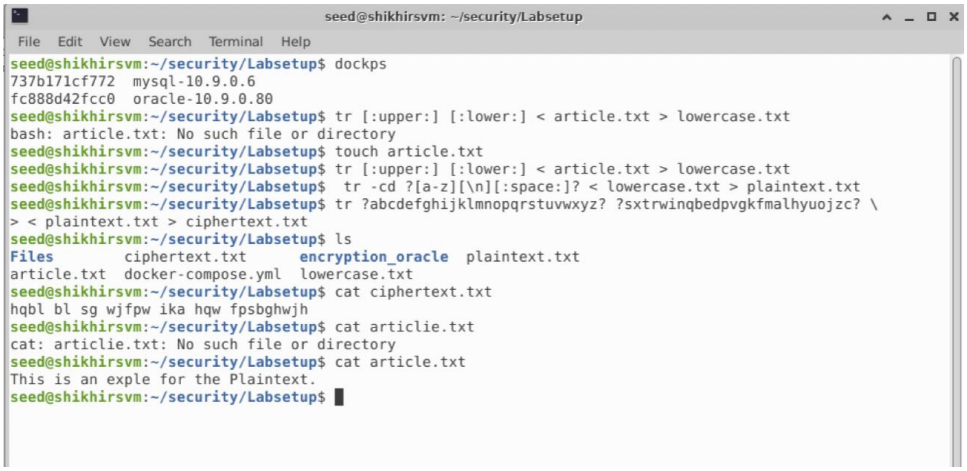
CS 201P Project #7— Secret-Key Encryption Lab

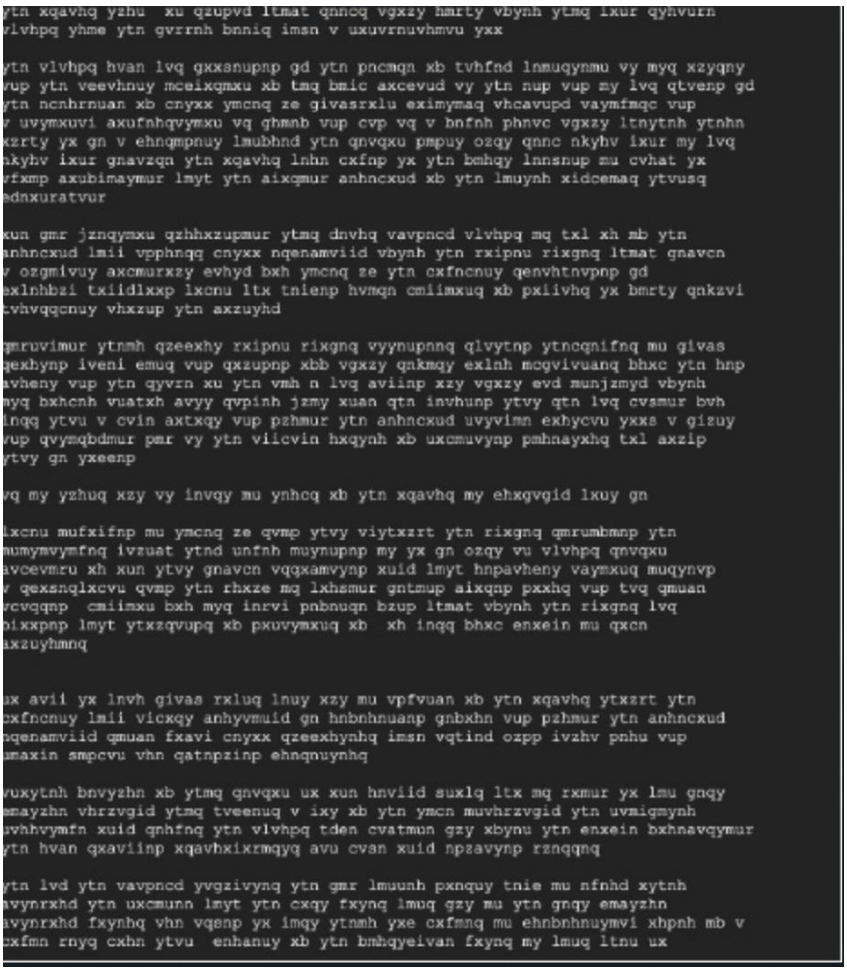
Name: Shikhir Goel

UCI ID: shikhirg@uci.edu

Computing/Cloud Platform Chosen: Google Cloud platform

Task 1: Frequency Analysis





This is out ciphertext provided to us in the labsetup file. Using the frequency analysis method, we will try to decode this ciphertext into plaintext.

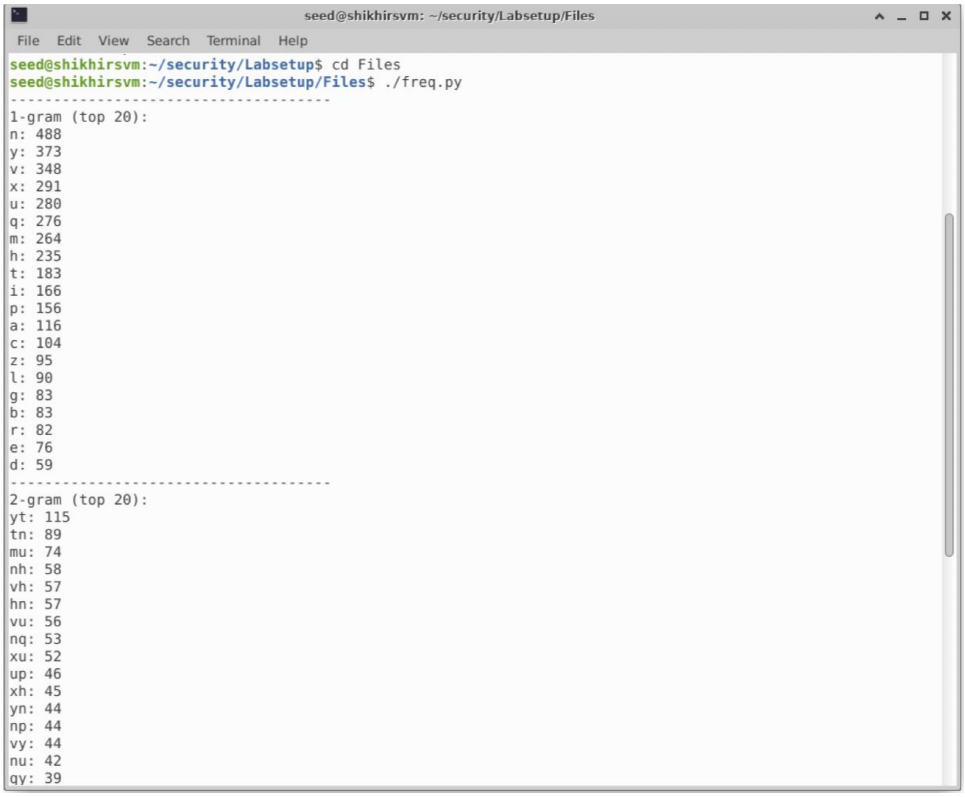
Bigrams

Of 2,383,373,483 bigrams scanned:

1. th (92535489, 3.882543%)
2. he (87741289, 3.681391%)
3. in (54433847, 2.283899%)
4. er (51910883, 2.178042%)
5. an (51015163, 2.140460%)
6. re (41694599, 1.749394%)
7. nd (37466077, 1.571977%)
8. on (33802063, 1.418244%)
9. en (32967758, 1.383239%)
10. at (31830493, 1.335523%)
11. ou (30637892, 1.285484%)
12. ed (30406590, 1.275779%)
13. ha (30381856, 1.274742%)
14. to (27877259, 1.169655%)
15. or (27434858, 1.151094%)
16. it (27048699, 1.134891%)
17. is (26452510, 1.109877%)
18. hi (26033632, 1.092302%)
19. es (26033602, 1.092301%)
20. ng (25106109, 1.053385%)

Trigrams
Of 1,699,542,842 trigrams scanned:
1. the (59623899, 3.508232%)
2. and (27088636, 1.593878%)
3. ing (19494469, 1.147042%)
4. her (13977786, 0.822444%)
5. hat (11059185, 0.650715%)
6. his (10141992, 0.596748%)
7. tha (10088372, 0.593593%)
8. ere (9527535, 0.560594%)
9. for (9438784, 0.555372%)
10. ent (9020688, 0.530771%)
11. ion (8607405, 0.506454%)
12. ter (7836576, 0.461099%)
13. was (7826182, 0.460487%)
14. you (7430619, 0.437213%)
15. ith (7329285, 0.431250%)
16. ver (7320472, 0.430732%)
17. all (7184955, 0.422758%)
18. wit (6752112, 0.397290%)
19. thi (6709729, 0.394796%)
20. tio (6425262, 0.378058%)

We look at the n-gram frequencies of most commonly occurring 2-gram and 3-gram words in the English language.

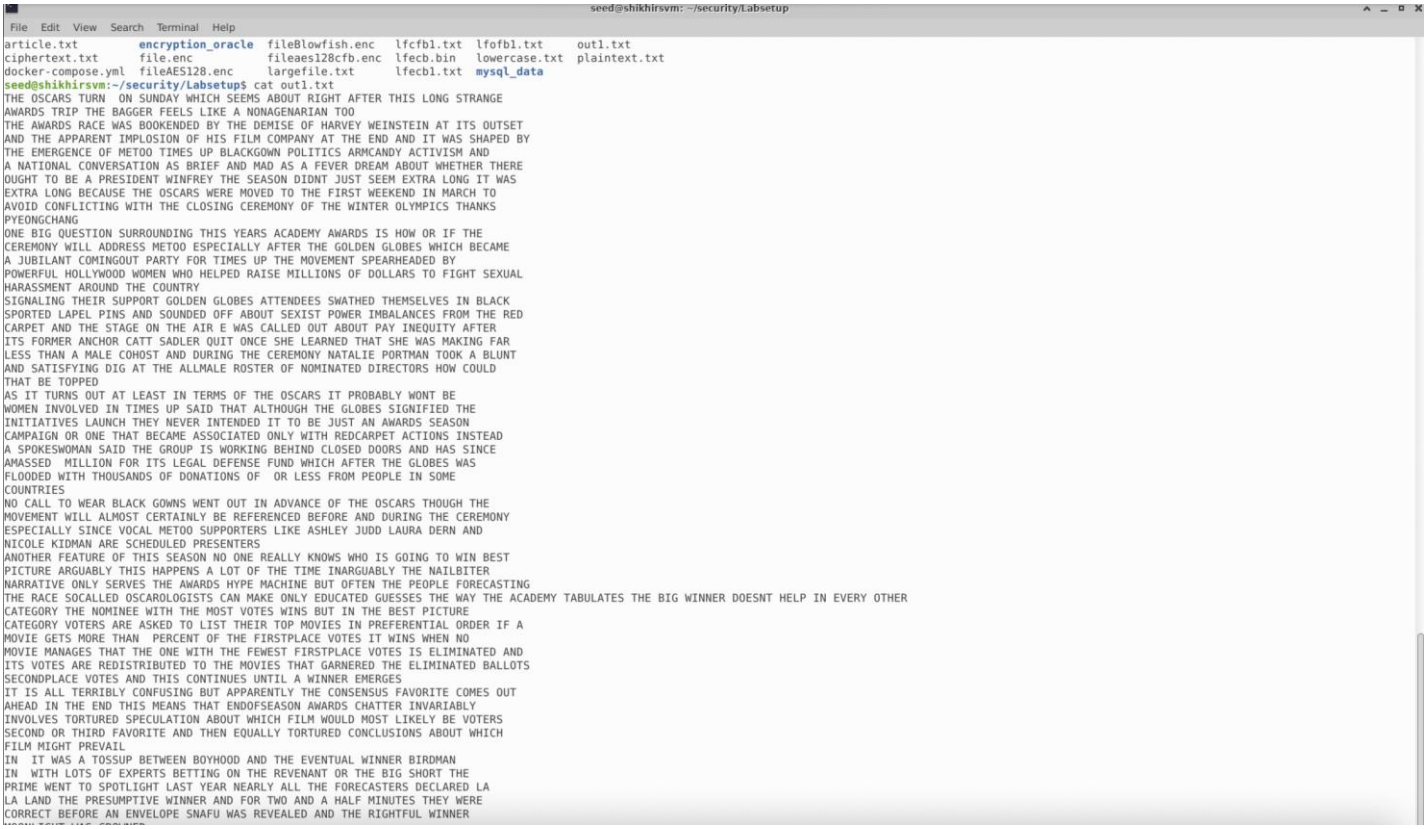


Using the freq.py file we can see the frequencies of the most commonly occurring n-grams in our ciphertext. Using the frequency analysis and the other links provided in the document, we map the ciphertext alphabets to plain text alphabets.

Using the translate function we replace the lowercase ciphertext with capital plaintext and the final output is shown below.

I have used the following command:

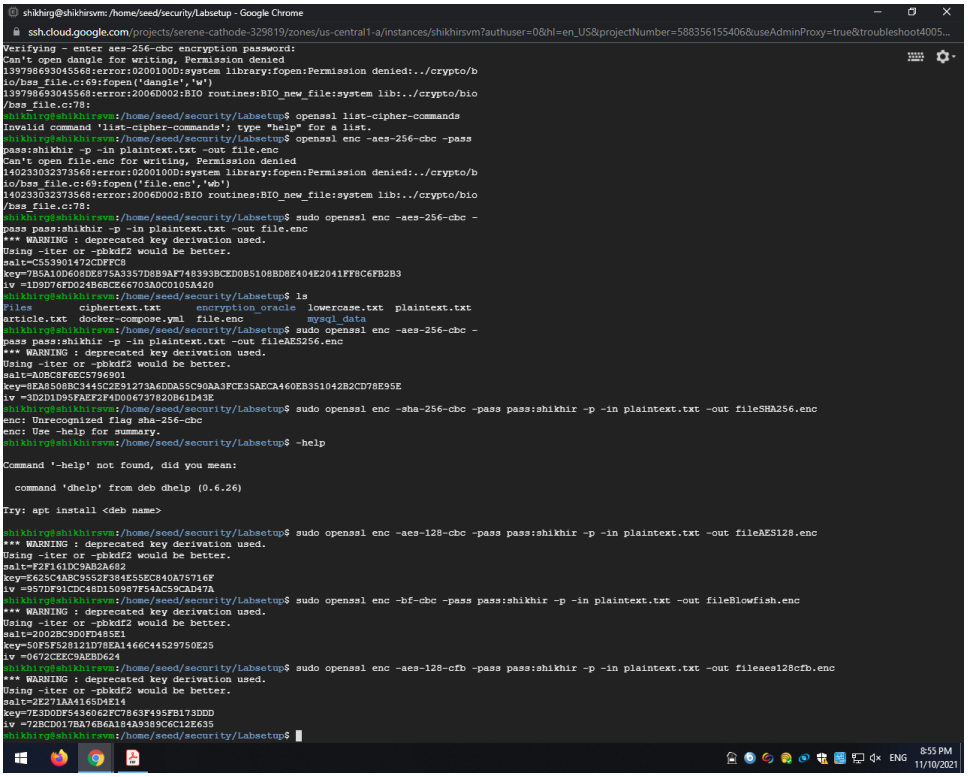
tr "ntyhquvmxbpzfrceialdgoswjk" "EHTRSNAIOFDUVGMPLCWYBJKMQX” < ciphertext.txt > out1.txt

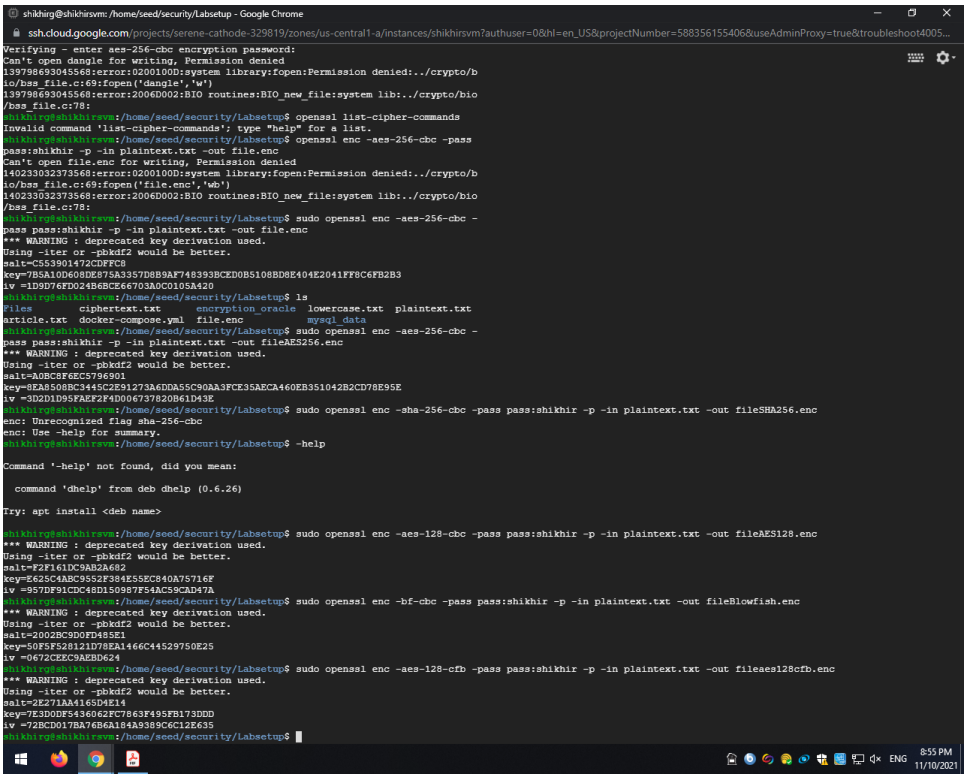


We can see that we have successfully translated/decrypted the ciphertext into plaintext.

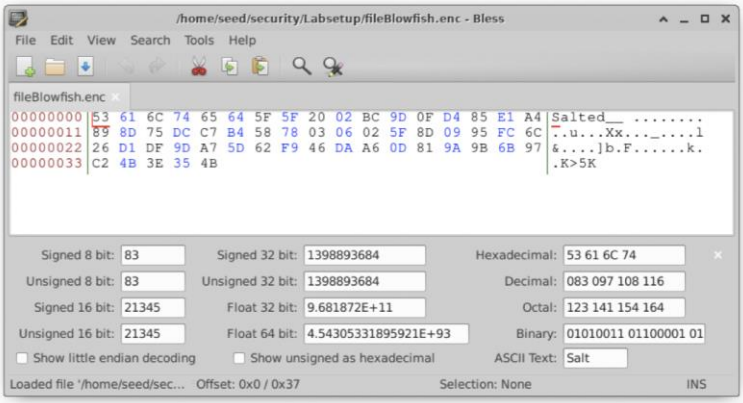
Task 2: Encryption using Different Ciphers and Modes

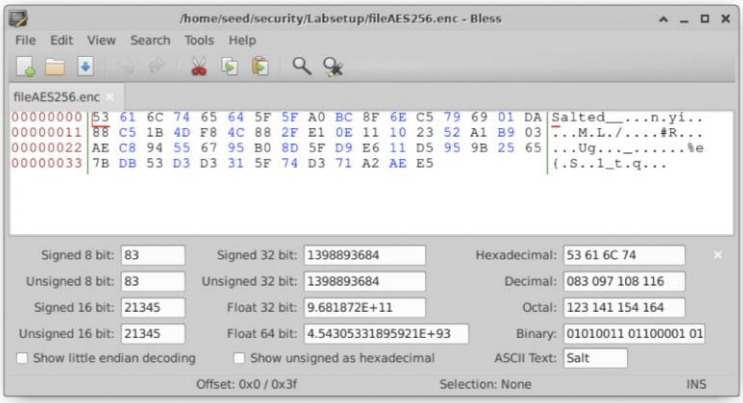
In this task we encrypt the files using different ciphers and different modes.

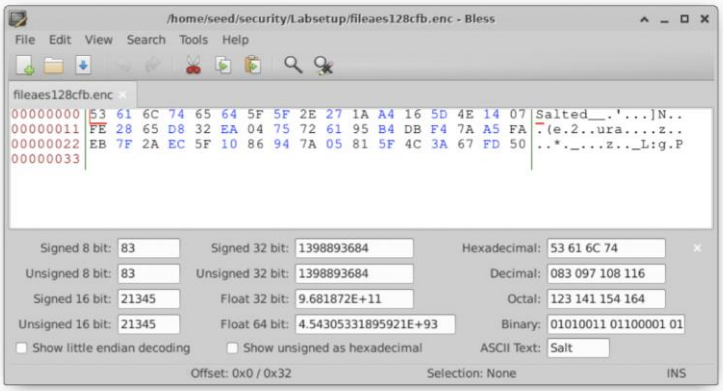


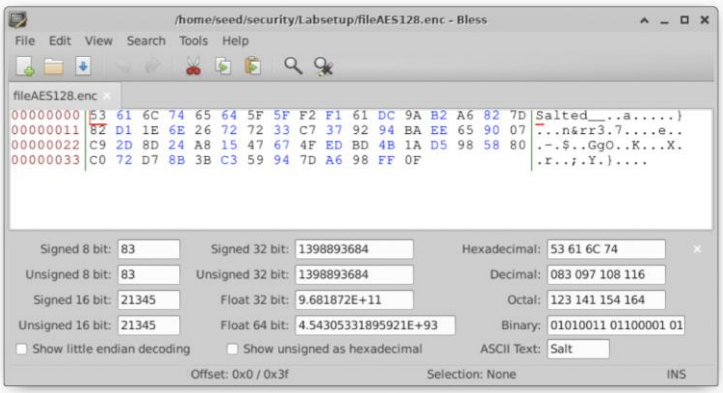


The ciphers and modes used are AES-128-CFB, Blowfish-CBC, AES-256 and AES-128-CBC.


/home/seed/security/Labsetup/fileBlowfish.enc - Bless
File Edit View Search Tools Help
fileBlowfish.enc
00000000 53 61 6C 74 65 64 5F 5F 20 02 BC 9D 0F D4 85 E1 A4 Salted__ ....
00000011 89 8D 75 DC C7 B4 58 78 03 06 02 5F 8D 09 95 FC 6C ..u...Xx..._....l
00000022 26 D1 DF 9D A7 5D 62 F9 46 DA A6 0D 81 9A 9B 6B 97 &....]b.F......k.
00000033 C2 4B 3E 35 4B .K>5K
Signed 8 bit: 83
Signed 32 bit: 1398893684
Hexadecimal: 53 61 6C 74
Unsigned 8 bit: 83
Unsigned 32 bit: 1398893684
Decimal: 083 097 108 116
Signed 16 bit: 21345
Float 32 bit: 9.681872E+11
Octal: 123 141 154 164
Unsigned 16 bit: 21345
Float 64 bit: 4.54305331895921E+93
Binary: 01010011 01100001 01
Show little endian decoding
Show unsigned as hexadecimal
ASCII Text: Salt
Loaded file '/home/seed/sec... Offset: 0x0 / 0x37
Selection: None
INS


/home/seed/security/Labsetup/fileAES256.enc - Bless
File Edit View Search Tools Help
fileAES256.enc
00000000 53 61 6C 74 65 64 5F 5F A0 BC 8F 6E C5 79 69 01 DA Salted__...n.yi..
00000011 88 C5 1B 4D F8 4C 88 2F E1 0E 11 10 23 52 A1 B9 03 ...M.L./....#R...
00000022 AE C8 94 55 67 95 B0 8D 5F D9 E6 11 D5 95 9B 25 65 ...Ug..._......%e
00000033 7B DB 53 D3 D3 31 5F 74 D3 71 A2 AE E5 (.S..1_t.q...
Signed 8 bit: 83
Signed 32 bit: 1398893684
Hexadecimal: 53 61 6C 74
Unsigned 8 bit: 83
Unsigned 32 bit: 1398893684
Decimal: 083 097 108 116
Signed 16 bit: 21345
Float 32 bit: 9.681872E+11
Octal: 123 141 154 164
Unsigned 16 bit: 21345
Float 64 bit: 4.54305331895921E+93
Binary: 01010011 01100001 01
Show little endian decoding
Show unsigned as hexadecimal
ASCII Text: Salt
Offset: 0x0 / 0x3f
Selection: None
INS

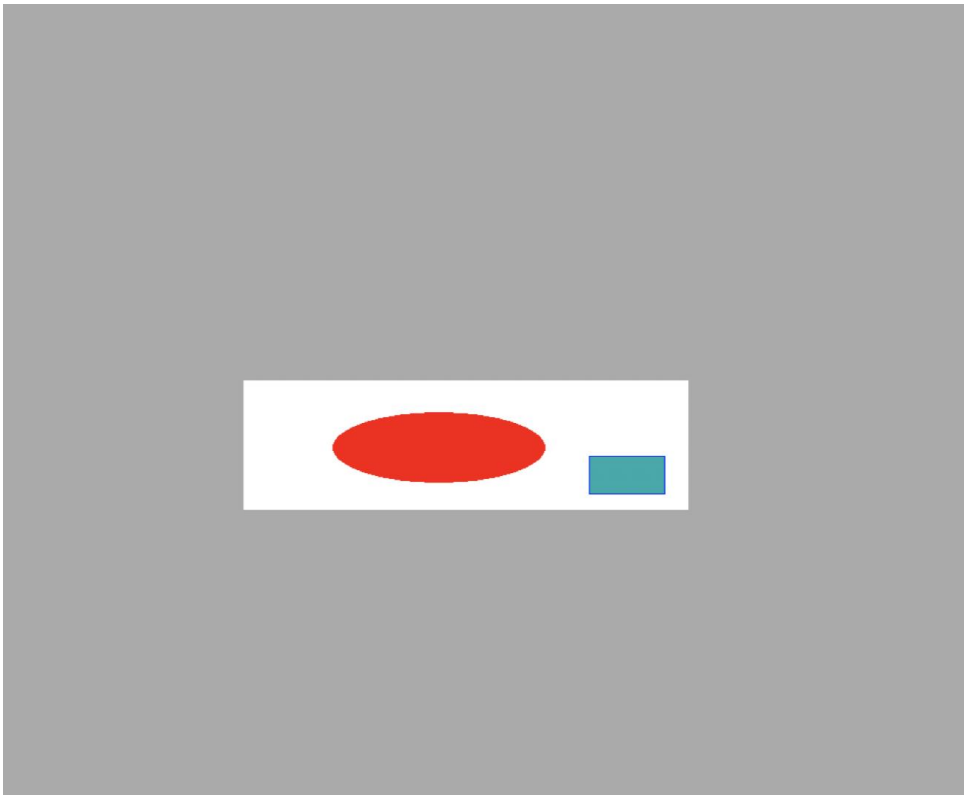


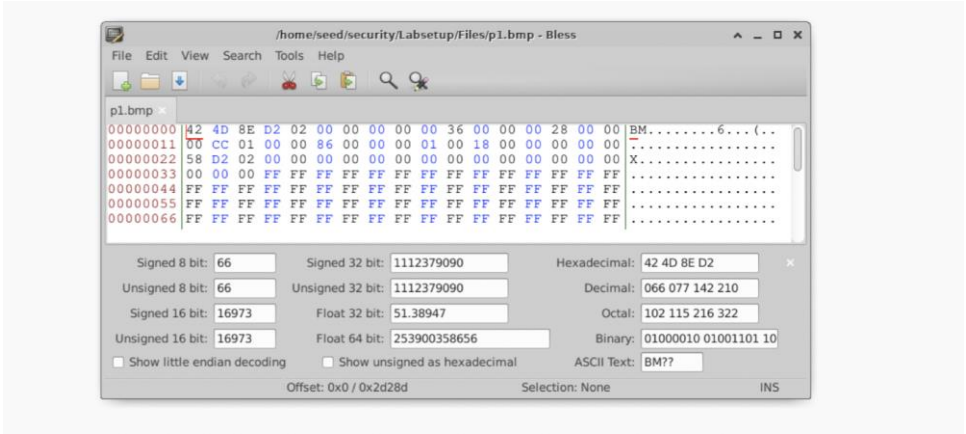


I have also displayed the hexcode for the generated enc files. The sizes of the files may vary depending on the cipher and the mode.

We observe that we have successfully encrypted the files using different ciphers and modes.

Task 3: Encryption Mode – ECB vs. CBC

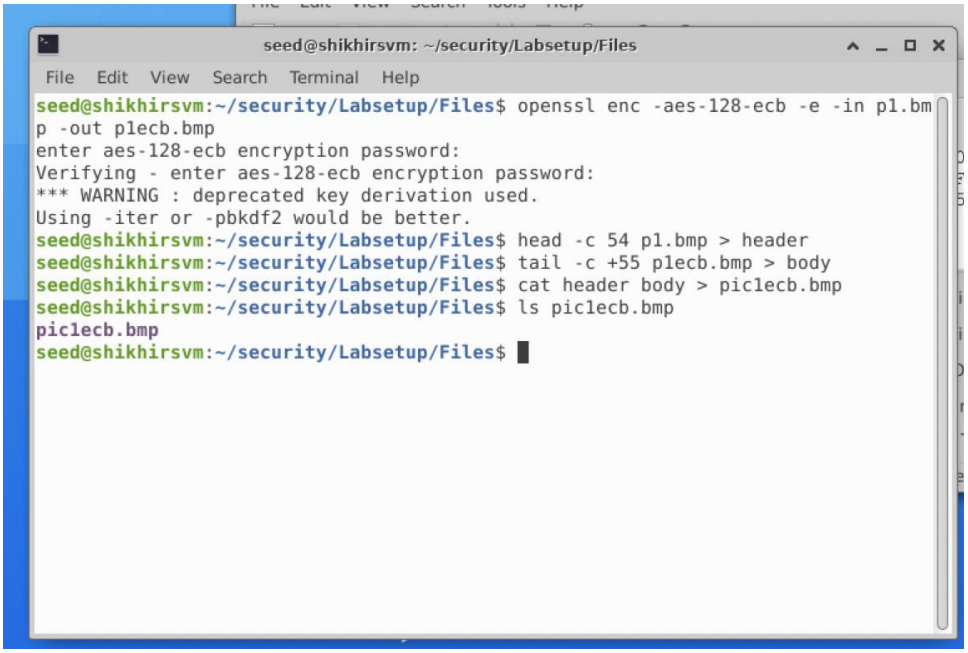


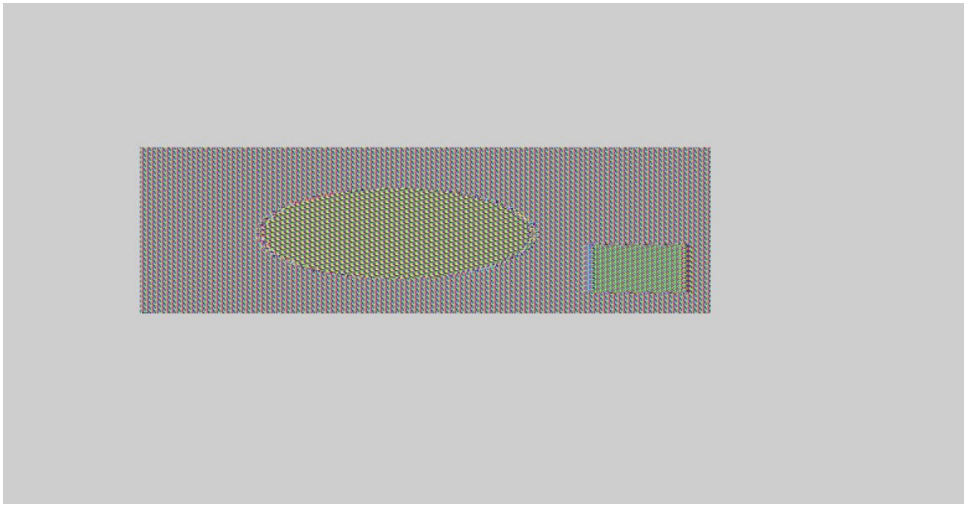


We first look at the original picture given to us; I have renamed it to p1 for the ease of use. We are going to encrypt it using different modes and use the original 54 bytes from the unencrypted image as the header for the bmp file and the body from the encrypted file.

1) Encrypting the image using ECB (Electronic Code Book)

Using the openssl command we encrypt the image with aes-128-ecb and use the 55+ bytes of the file as the body of the encrypted image. The header comes from the orignal image file.





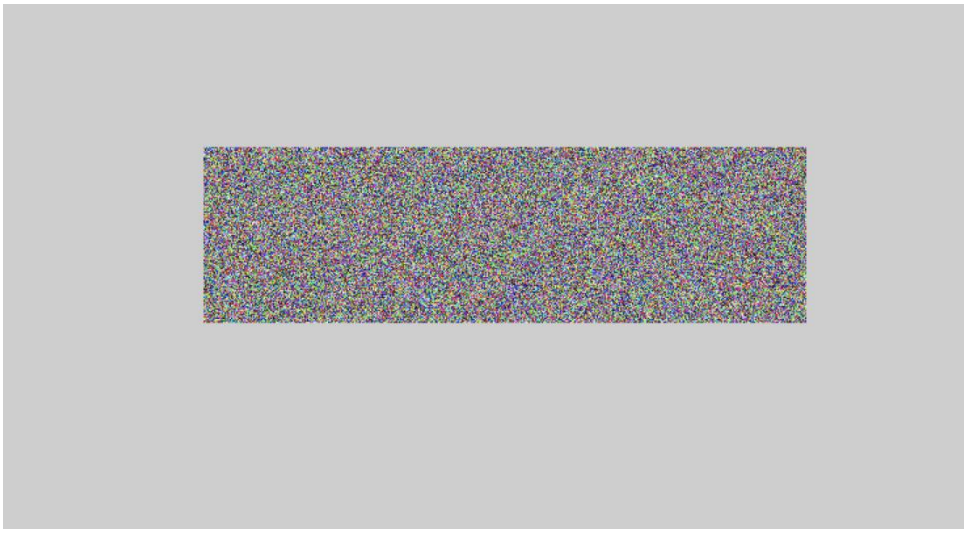
Here we can see that we have succesfully redirected the output to the bmp file. The first 54 bits of the encrypted bmp file are replaced with the first 54 bits of the original bmp file. The encrypted image can be seen using image viewer. Information about the original image can be observed in this encrypted bmp file because ECB mode generates same cipher text for repeating plain text. Even though the cipher image is different from the original image, most of the original image's information can be obtained from the cipher image and the shapes are preserved.

2) Encrypting the image using CBC (Cipher block chaining)

Block cipher encryption - each block of the plain text is XORed with the previously generated cipher text and then it is encrypted. So, each cipher block depends upon all plain texts that were processed.

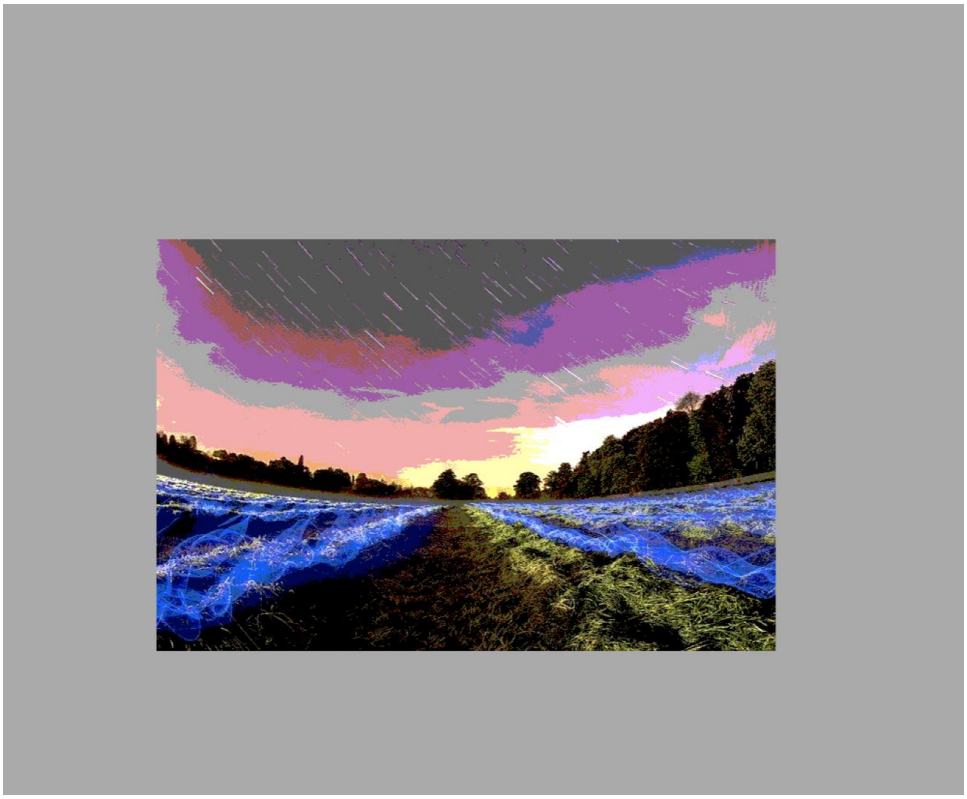
```
seed@shikhirsvm:~/security/Labsetup/Files$ cat header body > pic1ecb.bmp
seed@shikhirsvm:~/security/Labsetup/Files$ ls pic1ecb.bmp
pic1ecb.bmp
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cbc -e -in p1.bm
p -out p1ecb.bmp
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cbc -e -in p1.bm
p -out p1cbc.bmp
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ head -c 54 p1.bmp > header
seed@shikhirsvm:~/security/Labsetup/Files$ tail -c +55 p1cbc.bmp > body
seed@shikhirsvm:~/security/Labsetup/Files$ cat header body > pic1cbc.bmp
seed@shikhirsvm:~/security/Labsetup/Files$ ls pic1cbc.bmp
pic1cbc.bmp
seed@shikhirsvm:~/security/Labsetup/Files$
```

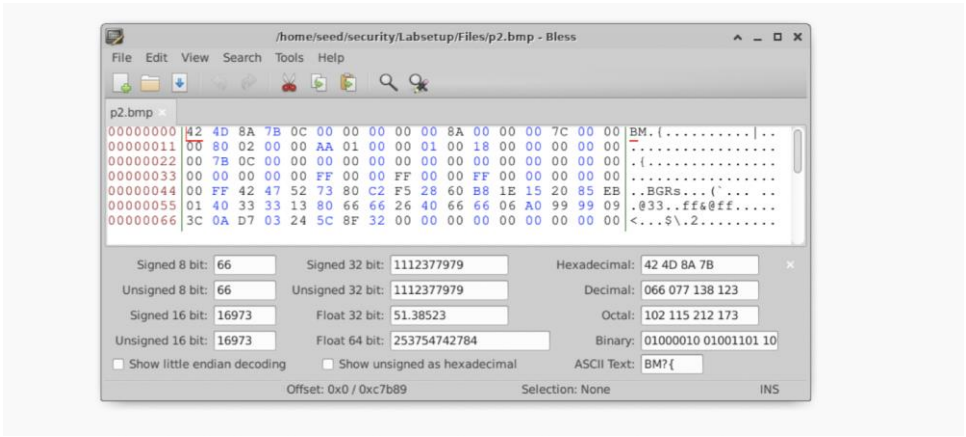
We use the openssl AES-CBC mode to encrypt the original bmp file.



The first 54 bits of the encrypted bmp file are replaced with the first 54 bits of the original bmp file. However, any information about the original image cannot be observed in this encrypted bmp file because CBC mode generates different cipher text for repeating plain text.

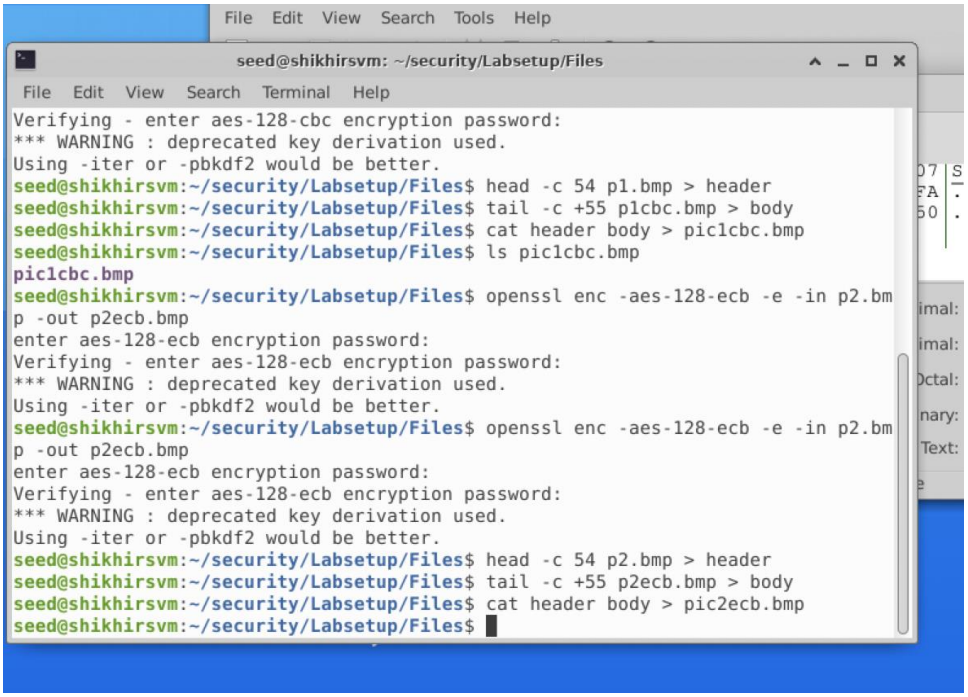
Trying the same experiment use our own sample bmp image



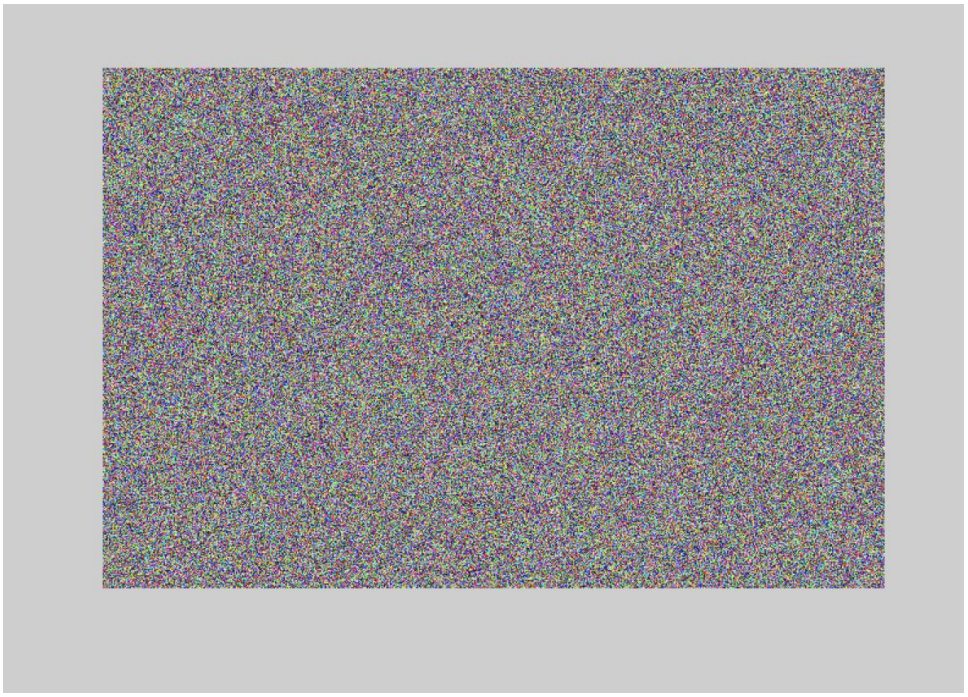


1) Encrypting the image using ECB (Electronic Code Book)

Using the openssl command we encrypt the image with aes-128-ecb and use the 55+ bytes of the file as the body of the encrypted image. The header comes from the orignal image file.



We use the openssl AES-ECB mode to encrypt the original bmp file.

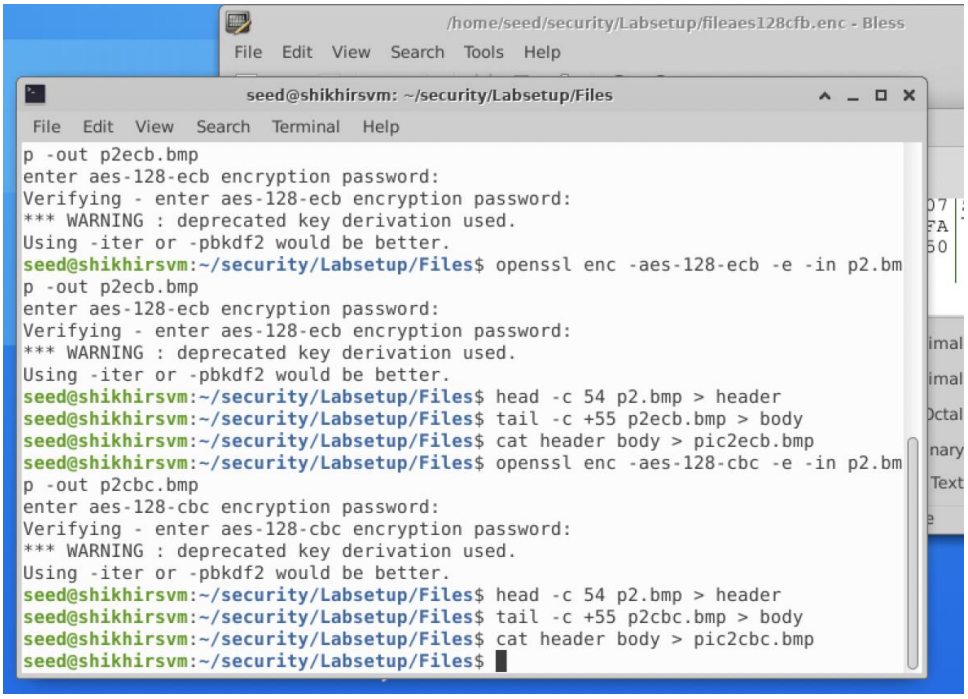


Here we can see that we have succesfully redirected the output to the bmp file. The first 54 bits of the encrypted bmp file are replaced with the first 54 bits of the original bmp file. The encrypted image can be seen using image viewer. Information about the original image can be observed in this encrypted bmp file because ECB mode generates same cipher text for repeating plain text. Even though the cipher image is different from the original image,

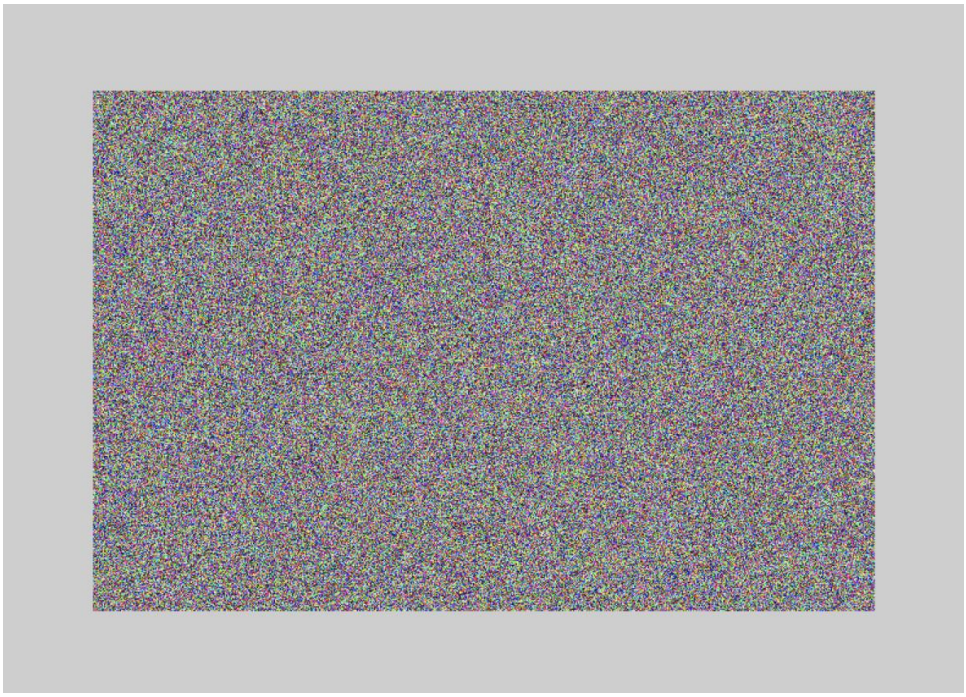
most of the original image's information can be obtained from the cipher image and the shapes are preserved.

2) Encrypting the image using CBC (Cipher block chaining)

Block cipher encryption - each block of the plain text is XORed with the previously generated cipher text and then it is encrypted. So, each cipher block depends upon all plain texts that were processed.



We use the openssl AES-CBC mode to encrypt the original bmp file.



The first 54 bits of the encrypted bmp file are replaced with the first 54 bits of the original bmp file. However, any information about the original image cannot be observed in this encrypted bmp file because CBC mode generates different cipher text for repeating plain text. Hence, we can conclude that the original shape in the image is not preserved.

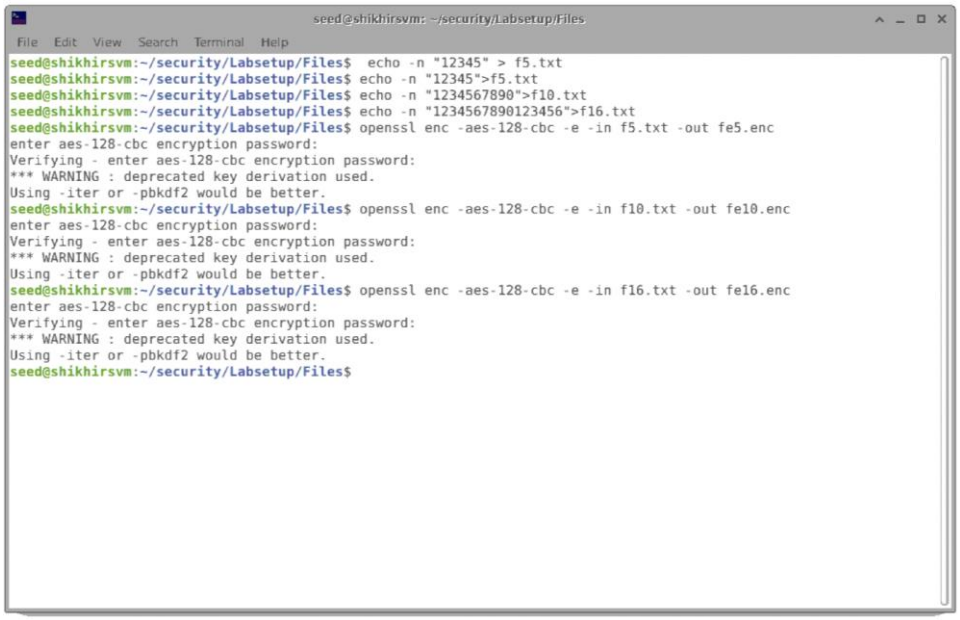
Task 4: Padding

We check for the Encryptions that use padding mechanisms to encrypt.

We first create 3 files of the size 5, 10, 16 bytes respectively and then encrypt them using different modes, we then decrypt them to check if there is some padding added to the file.

1) Encryption and Decryption using CBC

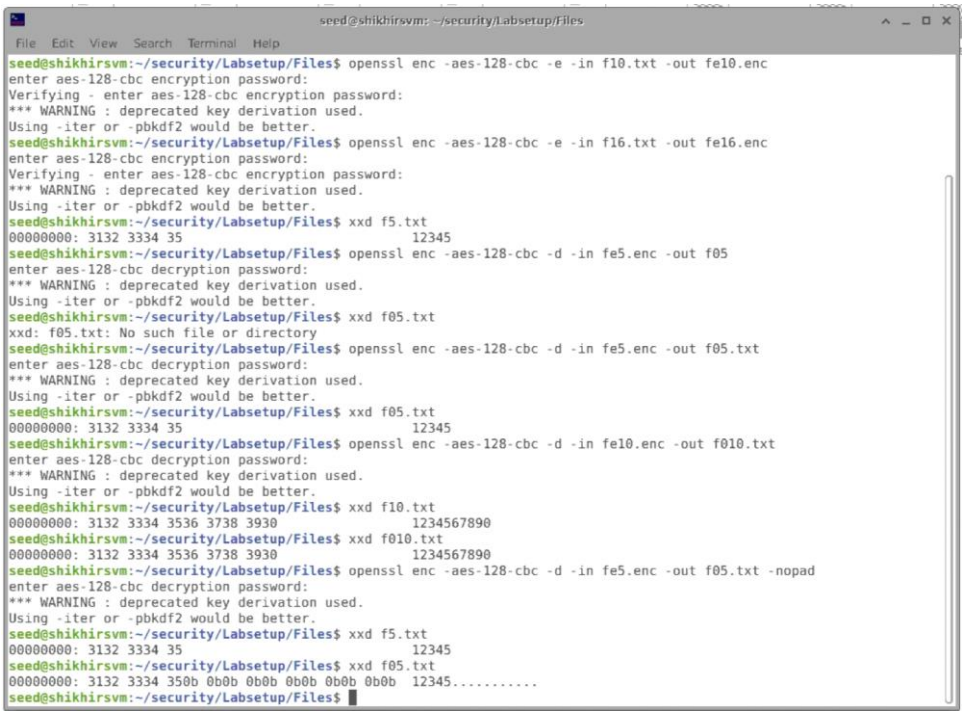
We encrypt all the 3 files using the CBC method, the size of the files is 32 bytes.

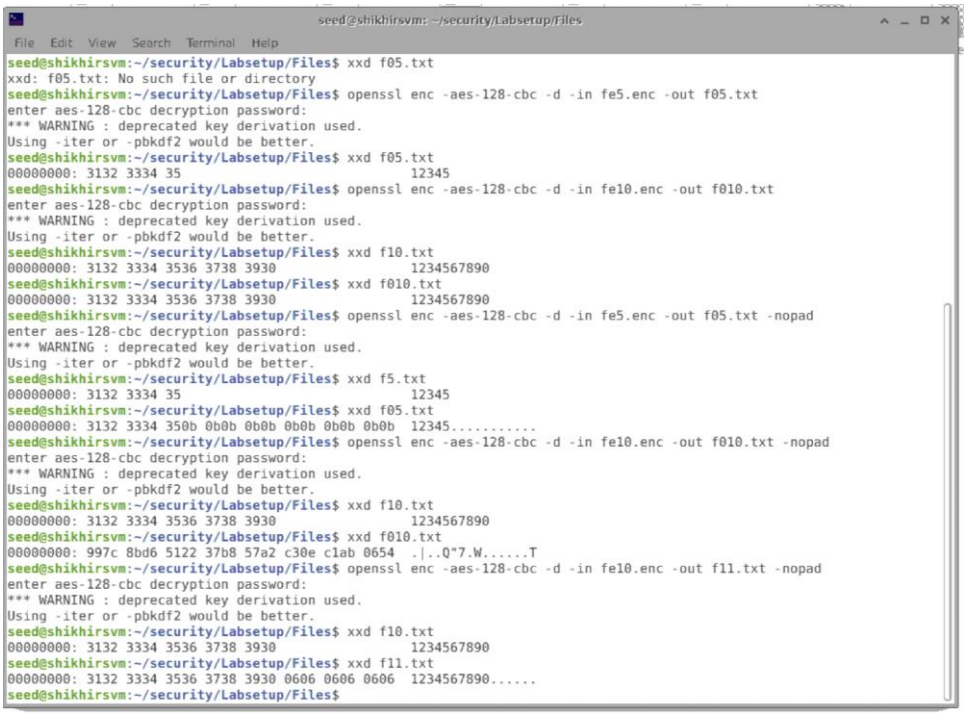


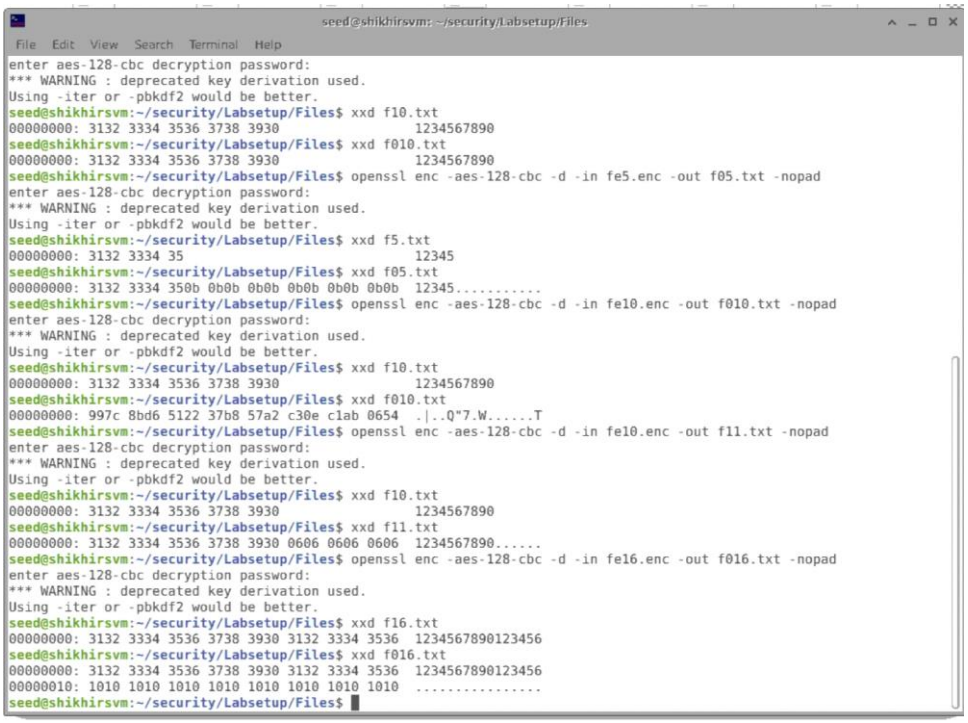
27 Bytes are appended to the 5 Byte file as padding.

22 Bytes are appended to the 10 Byte file as padding.

16 Bytes are appended to the 16 Byte file as padding.







The plain text is encrypted using AES-128-bit cipher with CBC mode. Padding is performed before the encryption starts.

2) Encryption and Decryption using CFB

The plain text for all the 3 files is encrypted using AES-128-bit cipher with CFB mode.

```
eed@shikhirsvm:~/security/Labsetup/Files$
eed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -e -in f5.txt -out f5cbc.bin -pass pass:123
** WARNING : deprecated key derivation used.
sing -iter or -pbkdf2 would be better.
eed@shikhirsvm:~/security/Labsetup/Files$ ls f5cbc.bin -lh
rw-rw-r-- 1 seed seed 21 Nov 12 10:37 f5cbc.bin
eed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f5cbc.bin -out f5cfb.txt -pass pass:123
** WARNING : deprecated key derivation used.
sing -iter or -pbkdf2 would be better.
eed@shikhirsvm:~/security/Labsetup/Files$ ls f5cfb.txt -lh
rw-rw-r-- 1 seed seed 5 Nov 12 10:40 f5cfb.txt
eed@shikhirsvm:~/security/Labsetup/Files$ xxd f5cbc.bin
0000000: 5361 6c74 6564 5f5f 3f64 4339 bcd4 6a7b  Salted__?dC9..j{
0000010: ee39 e6ec a3                              .9...
eed@shikhirsvm:~/security/Labsetup/Files$ xxd f5cfb.txt
0000000: 3132 3334 35                              12345
eed@shikhirsvm:~/security/Labsetup/Files$ xxd f5cfb.bin
xd: f5cfb.bin: No such file or directory
eed@shikhirsvm:~/security/Labsetup/Files$ xxd f5cfb.txt
0000000: 3132 3334 35                              12345
eed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f5cbc.bin -out f5cfb.txt -pass pass:123 -nopad
** WARNING : deprecated key derivation used.
sing -iter or -pbkdf2 would be better.
eed@shikhirsvm:~/security/Labsetup/Files$ xxd f5cfb.txt
0000000: 3132 3334 35                              12345
eed@shikhirsvm:~/security/Labsetup/Files$
```

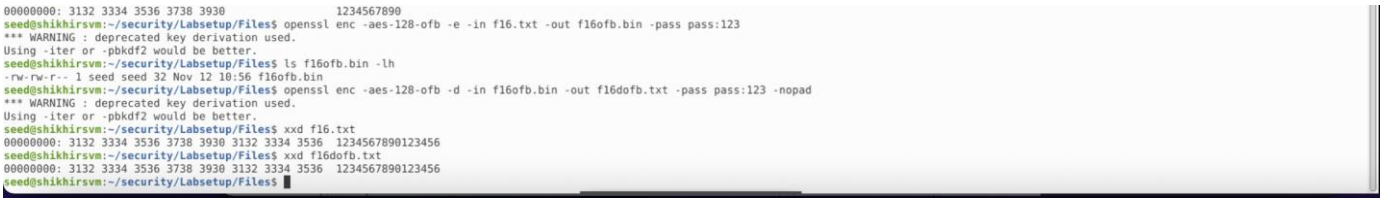
```
00000000: 3132 3334 35                             12345
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -e -in f10.txt -out f10cfb.bin -pass pass:123
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ ls f10cfb.bin -lh
-rw-rw-r-- 1 seed seed 26 Nov 12 10:44 f10cfb.bin
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f10cfb.bin -out f10cfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10cfb.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10cfb.bin
00000000: 5361 6c74 6564 5f5f 8544 9edf 2344 4425  Salted__.D..#DD%
00000010: 65ad f4da 202b 3070 cf4d                 e... +0p.M
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f10cfb.bin -out f10cfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10cfb.bin
00000000: 5361 6c74 6564 5f5f 8544 9edf 2344 4425  Salted__.D..#DD%
00000010: 65ad f4da 202b 3070 cf4d                 e... +0p.M
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f10cfb.bin -out f10dcfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10dcfb.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
seed@shikhirsvm:~/security/Labsetup/Files$
```

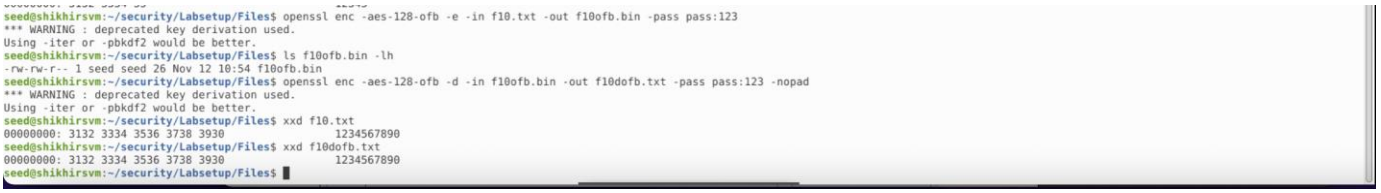
```
00000000: 5361 6c74 6564 5f5f 8544 9edf 2344 4425  Salted__.D..#DD%
00000010: 65ad f4da 202b 3070 cf4d                 e... +0p.M
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f10cfb.bin -out f10cfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10cfb.bin
00000000: 5361 6c74 6564 5f5f 8544 9edf 2344 4425  Salted__.D..#DD%
00000010: 65ad f4da 202b 3070 cf4d                 e... +0p.M
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f10cfb.bin -out f10dcfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f10dcfb.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -e -in f16.txt -out f16cfb.bin -pass pass:123
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ ls f16cfb.bin -lh
-rw-rw-r-- 1 seed seed 32 Nov 12 10:48 f16cfb.bin
seed@shikhirsvm:~/security/Labsetup/Files$ openssl enc -aes-128-cfb -d -in f16cfb.bin -out f16dcfb.txt -pass pass:123 -nopad
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f16.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
seed@shikhirsvm:~/security/Labsetup/Files$ xxd f16dcfb.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
seed@shikhirsvm:~/security/Labsetup/Files$
```

We encrypt the files using the CFB mode of encryption. The encrypted file size varies according to the size of the original text file (i.e., 21, 26, 32 bytes). When we decrypt (with "-nopad" argument), we get the exact same text as our original plaintext. Hence, no padding is done.

3) Encryption and Decryption using OFB

Perform encryption using OFB mode. We get varying sizes of the encrypted file.

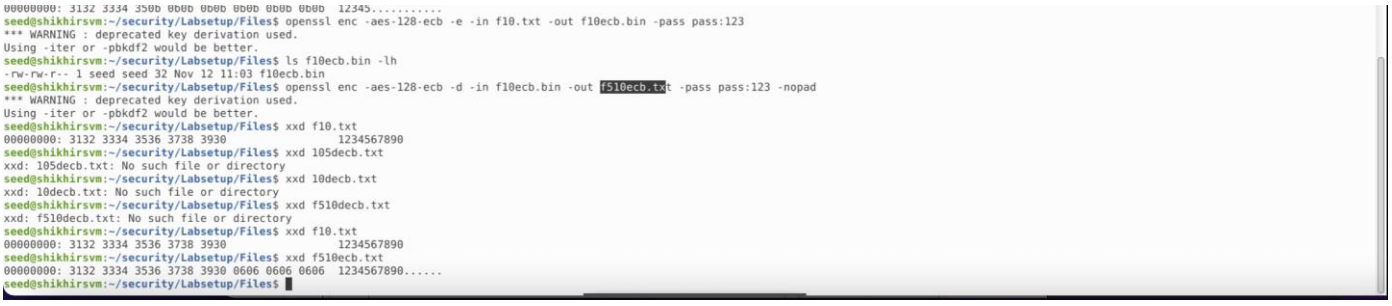




When we decrypt with “-nopad” argument, we get the same text as our original plain text. Hence, no padding is done.

4) Encryption and Decryption using ECB

In ECB, we get greater size of encrypted files than OFB and CFB modes.

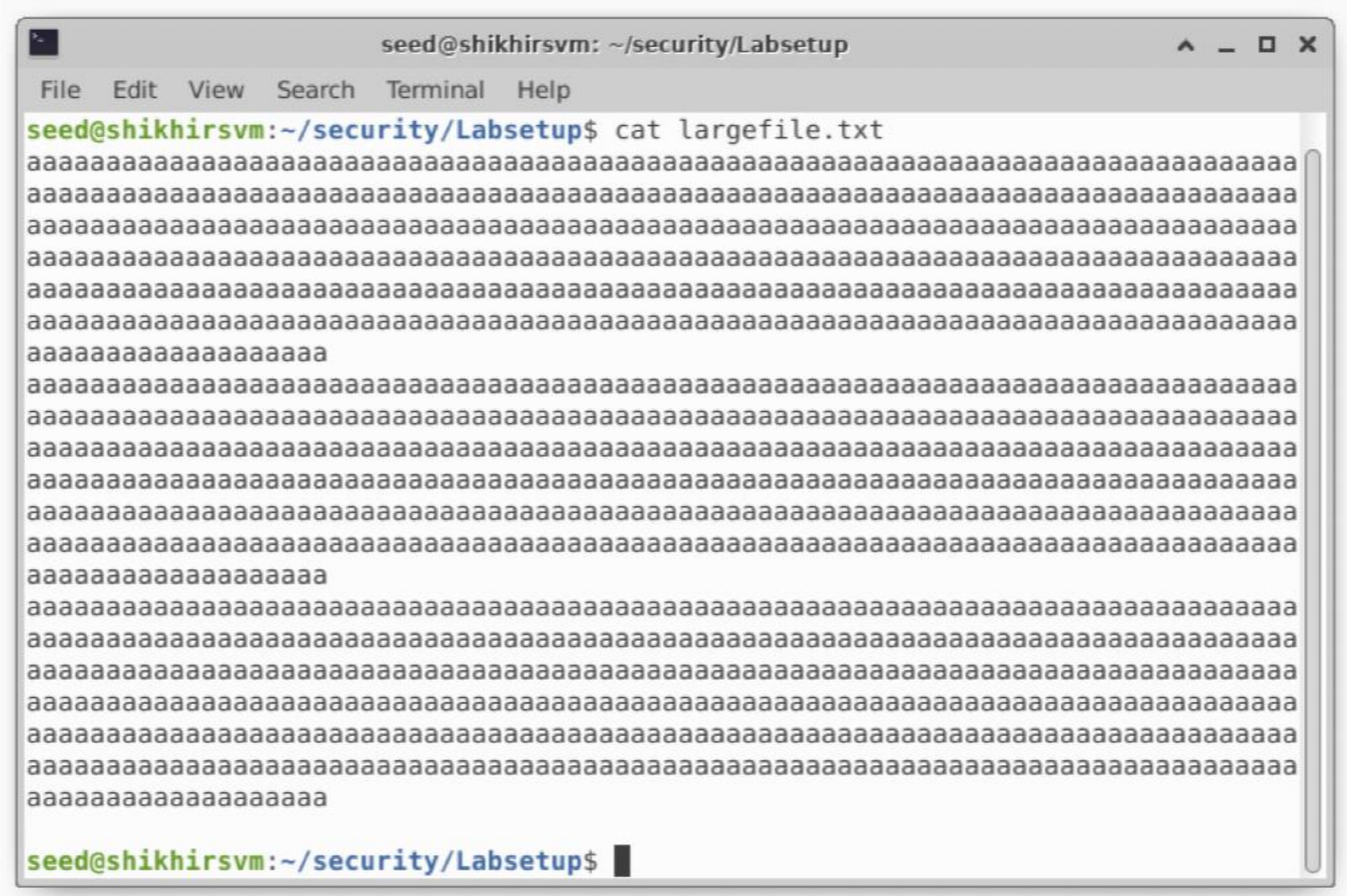


On Decrypting we see a "." character which suggests that padding is used in ECB encryption mode.

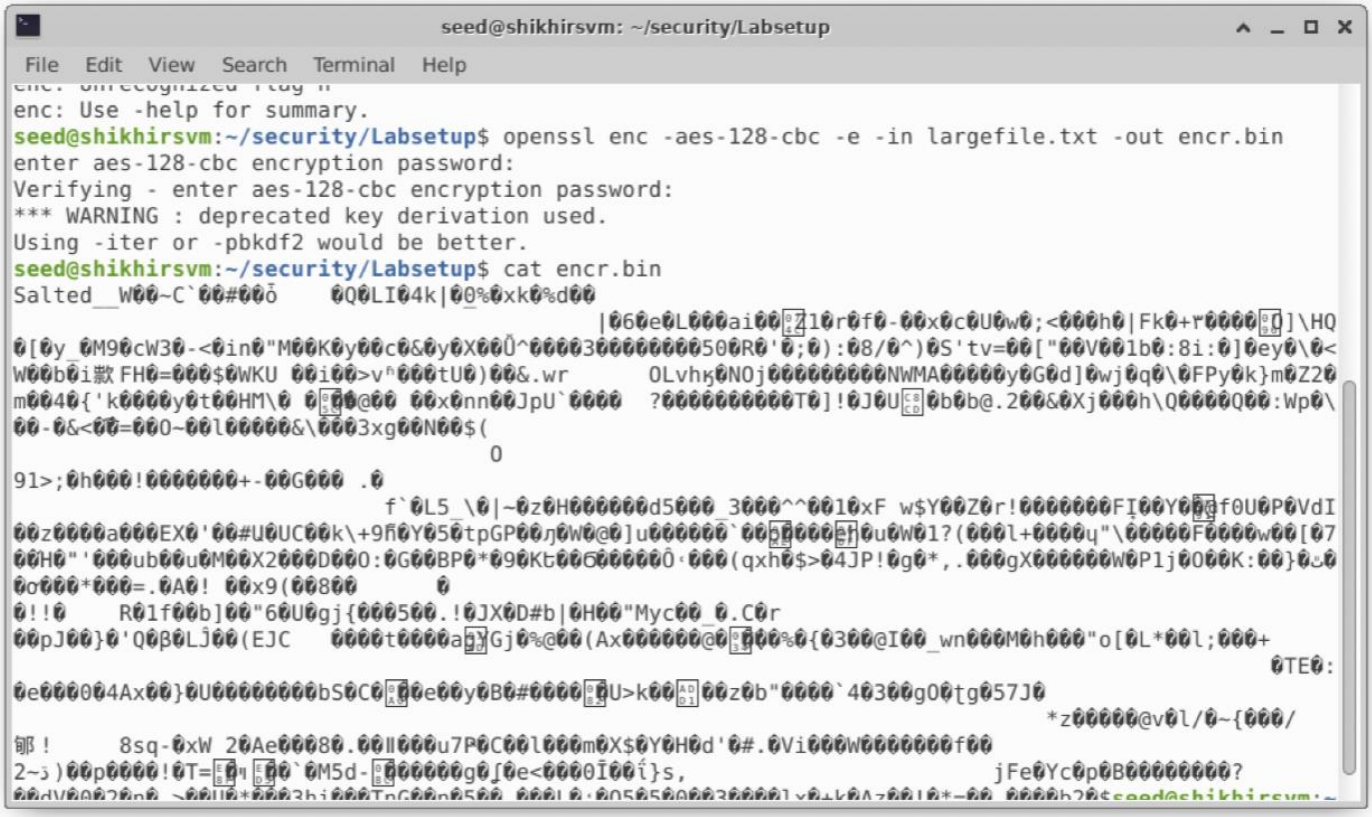
In CFB and OFB, padding is not required as plaintext is XORed with the output of previous cipher block. The first x-bytes of key stream of previous block is XORed with the last x bytes of plaintext, producing x-bytes of cipher text.

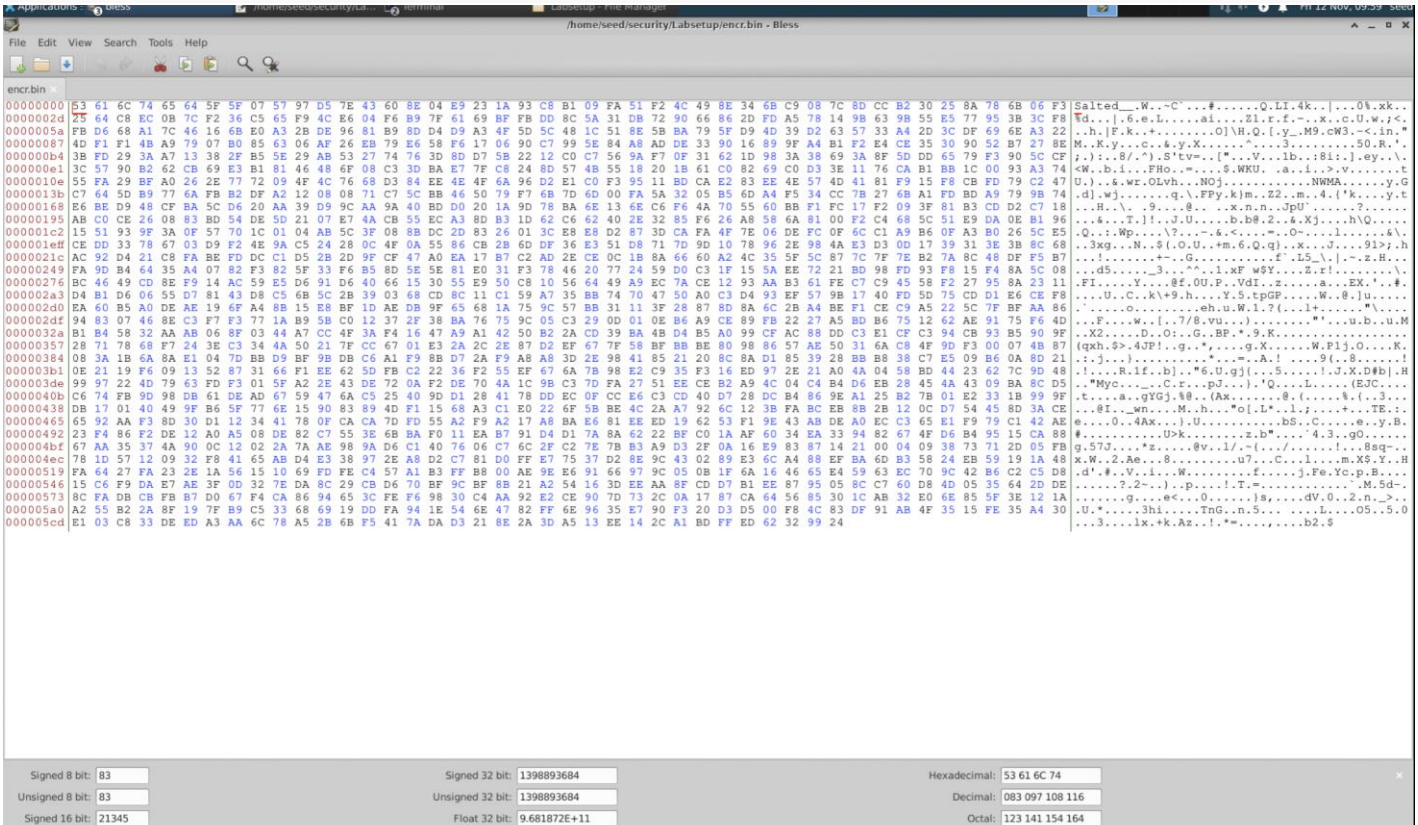
Task 5: Error Propagation – Corrupted Cipher Text

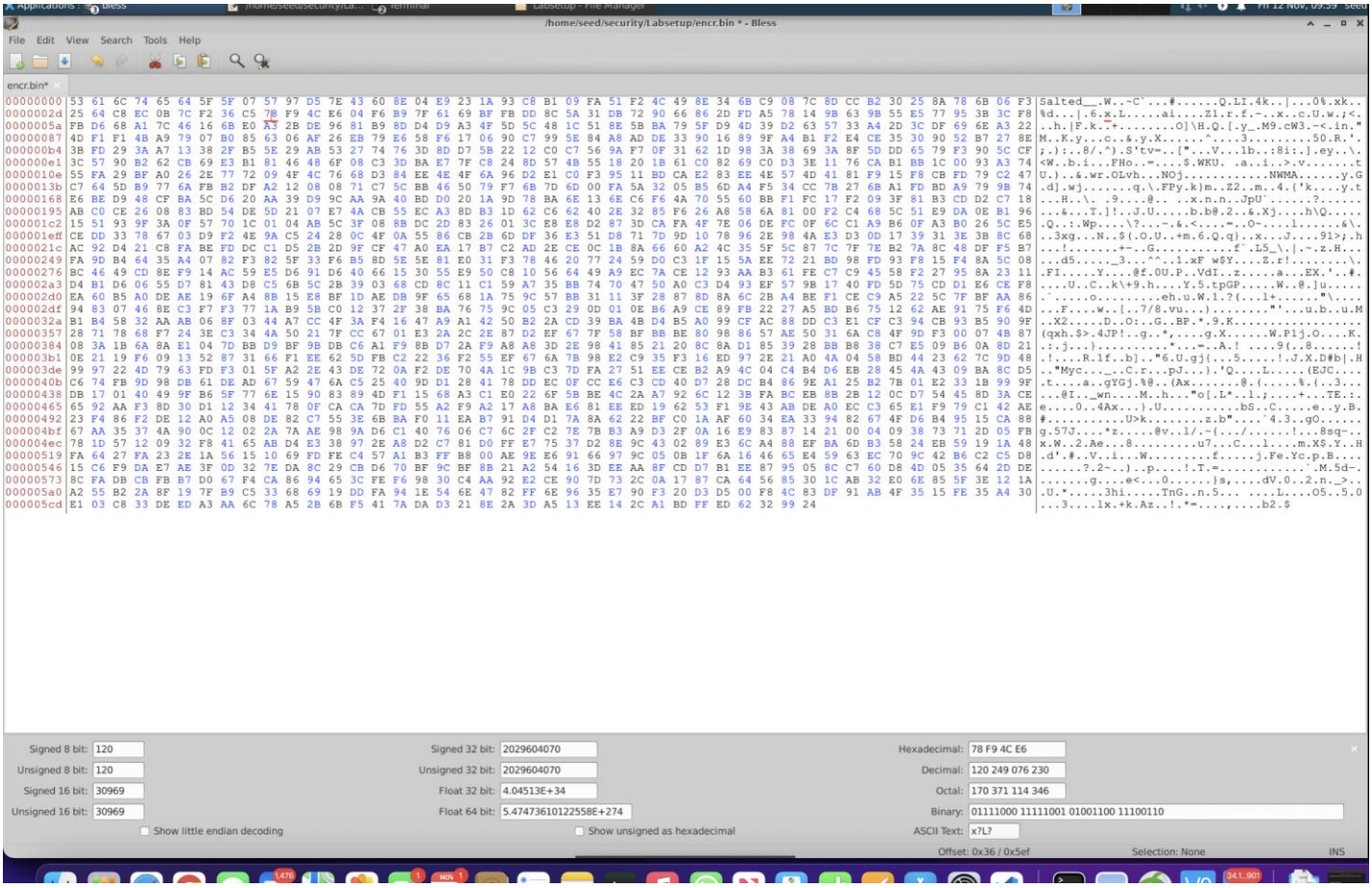
Generate cipher from a random paragraph, change the 55th byte and notice its effect on the decrypted text.



Encryption and decryption using cbc

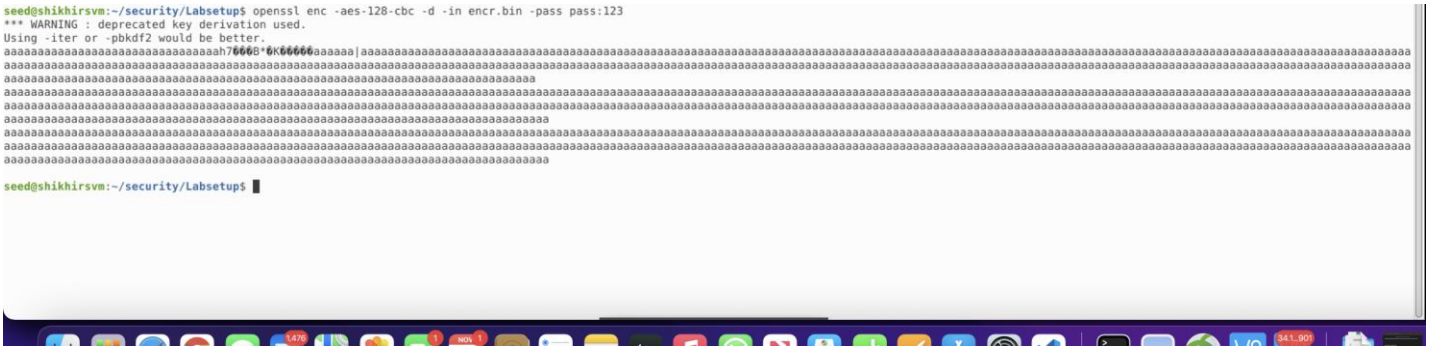

seed@shikhirsvm: ~/security/Labsetup
File Edit View Search Terminal Help
enc: Use -help for summary.
seed@shikhirsvm:~/security/Labsetup$ openssl enc -aes-128-cbc -e -in largefile.txt -out encr.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup$ cat encr.bin
Salted__


/home/seed/security/Labsetup/encr.bin - Bless
File Edit View Search Tools Help
encr.bin
Signed 8 bit: 83
Unsigned 8 bit: 83
Signed 16 bit: 21345
Unsigned 16 bit: 21345
Signed 32 bit: 1398893684
Unsigned 32 bit: 1398893684
Float 32 bit: 9.681872E+11
Float 64 bit: 4.543053318959221E+93
Hexadecimal: 53 61 6C 74
Decimal: 083 097 108 116
Octal: 123 141 154 164
Binary: 01010011 01100001 01101100 01110100
ASCII Text: Salt
Show little endian decoding
Show unsigned as hexadecimal
Offset: 0x0 / 0x5ef
Selection: None
INS



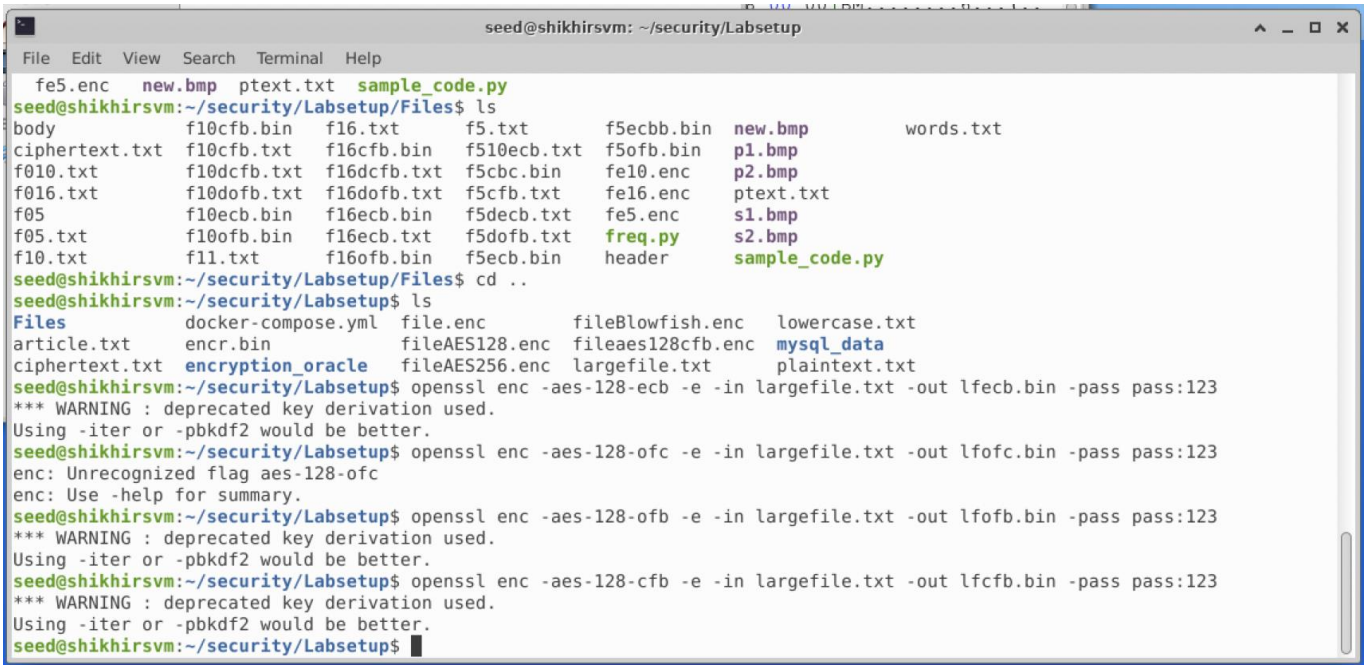
Modifying 55th byte and decrypting.

Output for decrypted cbc file

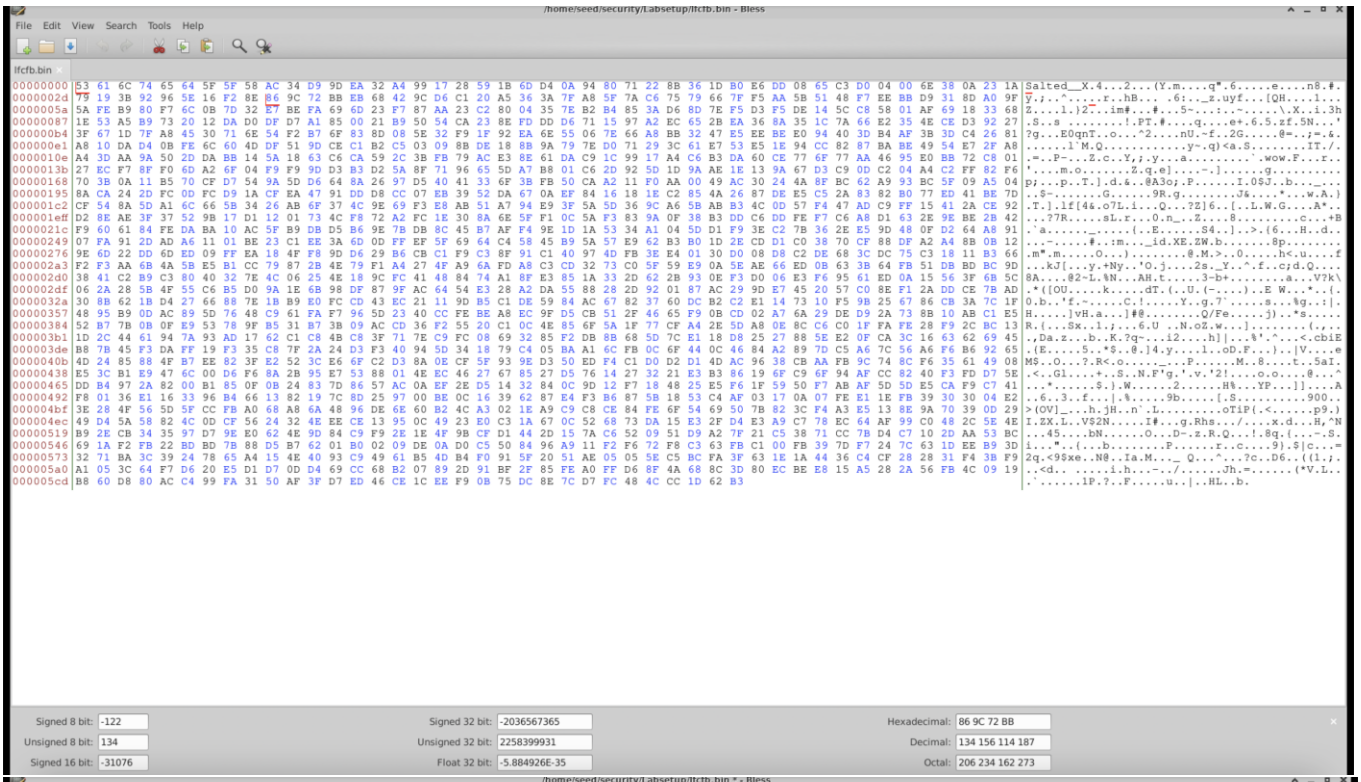


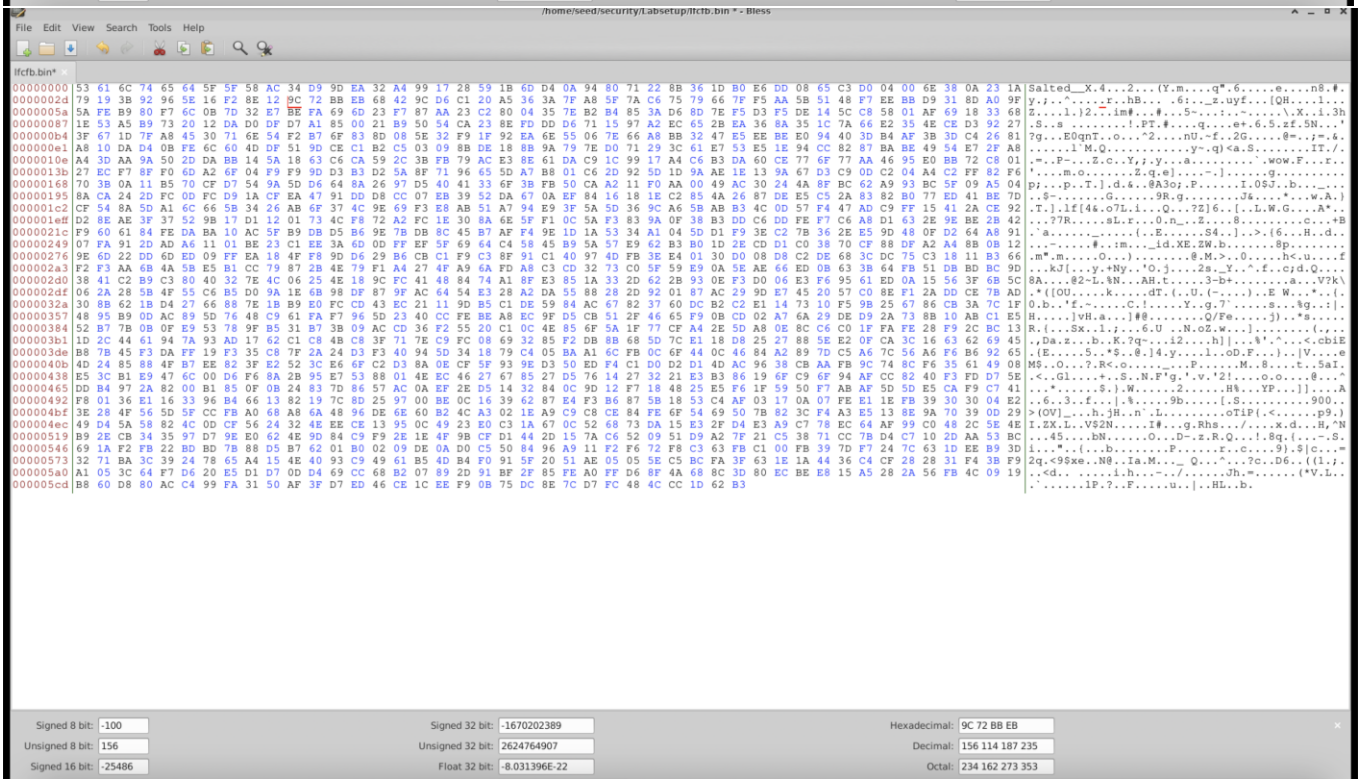
By changing the 55th byte, the decryption has significant changes. It completely distorts the text in a particular region and the transmitted file may not be readible.

Encrypting the same file using different modes – cfb, ecb and ofb

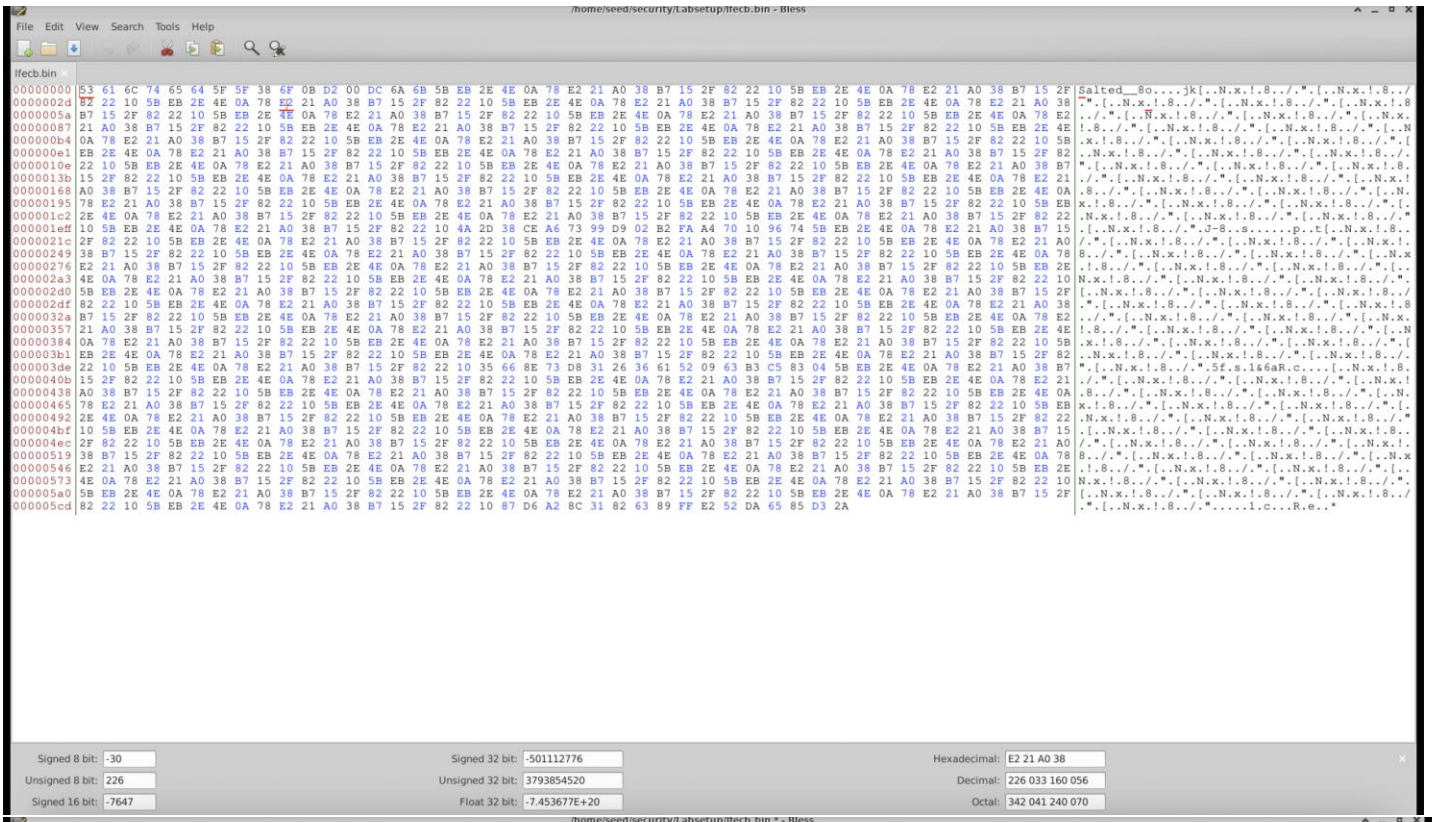


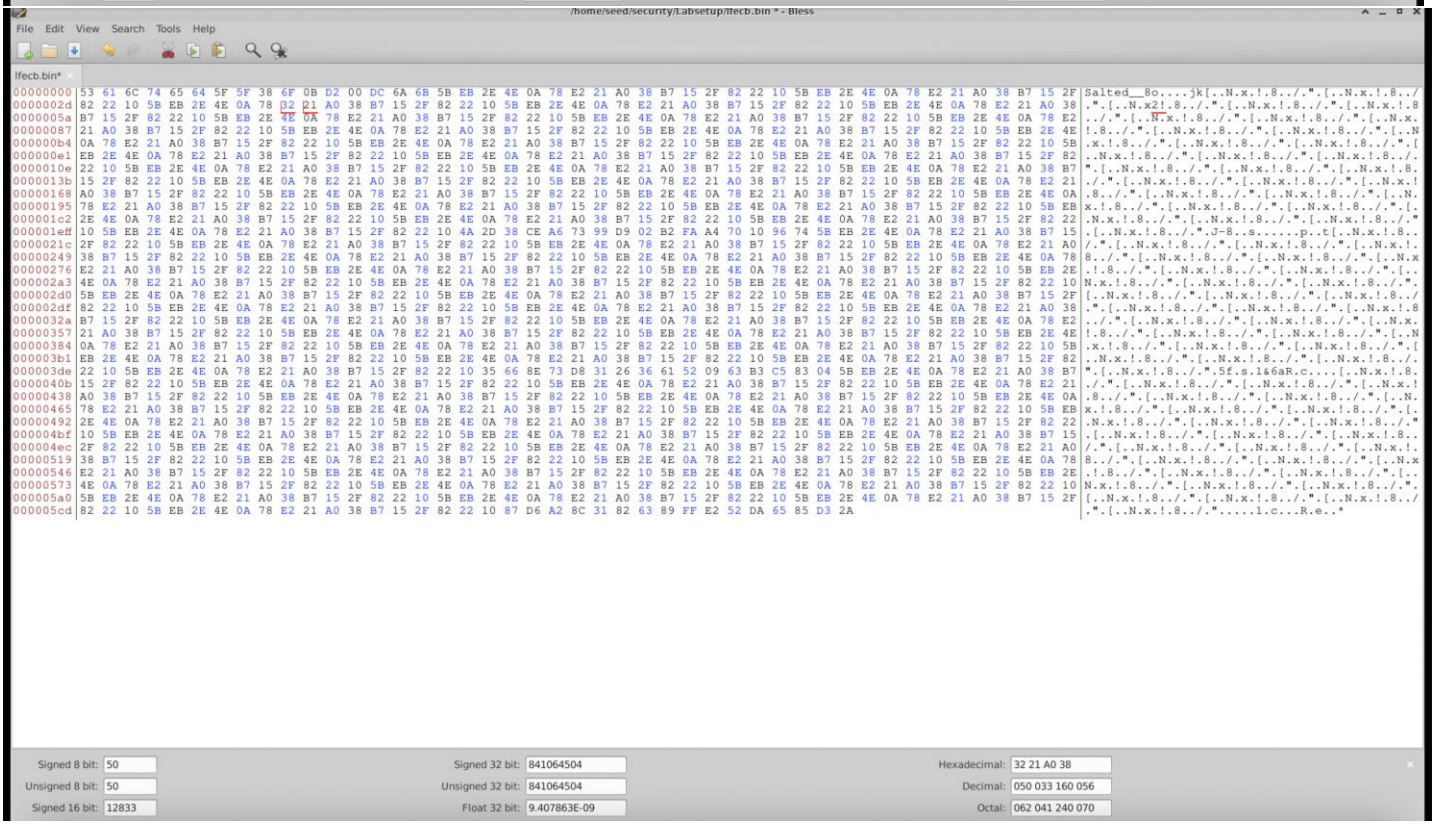
Corrupting the hexcode of the cfb bin file – using bless



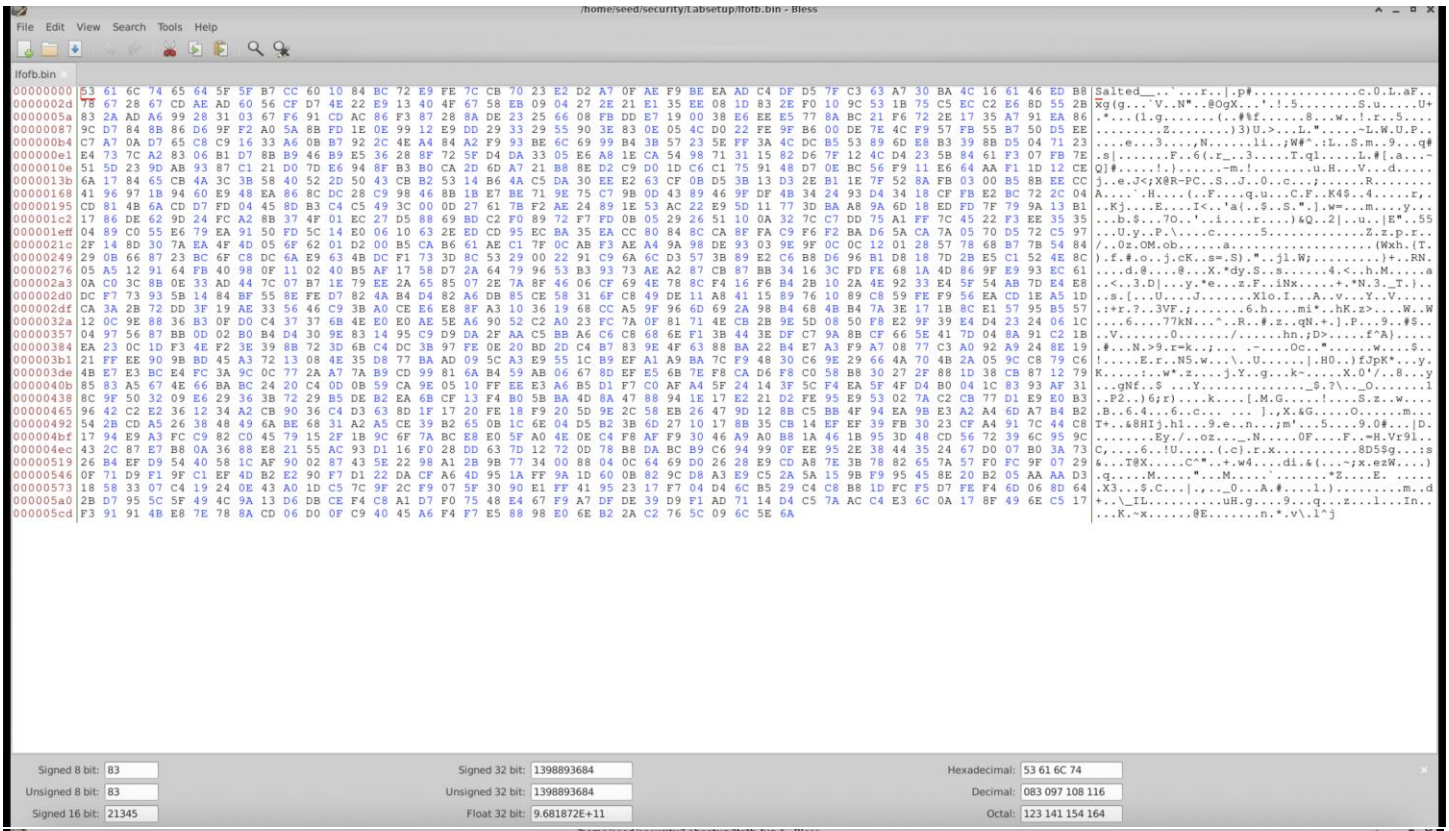


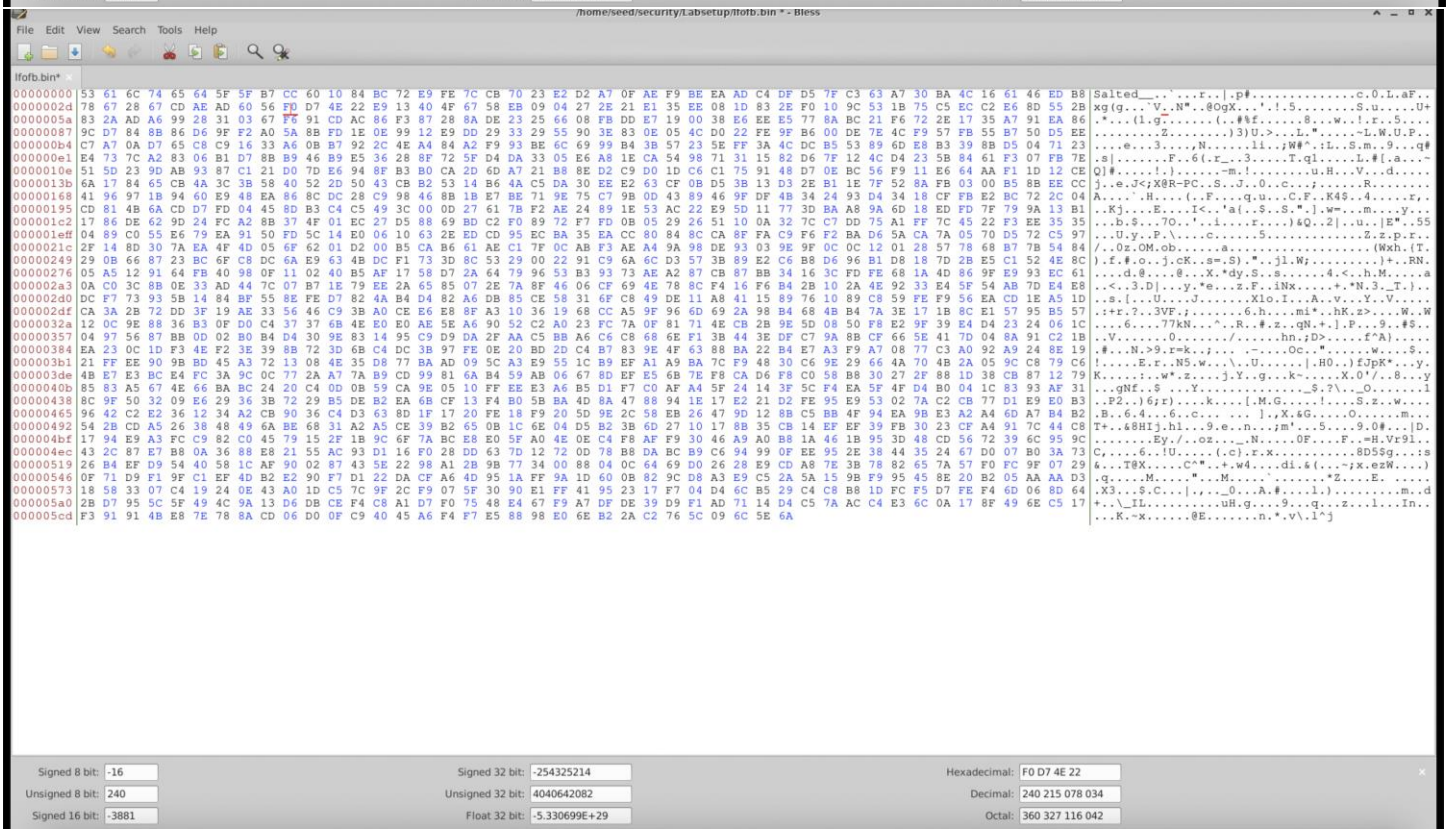
Corrupting the hexcode of the ecb bin file – using bless





Corrupting the hexcode of the ofb bin file – using bless

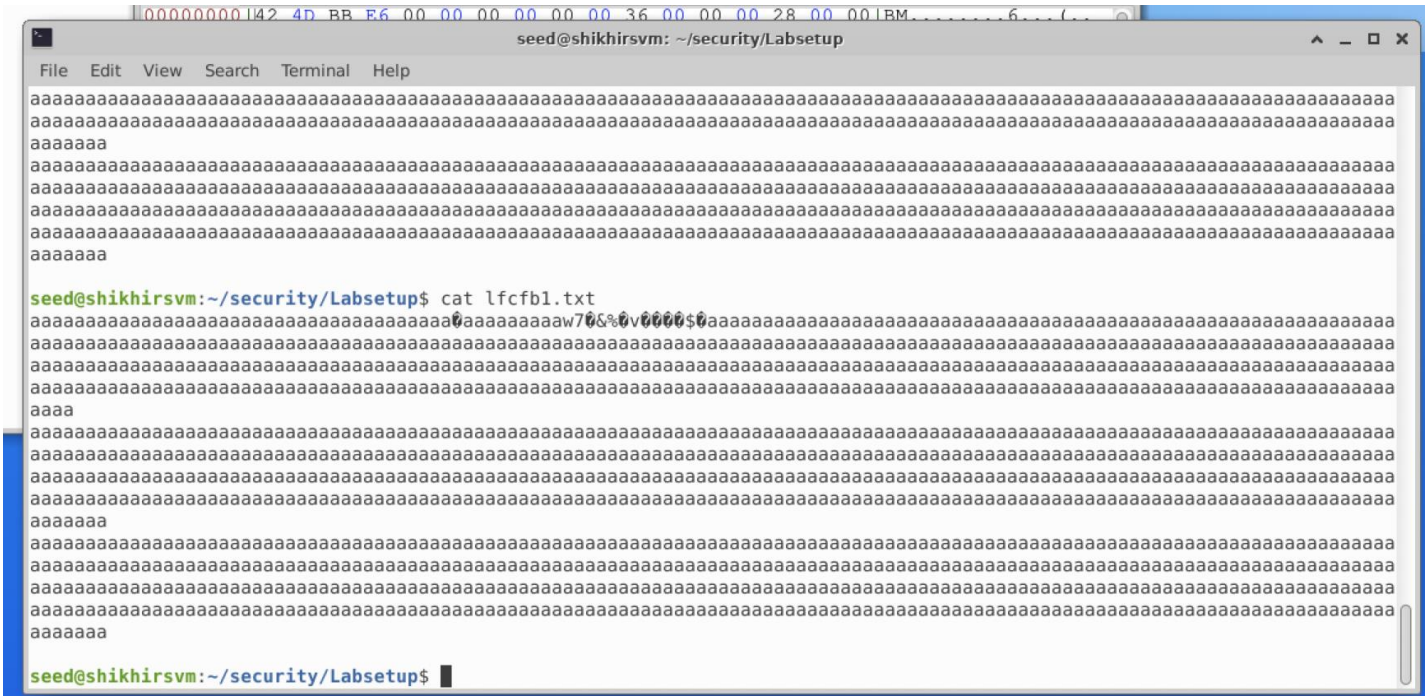




Decrypting all files using different modes

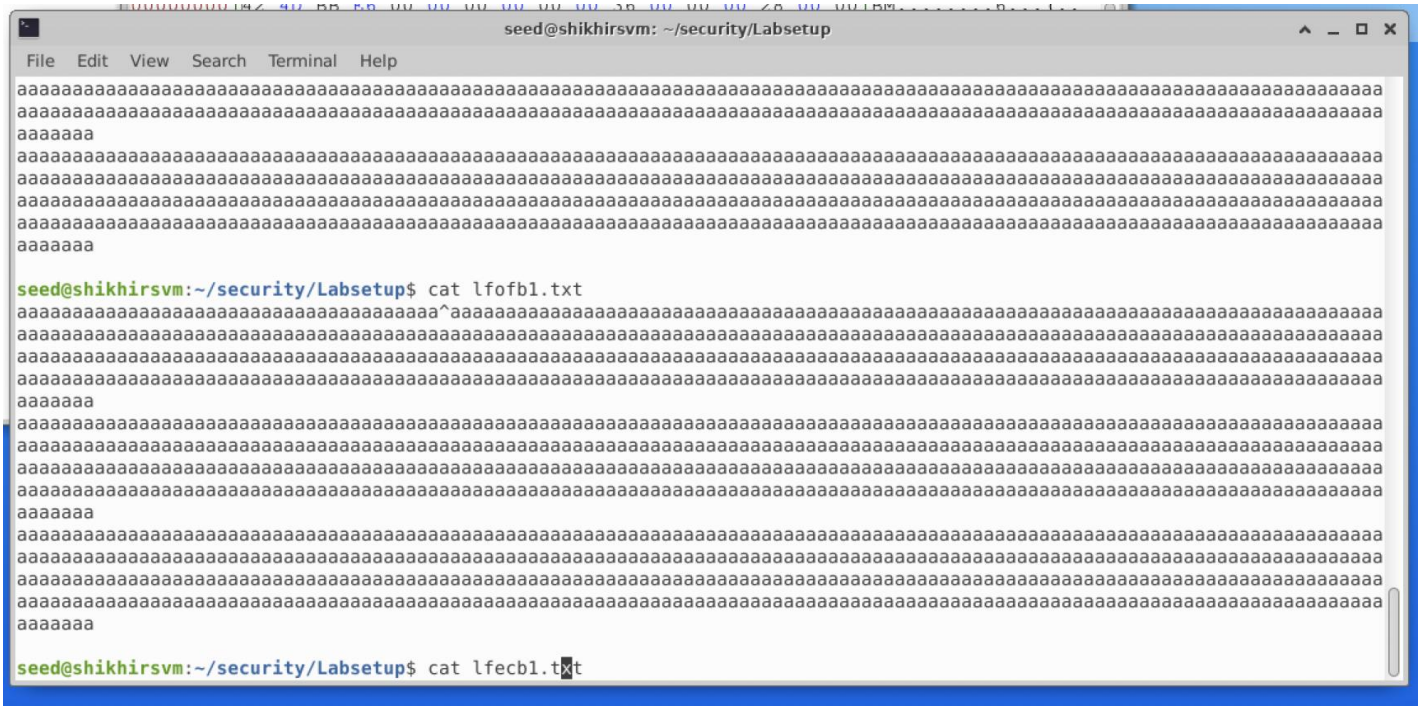
```
seed@shikhirsvm:~/security/Labsetup$ openssl enc -aes-128-ecb -d -in lfecb.bin -out lfecb1.txt -pass pass:123
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup$ openssl enc -aes-128-ofb -d -in lfofb.bin -out lfofb1.txt -pass pass:123
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup$ openssl enc -aes-128-cfb -d -in lfcfb.bin -out lfcfb1.txt -pass pass:123
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
seed@shikhirsvm:~/security/Labsetup$
```

Output for decrypted cfb file



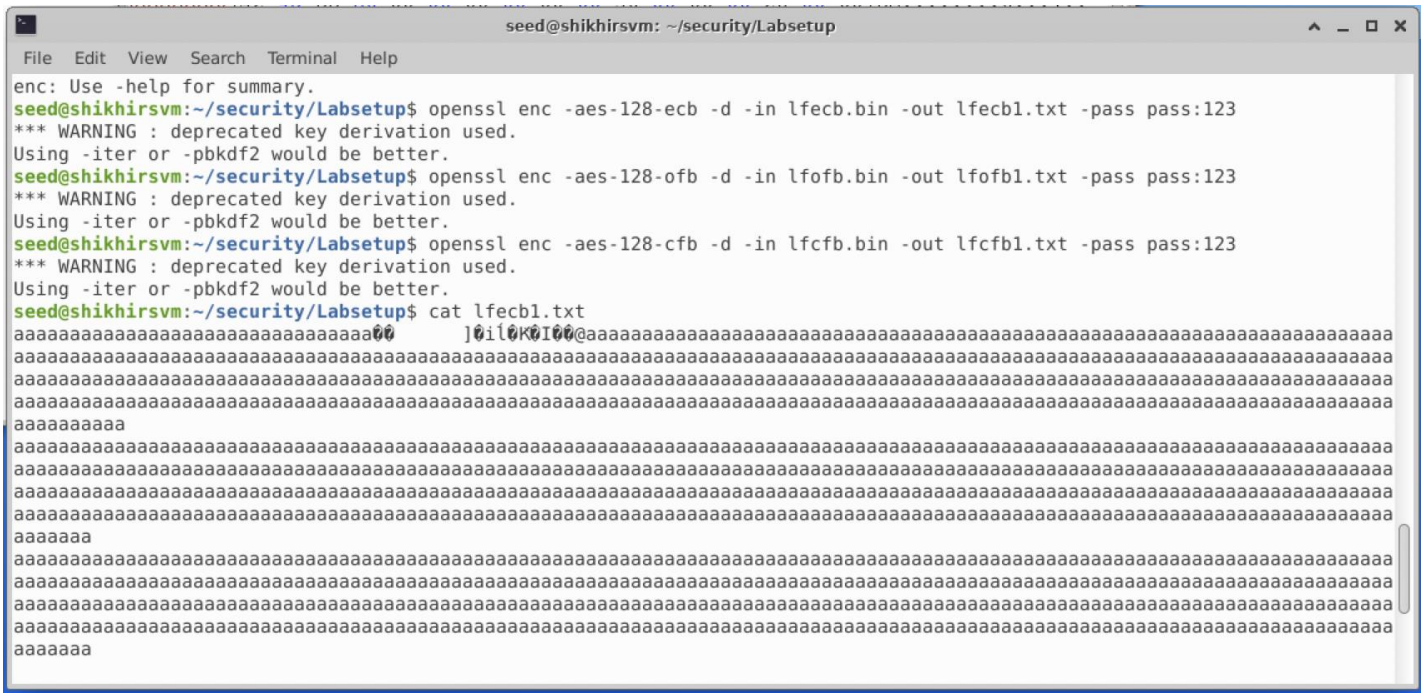
By changing the 55th byte, the decryption has significant changes. It completely distorts the text in a particular region and the transmitted file may not be readable.

Output for decrypted ofb file



By changing the 55th byte, the decryption has very minor changes. It distorts the text byte in a particular region.

Output for decrypted ecb file



By changing one byte, the decryption has very noticeable changes. It completely distorts the text in a particular region.