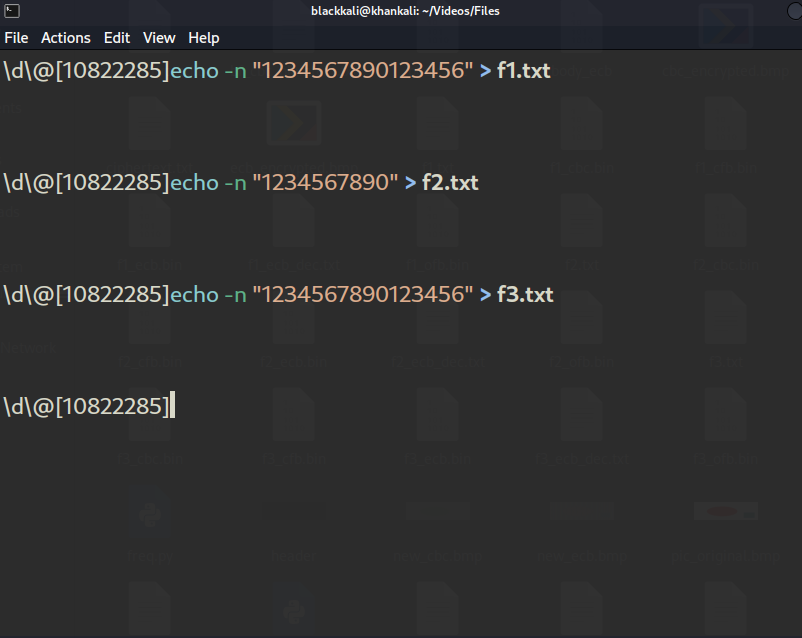


* Report which modes (ECB, CBC, CFB, and OFB) have paddings and which do not. For those that do not need paddings, please explain why.

1. **Create Files**

First, create three text files of different sizes (5 bytes, 10 bytes, and 16 bytes):

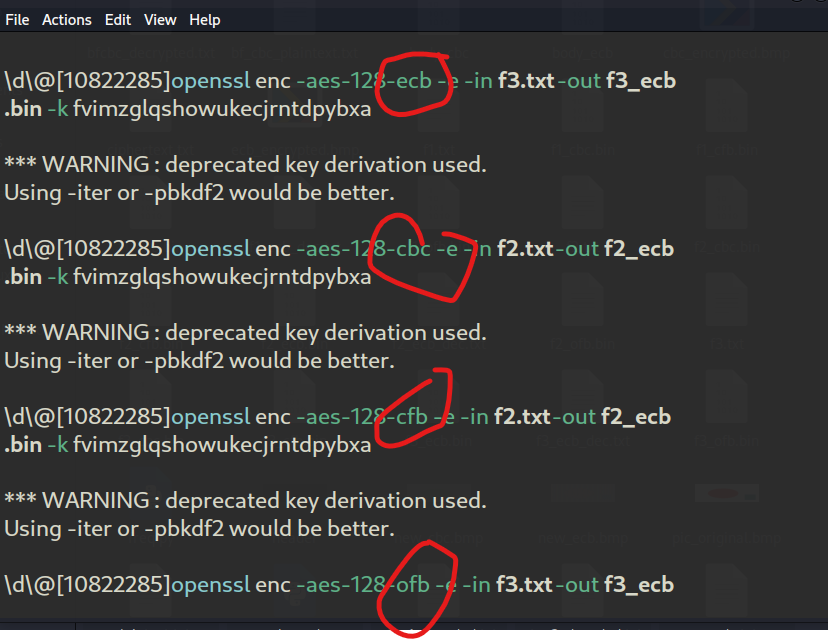
1. **Create the 5-byte file**:
2. **Create the 16-byte file**:
3. **Create the 10-byte file**:



**Step 2: Encrypt the Files**

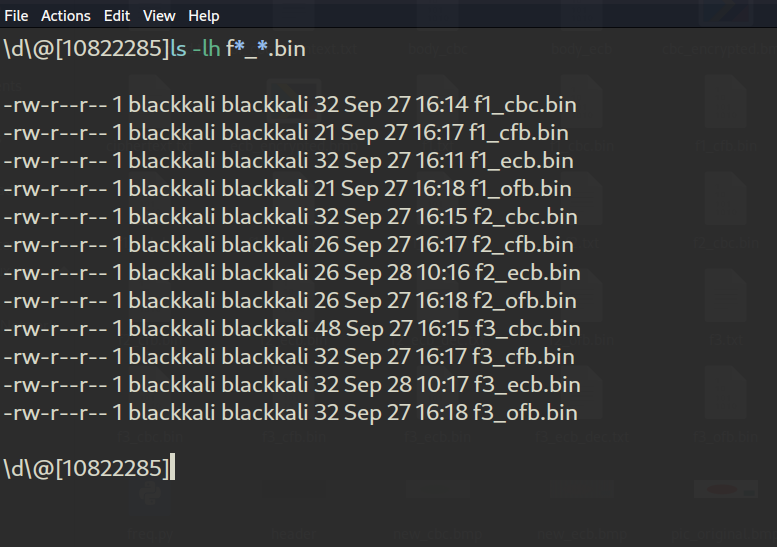
Now, encrypt each file using AES-128 in ECB, CBC, CFB, and OFB modes. Use a key

1. **Encrypt with ECB mode**:
2. **Encrypt with OFB mode**:
3. **Encrypt with CFB mode**:
4. Encrypt with CBC mode



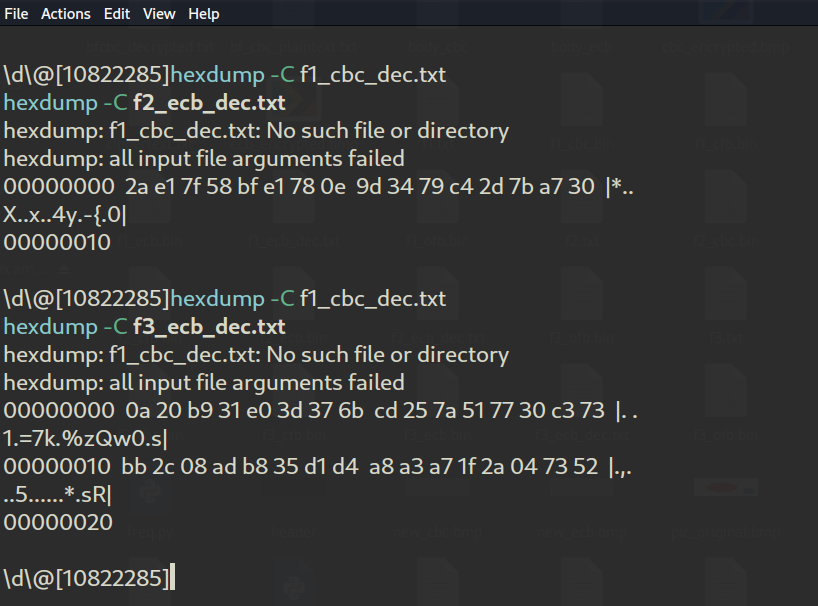
**Step 3: Check the Size of Encrypted Files**

Check the size of each encrypted file to observe how padding affects the output file size:



**Step 4: Use Hex Dump to Display Padding**

Use hexdump to examine the decrypted files and observe the padding:



Analyze padding

* + **ECB**: Yes, padding is required as the input size is not a multiple of the block size.
  + **CBC**: Yes, padding is required for the same reason.
  + **CFB**: No, CFB mode does not require padding as it can handle any length of input data.
  + **OFB**: No, OFB mode does not require padding for the same reason.