Lab 2 - Documentation

February 22, 2020

1 Lab 2 - Secret Key Encryption Lab

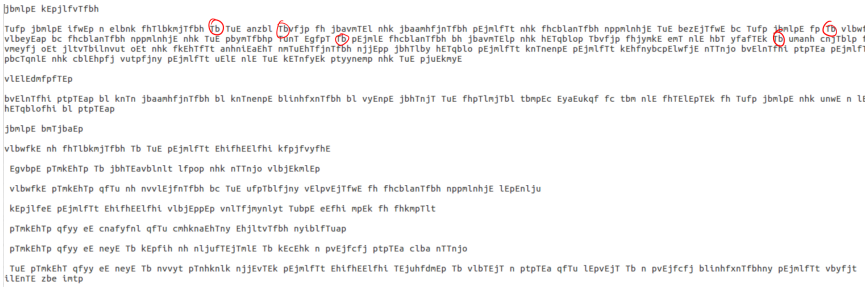
1.1 ## By: Laura Pereda

1.1.1 Task 1 - Freq. Analysis against Monoalphabetc Substitution Cipher

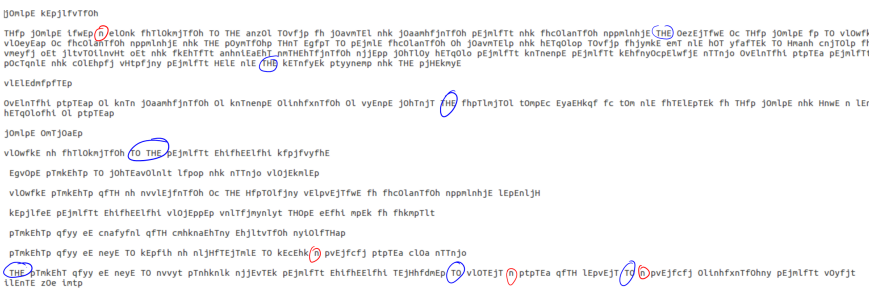
By running a simple frequency analysis, the characters 'R' and 'S' were the top 2. Thus, it would make sense to map the character 'E' = 'R' and 'T' = 'S'.

R	S	F	P	N	B	H	L	J	M	K	V	T	Y	U	A	I	C	E	Q	O	W	Z	D	G	X
153	137	123	111	104	101	101	101	83	57	52	44	37	37	33	30	25	20	20	12	10	8	3	2	2	2
10.9	9.7	8.7	7.9	7.4	7.2	7.2	7.2	5.9	4.0	3.7	3.1	2.6	2.6	2.3	2.1	1.8	1.4	1.4	0.9	0.7	0.6	0.2	0.1	0.1	0.1

From the substitution, I was able to notice that there were a lot of 'Tb', which could indicate that 'B' = 'O'. The word would then form to be 'TO':

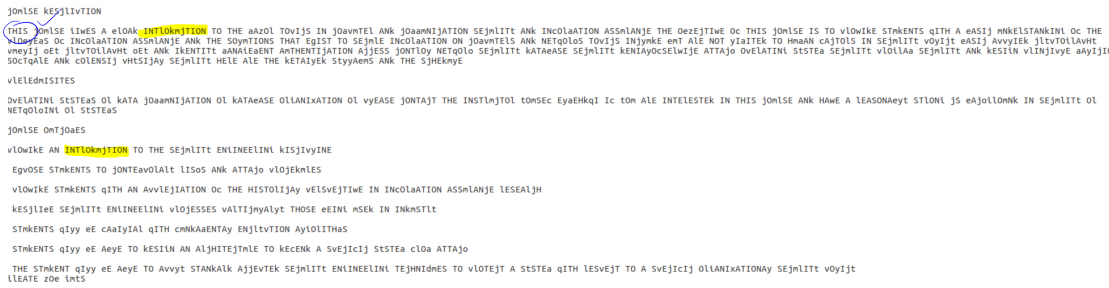


A very common word within the English language is 'THE', which is similar to 'TUE'. This would indicate that 'U' = 'H'. Substituting these two letters results in the following:

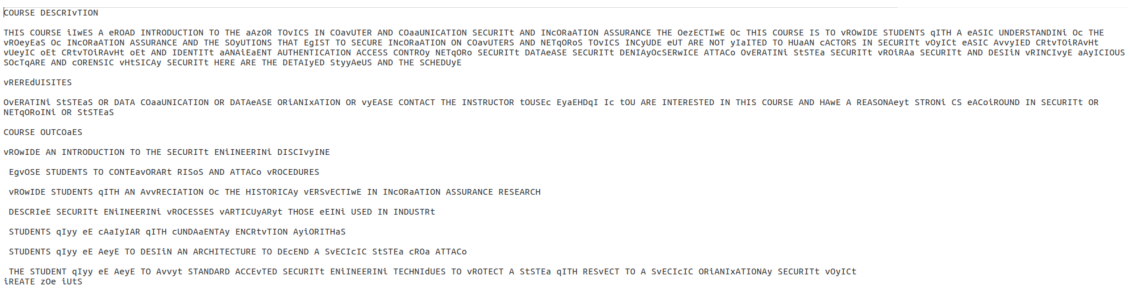


With this substitution, I noticed that the letter 'N' was always alone which could be a replacement

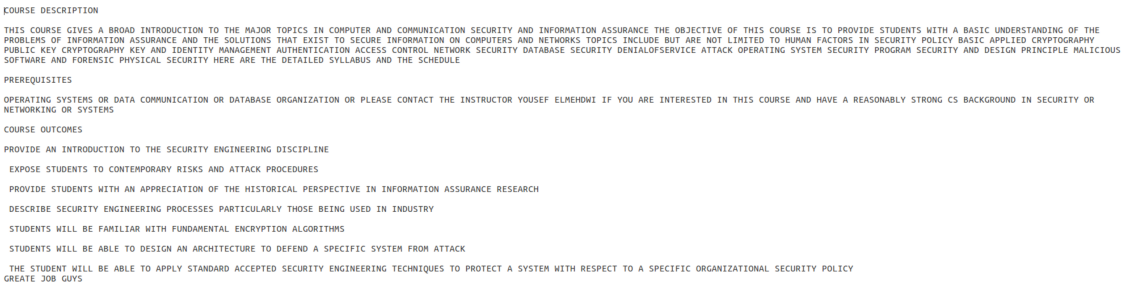
for ‘A’. Replacing that character led to a wild guess of replacing ‘FP’ with ‘IS’. This discovery lead to observing ‘IH’, so another possible replacement ‘H’ for ‘N’.



I managed to complete one four-letter word, but I was more interested in the highlighted part of the picture. It seems to form the word ‘INTRODUCTION’, so ‘L’ = ‘R, ’K’ = ‘D’, ‘M’ = ‘U’, and ‘J’ = ‘C’:



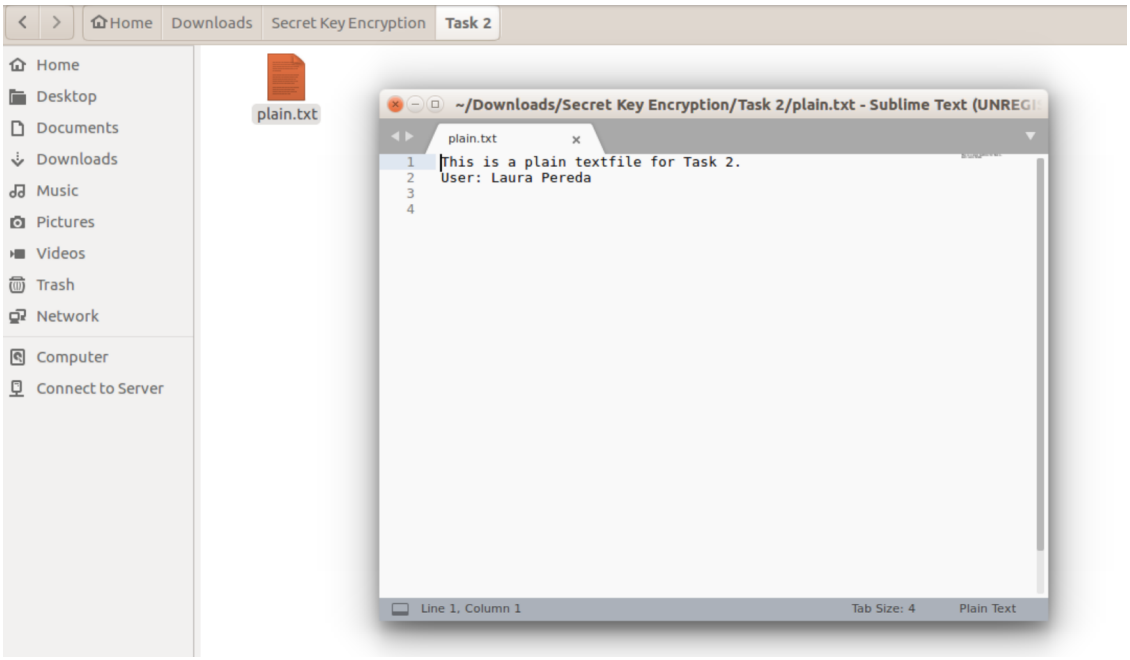
From there it was just finishing the last couple of words to finish the decryption. Final outcome:



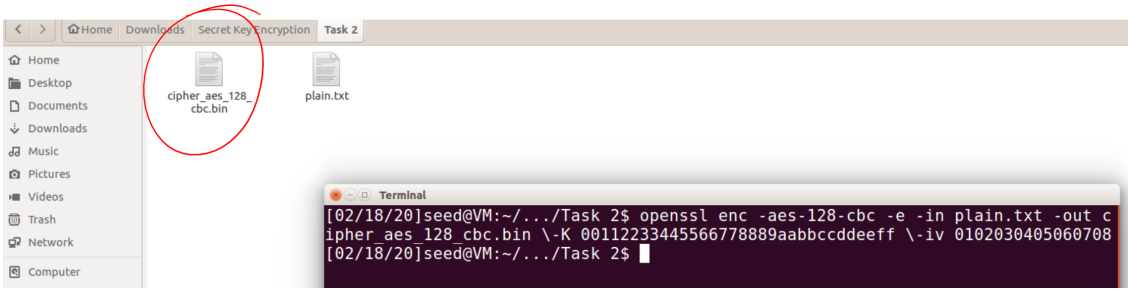
1.1.2 Task 2 - Encryption using Different Ciphers and Modes

This task indicates to try out different ciphertypes with different modes. The algorithm seems to follow a pattern of **AlgName-keySize-encryptionMode**. The familiar encryption mode for block cipher are Electronic Codebook (ECB) and Cipher Block Chaining (CBC). Because of this, I will be using the ciphertype examples to encrypt and decrypt, while increasing size if possible.

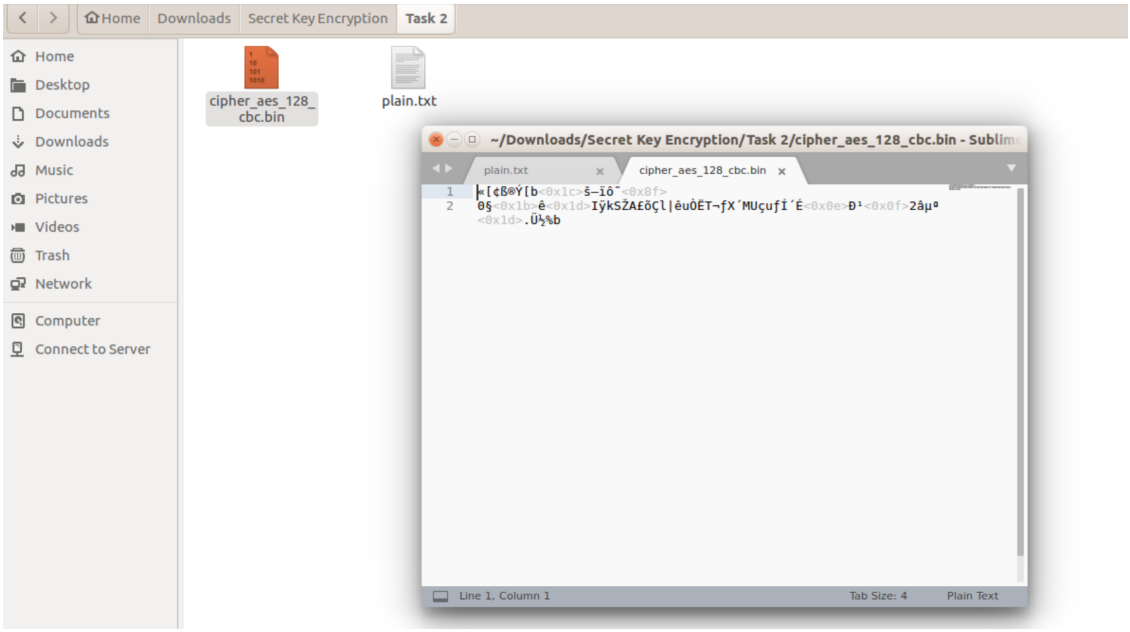
To implement the encryption methods, I created a plain textfile with the following content:



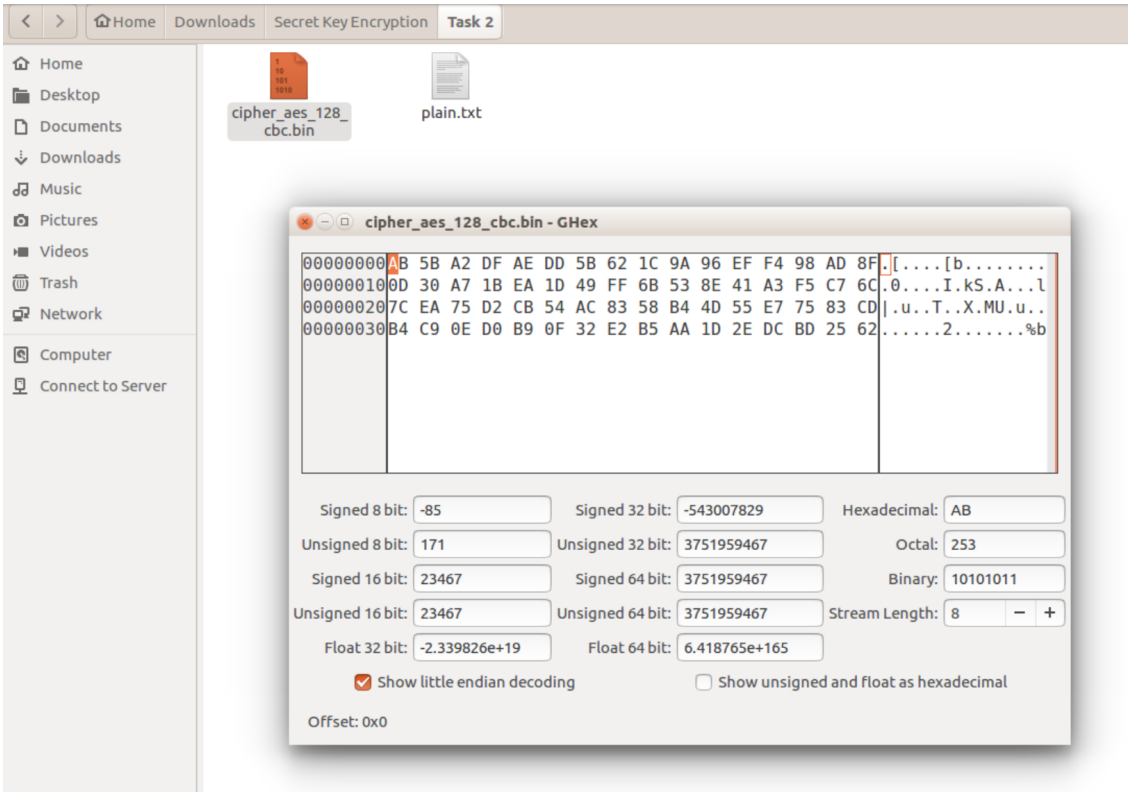
Using the cipher and mode: -aes-128-cbc, I encrypt the plain text file.



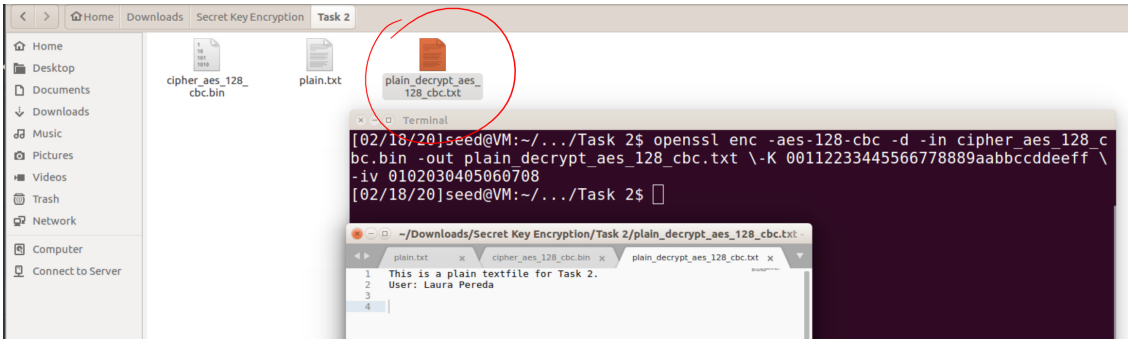
As shown below, using the cipher -aes-128-cbc resulted in an encrypted file that is not easily visible to understand when opening in a normal text editor.



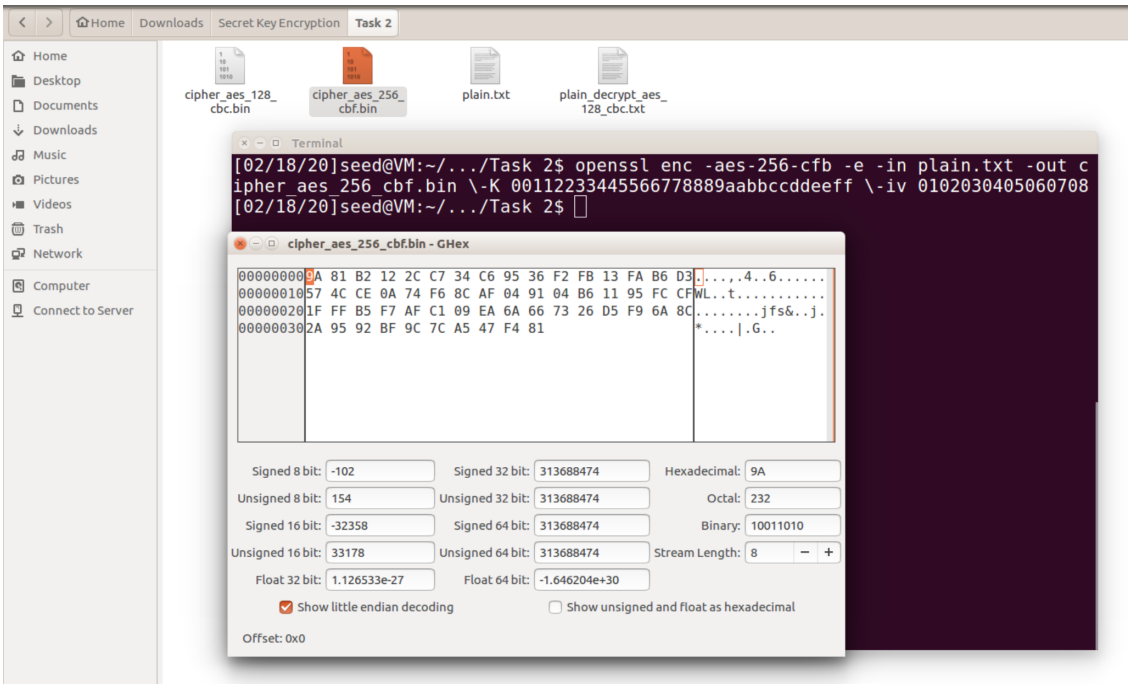
However, after doing some research, I realized GHEX was installed in the SeedUbuntu, which will let you view and modify files of binary format. This is just a simple demonstration for the first encryption.



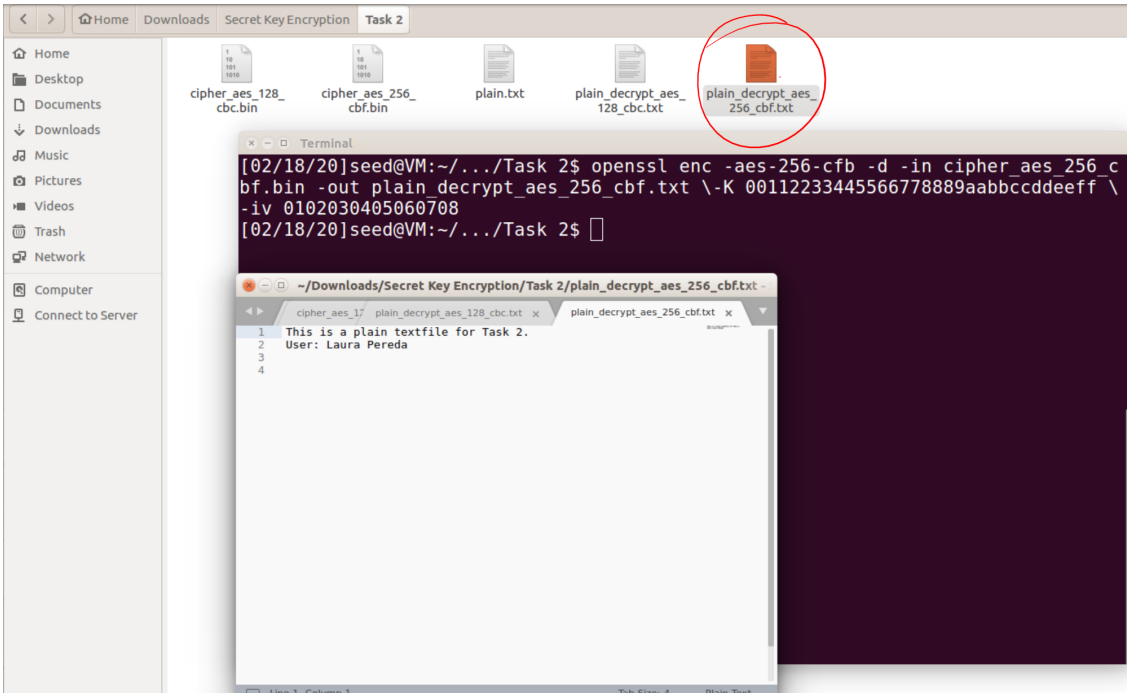
Here is the related decryption:



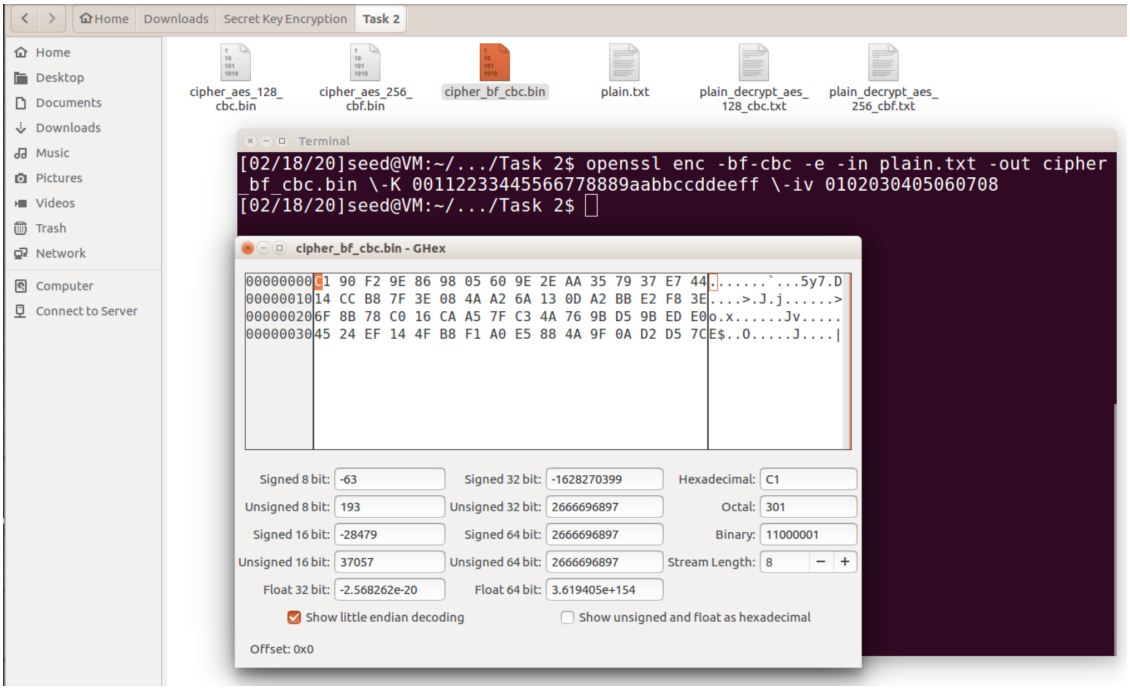
Using the cipher and mode: -aes-256-cfb (cbc with size 256 failed at decryption everytime for some reason) while using the same plain text file.



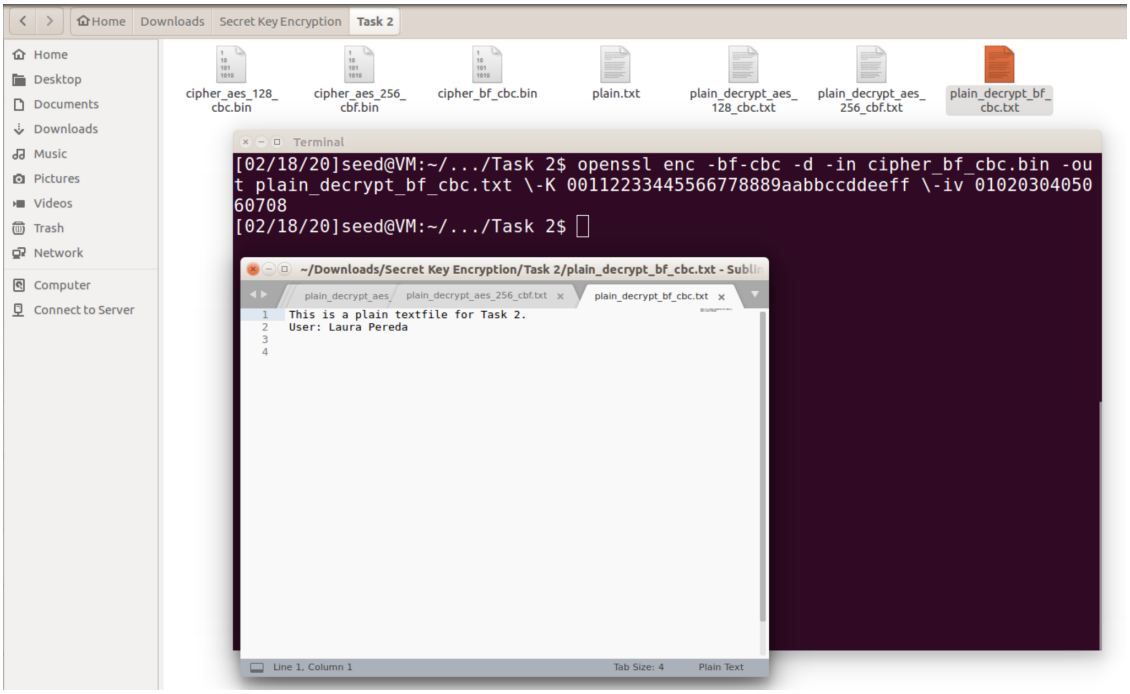
The decryption:



Using the cipher and mode: -bf-cbc while using same plain text file



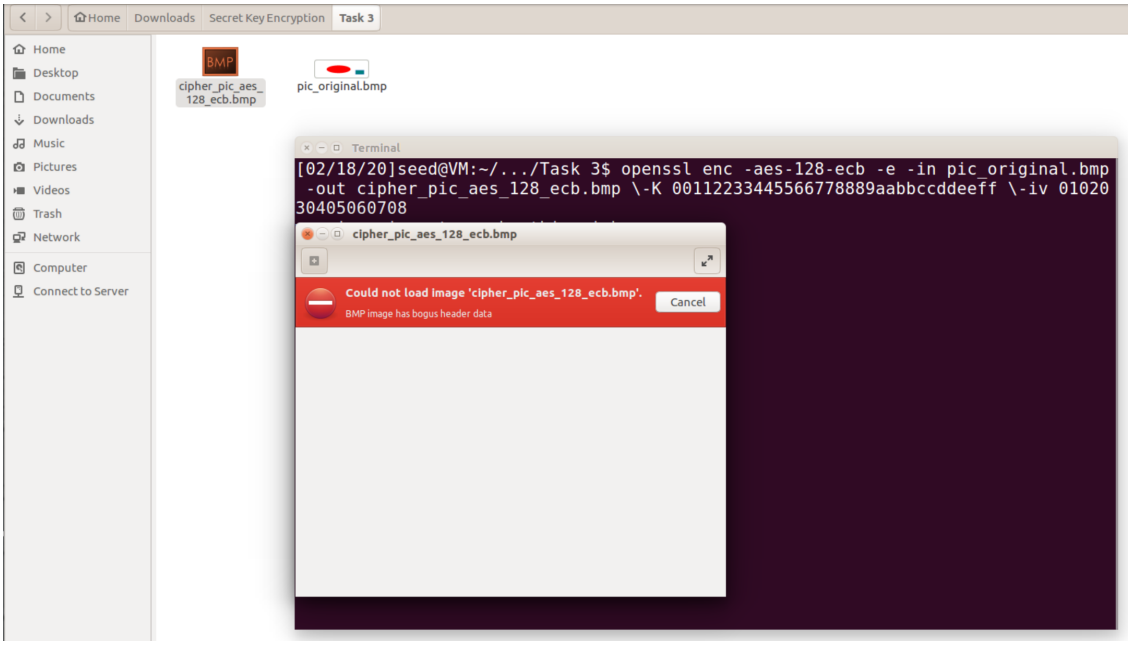
The decryption:



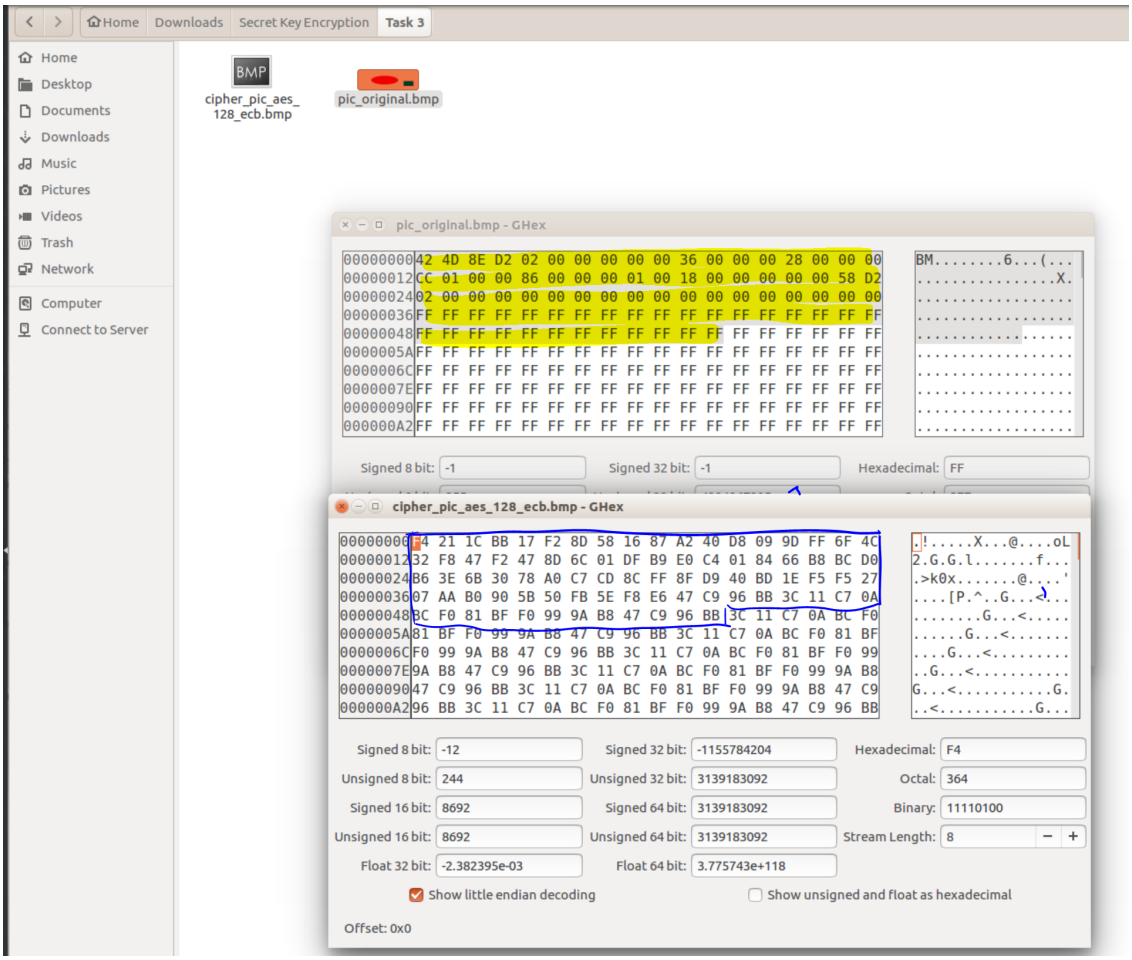
1.1.3 Task 3 - Encryption Mode: ECB vs CBC

Since the type of ECB or CBC mode is not defined, I decided to use the same ciphertype and mode above (but using ECB and CBC).

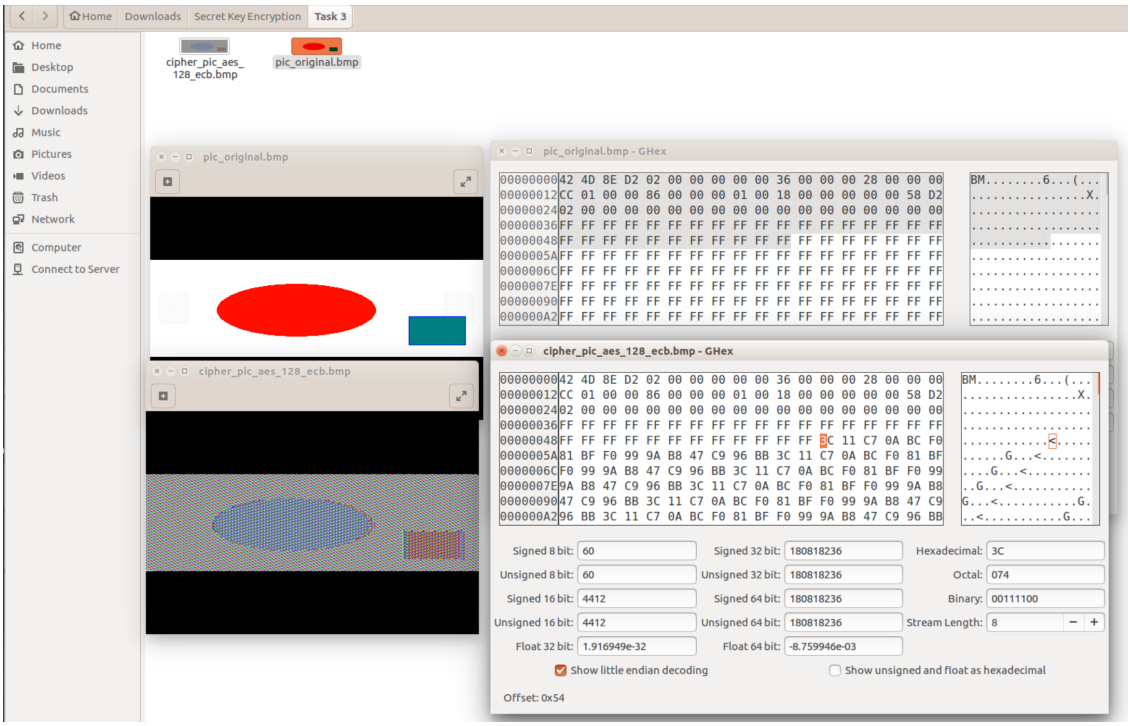
Using the image provided in Blackboard, I encrypted the image by using **-aes-128-cbc**. When I attempted to view the image through the ImageViewer, an error occurred stating that there was bogus header data.



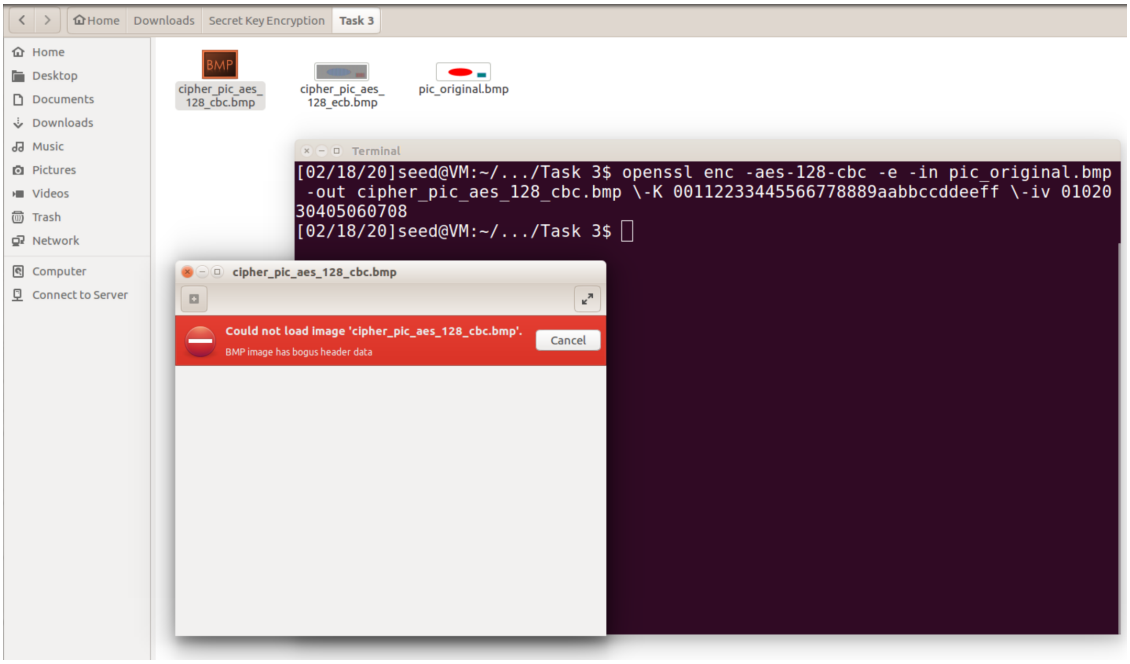
The task 3 description indicates that the first 54 bytes contain the header information about the picture, but we have to set it correctly in order for it to be treated as a legitimate **.bmp** file.



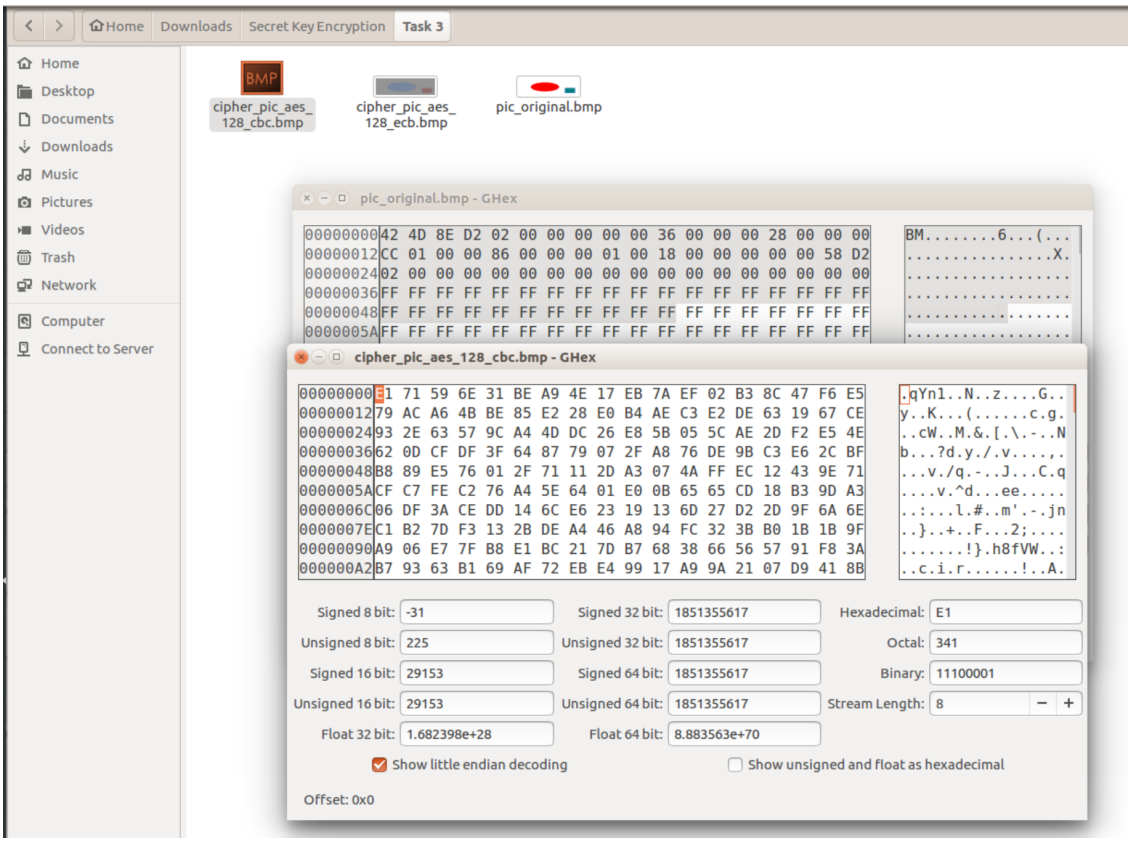
By replacing the first 54 bytes of cipher with the original, I was able to view the image.



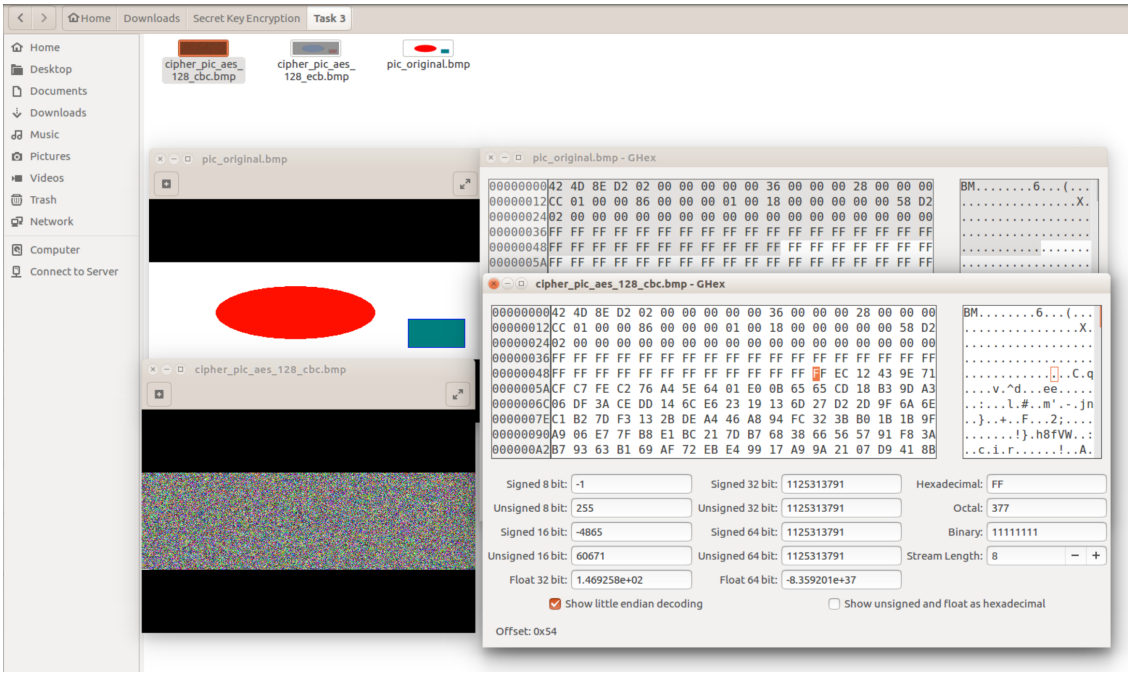
The same is done with the CBC encryption.



Identifying first 54 bytes to change:



Modifying to view image:



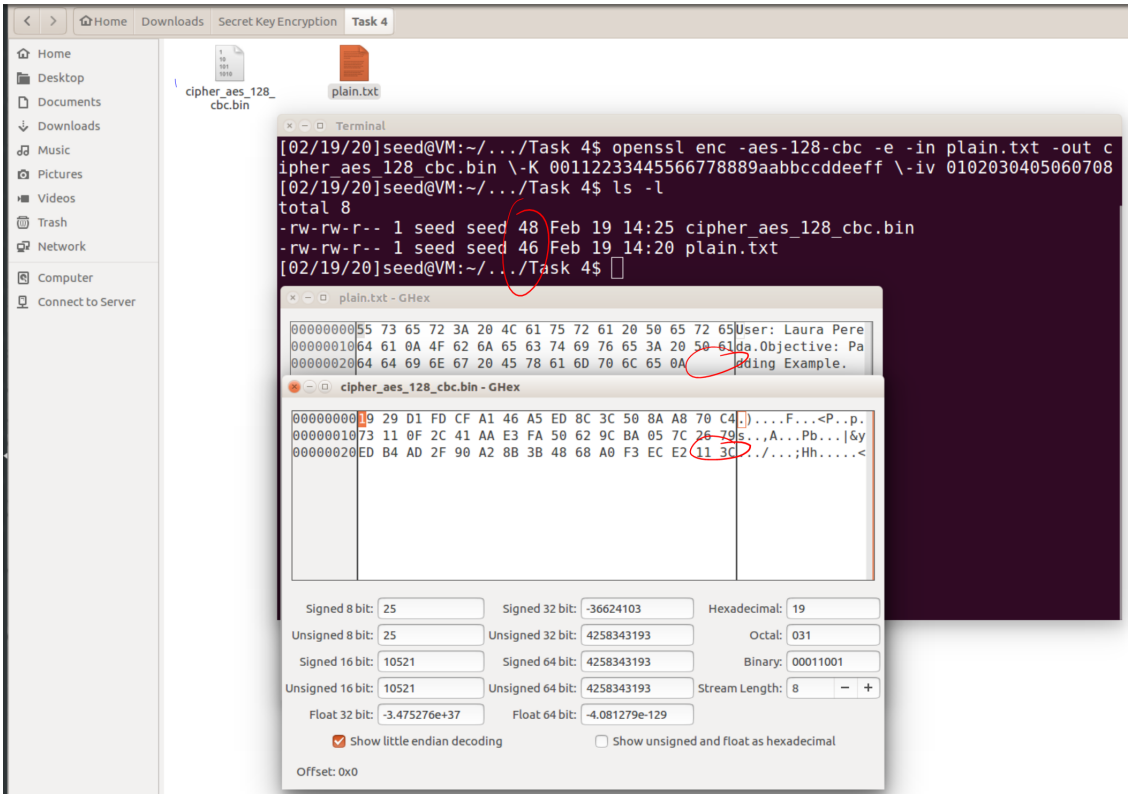
As shown above, the main disadvantage of ECB is when we are encrypting identical plain text

blocks into identical cipher text blocks. The patterns are still to visible, which then leads to lack of confidentiality. CBC, on the otherhand, has a more secure system since it utilizes the XOR operation before encrypting each cipher text block.

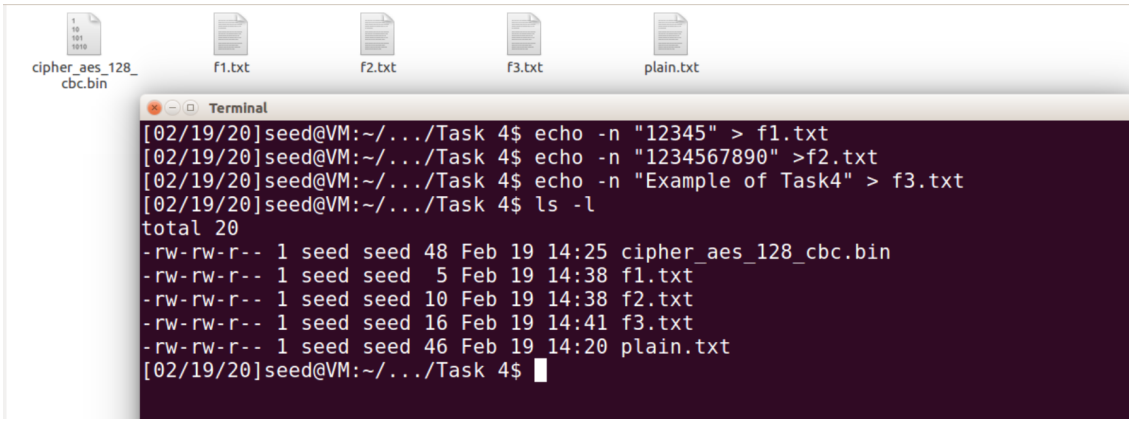
1.1.4 Task 4 - Padding

In this task, we are told to choose between using ECB, CBC, CFB, and OFB and indicating if any of these require padding, and if not, why they do not require padding. From the reference to "Block cipher mode of operation" in the pdf, cipher modes such as ECB and CBC require that the initial input be an exact multiple of the block size. IF the plain text that will be encrypted is not an exact multiple, then padding is required before encrypting (usually an additional string). When decrypting, the receiver will need to know how to remove the correct amount of padding in an umbiguous manner.

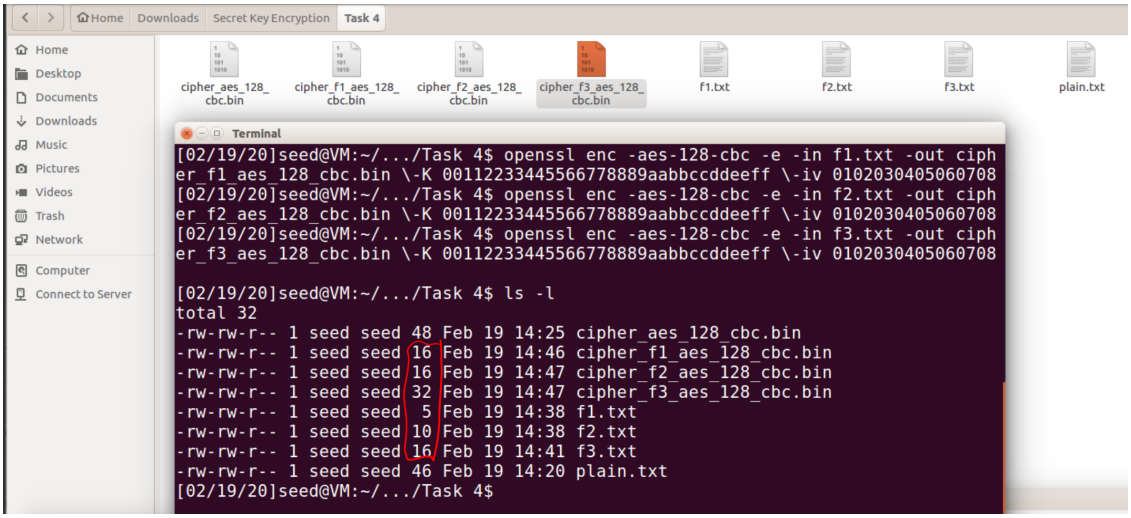
As an example of padding, I created a simple plain text file the size of 46. But after encrypting through -aes-128-cbc, the file grew to be the size of 48 bytes which is a multiple of 8 (standard size for block). The size of one block for 128/8 is 16 bytes. The 2 additional bytes were added to fulfill that requirement.



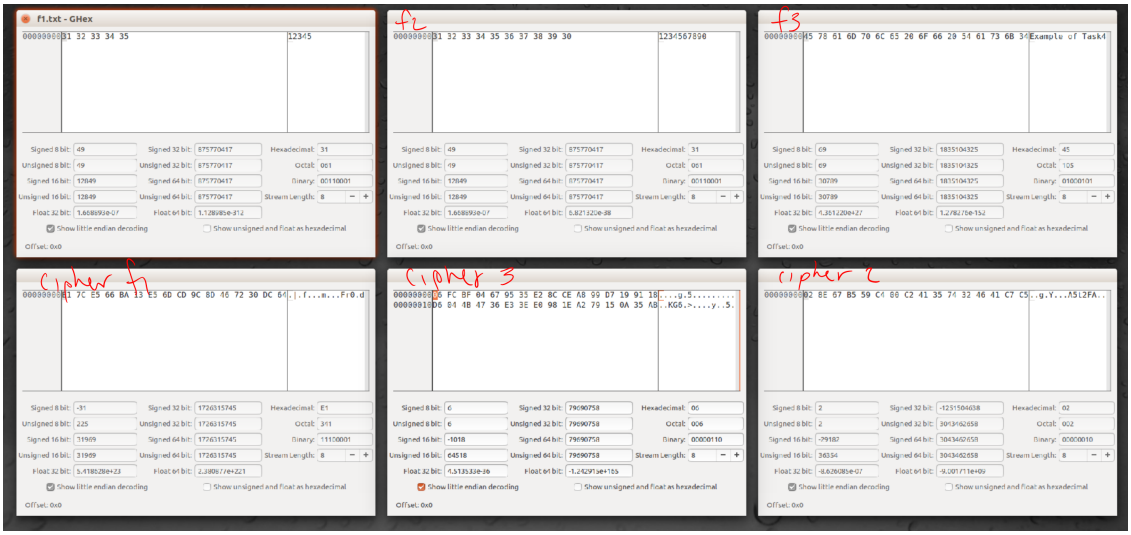
The next part of the task is to create 3 files the length of 5, 10 and 16. Which is shown below:



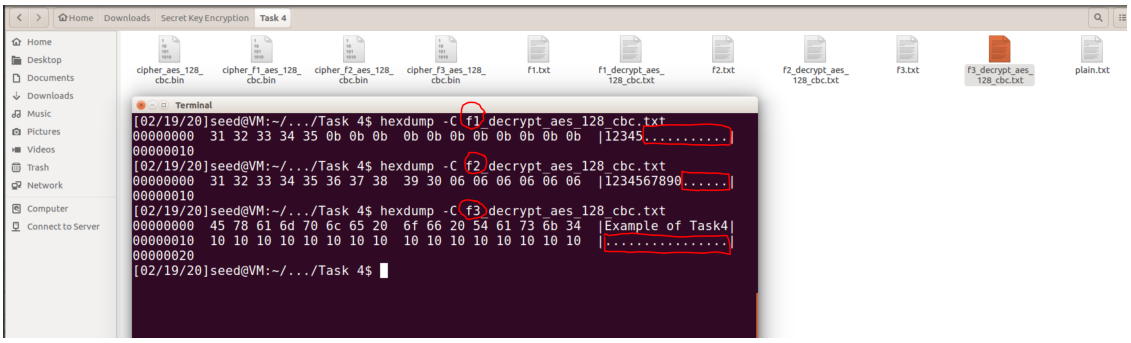
After encrypting them using -aes-128-cbc, we can see the additional padding incorporated into the files through GHEX. It's interesting to note that the original f3 files of 16 bytes was encrypted with an outcome of 32 bytes instead of 16 bytes like the other two.



Opening the original files and the encrypted files with GHEX, we can view the difference:



After decrypting the encrypted files, to view the actual content of the padding we can use a hex tool. Below is the following content of the padding for f1, f2 and f3 decrypted files:



1.1.5 Task 5 - Error Propagation – Corrupted Cipher Text

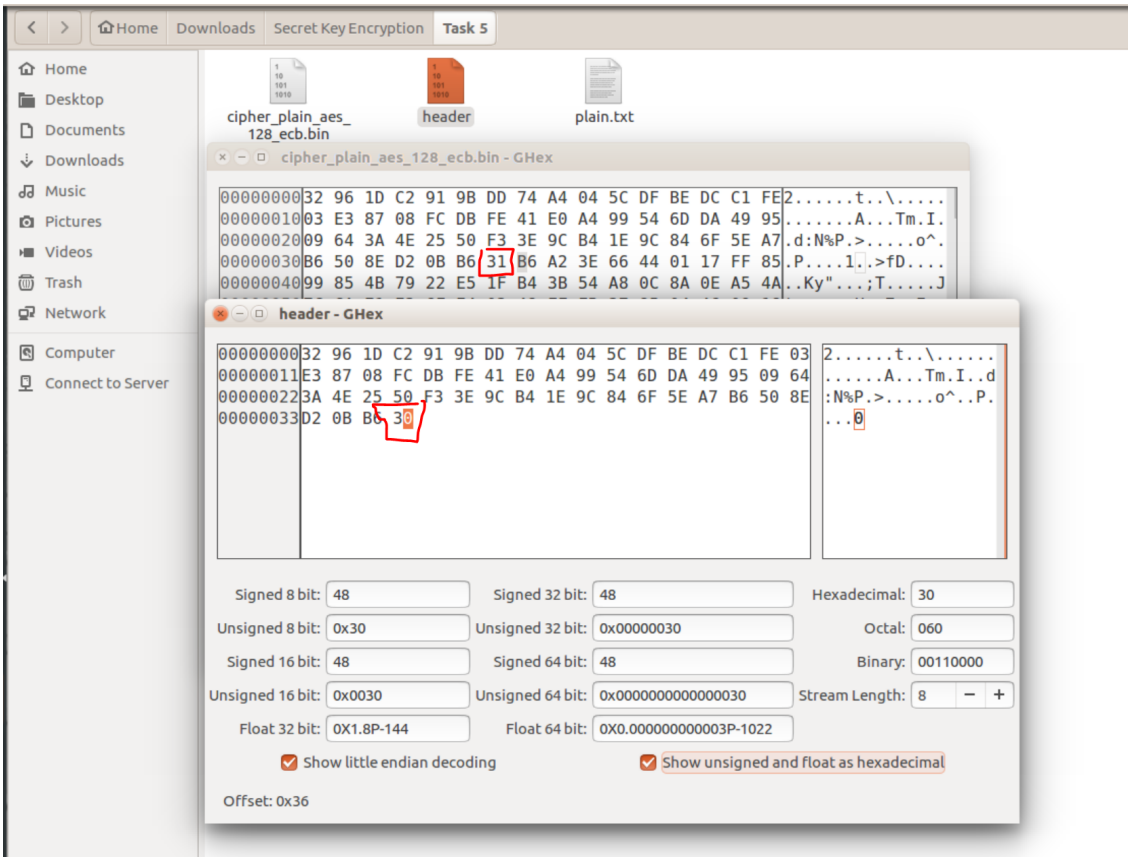
In this task, we want to answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?

Since I must answer this question before finishing the task, I will base my answer on the resul had compared ECB vs CBC and discovered that ECB fails in comparison to CBC when it comes to en does not imply that CBC will also have the same results in recovering a cipher. Most likely the dependent on other blocks of encrypted blocks would most likely be the best at recovering info random guess, I choose OFB.

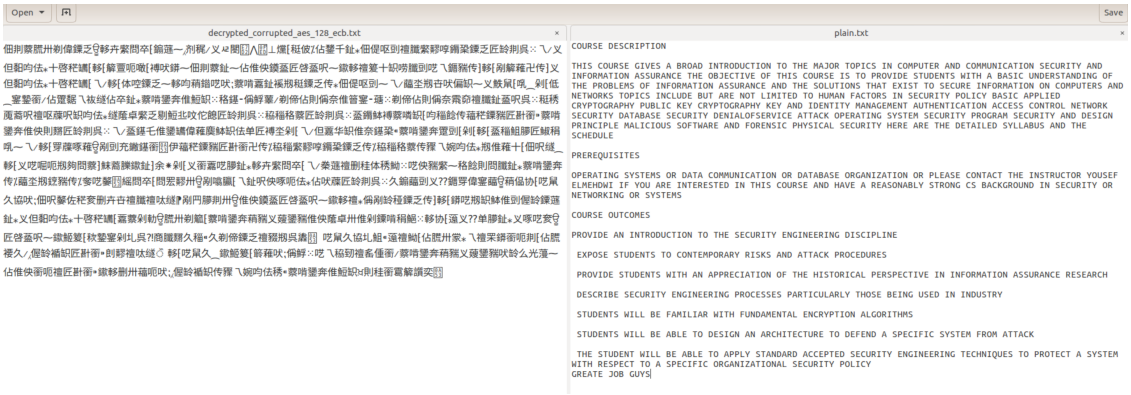
The task indicates to create a file at least 1000 bytes long, so I'll be using the original plain text from Task 1 since it is 1656 bytes long.

ECB Corrupted File

1. After encrypting the plain text, I modified the 55th byte of the encrypted text as shown below. I only changed the second bit from 0 to 1.

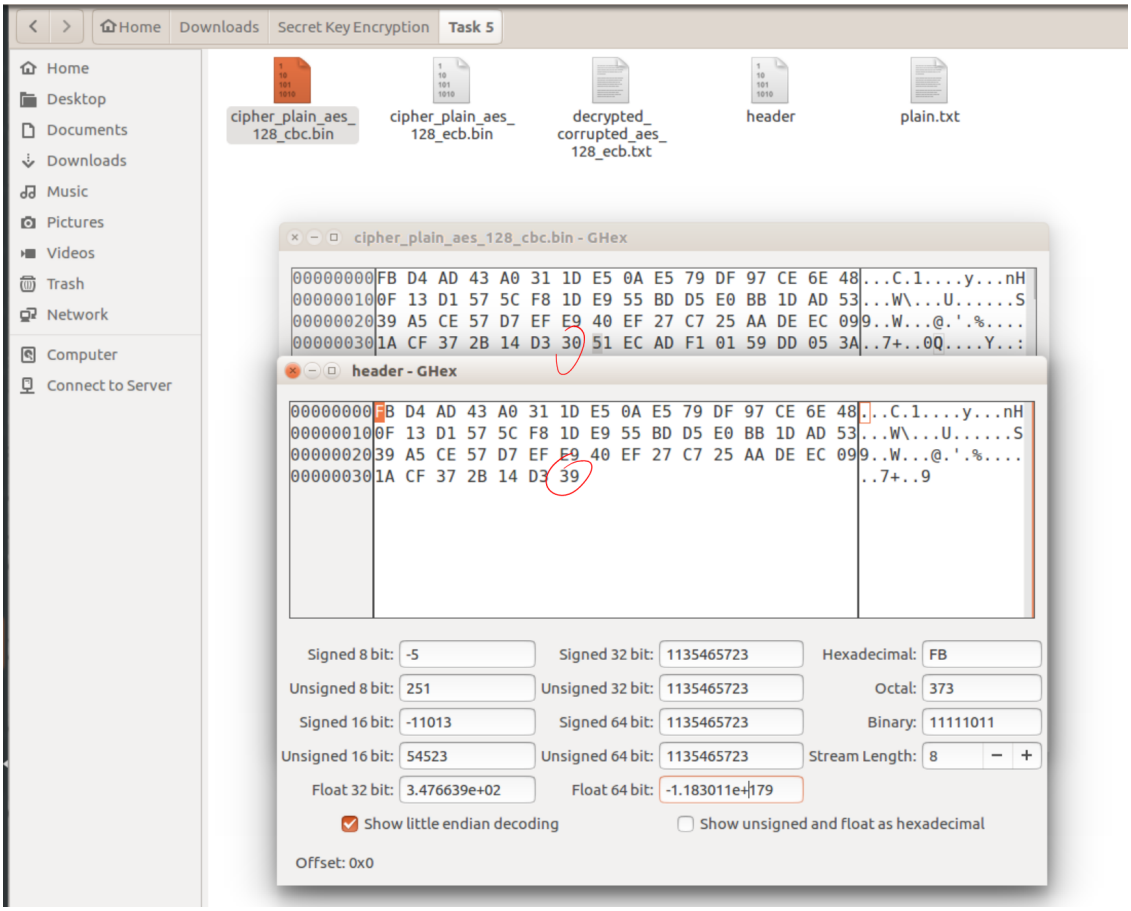


2. After corrupting, I decrypted the text and opened it within a text editor to compare the original plaintext to the corrupted decryption. As shown below, the one change resulted in being unable to view the original content at all.

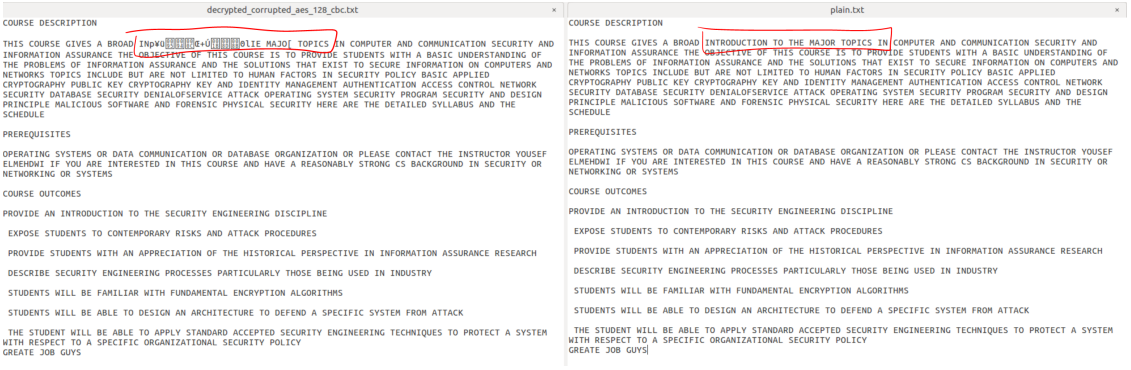


CBC Corrupted File

1. Repeat by corrupting 55th byte of encrypted text. Changed second bit from 9 to 0.

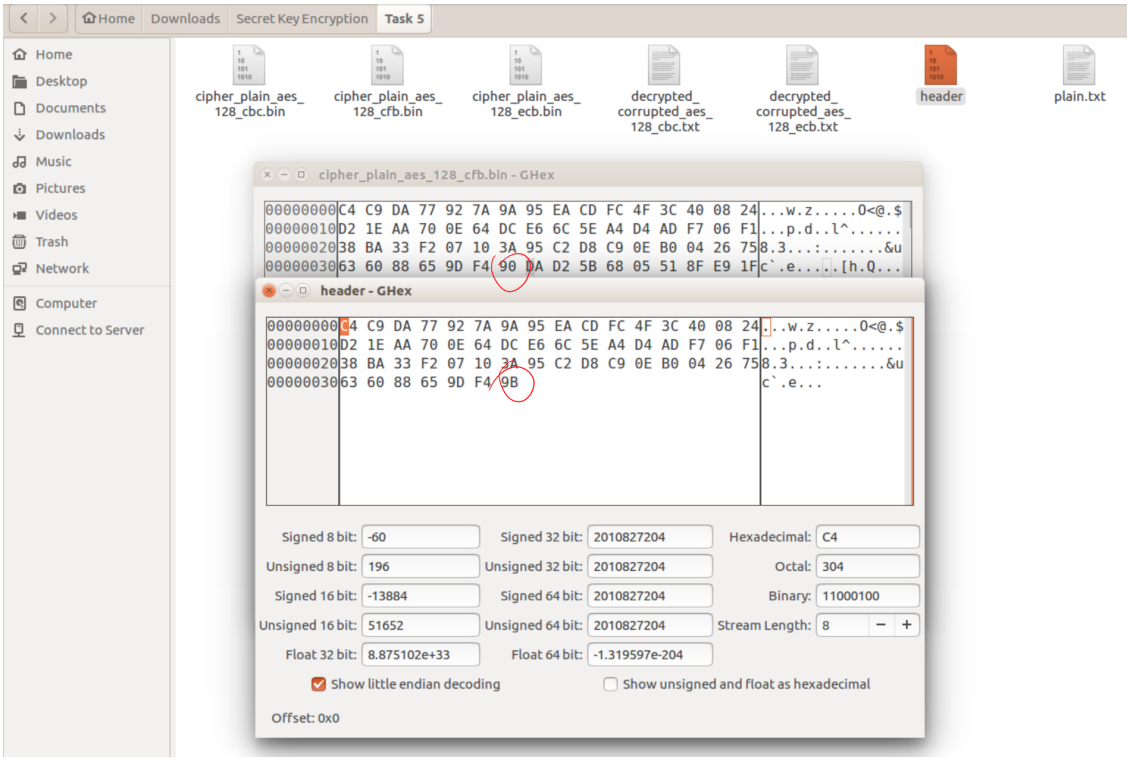


2. After corrupting, these are the results of decrypting corrupted file. Majority of the text file is still understandable with only that small portion being affected.

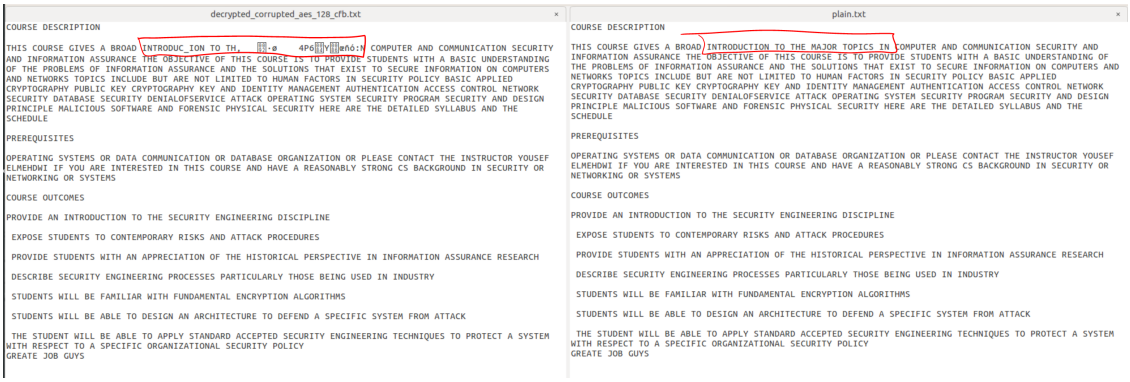


CFB Corrupted File

1. Repeat by corrupting 55th byte of encrypted text. Changed second bit from B to 0.

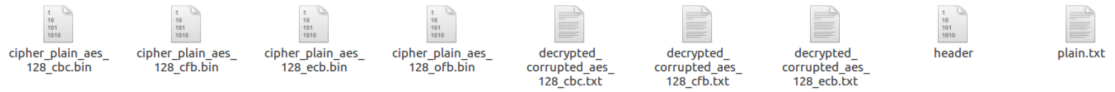


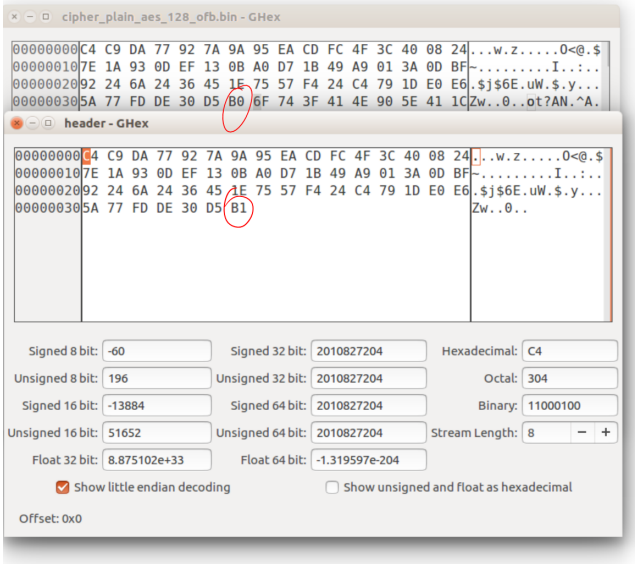
2. After corrupting, these are the results of decrypting corrupted file. Again, majority of text file is understandable, however, the area in which the corruption occurs is extended just a little more compared to CBC.



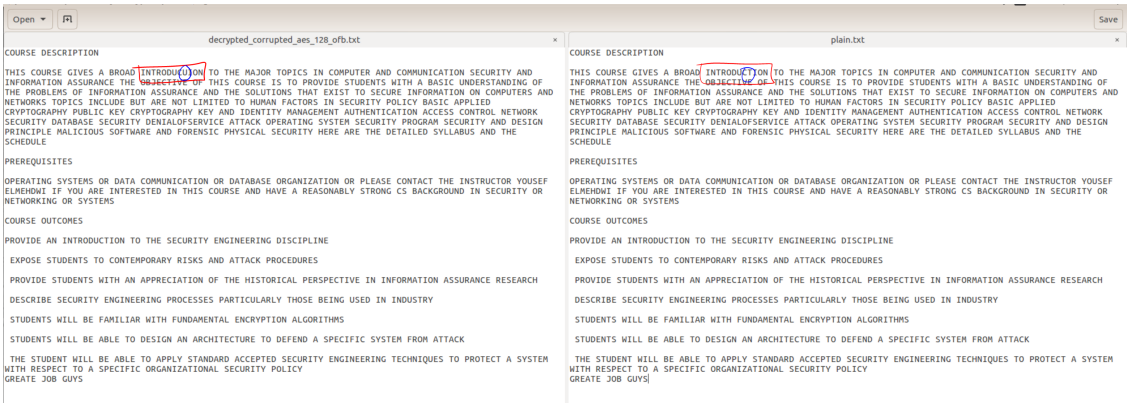
OFB Corrupted File

1. Repeat by corrupting 55th byte of encrypted text. Changed second bit from 1 to 0.





2. After corrupting, these are the results of decrypting corrupted file. With this one, I am pretty amazed with the results. Only one letter within the word where the corruption occured was affected.



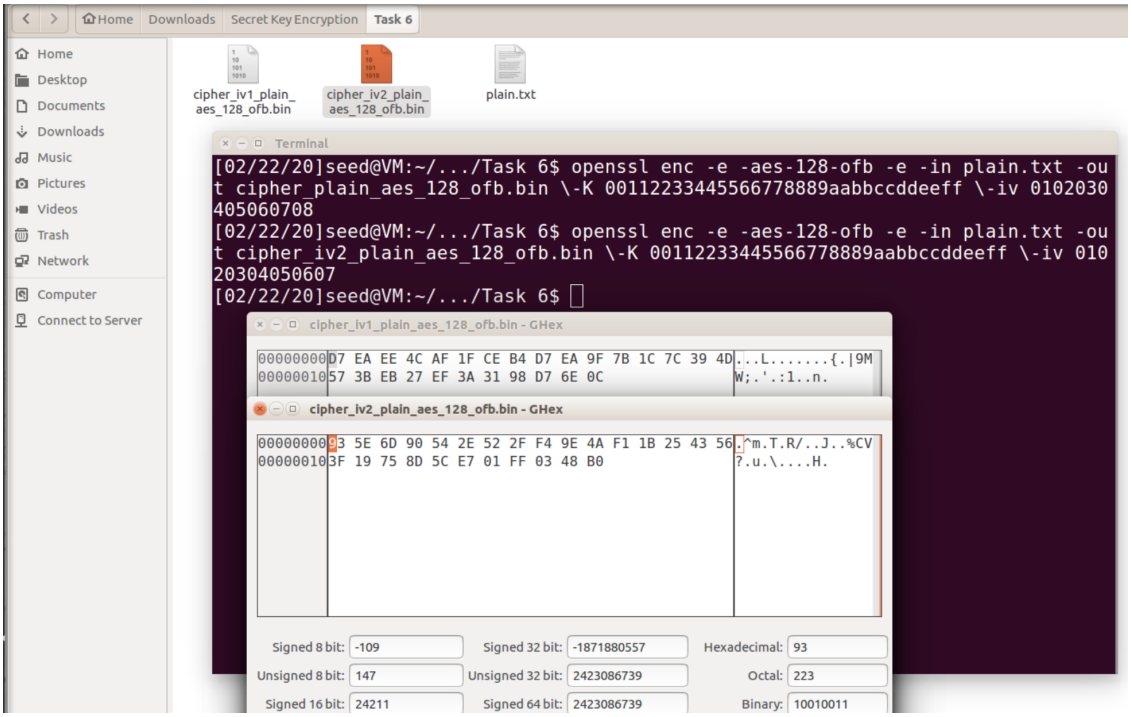
From these results, it is clear that OFB is able to recover or safe the most from a corruption.

1.1.6 Task 6 - Initial Vector (IV)

In this task, we are analyzing the usage of IV when encrypting. Specifically, the uniqueness.

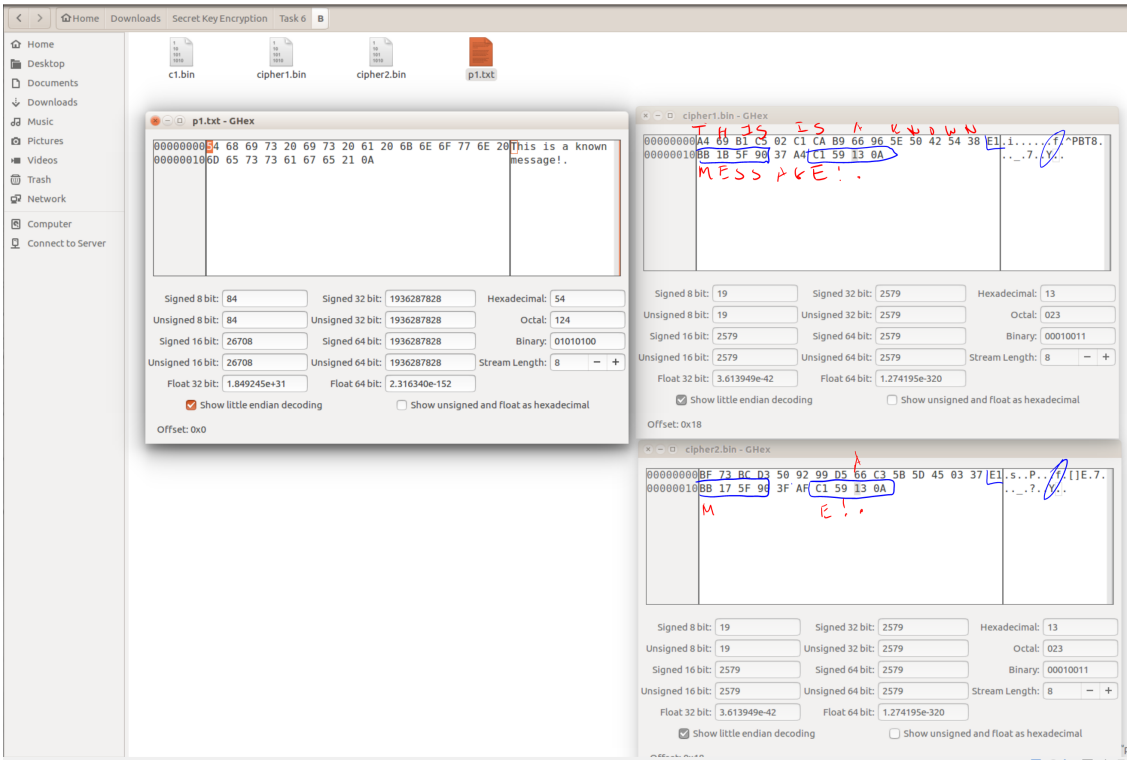
A. First we compare the encryption of the same text file using two different IVs. For the first encryption, I used IV = 0102030405060708, but for the second encryption the IV = 01020304050607. I dropped the last two numbers. Both encryptions use the same plain text and key, just two different

keys. As shown below, the usage of unique IVs on the same plain text file produced two different encryptions.



The next picture shows that if you utilize the same IV, the encryption still remained the same for both. This is important because it indicates that the encryption is not secured. If we wish to encrypt the same text twice or more times, we do not want to make it obvious to the attackers.

B. Some argue that using a different plaintext with the same IV is safe. Thus, we are attempting to figure out plaintext 2 from the given information: plaintext 1 and ciphertext 1, and unknown plaintext 2 and cipher text 2. From the picture below, I tried examining the two cipher texts and realized there were a few similarities between them. For example, the ending trail of both ciphers are similar (outlined in blue). I tried mapping each character from known p1 to c1, but it doesn't seem to apply to c2. Either way, I'm sure a professional attack would be able to utilize this pair of (p1, c1) to figure out p2 from c2. OFB is suppose to a more safer form of encryption compared to CFB, which means that if I was able to see few similarities from using the same IV, then CFB would probably reveal even more.


Home
Downloads
Secret Key Encryption
Task 6
B
c1.bin
cipher1.bin
cipher2.bin
p1.txt
p1.txt - GHex
00000000 54 68 69 73 20 69 73 20 61 20 6B 6E 6F 77 6E 20 This is a known
00000010 6D 65 73 73 61 67 65 21 0A message!.
cipher1.bin - GHex
00000000 A4 69 B1 C5 02 C1 CA B9 66 96 5E 50 42 54 38 E1
00000010 BB 1B 5F 90 37 A4 C1 59 13 0A
cipher2.bin - GHex
00000000 BF 73 BC D3 50 92 99 D5 66 C3 5B 5D 45 03 37 E1
00000010 BB 17 5F 90 3F AF C1 59 13 0A

[ ]: