

Secret-Key Encryption

Candidate Name: Alizeh Jafri

Table of Contents

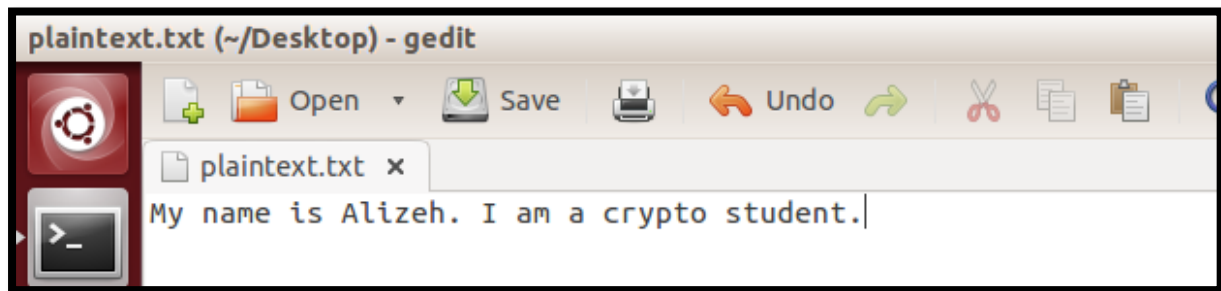
Introduction:	3
Task 1: Encryption using different ciphers and modes.....	3
Task 2: Encryption Mode – ECB vs. CBC.....	5
Task 3: Encryption Mode–Corrupted Cipher Text	8
Task 4: Initial Vector (IV)	15

Introduction:

The purpose of this lab exercise is to get familiarized with the concepts of secret-key encryption. This would allow the candidates to get a hand on experience with the encryption modes, encryption algorithms, padding and initial vector (IV).

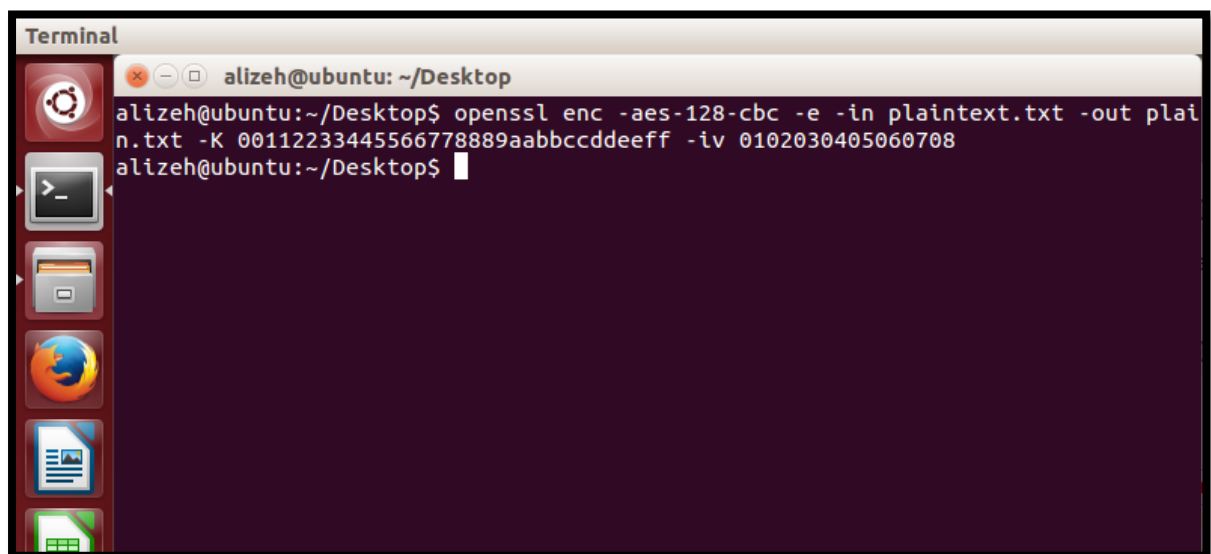
Task 1: Encryption using different ciphers and modes

The file name 'plaintext.txt' was created and some text was typed as shown in the screenshot below:

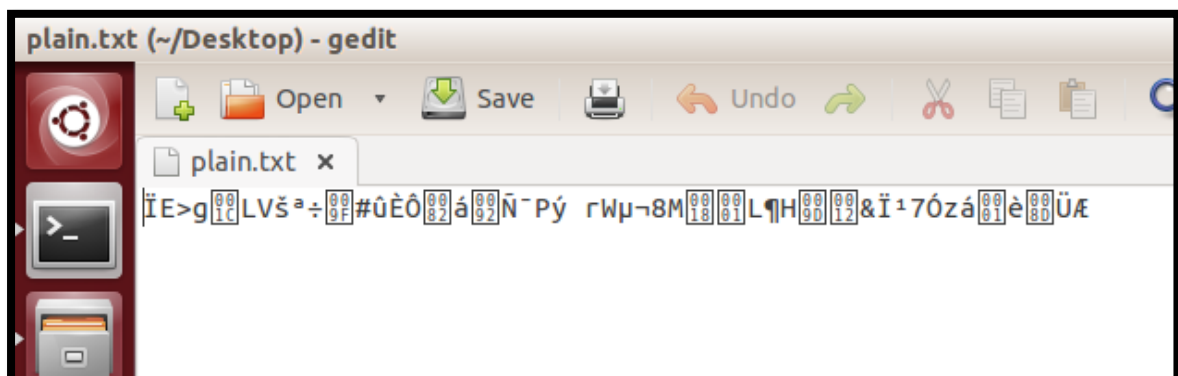


Encryption:

Next, in order to encrypt this file the command was typed and the encrypted output appeared in the 'plain.txt' file as shown below:



The *encrypted* form of text in the file 'plain.txt' is shown below:

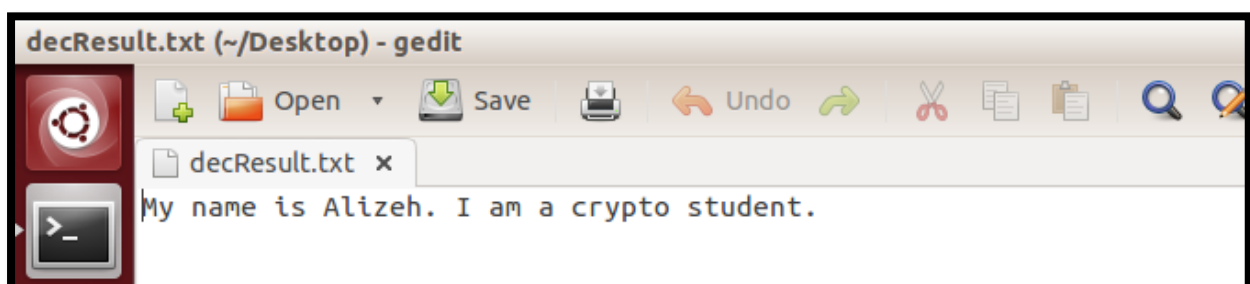


Decryption:

Furthermore, the file 'plain.txt' was decrypted using the following command as shown below:

```
alizeh@ubuntu:~/Desktop$  
alizeh@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -d -in plain.txt -out decResult.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708  
alizeh@ubuntu:~/Desktop$
```

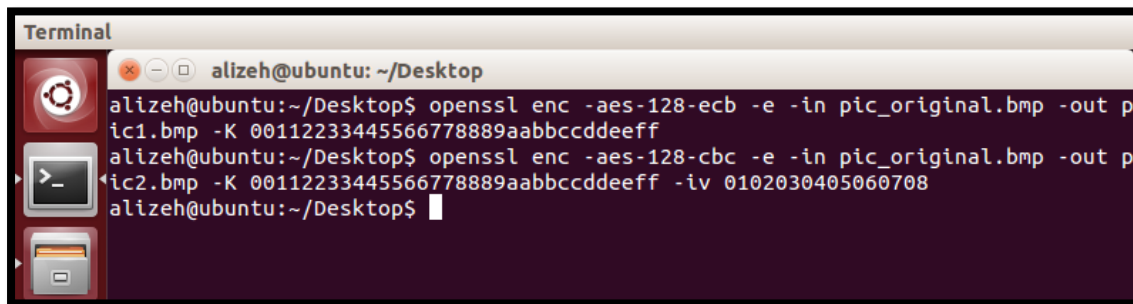
There is no difference between the decrypted file 'decResult.txt' and the file 'plaintext.txt'. This means, what happened was that the file named plaintext.txt was created and encrypted using an encryption command. Then, the encrypted file 'plain.txt' which was the output file from the encryption was decrypted using the decryption command and decrypted file named 'decResult.txt' was created showing the same normal text. The result after the decryption is shown below:



Task 2: Encryption Mode – ECB vs. CBC

In this task the file name 'pic_original.bmp' will be encrypted. So, the file can only be seen by the people who has the encryption keys.

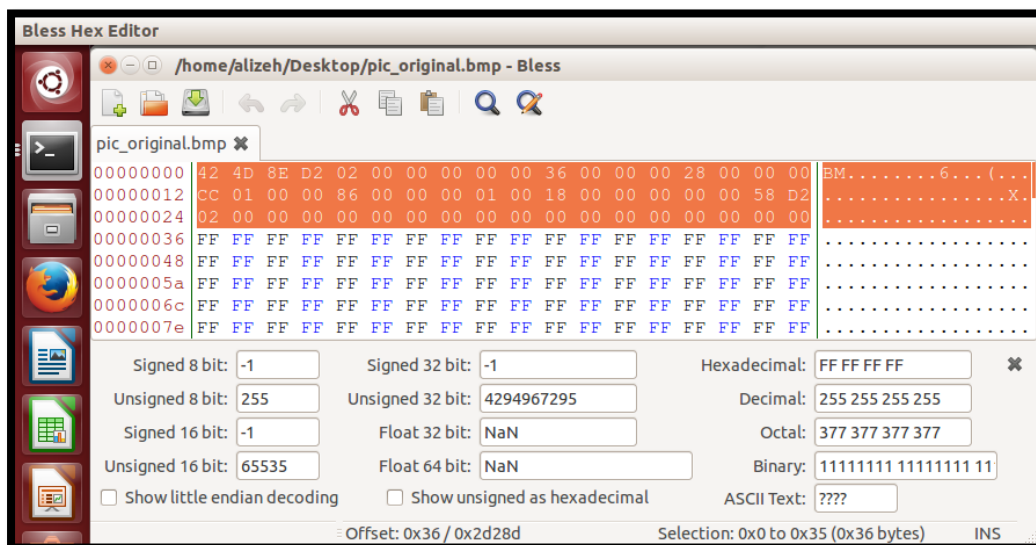
The screen shot below shows the command for encrypting the 'pic_original.bmp' file in **ECB** and **CBC** modes:



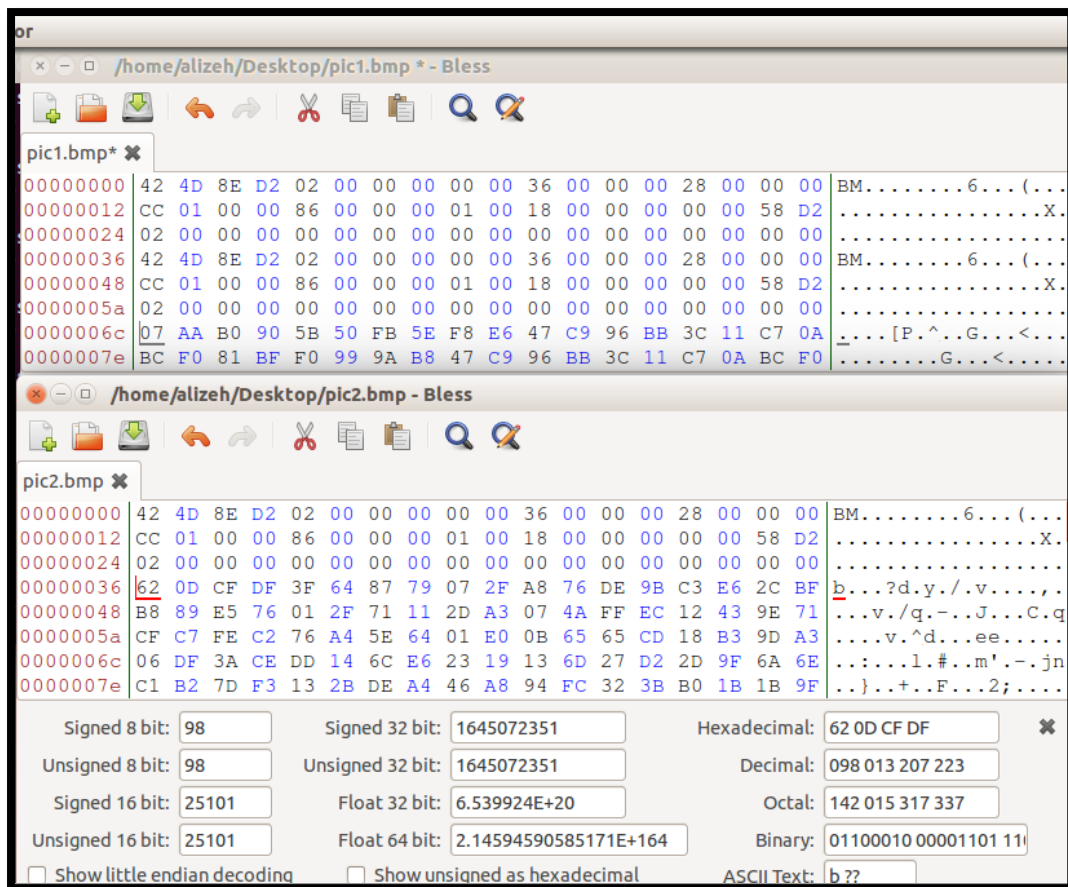
```
Terminal
alizerh@ubuntu: ~/Desktop
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic1.bmp -K 00112233445566778889aabbccddeeff
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic2.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
alizerh@ubuntu:~/Desktop$
```

replace the header (first 54 bytes) of the encrypted picture with that of the original picture. We can use the **bless hex editor** tool (already installed on our VM) to directly modify binary files.

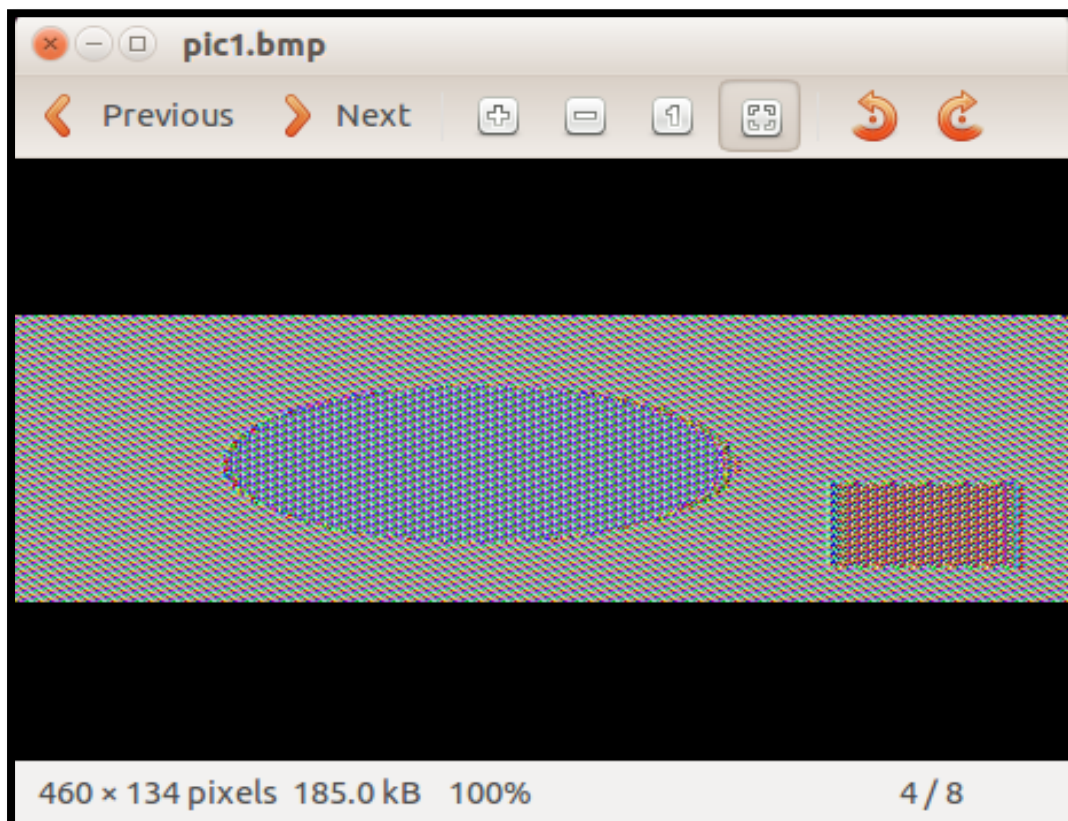
The first 54 bytes of the picture is shown below:



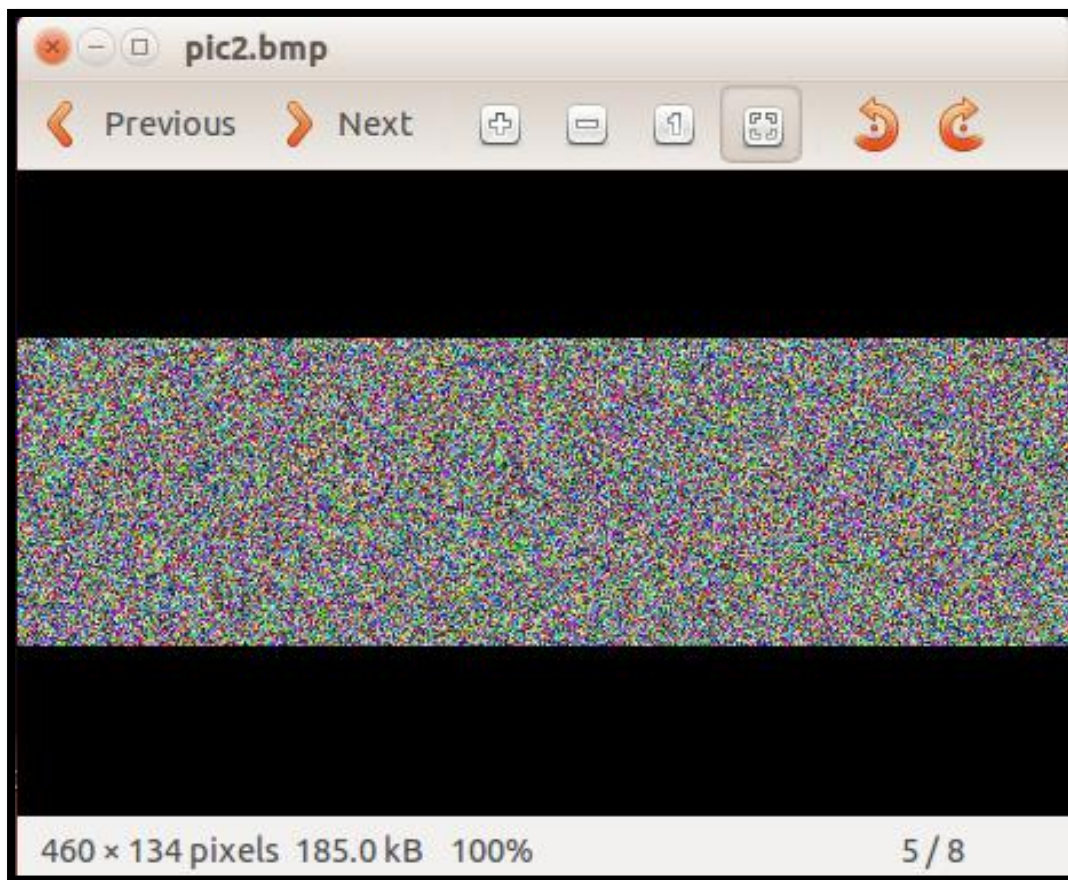
The header of the encrypted picture is replaced with the original picture. For this, the **Bless Hex Editor** tool is used as shown below:



The screen shot below shows the result of the **ECB mode**:



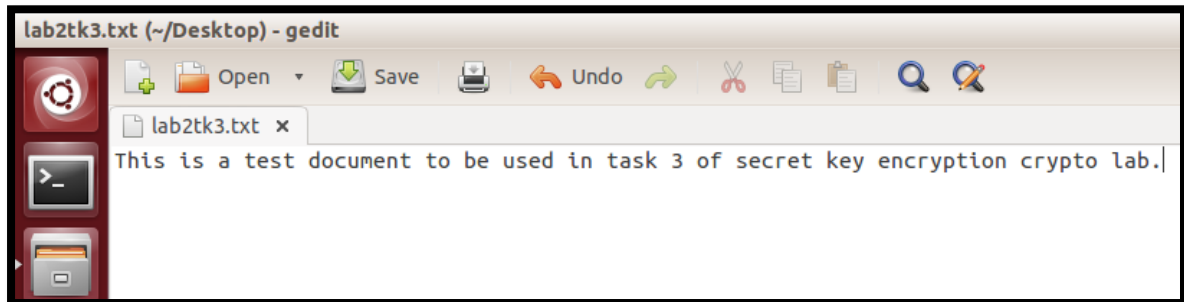
The screen shot below shows the result of the **CBC mode**:



From the results shown above, useful information about the original picture from the encrypted picture under ECB and CBC can be derived. As in ECB the picture is half visible whereas in CBC mode nothing is visible i.e the picture is fully encrypted. This is the reason why CBC can protect you from deriving information about the picture because in CBC iv is present while in ECB there is no iv. Therefore, because of having iv in the CBC mode, the image is completely encrypted.

Task 3: Encryption Mode–Corrupted Cipher Text

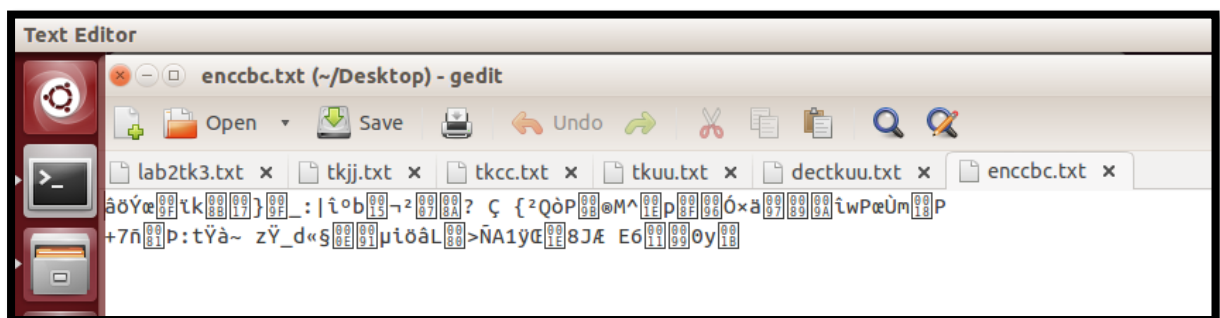
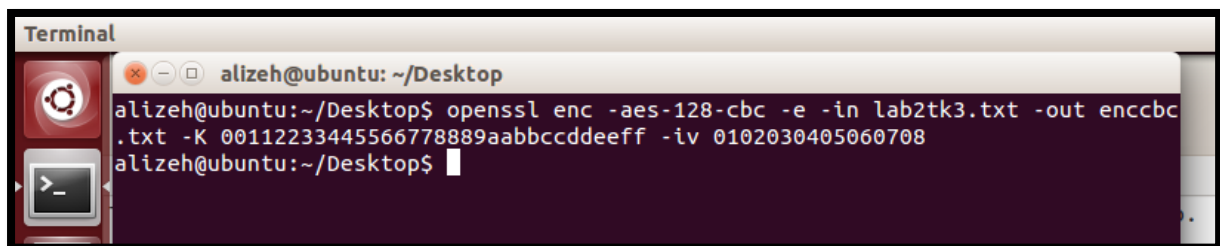
A text file was created.



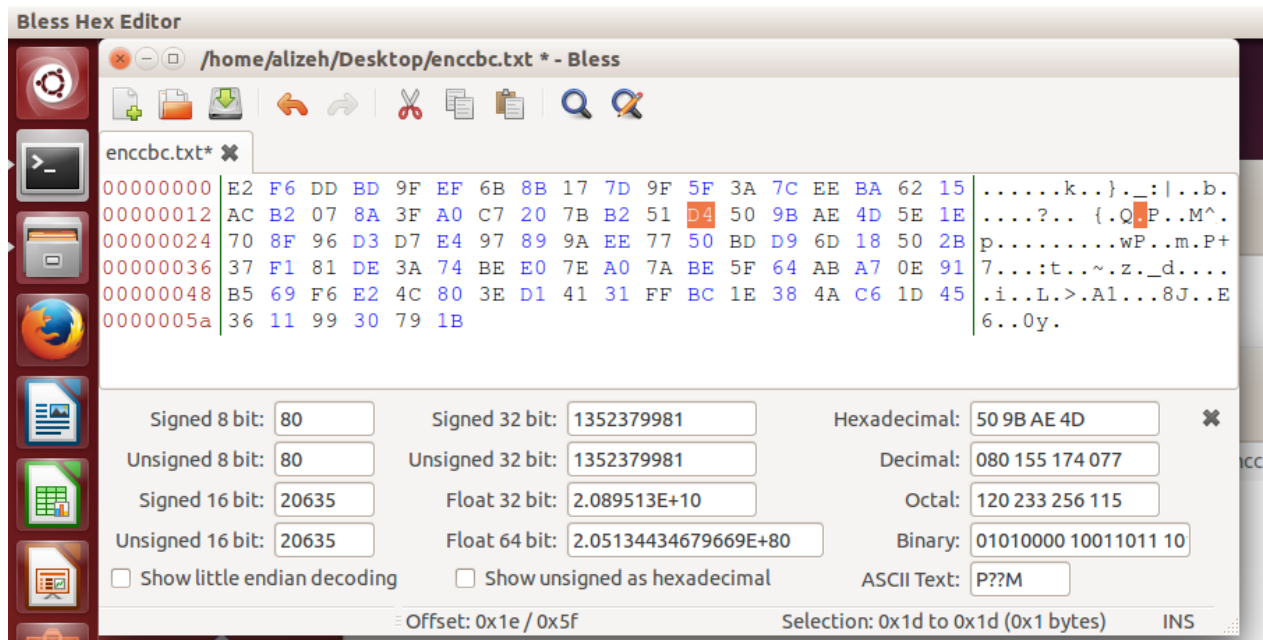
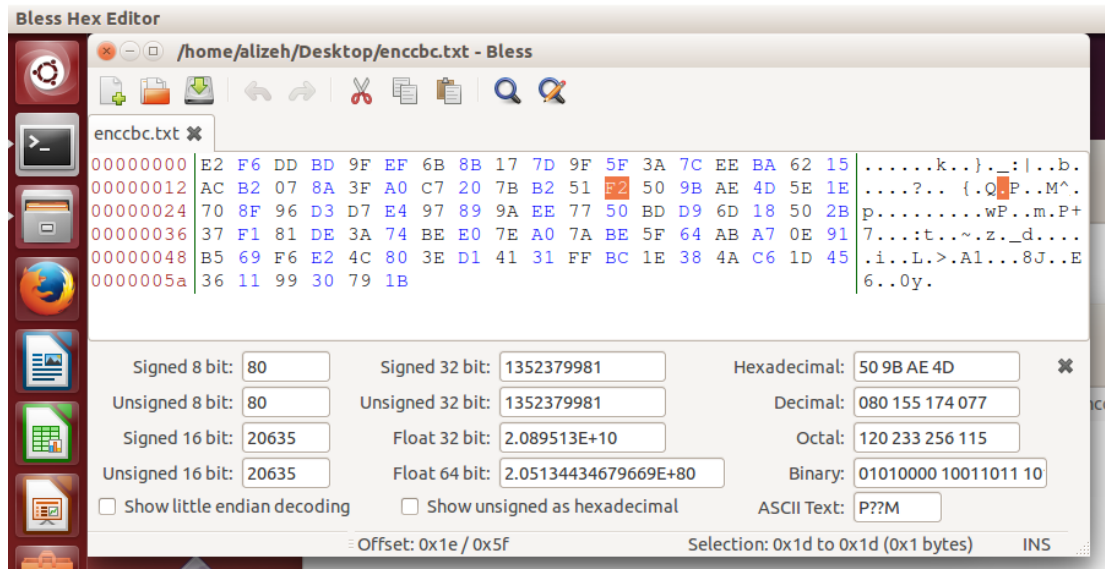
CBC

Encryption:

The screen shots below shows the encryption commands and the results from the encrypted file for CBC mode:

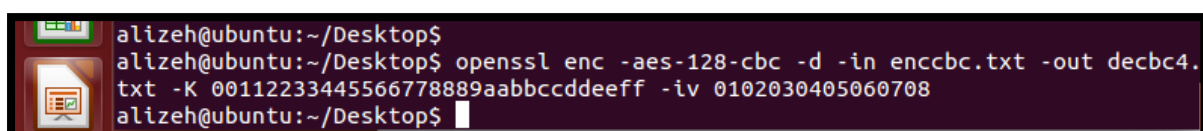


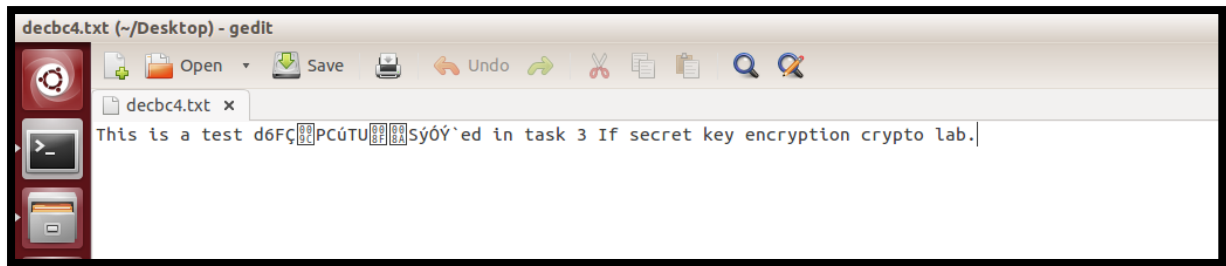
File corruption:



Decryption:

The screen shots below shows the decryption commands and the results from the decrypted file for CBC mode:

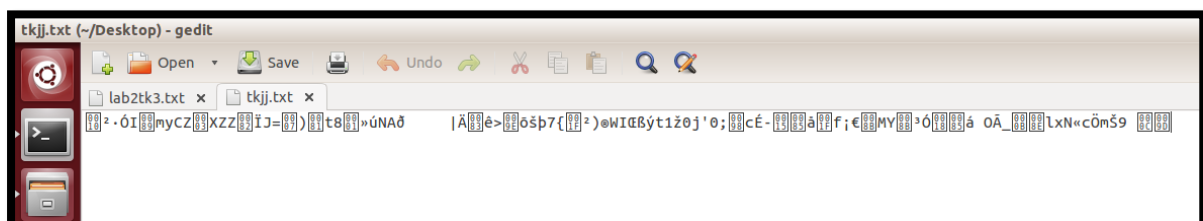
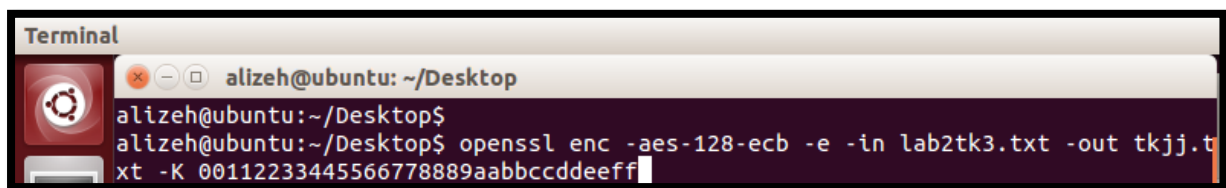




Here, most of the text is visible.

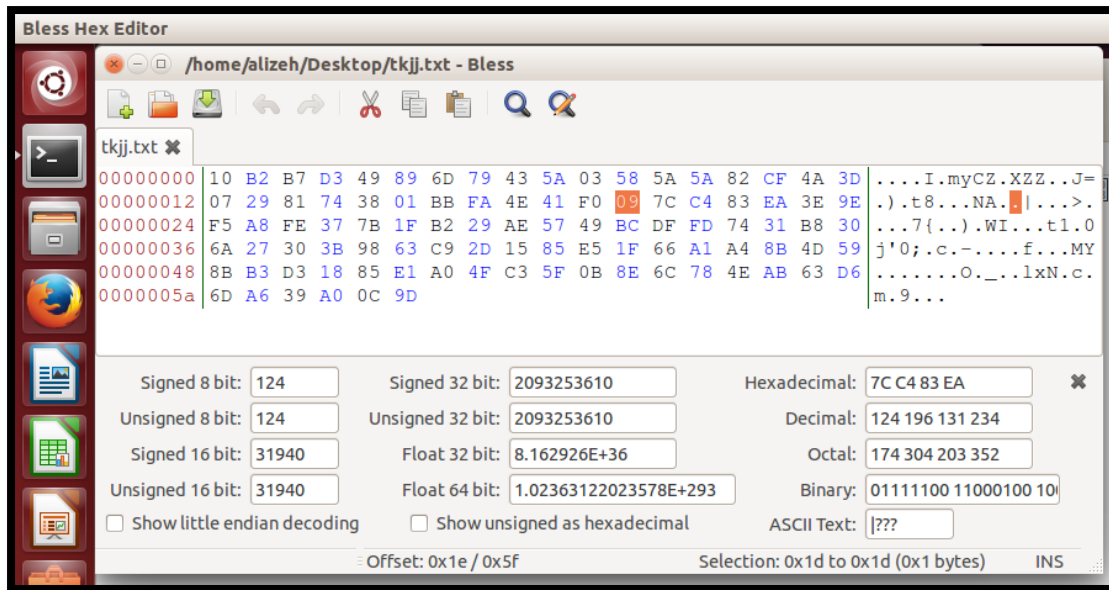
ECB (no iv):

The screen shots below shows the encryption commands and the results from the encrypted file for ECB mode:

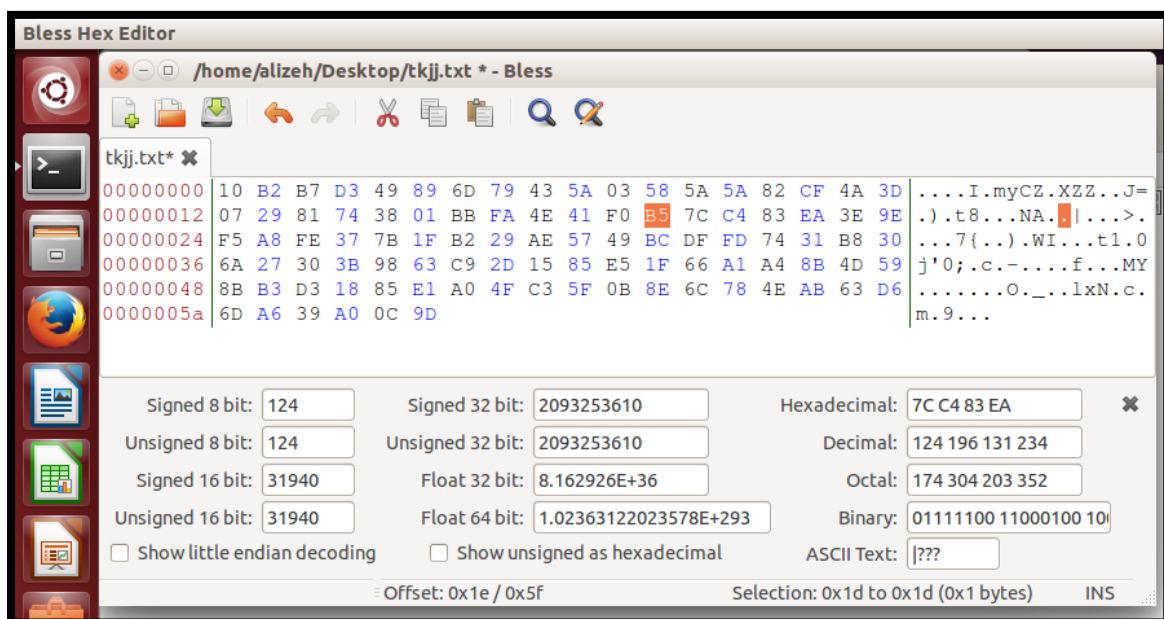


The screen shots below shows corruption of the 30th byte in the encrypted file

From



to

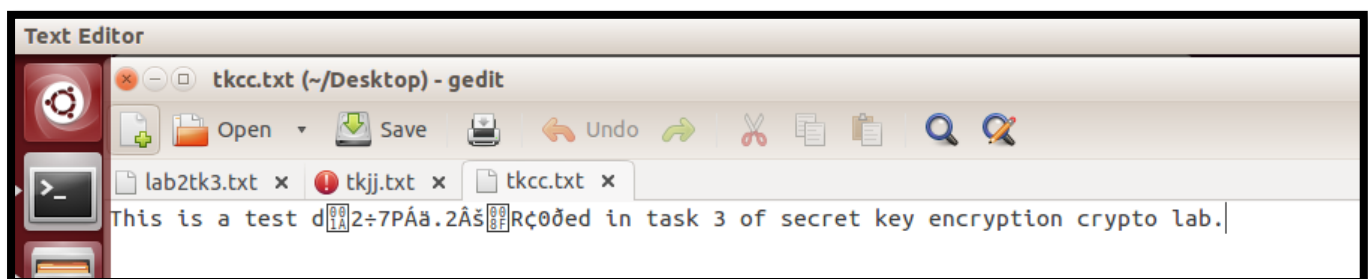


Decryption:

The screen shots below shows the decryption commands and the results from the decrypted file for ECB mode:

```
alizerh@ubuntu: ~/Desktop
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -e -in plaintext.txt -out plain.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -e -in plain.txt -out decResult.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
alizerh@ubuntu:~/Desktop$
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -d -in plain.txt -out decResult.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
alizerh@ubuntu:~/Desktop$
```

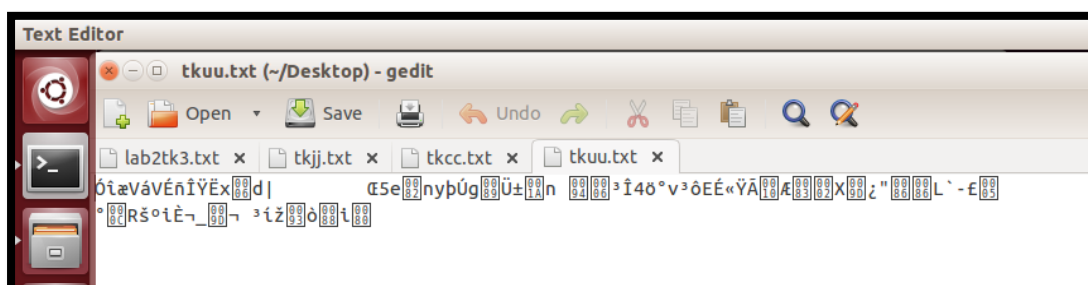
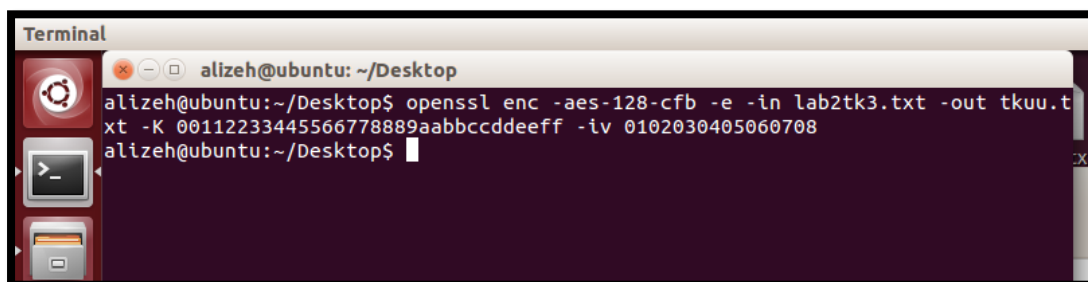
Here, most of the text can be seen as iv is not present.



CFB

Encryption:

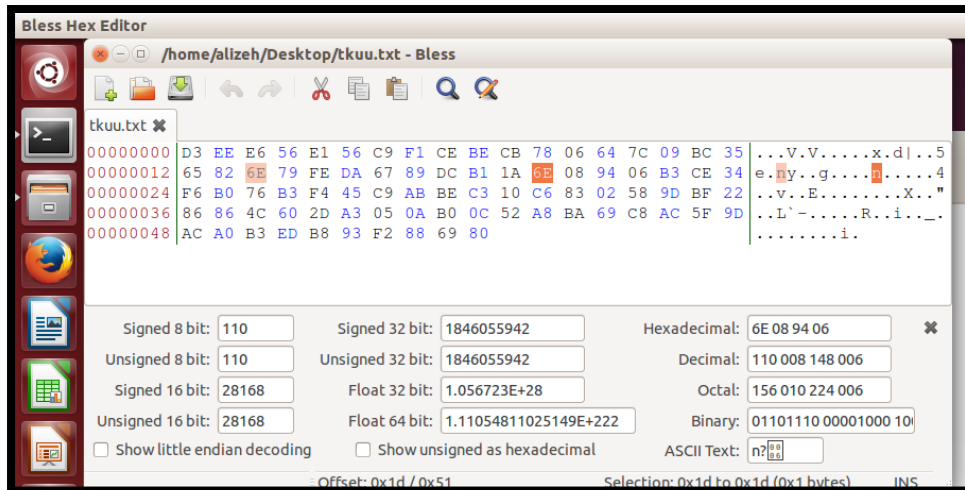
The screen shots below shows the encryption commands and the results from the encrypted file for CFB mode:



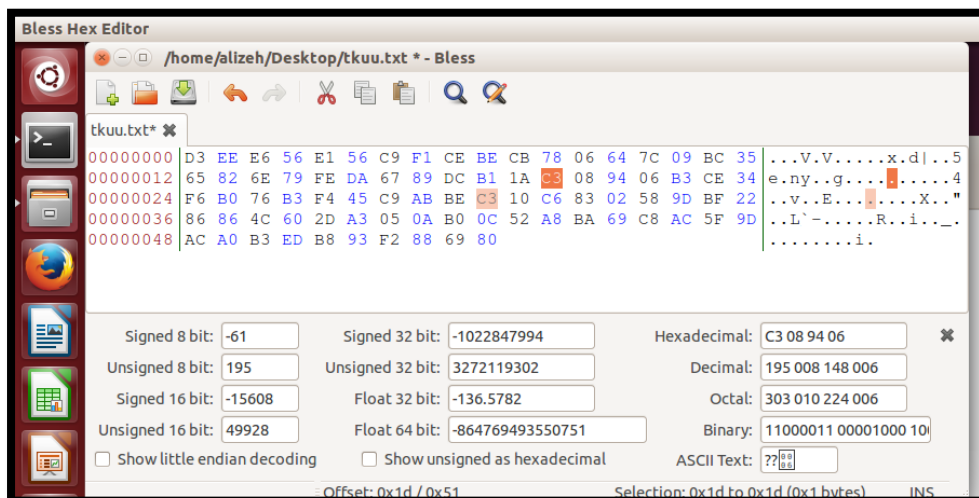
Corruption:

The screen shots below shows corruption of the 30th byte in the encrypted file in CFB.

FROM:



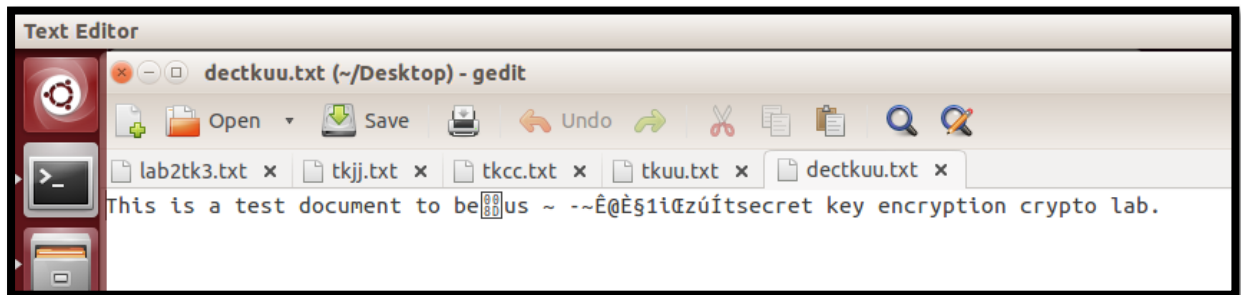
TO:



Decryption:

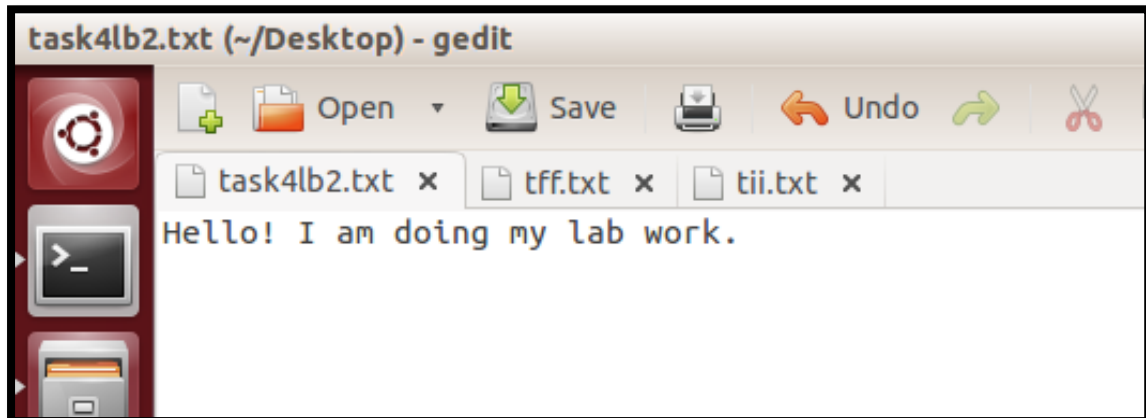
The screen shots below shows the decryption commands and the results from the decrypted file for CFB mode:

```
alizerh@ubuntu:~/Desktop$  
alizerh@ubuntu:~/Desktop$ openssl enc -aes-128-cfb -d -in tkuu.txt -out dectkuu.t  
xt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

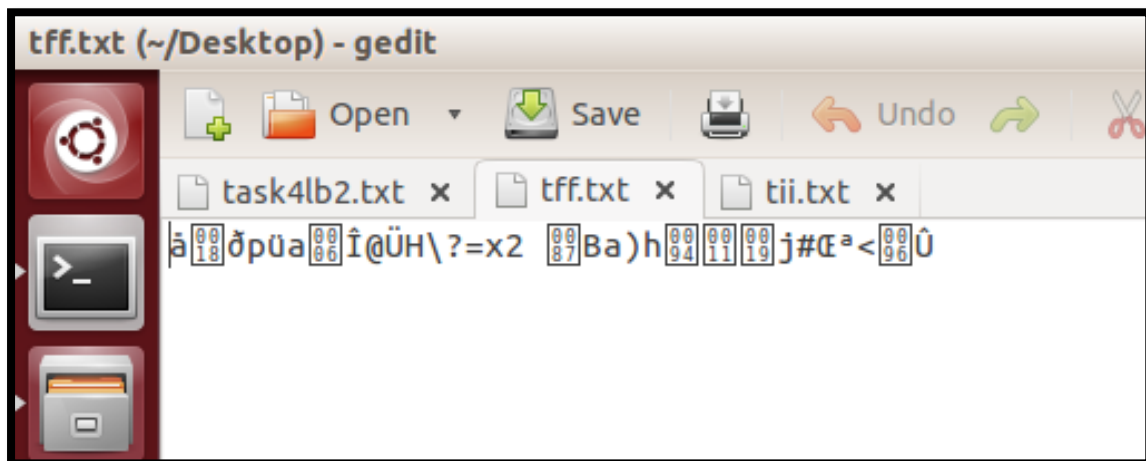


Task 4: Initial Vector (IV)

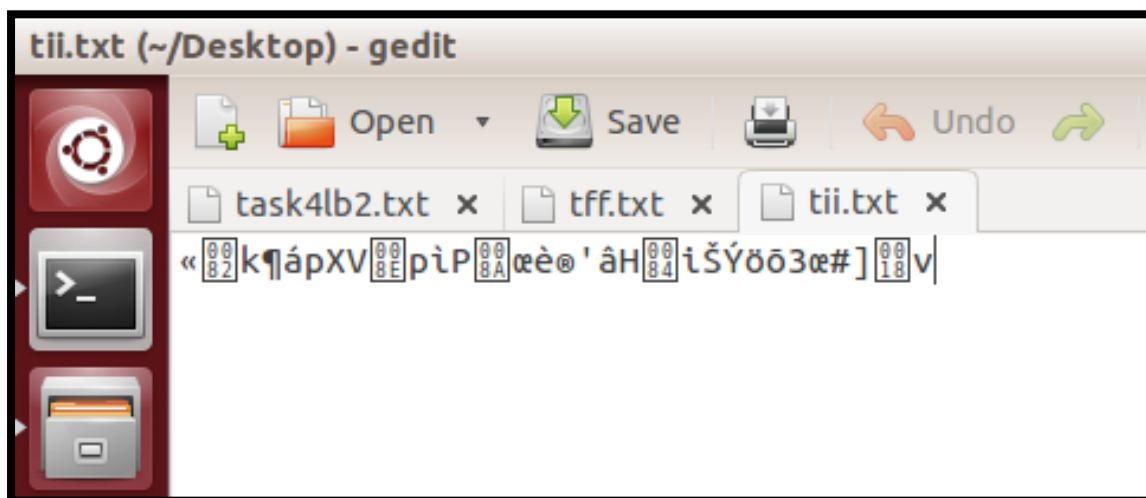
Command for different iv's

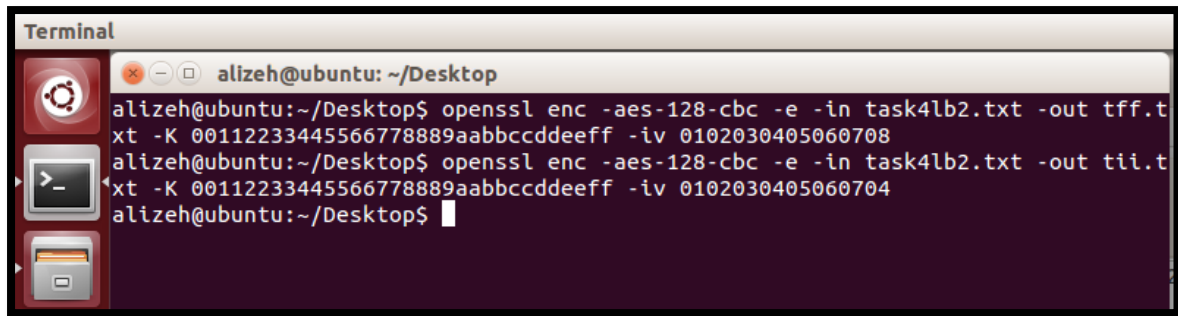


Result with first iv



With second iv



A terminal window titled "Terminal" with a dark purple background. The window shows a user named "alizabeth@ubuntu" in the directory "~/Desktop". The user has entered two commands to encrypt a file named "task4lb2.txt". The first command uses a key "00112233445566778889aabbccddeeff" and an IV "0102030405060708", resulting in an output file "tff.txt". The second command uses the same key but a different IV "0102030405060704", resulting in an output file "tii.txt".

```
alizabeth@ubuntu: ~/Desktop
alizabeth@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -e -in task4lb2.txt -out tff.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
alizabeth@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -e -in task4lb2.txt -out tii.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060704
alizabeth@ubuntu:~/Desktop$
```

Explanation:

It is essential to choose the **iv's** very carefully because if otherwise, the encrypted data would not be secured. Also, it is essential that the iv chosen is a unique one and should never be used again for the same key. However, using the iv's that are predictable will reduce security of the message (let's say m1) by a factor of N (number of **ciphertext** created). Therefore, the adversary can recover one of the **ciphertext**, but which one, that is not in control of the attacker. Whereas, in this case the observation was that when I encrypted a text with one iv the encrypted format changed using the other iv. This shows us that if an iv is unique, the data is protected from the **adversaries**. Moreover, to prove that having a unique iv is important we can see that after using the XOR function, even though the key is eliminated, the iv would still be present. This proves that unique iv can protect the message (m1) from the unauthorized access of the adversaries.