

**Information Security**

**Assignment**

**M. Umer Hassan**

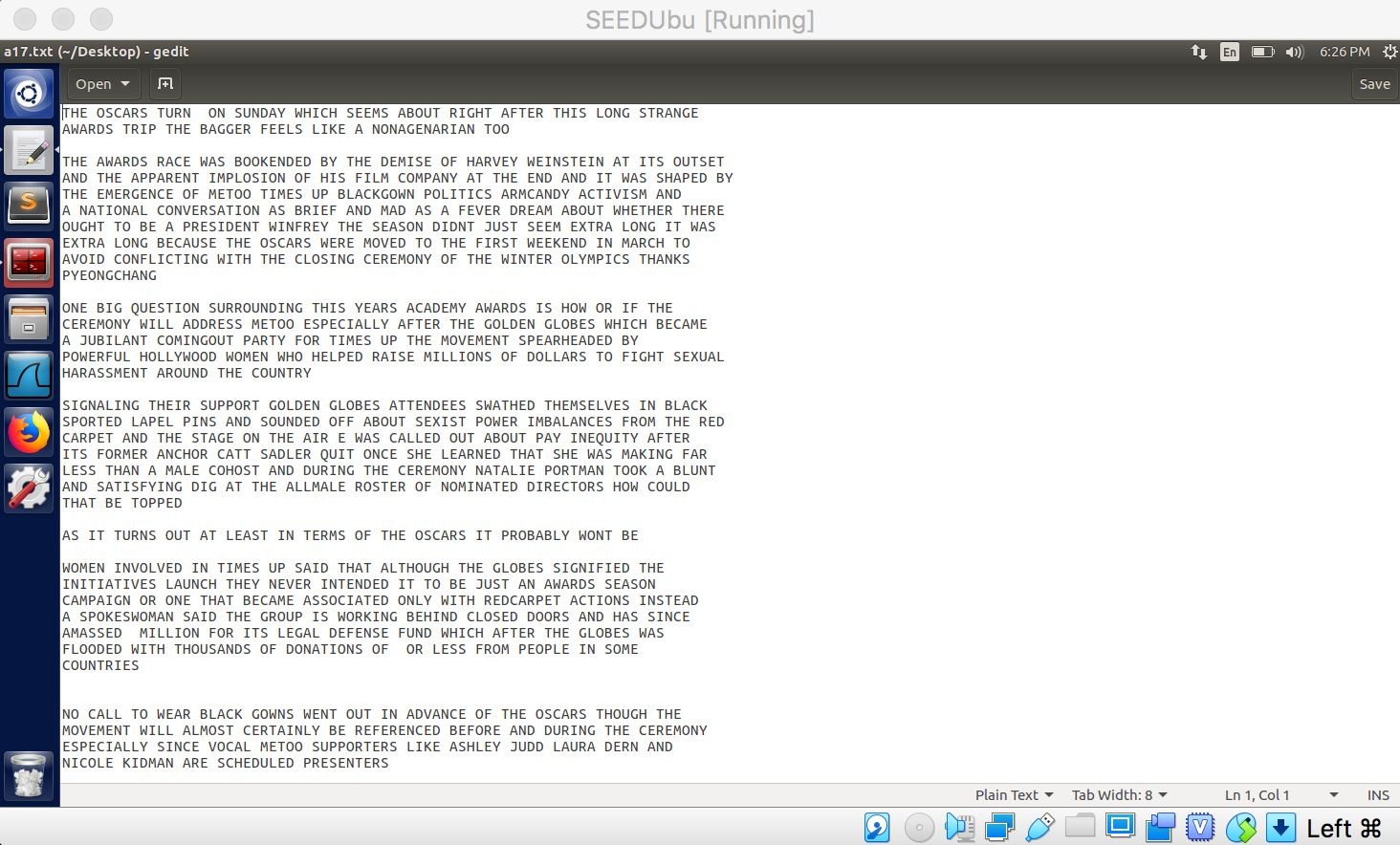
**BS-CS-18-56-7th-M**

# Secret-Key Encryption Lab

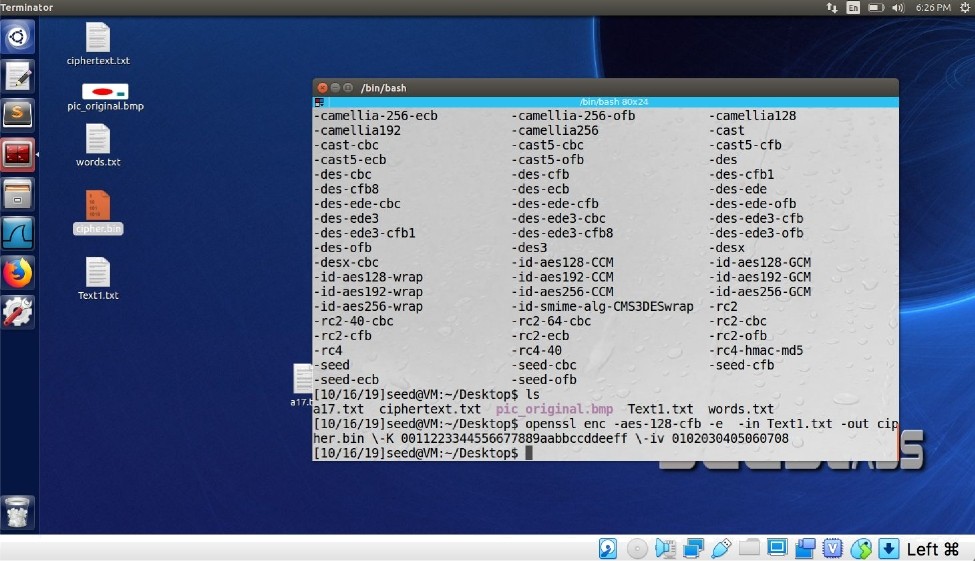
|  |  |  |
| --- | --- | --- |
| **Title** | **Author** | **Date** |
| Secret-Key Encryption Lab(Crypto Encryption) | M.Umer Hassan | 15-12-2021 |

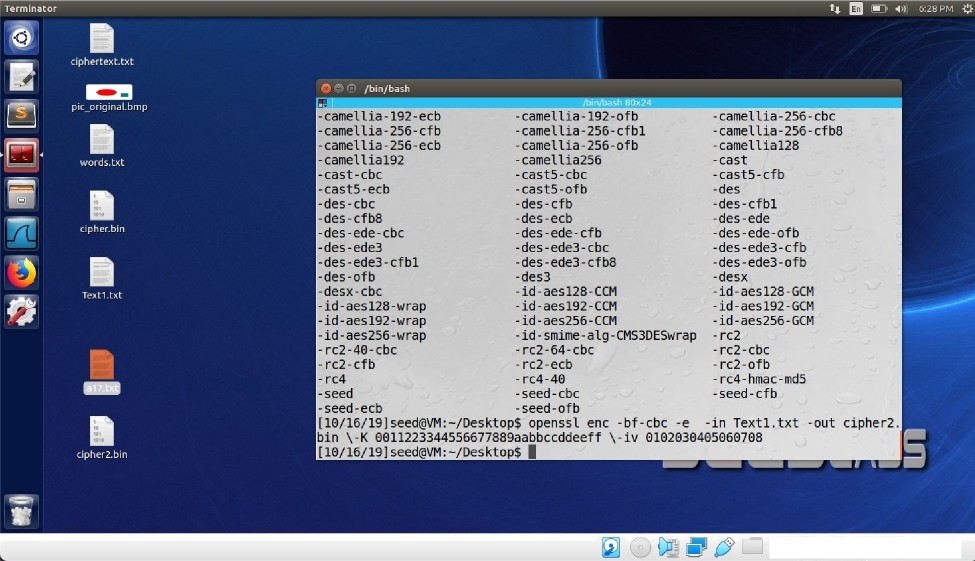
**Task 2: Encryption using Different Ciphers and Modes**

We used the provided ciphertext. Below is the decrypted result in plaintext.

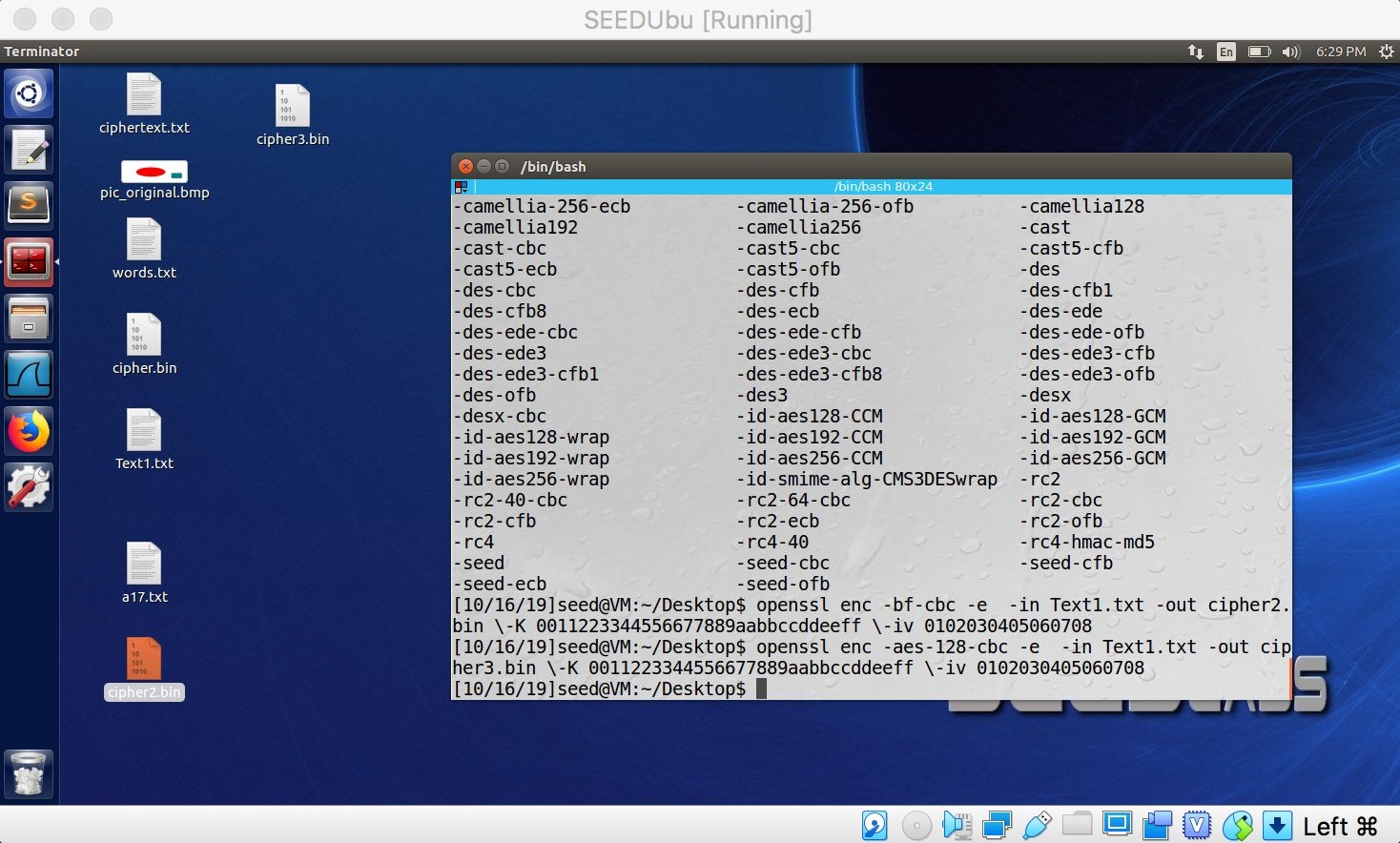


Three different types of encryption below.



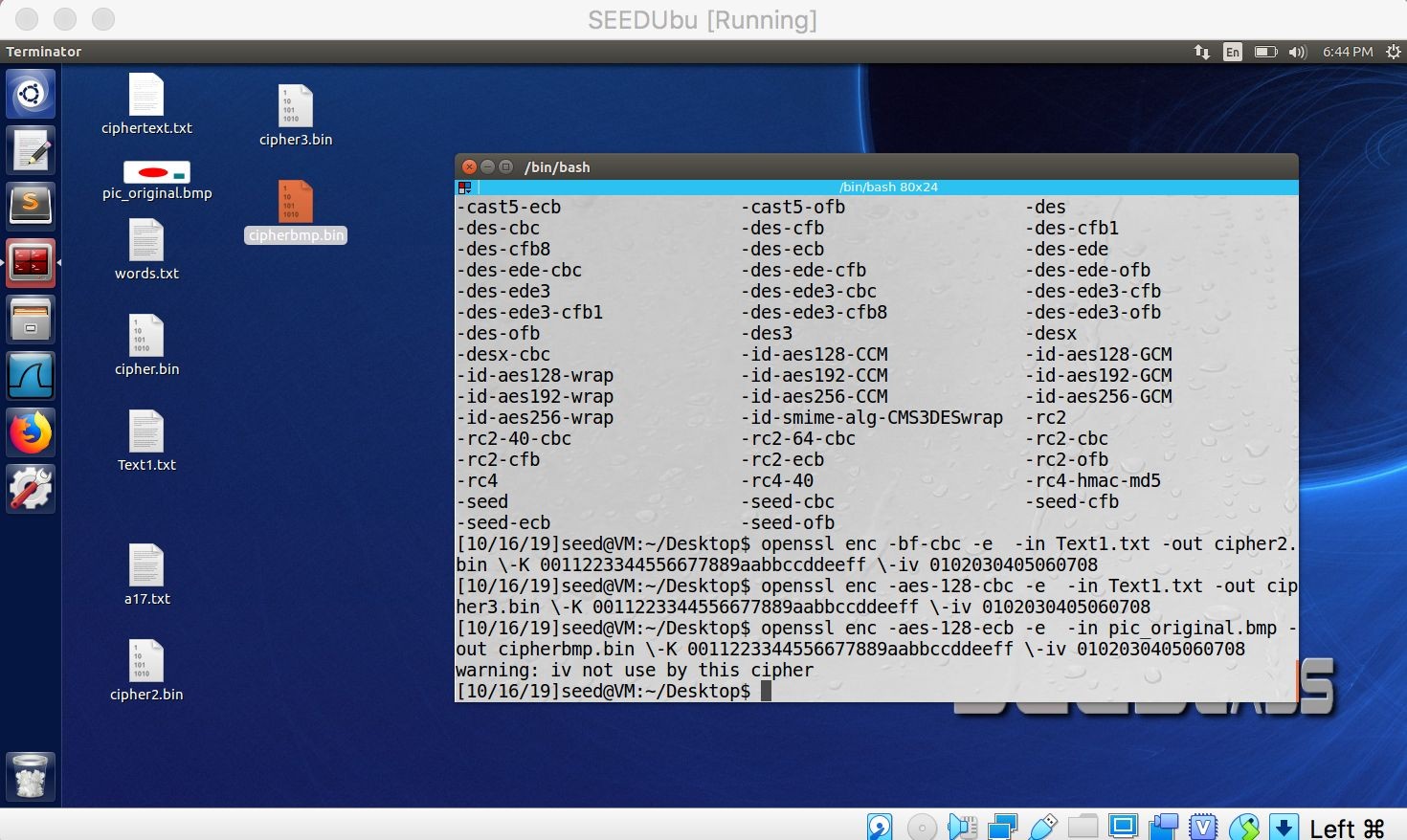


@ @ @ @ @ Left 9g

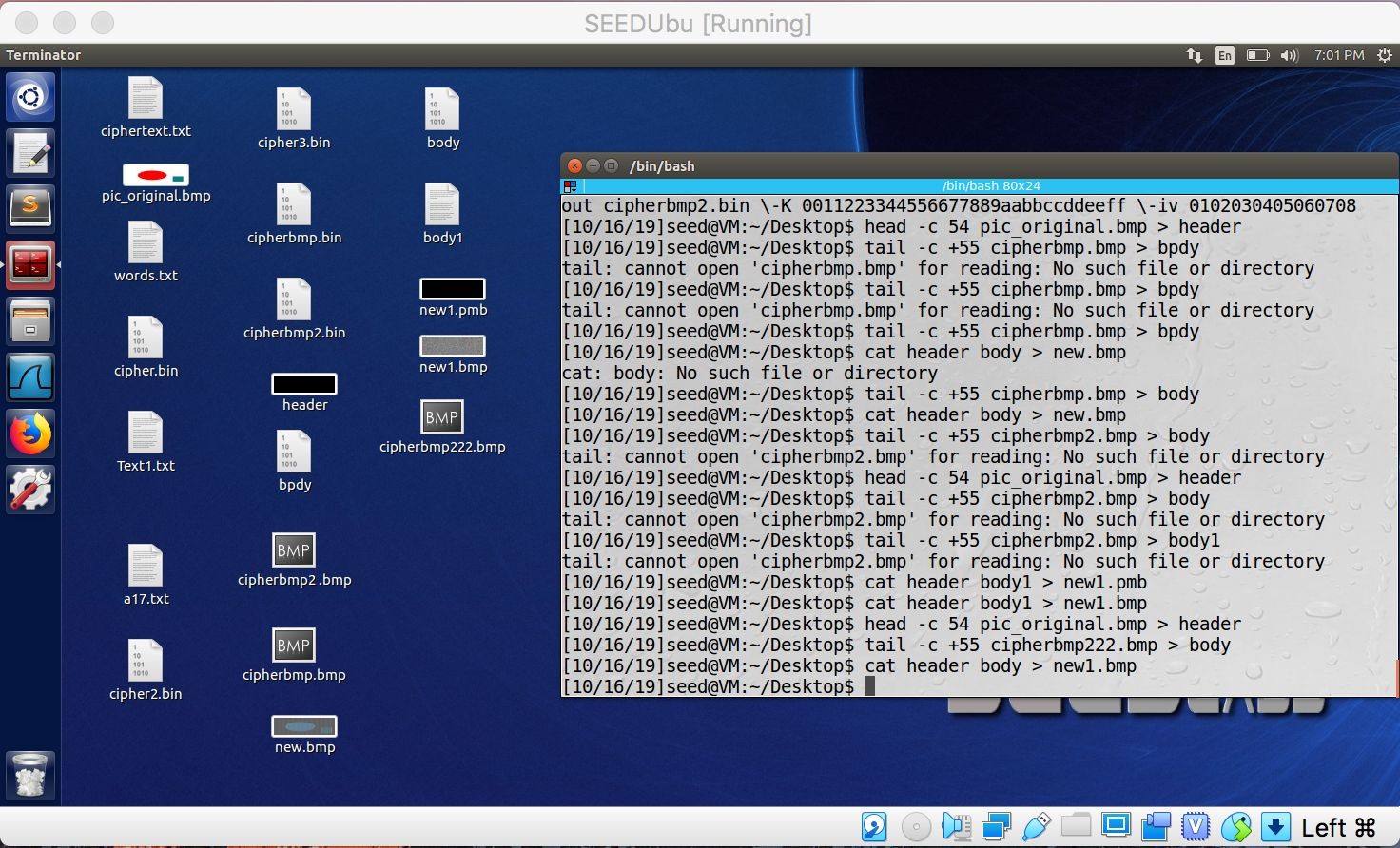


# Task 3: Encryption Mode – ECB vs. CBC

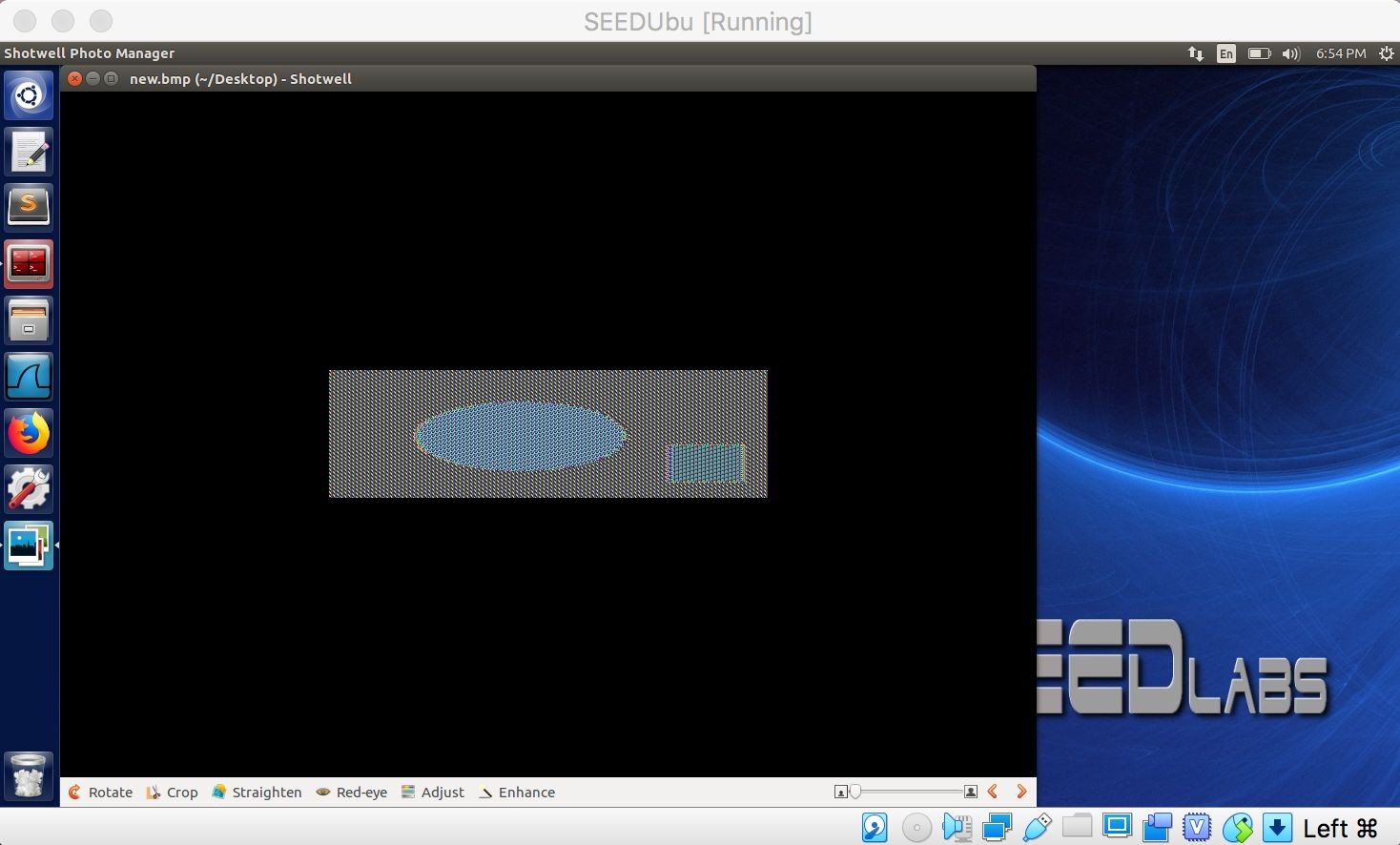
Image encryption using ECB and CBC



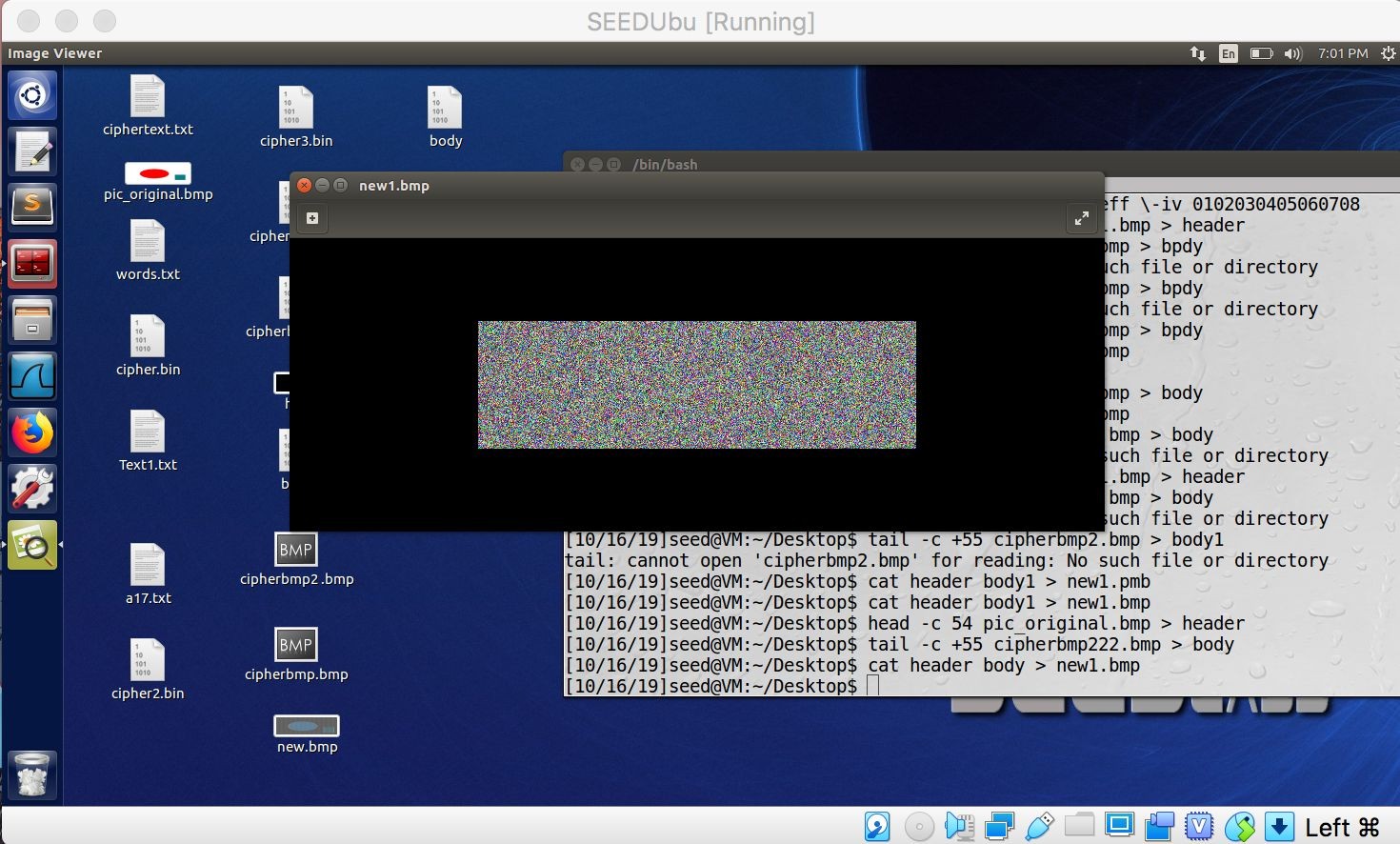
Making encrypted file viewable as image.



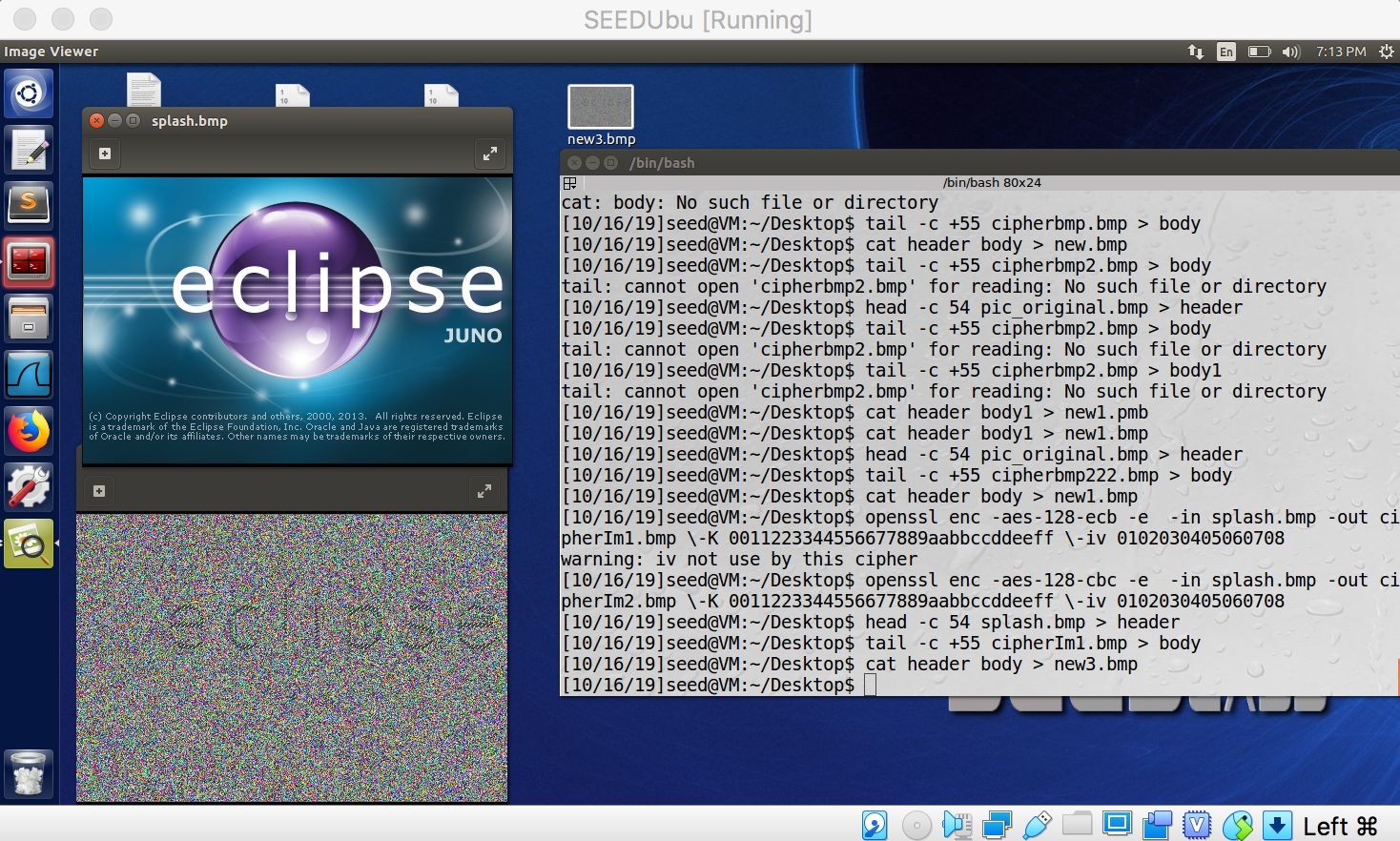
Viewing ECB encrypted image.



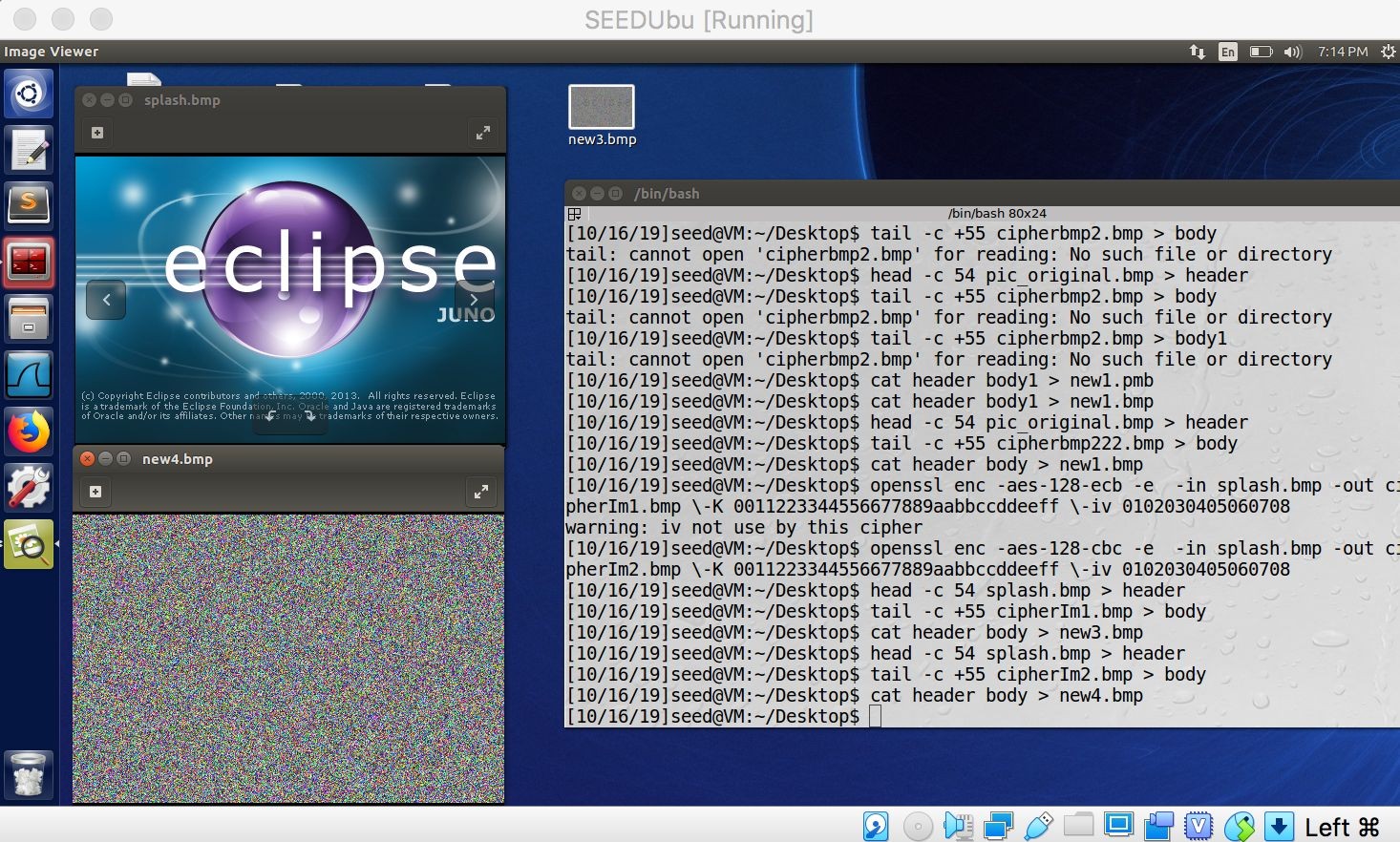
Viewing CBC encrypted image.



Viewing ECB encrypted image. Can make out some of the original image letters.



Viewing CBC encrypted file. Cannot make out any of the original image’s letters.



**2.** Yes, using the ECB encryption it is very easy to make out the shapes of the encrypted image. When using the CBC encryption I cannot see any relationship between the actual image and the encrypted image. When I ran the above encryption again with a newly chosen image, the results were the same. With ECB, I could make out some of the original image in the encrypted image. With CBC, I was unable to recognize any of the original image in the encrypted image.

# Task 4: Padding

1. I began by creating a text file of size 27 bytes. After encrypting the 27 byte file I got the following results with different encryption:

ECB - file size became 32 bytes. Padding was used. CBC - file size became 32 bytes. Padding was used. CFB - files size remained 27 bytes. No padding.

OFB - file size remained 27 bytes. No padding.

Conclusion is that CFB and OFB do not use padding because they are stream ciphers and therefore the plaintext message does not need to be a multiple of the block-size.

Creating three files size 5, 10, 16 bytes

Image of encrypted 5 bytes file, size is 16 bytes.

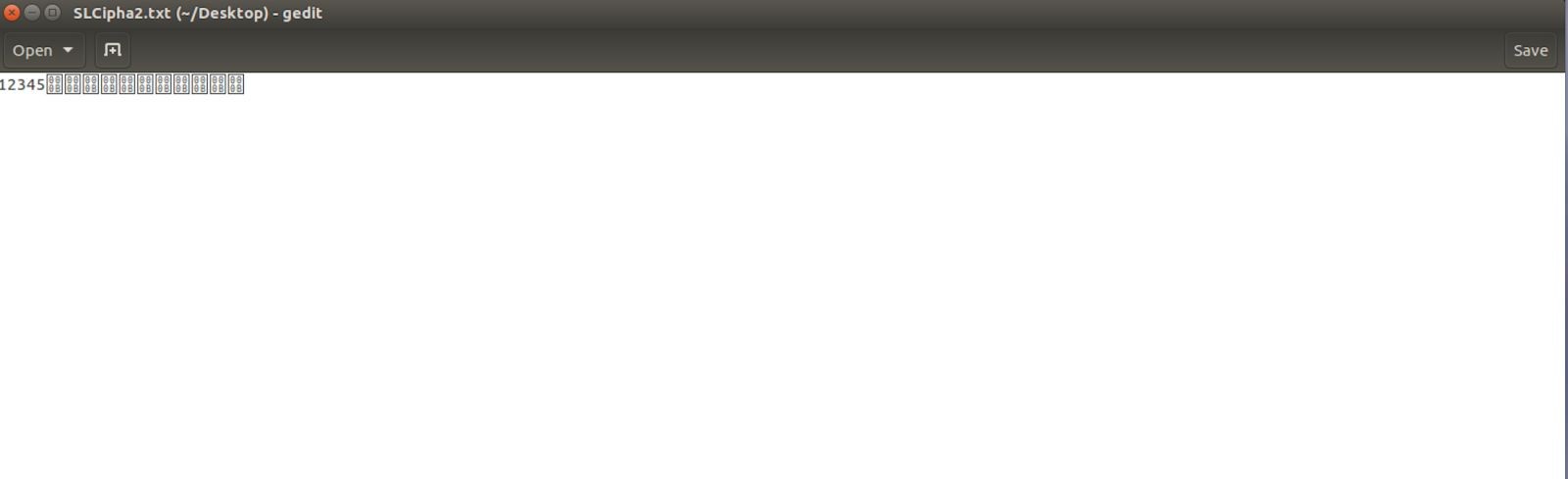
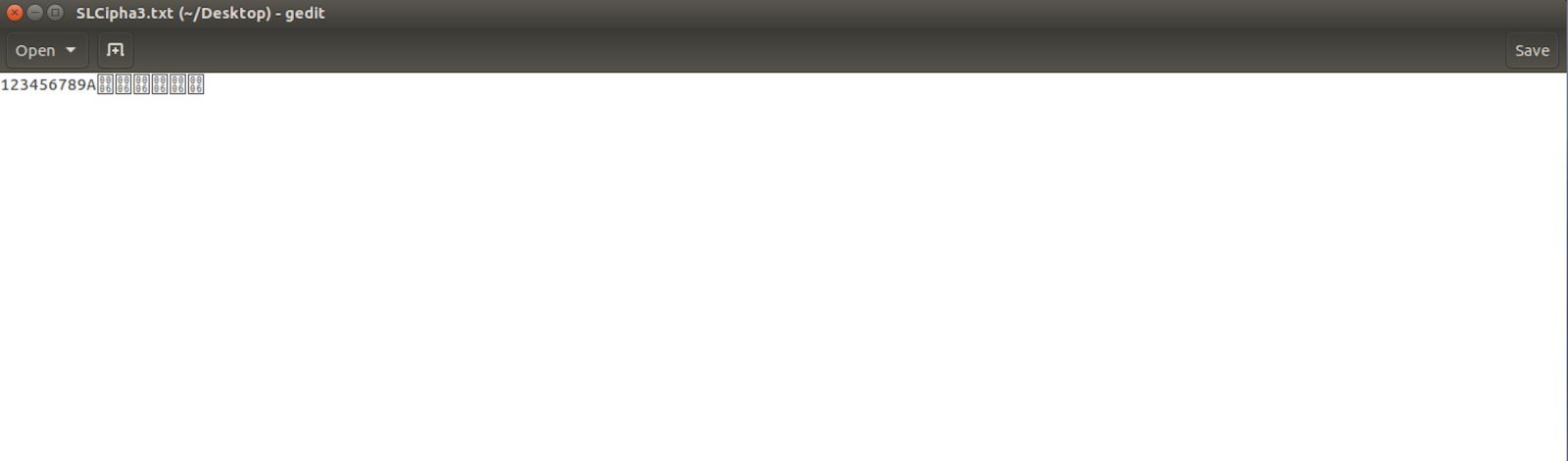
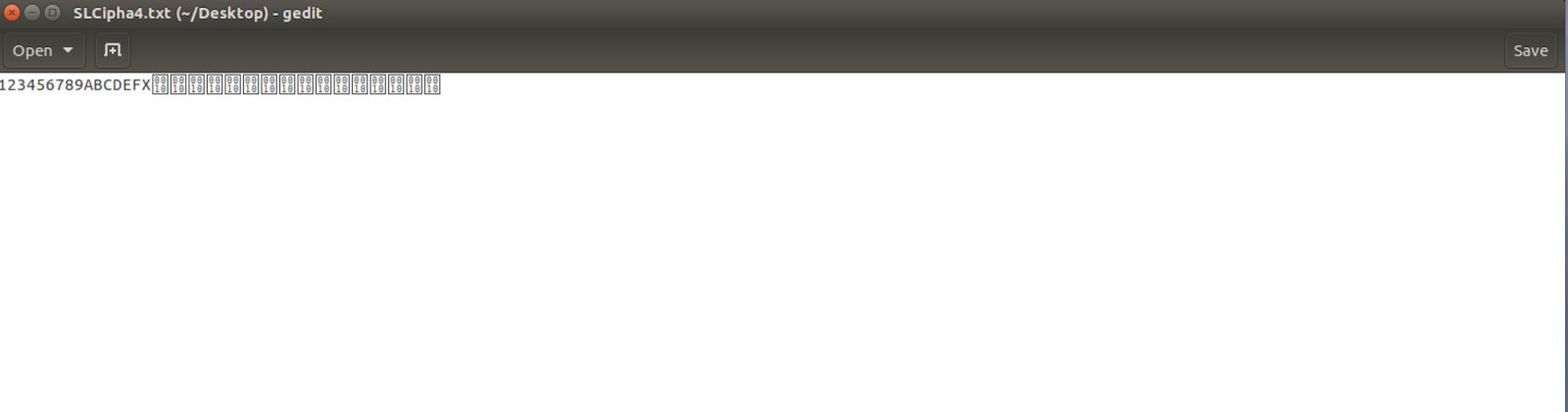
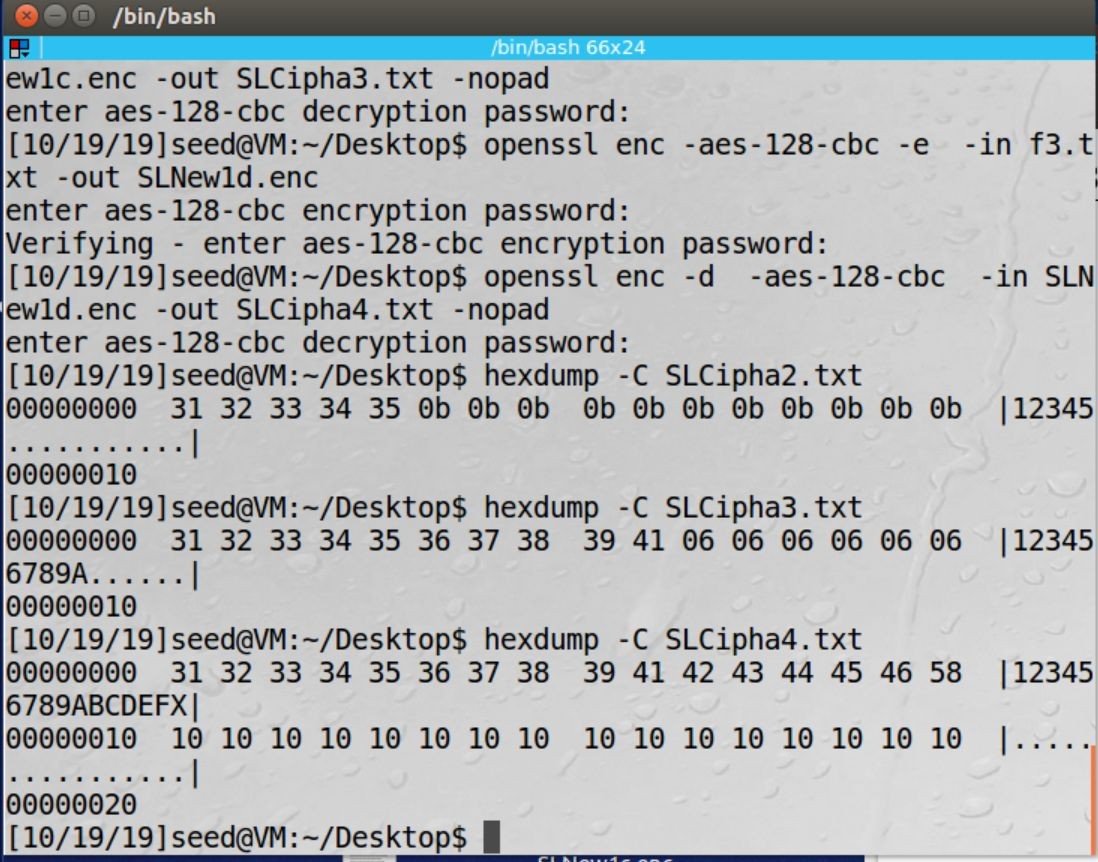


Image of encrypted 10 bytes file, size is 16 bytes.



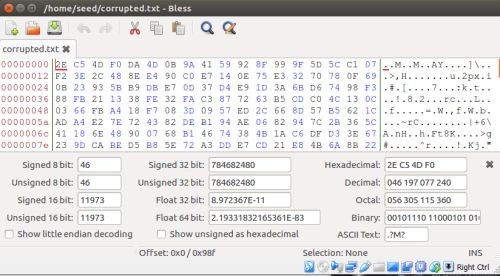
Image of encrypted 16 bytes file, size is 32 bytes.

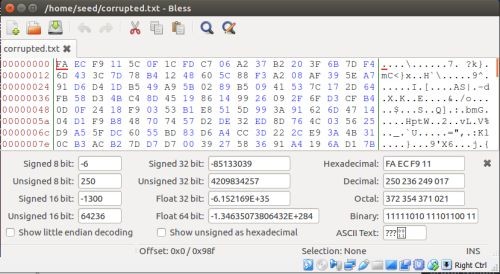
Hex Dump results for finding how each file was padded



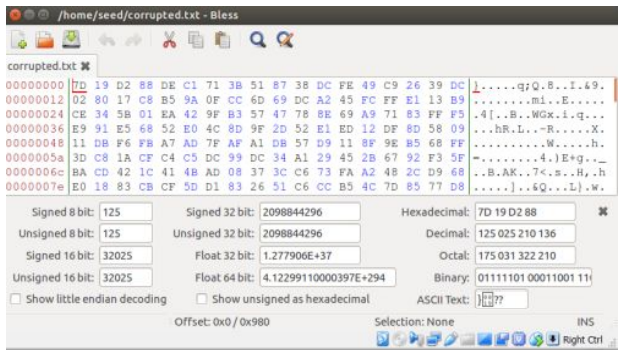
**Task 5: Error Propagation – Corrupted Cipher Text**

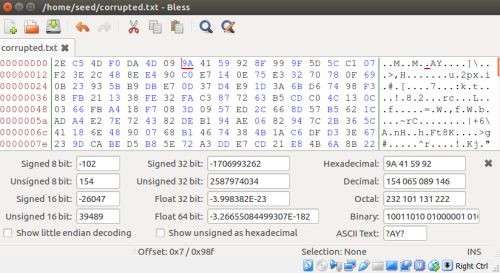
Created a text file greater than 1000 bytes in size. The file is essentially a repeat of NazimZerrouki and GregGertsen (our full names) until we reached or exceeded 1000 bytes. We then encrypted the file using AES 128 bit block ciphers using ECB, CBC, CFB, and OFB. The key and IV that were used was ‘00112233445566778899aabbccddeeff’ and ‘aabbccddeeff998877665544332211’ unless stated otherwise. The files prior to corrupting the 55th bit for ECB, CBC, CFB, and OFB respectively which are shown below:



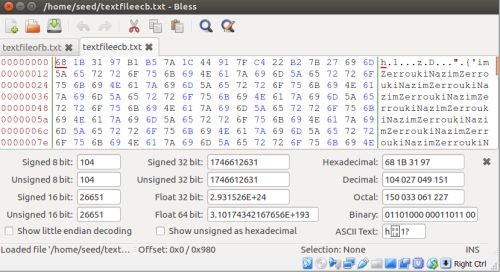




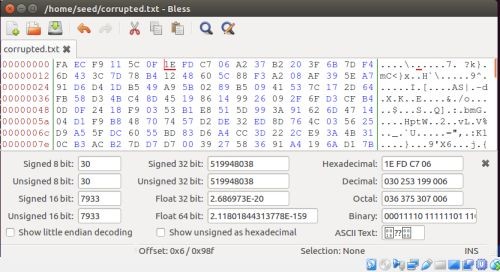


Corrupted encryption by changing 1 bit of the encrypted file for each type of encryption are shown below in the same orde

# Observation:

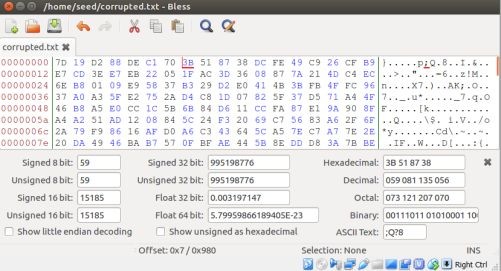


Using ECB to decrypt the corrupted file, we noticed that the first 16 bytes (64 bits) were corrupted while the rest of the bits were completely unaffected.



# Observation:

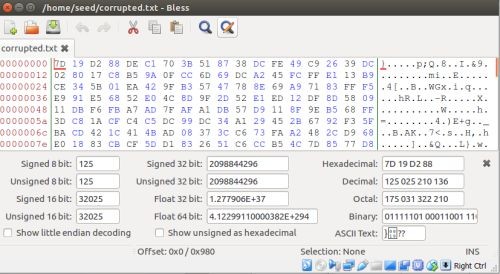
For CBC, after decrypting the corrupted file, we noticed that the majority of the first 23 bytes



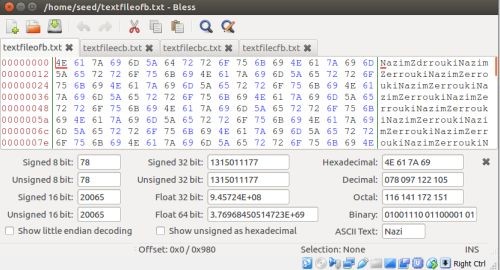
(roughly first 72 bits) were corrupted while the rest of the bits remained unaffected.

# Observation:

When decrypting using CFB, we noticed that the 7th byte was corrupted and bytes 17-21 and bytes 23-32 were corrupted as well. The remaining file was left completely unaffected.



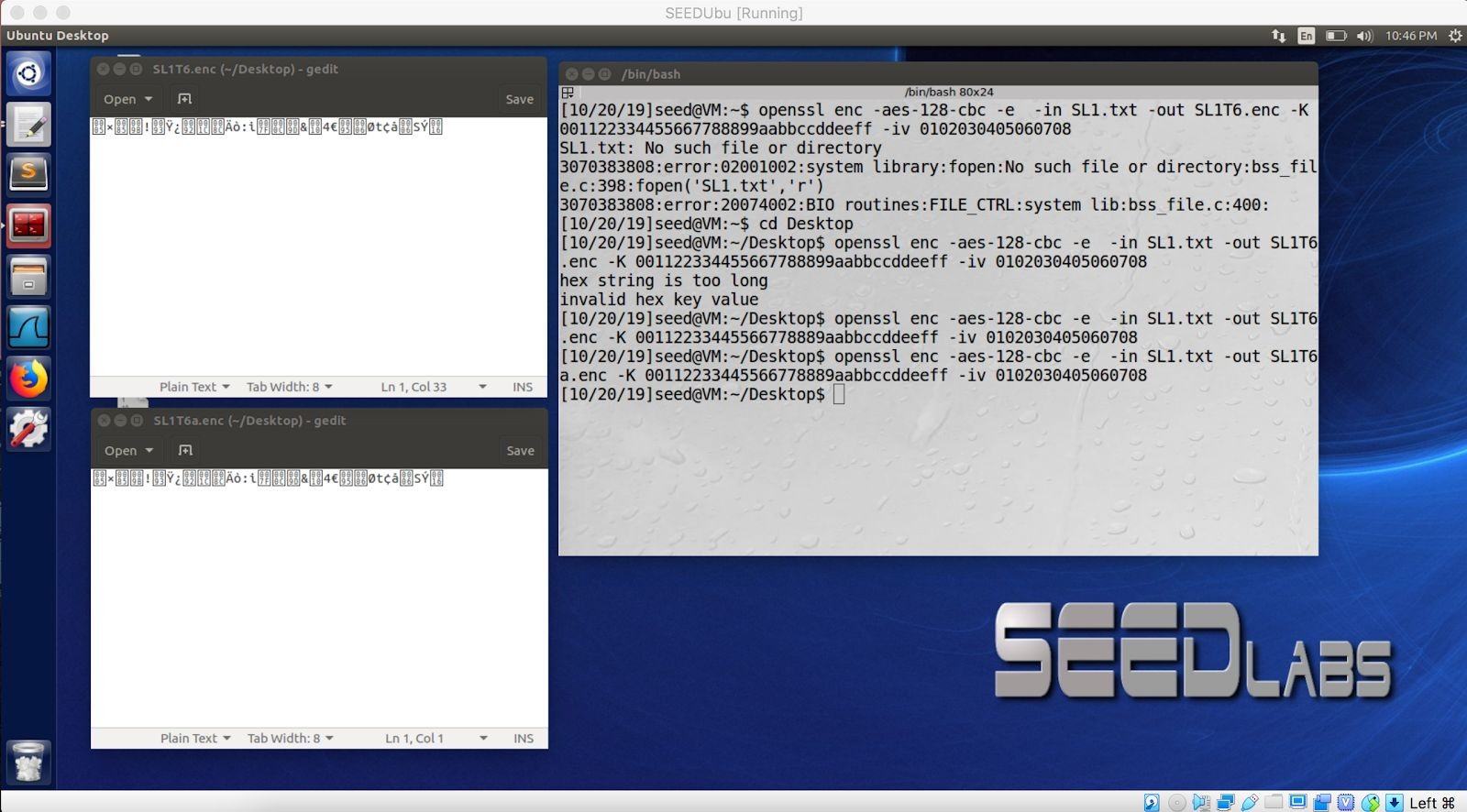
# Observation:

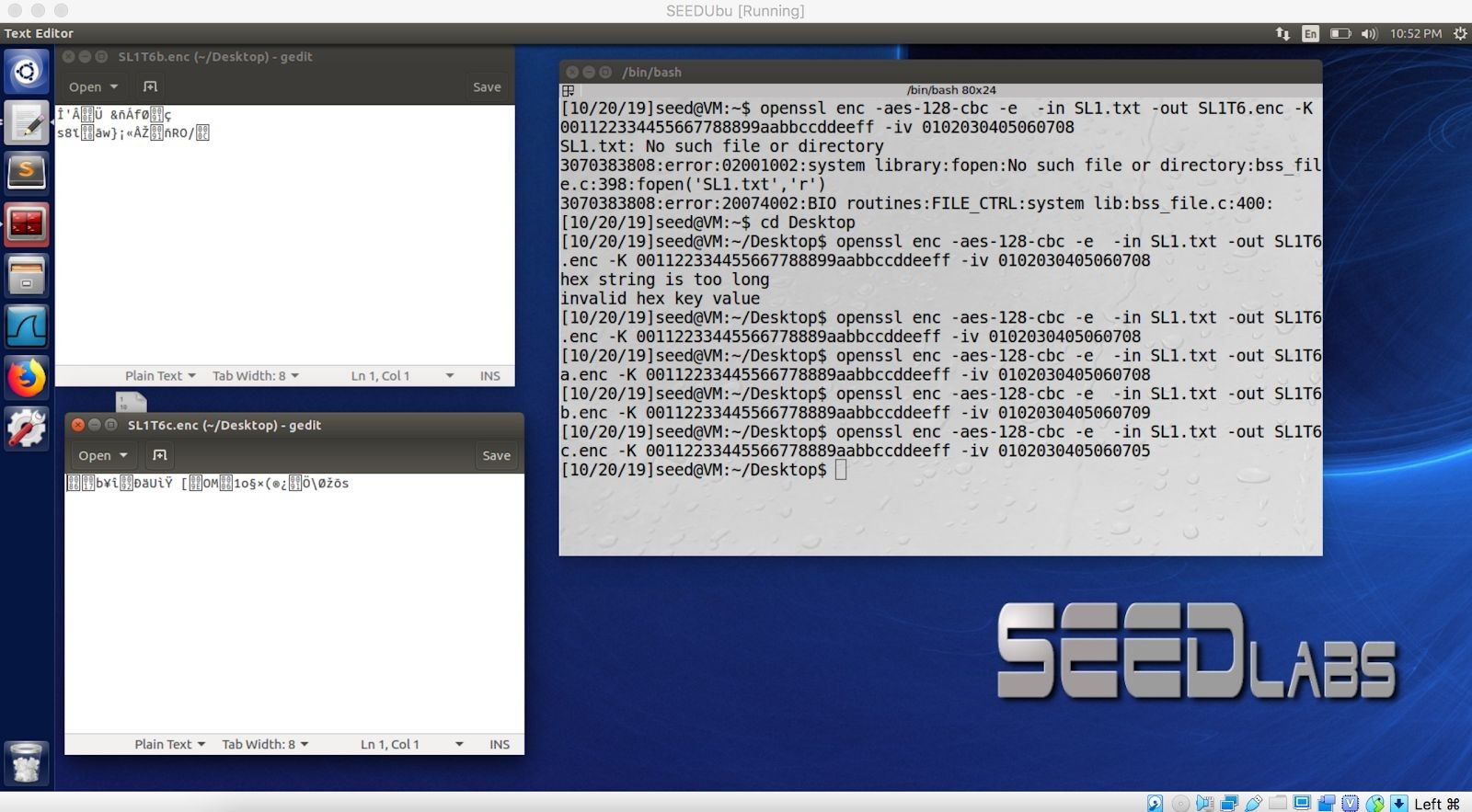


When decrypting using OFB, we noticed that only the 7th byte was corrupted while the rest of the bytes remained intact.

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

**Task 6.1. IV Experiment**

Using the same IV twice on the same plaintext file resulted in and identical output for the encryption.

With different IVs the encryption results were completely different.

The IV needs to be unique because if we are using the same plaintext more than once, not having a unique IV each time will result in an identical ciphertext and a pattern will start to emerge if there is an adversary observing our encryption.

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

When reusing OFB and same IV for encryption, even if our two plaintext messages to be encrypted are not identical there patterns will start to emerge and the algorithm will become deterministic. IN the case of P1> C1, P2>C2 there are some repeats in the ciphertext between C1 and C2, this can be used to crack the encryption. By looking at the two ciphertext it looks like maybe both plaintext messages end with an ‘!’ mark.

When using CFB the attacker can only know the first block of the message. **Task 6.3. Common Mistake: Use a Predictable IV**

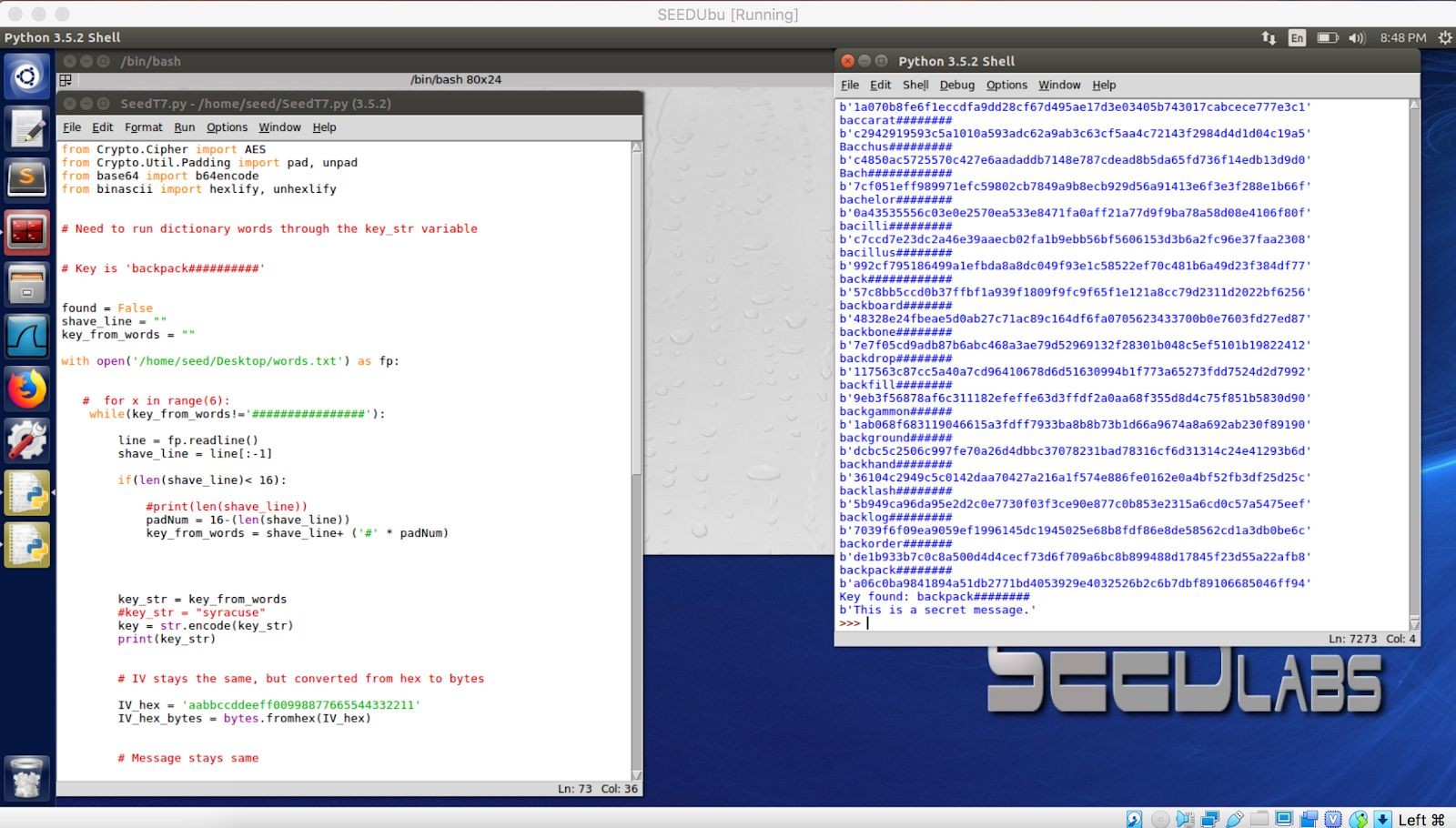
If you know that the plaintext is either ‘yes’ or ‘no’ and you need to guess which one it is from cipher text and you can predict the IV. Then it is possible to verify your guess by submitting either ‘yes’ or ‘no’ with a known IV.

# Task 7: Programming using the Crypto Library

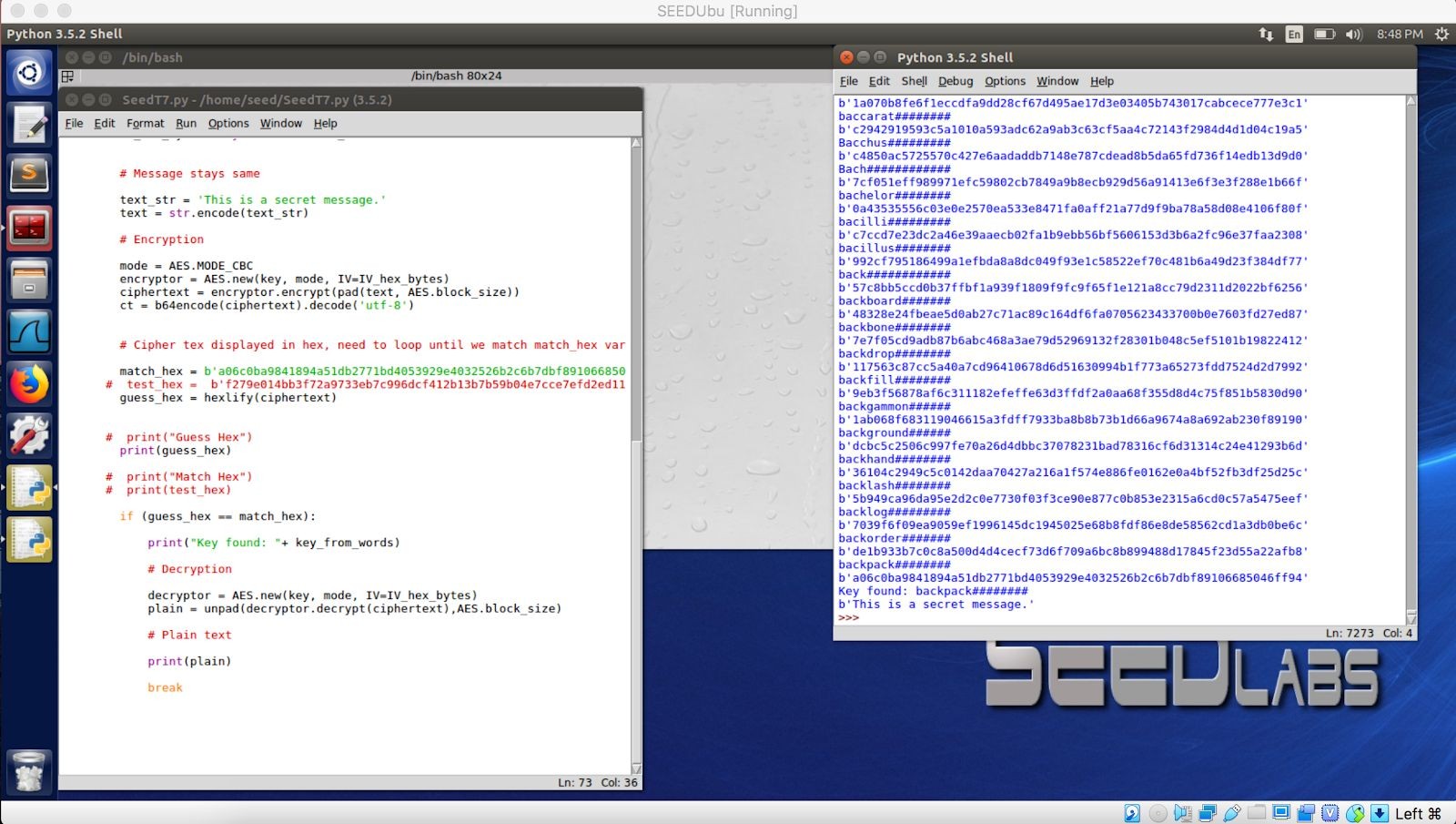
For task 7 we wrote a program in Python using the Pycrypto library formaking a dictionary attack on the AES-128-CBC encryption. The program works and I have demonstrated this below with screenshots. Basically, when given a ciphertext, message, and IV. It will run through a word list + padding them with #s to be 16 bytes long, and when it outputs a match to the desired ciphertext then program ends and then displays the key that generated the result. The only issue was that perhaps due to the different implementation of the encryption library as compared to Openssl, I was unable to generate the same key that was given to us in the lab.

Therefore, I just generated a new ciphertext as the target and ran the program until it matched it. In this regard it worked fine. Please see below images

Program and output after finding the key, word was ‘backpack’ (‘backpack########’). This image shows top half of program



This image shows remaining bottom half of program. And demonstrates that we were able to run a successful dictionary attack on AES-128-CBC encryption when given the IV, message, and ciphertext.



GitHub Repository:

<https://github.com/M-Umer-Hassan/Secret-Key-Encryption-Lab-Crypto-Encryption-.git>