CS 458 LAB 2

Sahil Riyaz Sheikh

A20518693

Department of Computer ScienceIllinois
Institute of Technology

October 07, 2023

In this assignment I'll performing the following tasks:

- Frequency Analysis
- Encryption using different ciphers and modes
- Encryption Mode: ECB vs CBC
- Padding
- Error Propagation - Corrupted Cipher Text
- Initial Vector
  - Uniqueness of Initial vector
  - Common Mistake: Using same Initial Vector
  - Common Mistake: Make predictable Initial Vector

1. Frequency Analysis :

For the frequency analysis, I utilized the resources provided in the LAB. After performing frequency analysis, here are the results.

I compared the above results with pre-existing known frequencies

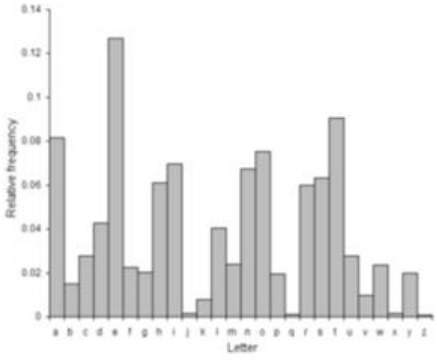
The frequency of the letters of the alphabet in English

The inventor of Morse code, Samuel Morse (1791-1872), needed to know this so that he could give the simplest codes to the most frequently used letters. He did it simply by counting the number of letters in sets of printers' type. The figures he came up with were:

12,000	E	2,500	F
9,000	T	2,000	W, Y
8,000	A, I, N, O, S	1,700	G, P
6,400	H	1,600	B
6,200	R	1,200	V
4,400	D	800	K
4,000	L	500	Q
3,400	U	400	J, X
3,000	C, M	200	Z

However, this gives the frequency of letters in English text, which is dominated by a relatively small number of common words . For word games, it is often the frequency of letters in English vocabulary, regardless of word frequency, which is of more interest. The following is a result of an analysis of the letters occurring in the words listed in the main entries of the ***Concise Oxford Dictionary*** (9th edition, 1995) and came up with the following table:

E	11.1607%	56.88	M	3.0129%	15.36
A	8.4966%	43.31	H	3.0034%	15.31
R	7.5809%	38.64	G	2.4705%	12.59
I	7.5448%	38.45	B	2.0720%	10.56
O	7.1635%	36.51	F	1.8121%	9.24
T	6.9509%	35.43	Y	1.7779%	9.06
N	6.6544%	33.92	W	1.2899%	6.57
S	5.7351%	29.23	K	1.1016%	5.61
L	5.4893%	27.98	V	1.0074%	5.13
C	4.5388%	23.13	X	0.2902%	1.48
U	3.6308%	18.51	Z	0.2722%	1.39
D	3.3844%	17.25	J	0.1965%	1.00
P	3.1671%	16.14	Q	0.1962%	(1)



Procedure I used to solve the problem:
So I made a error while doing this. Rather than replacing ‘polaz’ with ‘eario’ but I did the opposite. But I was able to observe “ya” converted to “yo”, which I believed could be “to” but I was not sure.

For out2 I replace ‘polaz’ with ‘eario’. Previously the “ya” terms were now “yi” and this made be sure that “a” should be replaced by “o”.

So for out3 I replaced “ya” to “to”, then I noticed “ho” repeating many times, I strongly believed “h” could be “s” to get the word “so”. I did this in out3 to get out3_1.

Now I tried to look back at the 4 outputs I had observed. I noticed my assumption for “ya” being “to” and “h” being “s” made sense for some words with length 2 but felt out of place in other words.

After studying Cipher text and out2 I was able to get a pattern. The word “oxp” was replaced to “axe” and I believed “x” has to be “r” to get the word “r” .

So I made a new file out4 in which I did the following operation: ‘pox’ -> ‘ear’
After doing so, some places ARE look perfect but as other it doesn’t fits well.

I’ll start from scratch again. I’ll only replace a single letter at a time. I believe “p” is “e” because of the frequency analysis results.

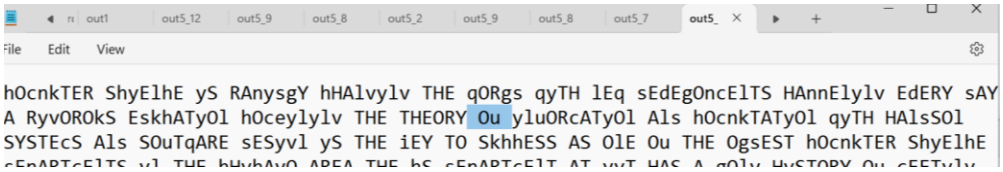
The letter with 2nd most frequency is “o”, I already tried replacing “o” with “a”. According to the known frequencies after “E”, “T” was the letter which was heavily used. In the research “T” occurred 9000 times, so I’ll replace “o” with “T”.

I can notice the “TxE” which could be “THE” so I’ll replace x with H now.
Now I notice the letter “z” is occurring as it self. The letter “a” is usually as it self for single letter word. I’ll replace “z” with “a”
Now as I observe further, I can see “HAa” and “Aa” coud be has and as respectively.
Now I noticed:

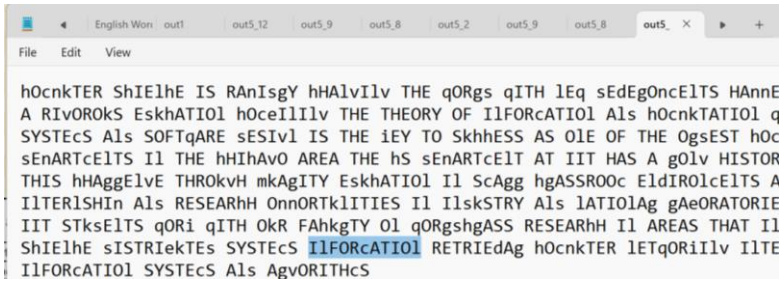
“Tf” , “yS” , “hS” , “AwEA”
Now Tf could be TO, this has good probability of happening
And AwEA could be AREA.

After doing so, I noticed “ THEORj” this has to be “THEORY”. Replacing “j” with “Y”

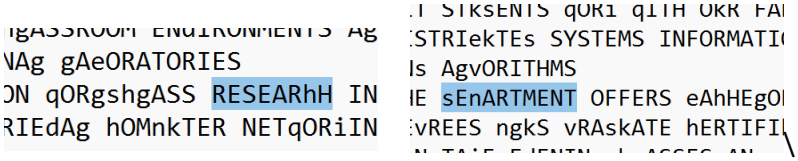
I noticed “HySTORY” which will be “HISTORY” and, “Ou” in many places which will be “OF”.



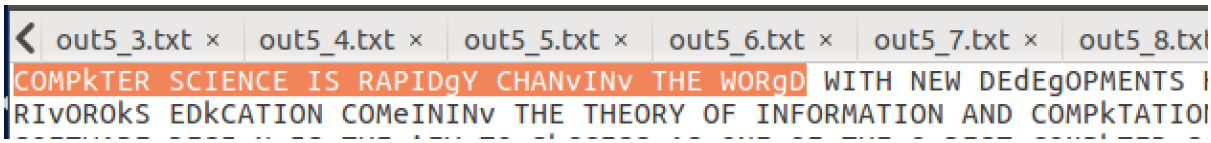
I identified two more possible words: IlFORcATIOl as INFORMATION. Replacing ‘lc’ ‘NM’



Noticed : “qITH” which will be “WITH” ,“RESEARhH” will be “RESEARCH”, “sEnARTMENT” will be “DEPARTMENT” . replacing ‘qhsn’ with ‘WCDP’



Can be Interpreted as COMPUTER SCIENCE IS RAPIDLY CHANGING
Replacing ‘kgvd’ with ‘ULGV’



Noticed:

will be KEY

will be BACHELOR

will be QUALITY

Replacing ‘iem’ ‘KBQ’

Noticed

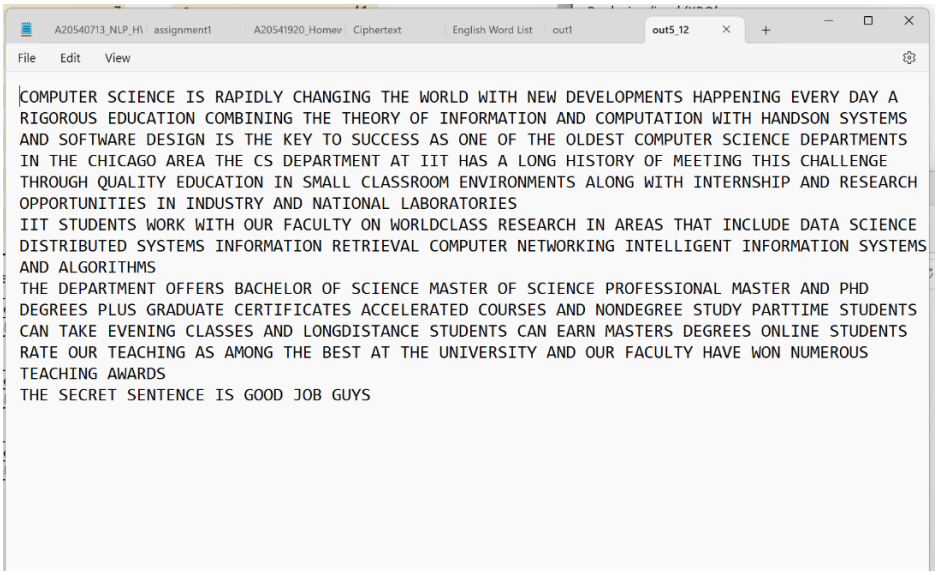
will be JOB

Successfully encrypted the msg.

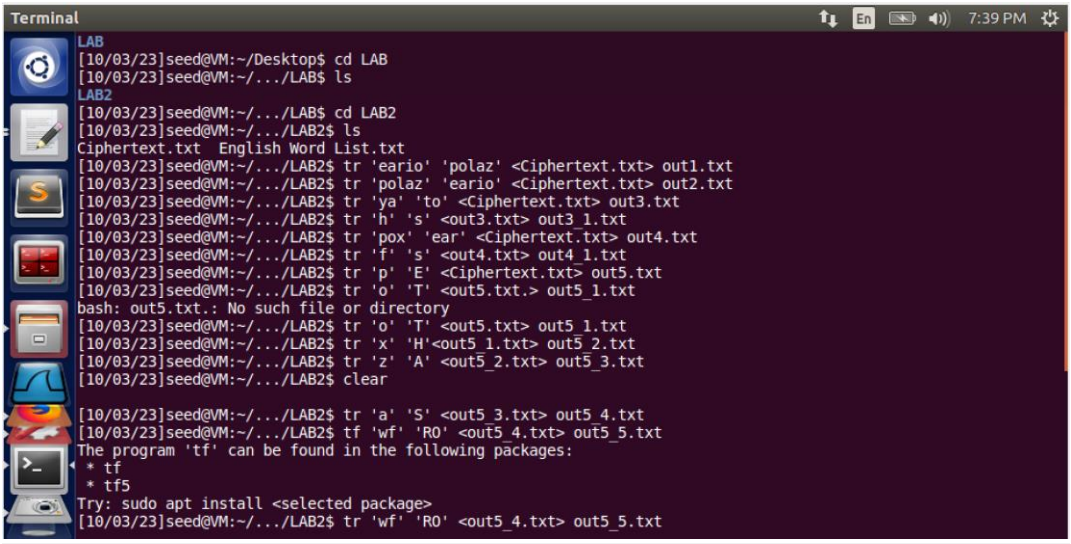
ENC_LETTER	A	B	C	D	E	F	G	H	I
ACTUAL_LETTER	S	J	M	V	B	O	L	C	K

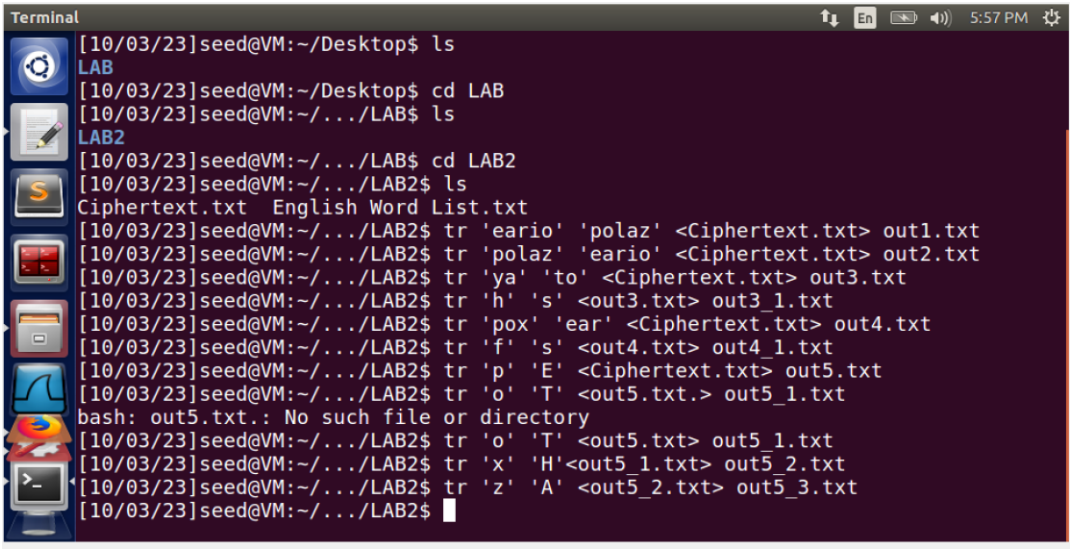
J	K	L	M	N	O	P	Q	R
Y	U	N	Q	P	T	E	W	Z

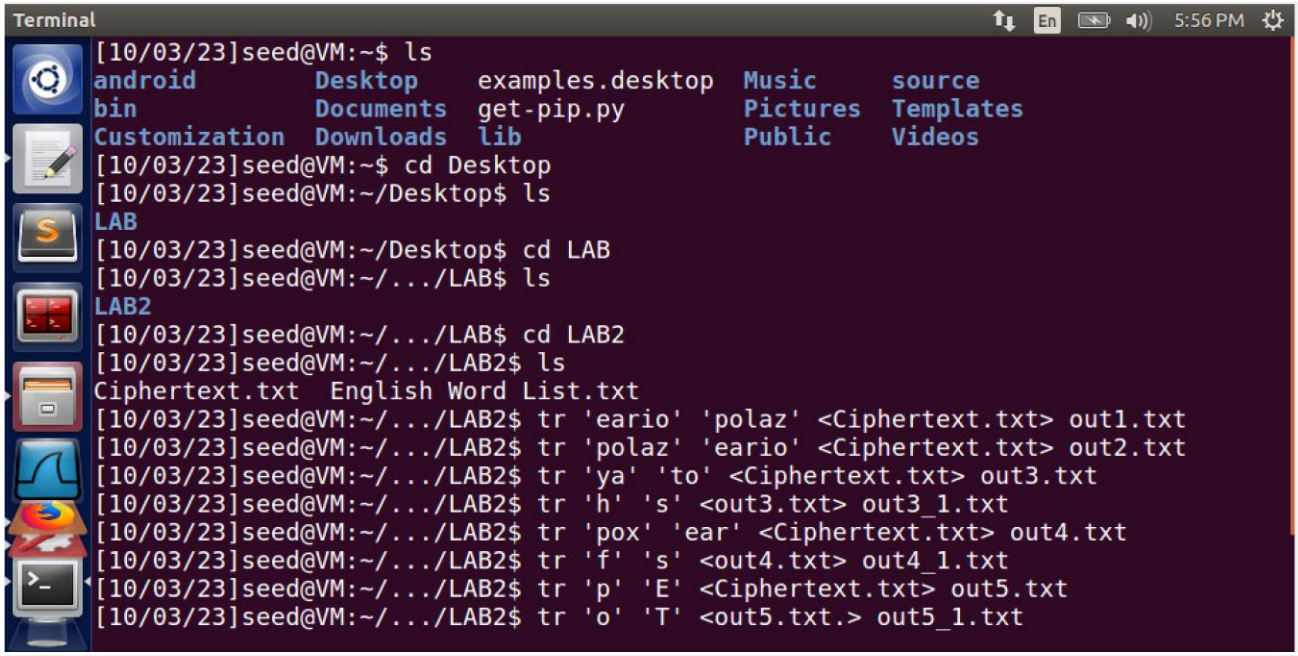
S	T	U	V	W	X	Y	Z
D	X	F	G	R	H	I	A

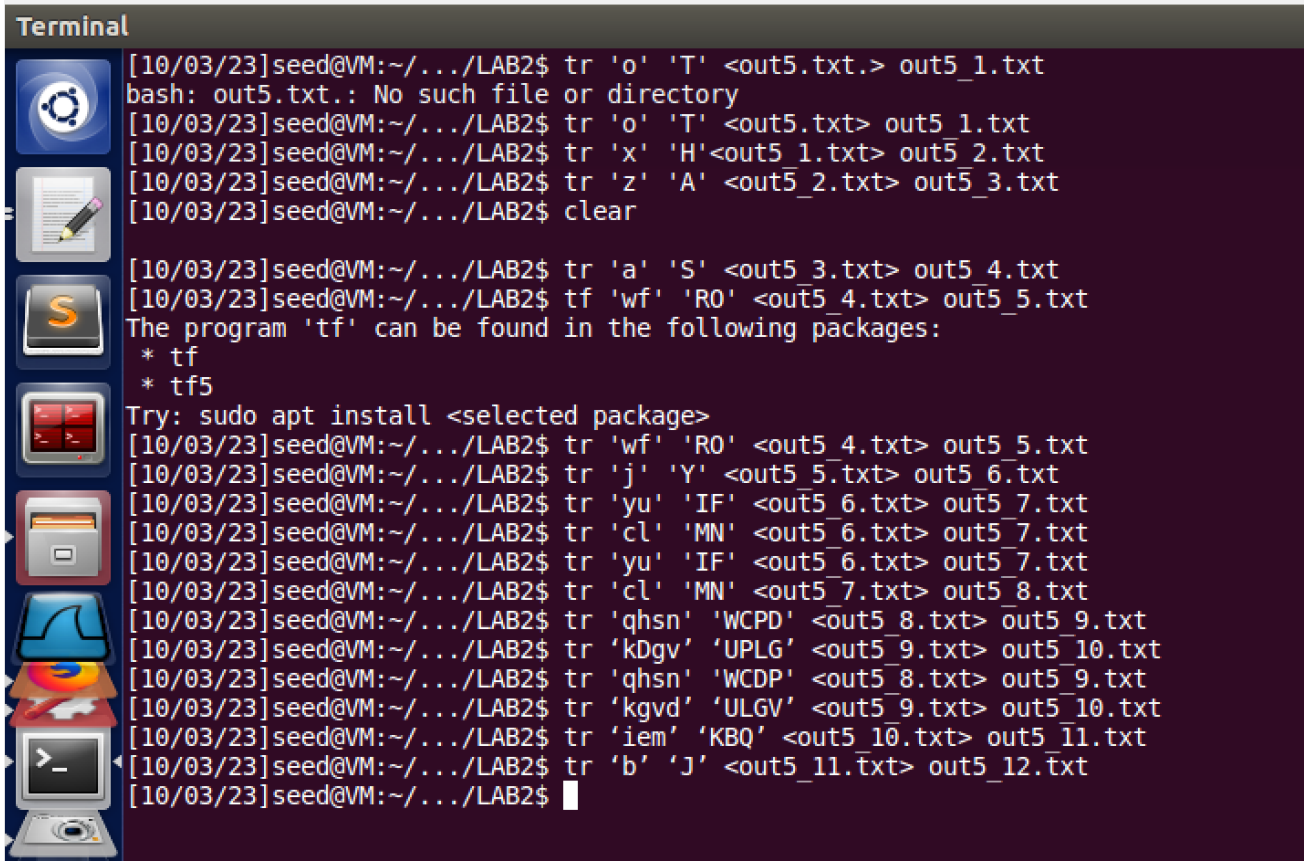


I have attached snippets of the code executed by me below. I have also submitted all the files which were created while trying to decrypt the message.









The final output is stored in file out5_12.txt

2. Encryption and Decryption using Different Cipher and Modes:

Running command man enc to get information about different types of ciphers and their corresponding modes:

```
aes-[128|192|256]-cbc  128/192/256 bit AES in CBC mode
aes-[128|192|256]      Alias for aes-[128|192|256]-cbc
aes-[128|192|256]-cfb  128/192/256 bit AES in 128 bit CFB mode
aes-[128|192|256]-cfb1 128/192/256 bit AES in 1 bit CFB mode
aes-[128|192|256]-cfb8 128/192/256 bit AES in 8 bit CFB mode
aes-[128|192|256]-ecb  128/192/256 bit AES in ECB mode
aes-[128|192|256]-ofb  128/192/256 bit AES in OFB mode
```

```
rc2-cbc             128 bit RC2 in CBC mode
rc2                 Alias for rc2-cbc
rc2-cfb             128 bit RC2 in CFB mode
rc2-ecb             128 bit RC2 in ECB mode
rc2-ofb             128 bit RC2 in OFB mode
rc2-64-cbc          64 bit RC2 in CBC mode
rc2-40-cbc          40 bit RC2 in CBC mode
```

```
  bf-cbc                Blowfish in CBC mode
  bf                    Alias for bf-cbc
  bf-cfb                Blowfish in CFB mode
  bf-ecb                Blowfish in ECB mode
  bf-ofb                Blowfish in OFB mode
```

For every cipher and corresponding mode I have implemented the following steps:

a. Creating a plain txt file
b. Using openssl to encrypt the said file
c. Using cat to print the .bin file created after encrypting
d. Using xxd command to observe encrypted contents of said .bin file in hexadecimal format

AES-128-CBC

```
bash: plain.txt: No such file or directory
[10/04/23]seed@VM:~/.../Q2$ clear
[10/04/23]seed@VM:~/.../Q2$ echo "This is my first text file" > plain.txt
[10/04/23]seed@VM:~/.../Q2$ ls
pic_original.bmp  plain.txt
[10/04/23]seed@VM:~/.../Q2$ man enc
[10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.bin -k 000102030405060708090aabbccddeeff
[10/04/23]seed@VM:~/.../Q2$ ls
cipher1.bin  pic_original.bmp  plain.txt
[10/04/23]seed@VM:~/.../Q2$ cat cipher1.bin
Salted__[illegible]
[10/04/23]seed@VM:~/.../Q2$ xxd cipher1.bin
00000000: 5361 6c74 6564 5f5f b7e5 cca6 1cc7 23f9  Salted__......#.
00000010: 6f55 f924 db5c 18be 1c7d 2e0a 2860 9576  oU.$.\...}..(`.v
00000020: dc44 aa19 55f3 da51 9498 52ea bb00 aa0a  .D..U..Q..R.....
[10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-cbc -d -in cipher1.bin -k 000102030405060708090aabbccddeeff
This is my first text file
[10/04/23]seed@VM:~/.../Q2$
```

AES-128-CFB

```
10/04/23]seed@VM:~/.../Q2$ echo "AES-128-CFB Decryption" > plain2.txt
10/04/23]seed@VM:~/.../Q2$ ls
ipher1.bin  pic_original.bmp  plain2.txt  plain.txt
10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-cfb -e -in plain2.txt -out cipher2.bin -k 000102030405060708090aabbccddeeff
10/04/23]seed@VM:~/.../Q2$ cat cipher2.bin
alted__s6H~[illegible][10/04/23]seed@VM:~/.../Q2$
10/04/23]seed@VM:~/.../Q2$ xxd cipher2.bin
00000000: 5361 6c74 6564 5f5f 7336 487e a8e0 8855  Salted__s6H~...U
00000010: af1d 35f3 cb00 a7cd 8f67 08b5 65bd ba7f  ..5......g..e...
00000020: 43c2 e363 cc4e c5                        C..c.N.
10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-cfb -d -in cipher2.bin -k 000102030405060708090aabbccddeeff
ES-128-CFB Decryption
10/04/23]seed@VM:~/.../Q2$
```

AES-128-OFB

```
[10/04/23]seed@VM:~/.../Q2$ echo "AES-128-OFB Decryption" > plain3.txt
[10/04/23]seed@VM:~/.../Q2$ ls
cipher1.bin  pic_original.bmp  plain3.txt
cipher2.bin  plain2.txt        plain.txt
[10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-ofb -e -in plain3.txt -out cipher3.bin -k 00010203040506070809aabbccddeeff
[10/04/23]seed@VM:~/.../Q2$ cat cipher3.bin
Salted__[illegible]
[illegible]Ux[illegible]Th[10/04/23]seed@VM:~/.../Q2$ xxd cipher3.bin
00000000: 5361 6c74 6564 5f5f dd82 c8f0 eddf e5d0  Salted__........
00000010: b97e 777e af0b d7f3 9907 2b55 06ee dc09  .~w~......+U....
00000020: 18dd 5578 8454 68                        ..Ux.Th
[10/04/23]seed@VM:~/.../Q2$ openssl enc -aes-128-ofb -d -in cipher3.bin -k 00010203040506070809aabbccddeeff
AES-128-OFB Decryption
[10/04/23]seed@VM:~/.../Q2$
```

RC2-CBC

```
Terminal
Terminal
[10/06/23]seed@VM:~$ cd Desktop
[10/06/23]seed@VM:~/Desktop$ cd LAB
[10/06/23]seed@VM:~/.../LAB$ cd LAB2
[10/06/23]seed@VM:~/.../LAB2$ cd Q2
[10/06/23]seed@VM:~/.../Q2$ ls
cipher1.bin  cipher4.bin  cipher7.bin  plain2.txt  plain5.txt  plain8.txt
cipher2.bin  cipher5.bin  cipher8.bin  plain3.txt  plain6.txt  plain9.txt
cipher3.bin  cipher6.bin  cipher9.bin  plain4.txt  plain7.txt  plain.txt
[10/06/23]seed@VM:~/.../Q2$ echo " RC2 - CBC Decryption" > plain.txt
[10/06/23]seed@VM:~/.../Q2$ openssl enc -rc2-cbc -e -in plain.txt -out cip.bin -
K 00010203040506070809aabbccddeeff -iv 0a0b0c0d0e0f010203040506070809
[10/06/23]seed@VM:~/.../Q2$ cat cip.bin
[illegible]J[illegible]r[illegible]w[illegible]f[illegible]avV[illegible]x[illegible]m[10/06/23]seed@VM:~/.../Q2$ xxd cip.bin
00000000: a5f5 1ba2 4afa 8003 72b0 ed77 178e bd66  ....J...r..w...f
00000010: ea61 7656 8d78 886d                      .avV.x.m
[10/06/23]seed@VM:~/.../Q2$ openssl enc -rc2-cbc -d -in cip.bin -K 0001020304050
6070809aabbccddeeff -iv 0a0b0c0d0e0f010203040506070809
 RC2 - CBC Decryption
[10/06/23]seed@VM:~/.../Q2$
```

RC2-CFB

```
[10/04/23]seed@VM:~/.../Q2$ echo "RC2-CFB Decryption" > plain5.txt
[10/04/23]seed@VM:~/.../Q2$ openssl enc -rc2-cfb -e -in plain5.txt -out cipher5.bin -k 00010203040506070809aabbccddeeff
[10/04/23]seed@VM:~/.../Q2$ cat cipher5.bin
Salted__ak[illegible](1"[illegible]fw[illegible]K5}[illegible]~3^[10/04/23]seed@VM:~/.../Q2$ xxd cipher5.bin
00000000: 5361 6c74 6564 5f5f 616b abb2 bfb7 2831  Salted__ak....(1
00000010: 22a5 e026 bf66 5790 a34b 357d c22a 0f8c  "..&.fW..K5}.*..
00000020: 7e33 5e                                  ~3^
[10/04/23]seed@VM:~/.../Q2$ openssl enc -rc2-cfb -d -in cipher5.bin
enter rc2-cfb decryption password:
RC2-CFB Decryption
[10/04/23]seed@VM:~/.../Q2$
```

RC2-OFB

```
[10/04/23]seed@VM:~/.../Q2$ echo "RC2-OFB Decryption" > plain6.txt
[10/05/23]seed@VM:~/.../Q2$ openssl enc -rc2-ofb -e -in plain6.txt -out cipher6.bin -k 00010203040506070809aabbccddeeff
[10/05/23]seed@VM:~/.../Q2$ cat cipher6.txt
cat: cipher6.txt: No such file or directory
[10/05/23]seed@VM:~/.../Q2$ cat cipher6.bin
Salted__D+[illegible]B*[illegible]q[illegible]/[illegible]2L%[illegible]
                        e4[illegible]cC[illegible][10/05/23]seed@VM:~/.../Q2$ xxd cipher6.bin
00000000: 5361 6c74 6564 5f5f 442b cf01 55c9 422a  Salted__D+..U.B*
00000010: 97c7 718f b42f 9b32 4c25 960b 6534 c263  ..q../.2L%..e4.c
00000020: 4301 47                                  C.G
[10/05/23]seed@VM:~/.../Q2$ openssl enc -rc2-ofb -in cipher6.bin -k 00010203040506070809aabbccddeeff
Salted__[illegible]g $<OY[illegible]Pw[illegible]q
The program 'q' can be found in the following packages:
 * python-q-text-as-data
 * python3-q-text-as-data
Try: sudo apt install <selected package>
[10/05/23]seed@VM:~/.../Q2$ openssl enc -rc2-ofb -d -in cipher6.bin -k 00010203040506070809aabbccddeeff
RC2-OFB Decryption
[10/05/23]seed@VM:~/.../Q2$
```

BF-CBC

```
[10/05/23]seed@VM:~/.../Q2$ echo "BF-CBC Decryption" > plain7.txt
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-cbc -e -in plain7.txt -out cipher7.bin -k 00010203040506070809aabbccddeeff
[10/05/23]seed@VM:~/.../Q2$ cat cipher7.bin
Salted__C[illegible]
[10/05/23]seed@VM:~/.../Q2$ xxd cipher7.bin
00000000: 5361 6c74 6564 5f5f 43db 2092 409d 9cb2  Salted__C. .@...
00000010: f696 5385 8097 80e9 7598 b0ce c63e 41c5  ..S.....u....>A.
00000020: 56c4 0ce1 0ad0 fb6c                      V......l
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-cbc -d -in cipher7.bin -k 00010203040506070809aabbccddeeff
BF-CBC Decryption
[10/05/23]seed@VM:~/.../Q2$
```

BF-CFB

```
[10/05/23]seed@VM:~/.../Q2$ echo "BF-CFB Decryption" > plain8.txt
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-cfb -e -in plain8.txt -out cipher8.bin -k 00010203040506070809aabbccddeeff
[10/05/23]seed@VM:~/.../Q2$ cat cipher8.bin
Salted__[illegible]10/05/23]seed@VM:~/.../Q2$ xxd cipher8.bin
00000000: 5361 6c74 6564 5f5f b924 d1b7 a516 04d8  Salted__.$......
00000010: b5e6 ce9e 4771 3f10 09e4 c8a3 6450 1519  ....Gq?.....dP..
00000020: ae15                                     ..
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-cfb -d -in cipher8.bin -k 00010203040506070809aabbccddeeff
BF-CFB Decryption
[10/05/23]seed@VM:~/.../Q2$
```

BF-OFB

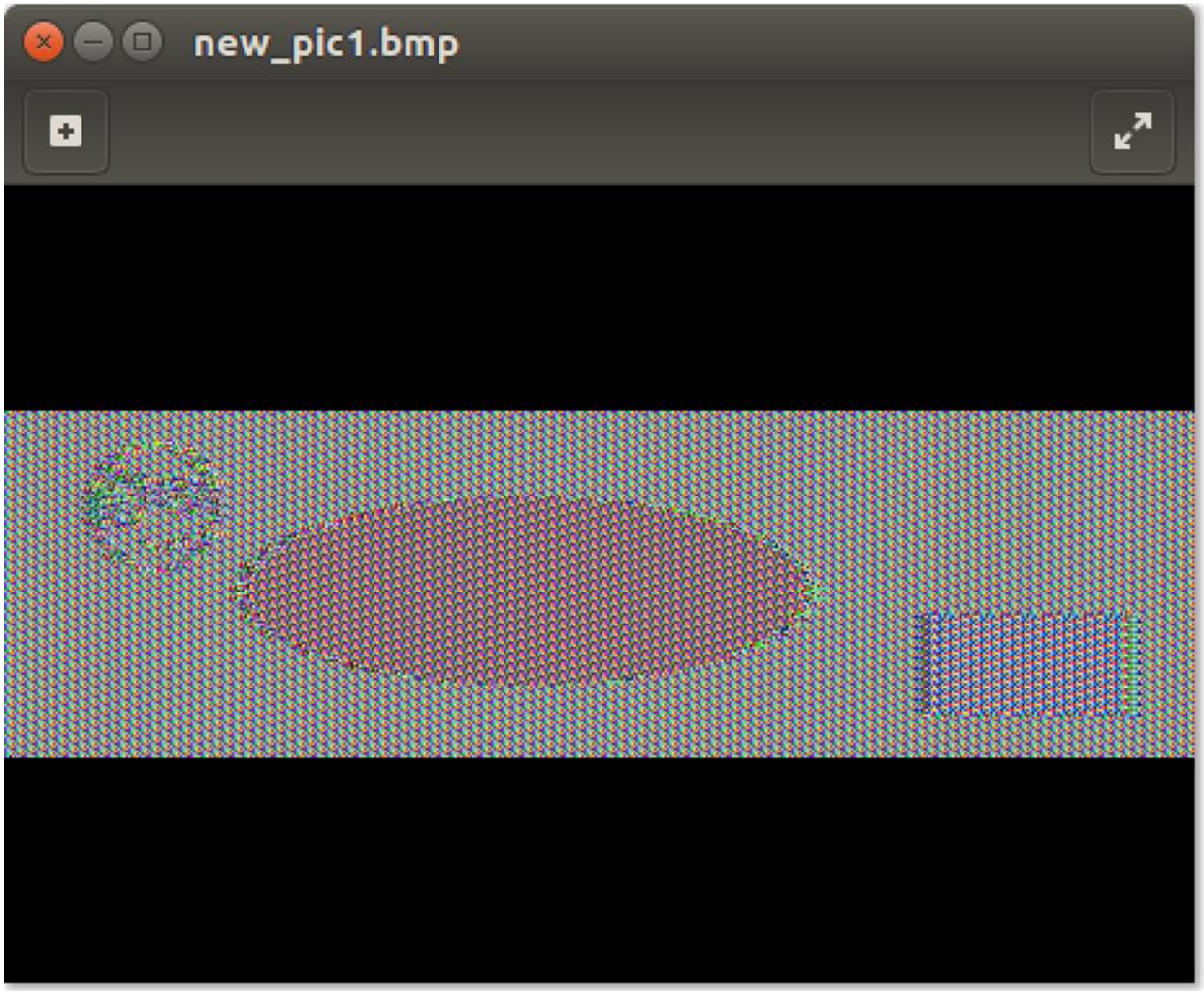
```
[10/05/23]seed@VM:~/.../Q2$ echo "BR-OFB Decryption" > plain9.txt
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-ofb -e -in plain9.txt -out cipher9.bin -k 00010203040506070809aabbccddeeff
[10/05/23]seed@VM:~/.../Q2$ cat cipher9.bin
Salted__[illegible][10/05/23]seed@VM:~/.../Q2$ xxd cipher9.bin
00000000: 5361 6c74 6564 5f5f 17a2 41e0 46d9 7d45  Salted__..A.F.}E
00000010: 6780 a290 87be 2ae2 1ec7 d3dc 70f6 5c88  g.....*.....p.\.
00000020: bedc                                     ..
[10/05/23]seed@VM:~/.../Q2$ openssl enc -bf-ofb -d -in cipher9.bin -k 00010203040506070809aabbccddeeff
BR-OFB Decryption
[10/05/23]seed@VM:~/.../Q2$
```

3. Encryption mode: ECB VS CBC

The given image is encrypted using both aes-128 cipher with ecb mode and cbc mode. Ther results are as follows:

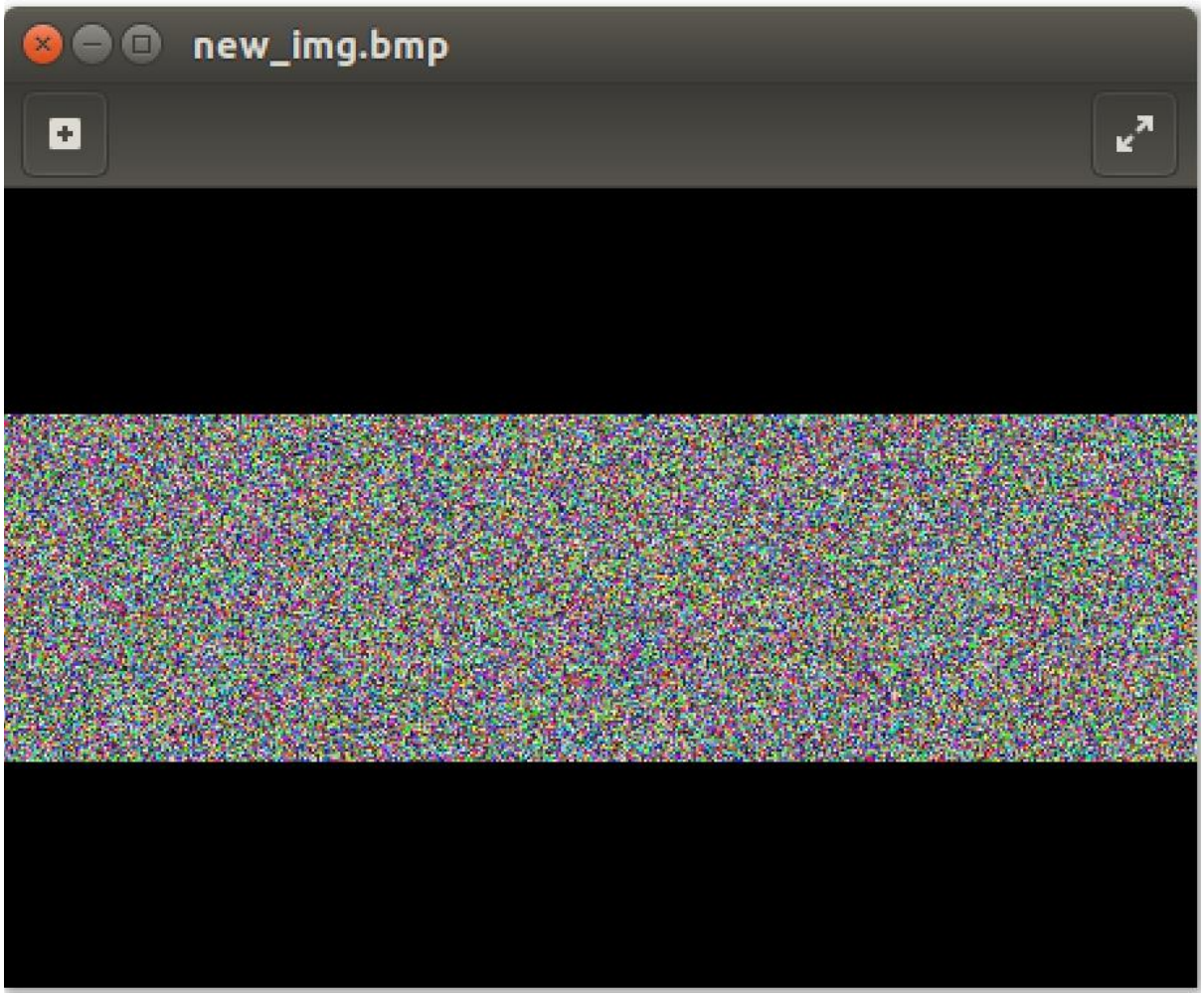
AES-128-ECB Mode

```
[10/06/23]seed@VM:~/.../Q3$ ls
pic_original.bmp
[10/06/23]seed@VM:~/.../Q3$ openssl enc -aes-128-ecb -in pic_original.bmp -out enc_img.bmp -K 696969
[10/06/23]seed@VM:~/.../Q3$ tail -c +55 enc_img.bmp > body
[10/06/23]seed@VM:~/.../Q3$ head -c +54 pic_original.bmp > head
[10/06/23]seed@VM:~/.../Q3$ cat head body > new_pic1.bmp
[10/06/23]seed@VM:~/.../Q3$ ls
body  enc_img.bmp  head  new_pic1.bmp  pic_original.bmp
[10/06/23]seed@VM:~/.../Q3$
```

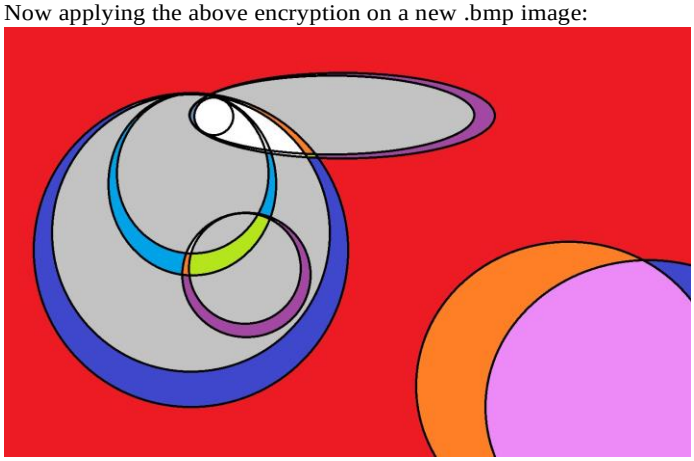


AES-128-CBC Mode

```
[10/06/23]seed@VM:~/.../Q3$ openssl enc -aes-128-cbc -in pic_original[10/06/23]seed@VM:~/.../Q3$ openssl enc -aes-128-cbc -in pic_original.bmp -out en
c_img.bmp -K 000102030405060708090aabbccddeeff -iv 0a0b0c0d0e0f010203040506070809
[10/06/23]seed@VM:~/.../Q3$ head -c +54 pic_original.bmp > h1
[10/06/23]seed@VM:~/.../Q3$ tail -c +55 enc_img.bmp > b1
[10/06/23]seed@VM:~/.../Q3$ cat h1 b1 > new_img.bmp
[10/06/23]seed@VM:~/.../Q3$
```

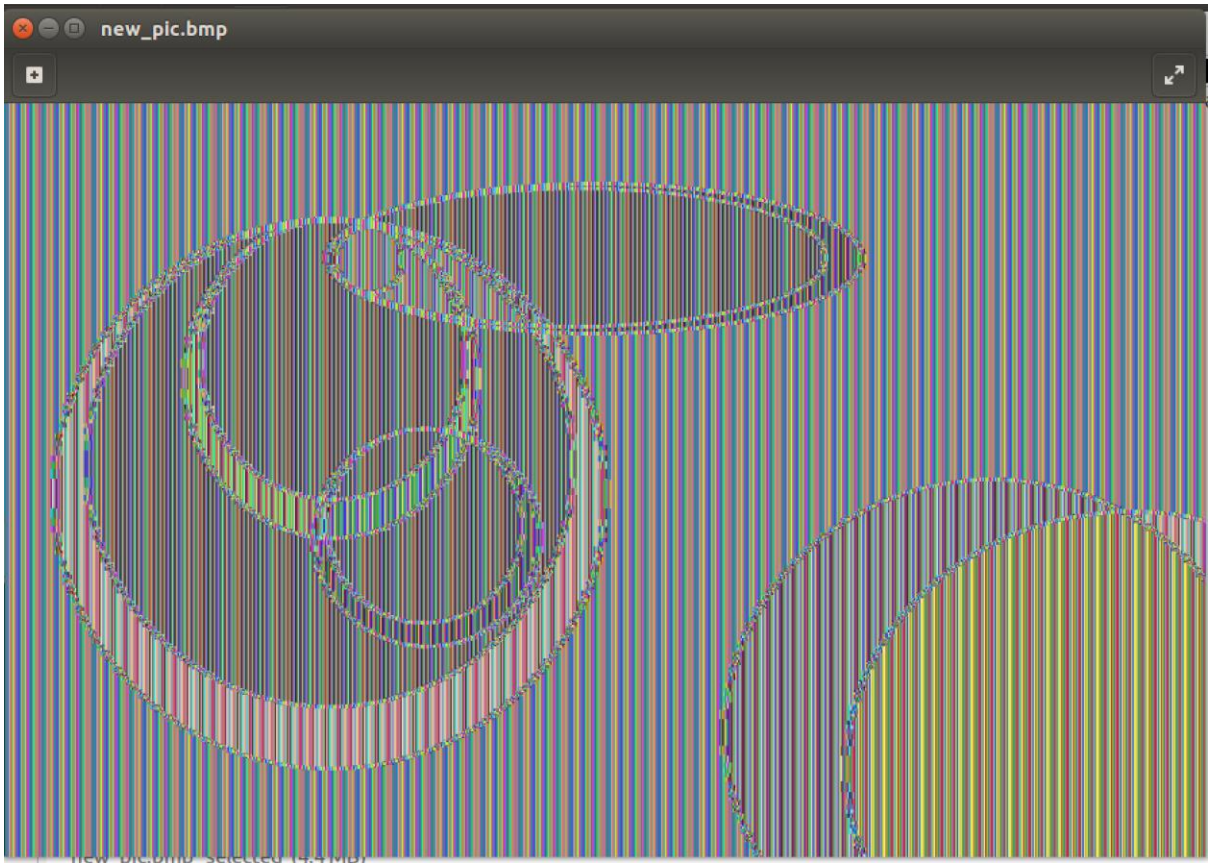


Now applying the above encryption on a new .bmp image:



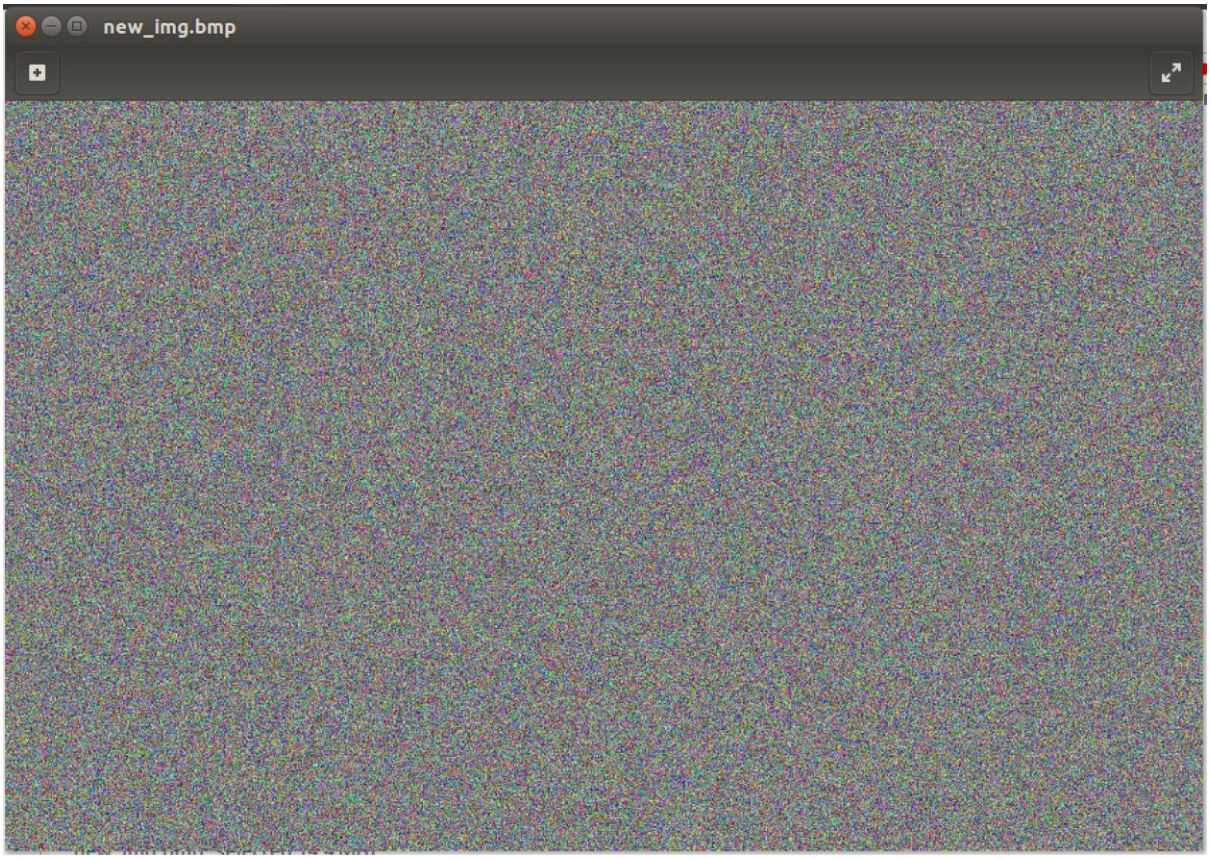
AES-128-ECB Mode

```
[10/06/23]seed@VM:~/.../Q3$ ls
body  enc_img.bmp  head  new_pic1.bmp  pic.bmp  pic_original.bmp
[10/06/23]seed@VM:~/.../Q3$ openssl enc -aes-128-ecb -e -in pic.bmp -out enc_p.bmp -K 696969
[10/06/23]seed@VM:~/.../Q3$ h -c +54 pic.bmp > h
h: command not found
[10/06/23]seed@VM:~/.../Q3$ head -c +54 pic.bmp > h
[10/06/23]seed@VM:~/.../Q3$ tail -c +55 enc_p.bmp > b
[10/06/23]seed@VM:~/.../Q3$ cat h b > new_pic.bmp
[10/06/23]seed@VM:~/.../Q3$
```



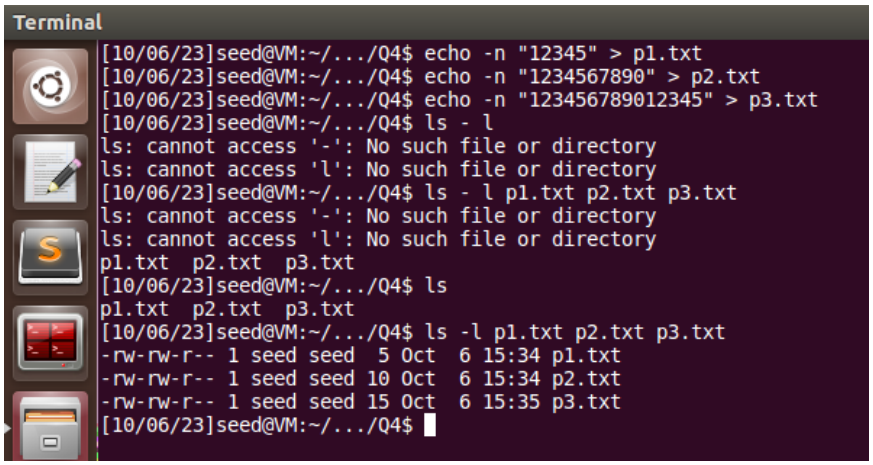
AES-128-CBC Mode

```
[10/06/23]seed@VM:~/.../Q3$ openssl enc -aes-128-cbc -in pic.bmp -out enc_img.bmp -K 00010203040506070809aabbccddeeff -iv 0a0b0c0d0e0f0102030405060708
09
[10/06/23]seed@VM:~/.../Q3$ head -c +54 pic.bmp > h1
[10/06/23]seed@VM:~/.../Q3$ tail -c +55 enc_img.bmp > b1
[10/06/23]seed@VM:~/.../Q3$ cat h1 b1 > new_img.bmp
[10/06/23]seed@VM:~/.../Q3$
```



4. Padding:

Created three files of size 5,10,15 bytes each. I selected AES-192 selection for this task.



Now I have applied the same procedure for each mode of cipher on 3 files.

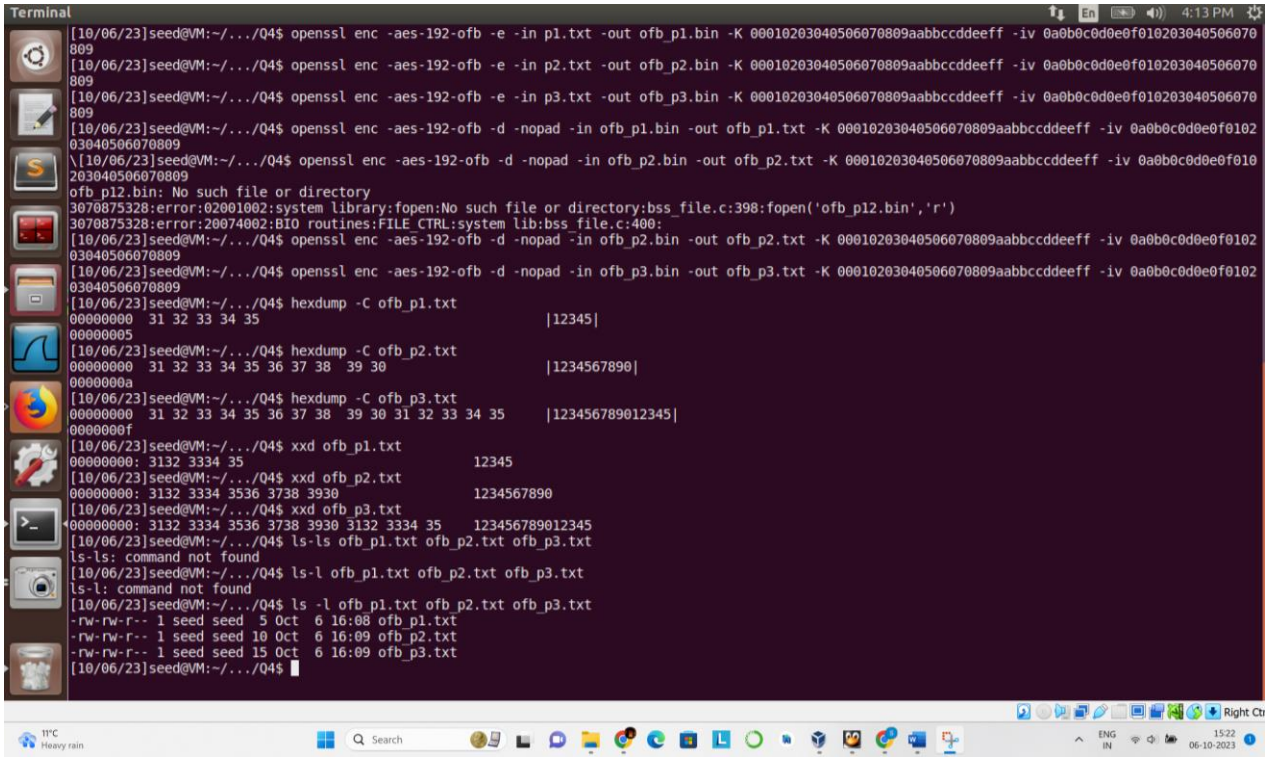
a. First, I encrypted all files with the selected cipher with required key and iv
b. Then I decrypted the said encrypted files
c. I printed the decrypted txt file in hex format using hexdump and xxd command
d. In the end I compared and printed the size of all the decrypted txt files

AES-192-CFB

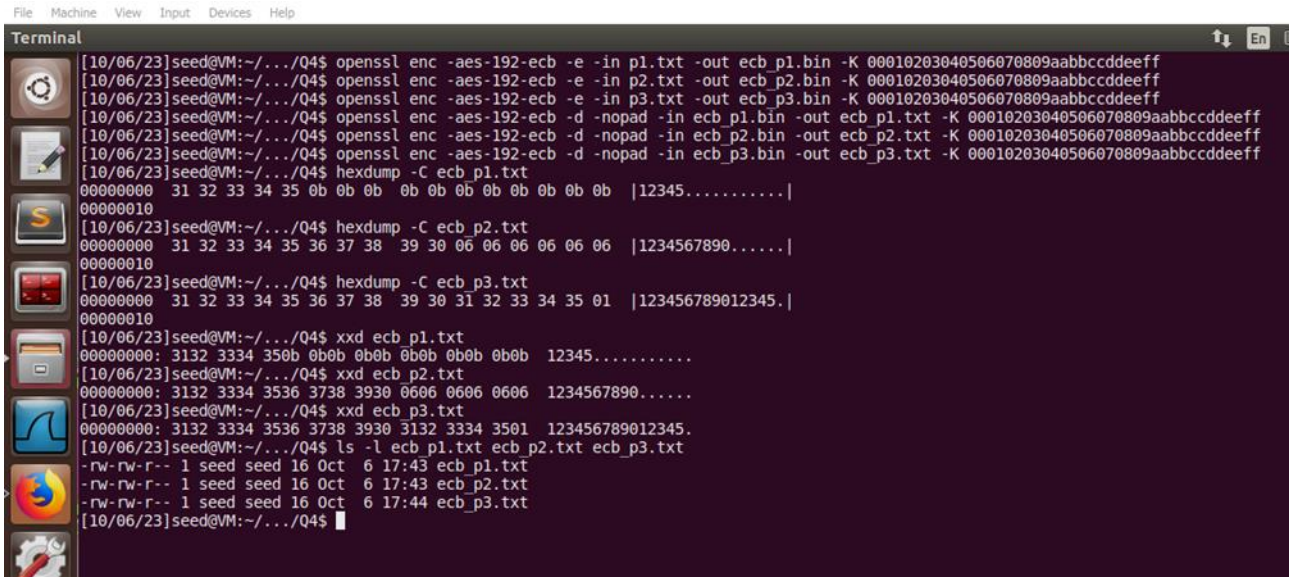
```
[10/06/23]seed@VM:~/.../Q4$ clear

[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -e -in p1.txt -out cfb_p1.bin -K 00010203040506070809
809
[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -e -in p2.txt -out cfb_p2.bin -K 00010203040506070809
809
[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -e -in p3.txt -out cfb_p3.bin -K 00010203040506070809
809
[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -d -nopad -in cfb_p1.bin -out cfb_p1.txt -K 000102030
03040506070809
[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -d -nopad -in cfb_p2.bin -out cfb_p2.txt -K 000102030
03040506070809
[10/06/23]seed@VM:~/.../Q4$ openssl enc -aes-192-cfb -d -nopad -in cfb_p3.bin -out cfb_p3.txt -K 000102030
03040506070809
[10/06/23]seed@VM:~/.../Q4$ hexdump -C cfb_p1.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[10/06/23]seed@VM:~/.../Q4$ hexdump -C cfb_p2.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[10/06/23]seed@VM:~/.../Q4$ hexdump -C cfb_p3.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35     |123456789012345|
0000000f
[10/06/23]seed@VM:~/.../Q4$ xxd cfb_p1.txt
00000000: 3132 3334 35                             12345
[10/06/23]seed@VM:~/.../Q4$ xxd cfb_p2.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
```

```
[10/06/23]seed@VM:~/.../Q4$ xxd cfb_p3.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 35    123456789012345
[10/06/23]seed@VM:~/.../Q4$ ls -s cfb_p1.txt cfb_p2.txt cfb_p3.txt
4 cfb_p1.txt  4 cfb_p2.txt  4 cfb_p3.txt
[10/06/23]seed@VM:~/.../Q4$ ls -l cfb_p1.txt cfb_p2.txt cfb_p3.txt
-rw-rw-r-- 1 seed seed  5 Oct  6 15:53 cfb_p1.txt
-rw-rw-r-- 1 seed seed 10 Oct  6 15:53 cfb_p2.txt
-rw-rw-r-- 1 seed seed 15 Oct  6 15:53 cfb_p3.txt
[10/06/23]seed@VM:~/.../Q4$
```

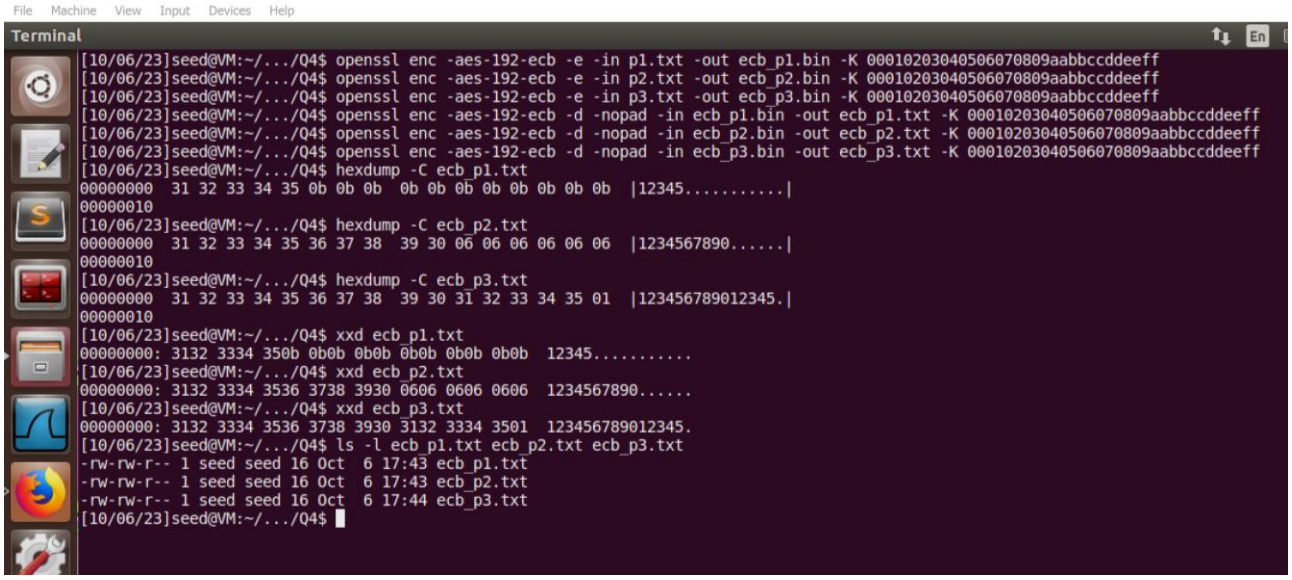
AES-192-OFB



AES-192-ECB



AES-192-CBC



I observed that, ECB and CBC require padding and, CFB and OFB don't require padding. I can say that as the byte size for CFB and OFB remains same, whereas for ECB and CBC the files size increases to 16 bytes for every file. The OFB and CFB methods create a secret code to scramble each bit/byte of original message. This secret code is inserted the original message to create encrypted text.

5. Error Propagation : Corrupted Cipher text

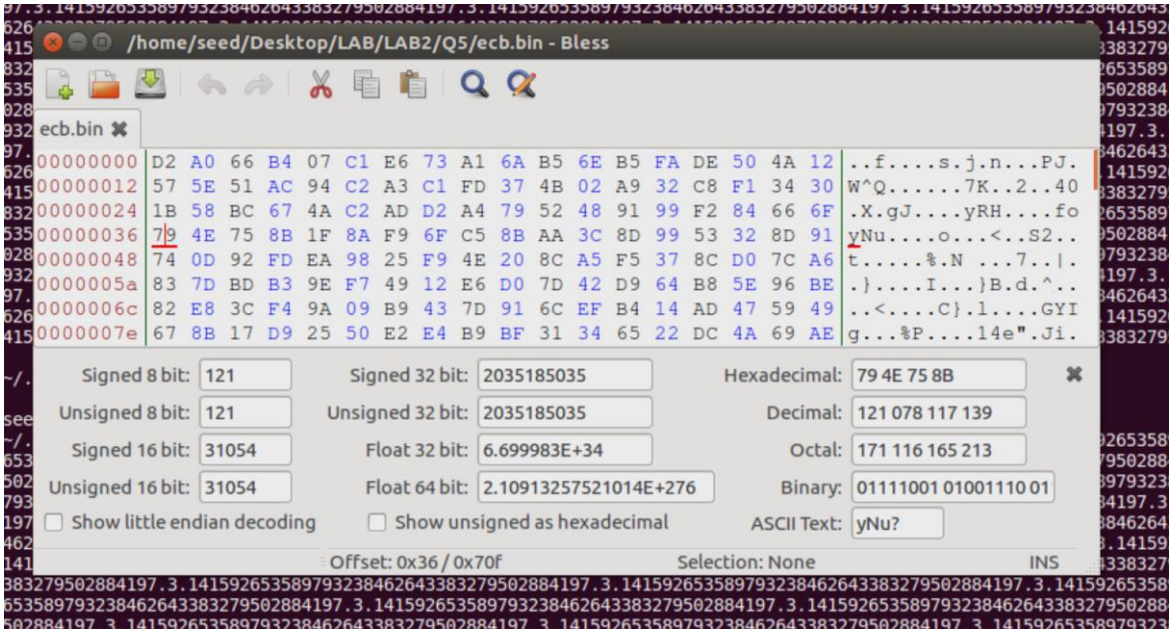
For this task I created a file with more than thousand bytes. I simply created a file with reoccurring values of pi (3.14...). The file ended being 1807 bytes or 1.8Kb. after this I encrypted the file using AAES-128 encryption with various modes, I'll corrupt the 55th byte of this encrypted file. Then I decrypted these encrypted .bin files to get .txt files.

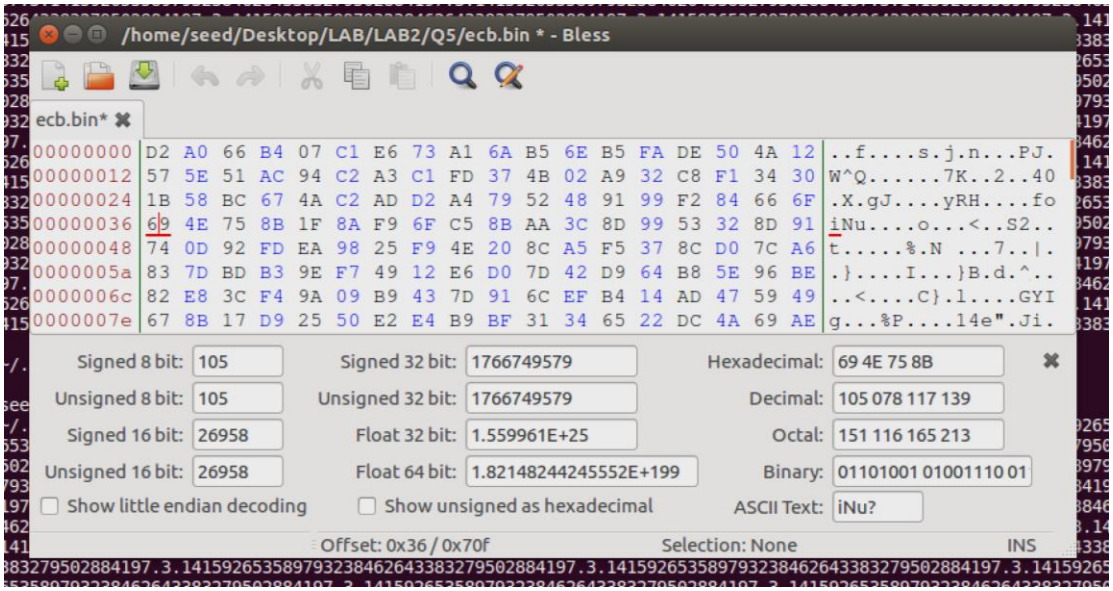
I believe the ECB mode will have the moderate data loss. This is so as in ECB mode every plain text is encrypted independently, this means that if one bye of ciphertext is corrupted then only that block will be affected and rest of the file will be decrypted without further issues.

For modes CBC and CFB, I think we would observe data loss. I assume this loss will be more significant for CBC as compared to CFB. As in CBC, when one block of ciphertext is corrupted this in turn affects the decryption of the following blocks.

The OFB mode will have the least data loss as compared to the other modes. I can say so, as in OFB mode only the corrupted bit's decrypted bit will be affected. Now let's verify my assumptions with the results:

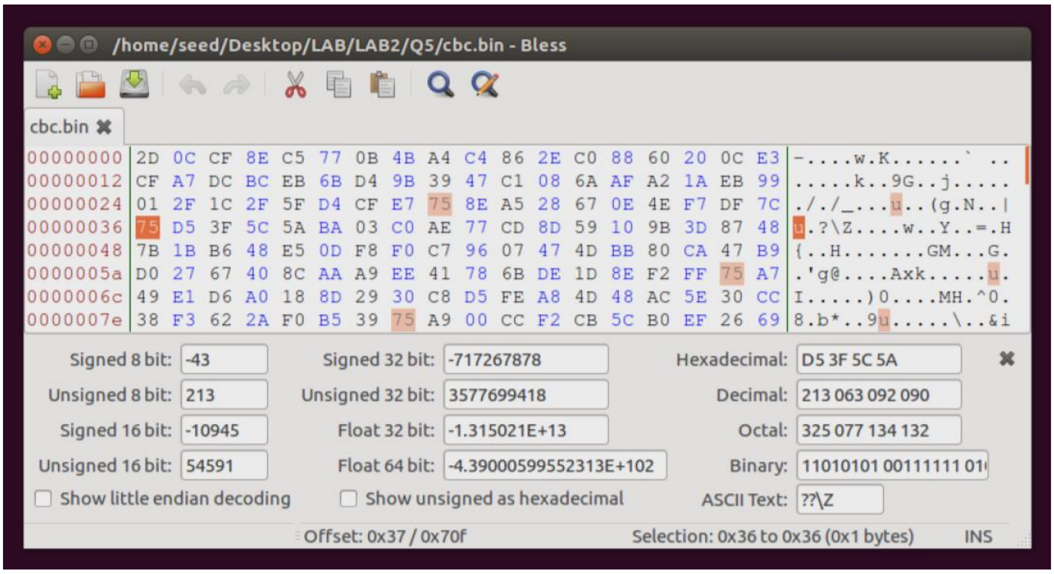
AES-128-ECB

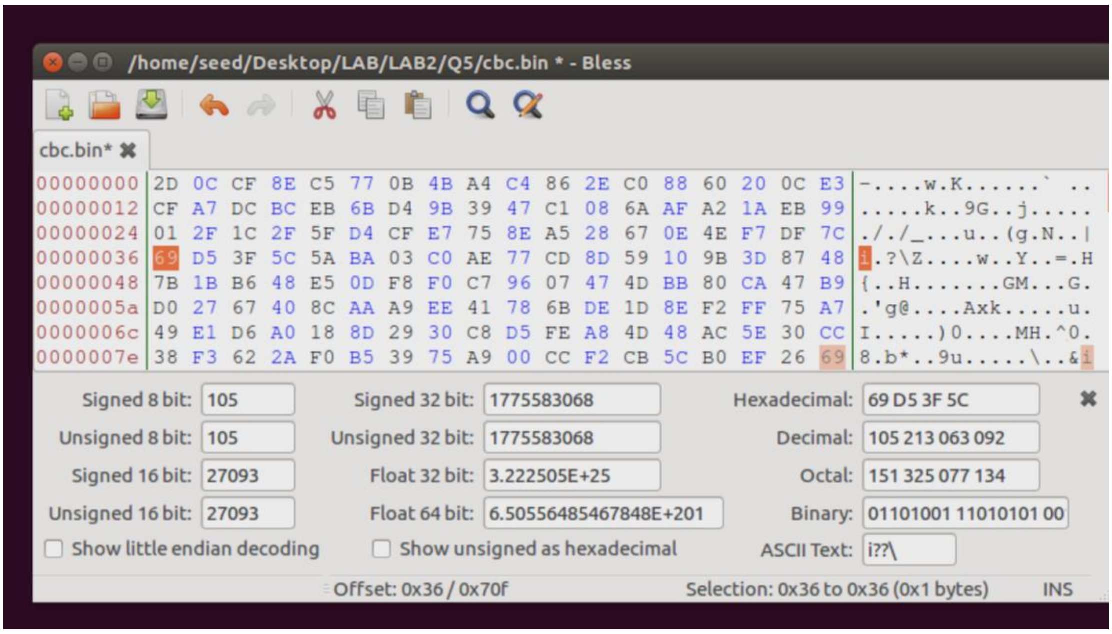


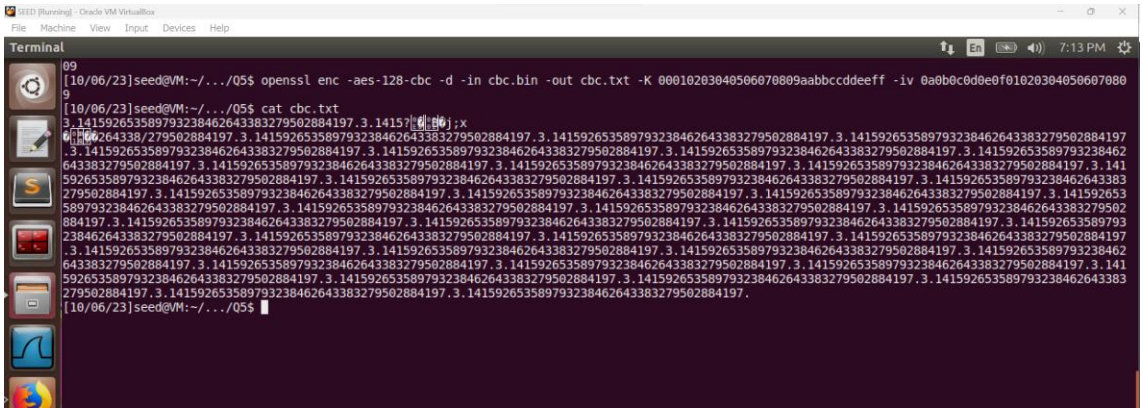



-rw-rw-r-- 1 seed seed 9408 Oct 6 18:33 file1
[10/06/23]seed@VM:~/.../Q5$ ls -l file.txt
-rw-rw-r-- 1 seed seed 1807 Oct 6 18:36 file.txt
[10/06/23]seed@VM:~/.../Q5$ openssl enc -aes-128-ecb -e -in file.txt -out ecb.bin -K 00010203040506070809aabbccddeeff
[10/06/23]seed@VM:~/.../Q5$ openssl enc -aes-128-ecb -d -in ecb.bin -out ecb.txt -K 00010203040506070809aabbccddeeff
[10/06/23]seed@VM:~/.../Q5$ cat ecb.txt
[10/06/23]seed@VM:~/.../Q5$

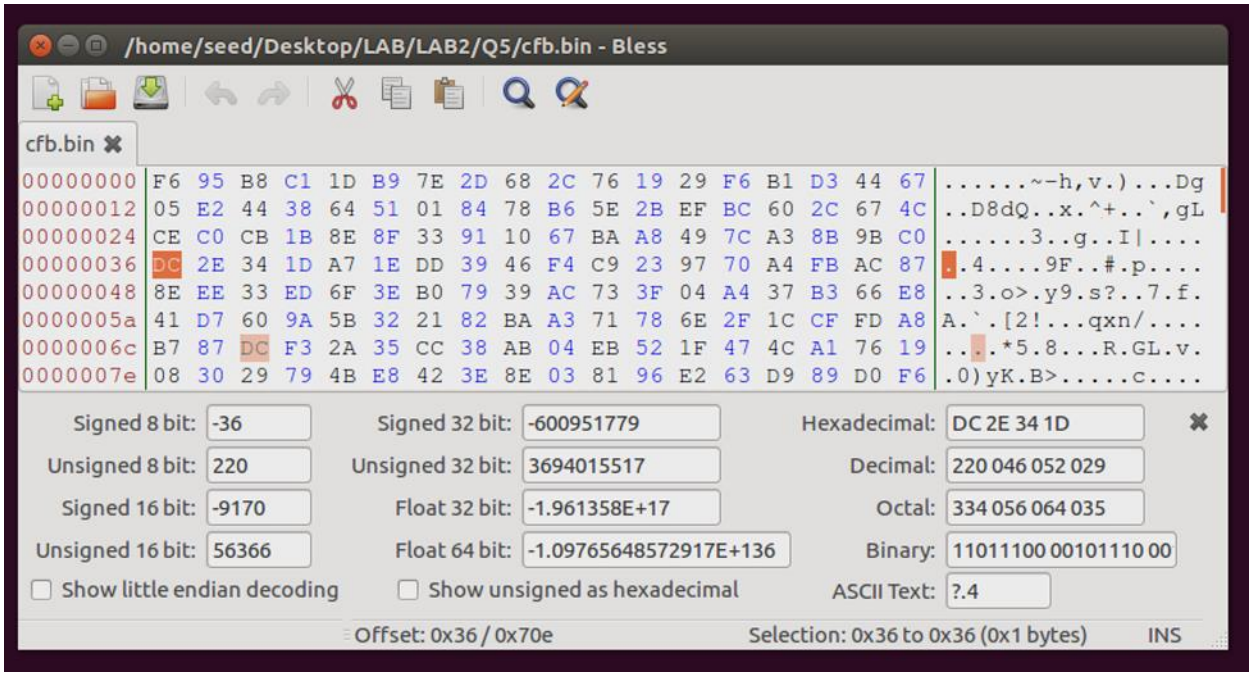
AES-128-CBC

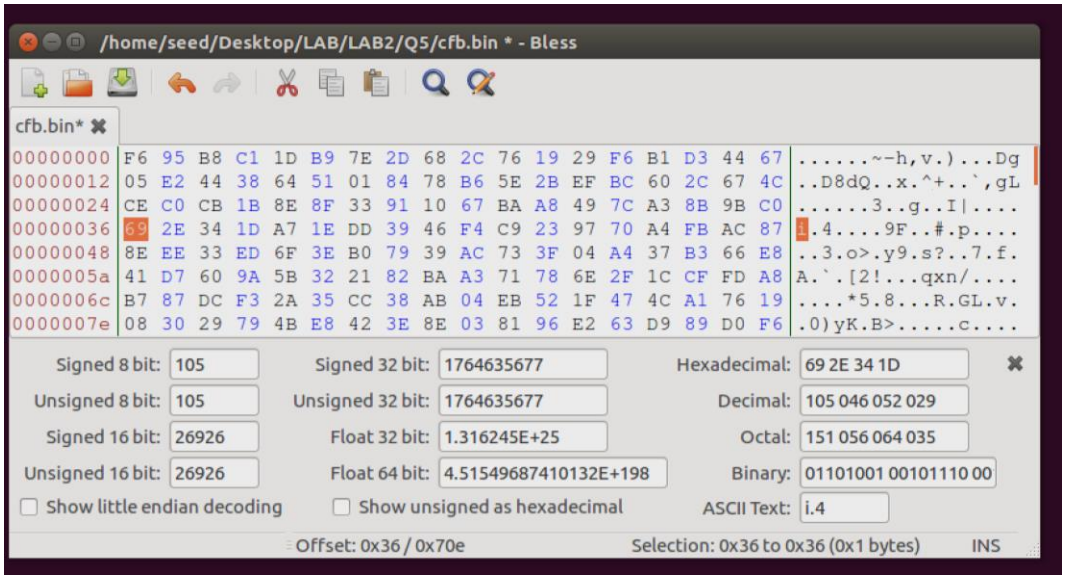

/home/seed/Desktop/LAB/LAB2/Q5/cbc.bin - Bless
cbc.bin
00000000 2D 0C CF 8E C5 77 0B 4B A4 C4 86 2E C0 88 60 20 0C E3
00000012 CF A7 DC BC EB 6B D4 9B 39 47 C1 08 6A AF A2 1A EB 99
00000024 01 2F 1C 2F 5F D4 CF E7 75 8E A5 28 67 0E 4E F7 DF 7C
00000036 75 D5 3F 5C 5A BA 03 C0 AE 77 CD 8D 59 10 9B 3D 87 48
00000048 7B 1B B6 48 E5 0D F8 F0 C7 96 07 47 4D BB 80 CA 47 B9
0000005a D0 27 67 40 8C AA A9 EE 41 78 6B DE 1D 8E F2 FF 75 A7
0000006c 49 E1 D6 A0 18 8D 29 30 C8 D5 FE A8 4D 48 AC 5E 30 CC
0000007e 38 F3 62 2A F0 B5 39 75 A9 00 CC F2 CB 5C B0 EF 26 69
Signed 8 bit: -43
Unsigned 8 bit: 213
Signed 16 bit: -10945
Unsigned 16 bit: 54591
Signed 32 bit: -717267878
Unsigned 32 bit: 3577699418
Float 32 bit: -1.315021E+13
Float 64 bit: -4.39000599552313E+102
Hexadecimal: D5 3F 5C 5A
Decimal: 213 063 092 090
Octal: 325 077 134 132
Binary: 11010101 00111111 01
ASCII Text: ??\Z
Show little endian decoding
Show unsigned as hexadecimal
Offset: 0x37 / 0x70f
Selection: 0x36 to 0x36 (0x1 bytes)
INS

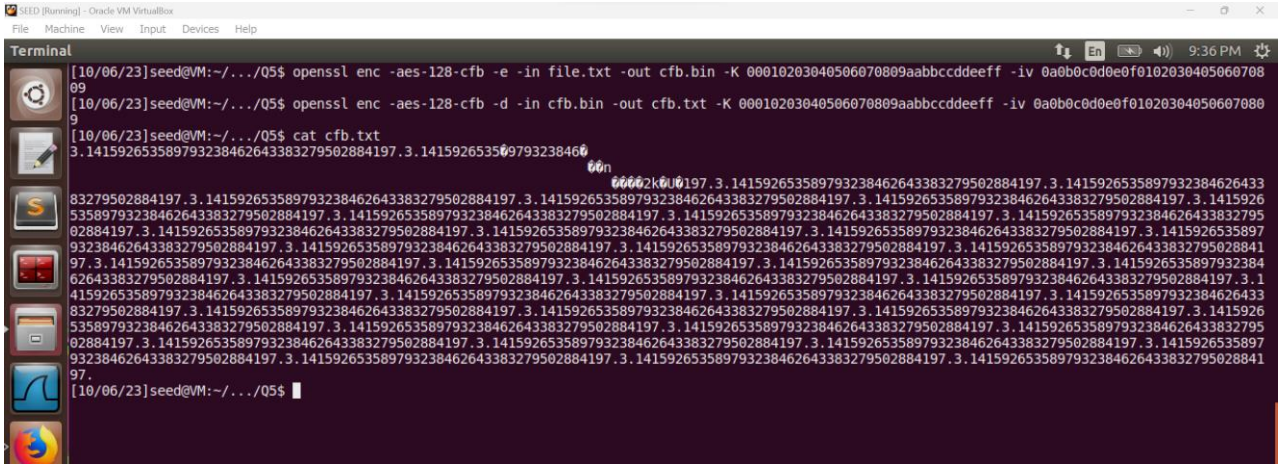

/home/seed/Desktop/LAB/LAB2/Q5/cbc.bin * - Bless
cbc.bin*
00000000 2D 0C CF 8E C5 77 0B 4B A4 C4 86 2E C0 88 60 20 0C E3
00000012 CF A7 DC BC EB 6B D4 9B 39 47 C1 08 6A AF A2 1A EB 99
00000024 01 2F 1C 2F 5F D4 CF E7 75 8E A5 28 67 0E 4E F7 DF 7C
00000036 69 D5 3F 5C 5A BA 03 C0 AE 77 CD 8D 59 10 9B 3D 87 48
00000048 7B 1B B6 48 E5 0D F8 F0 C7 96 07 47 4D BB 80 CA 47 B9
0000005a D0 27 67 40 8C AA A9 EE 41 78 6B DE 1D 8E F2 FF 75 A7
0000006c 49 E1 D6 A0 18 8D 29 30 C8 D5 FE A8 4D 48 AC 5E 30 CC
0000007e 38 F3 62 2A F0 B5 39 75 A9 00 CC F2 CB 5C B0 EF 26 69
Signed 8 bit: 105
Unsigned 8 bit: 105
Signed 16 bit: 27093
Unsigned 16 bit: 27093
Signed 32 bit: 1775583068
Unsigned 32 bit: 1775583068
Float 32 bit: 3.222505E+25
Float 64 bit: 6.50556485467848E+201
Hexadecimal: 69 D5 3F 5C
Decimal: 105 213 063 092
Octal: 151 325 077 134
Binary: 01101001 11010101 00
ASCII Text: i??\
Show little endian decoding
Show unsigned as hexadecimal
Offset: 0x36 / 0x70f
Selection: 0x36 to 0x36 (0x1 bytes)
INS


Terminal
7:13 PM
[10/06/23]seed@VM:~/.../Q5$ openssl enc -aes-128-cbc -d -in cbc.bin -out cbc.txt -K 00010203040506070809aabbccddeeff -iv 0a0b0c0d0e0f01020304050607080
9
[10/06/23]seed@VM:~/.../Q5$ cat cbc.txt
[10/06/23]seed@VM:~/.../Q5$

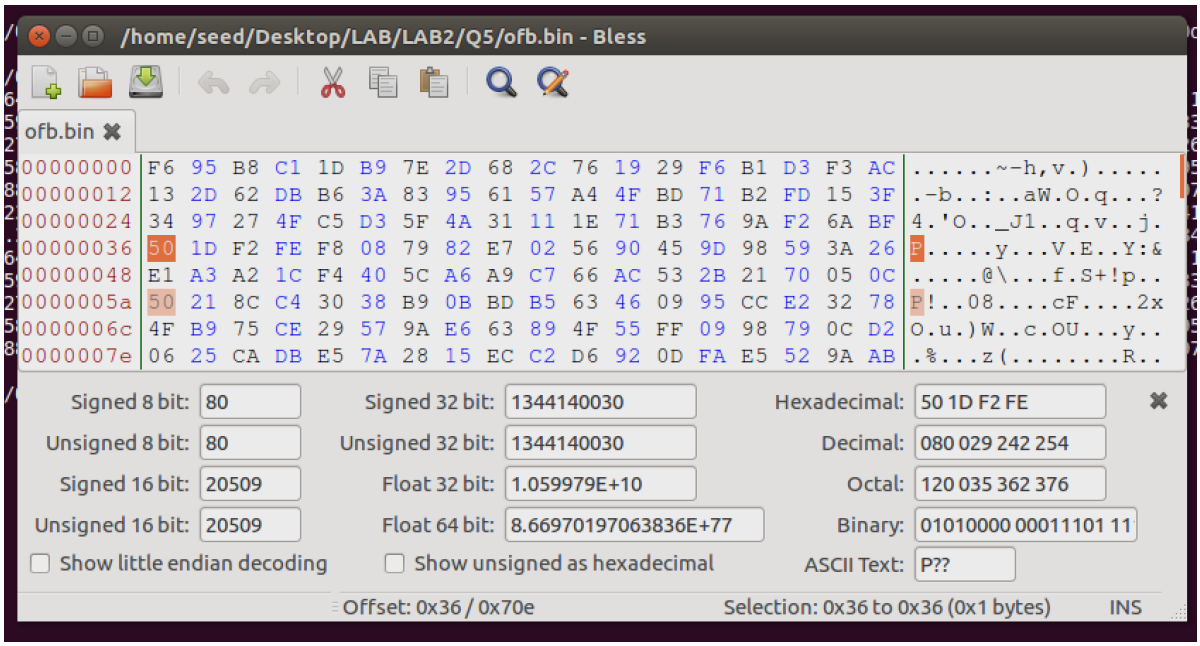
AES-128-CFB

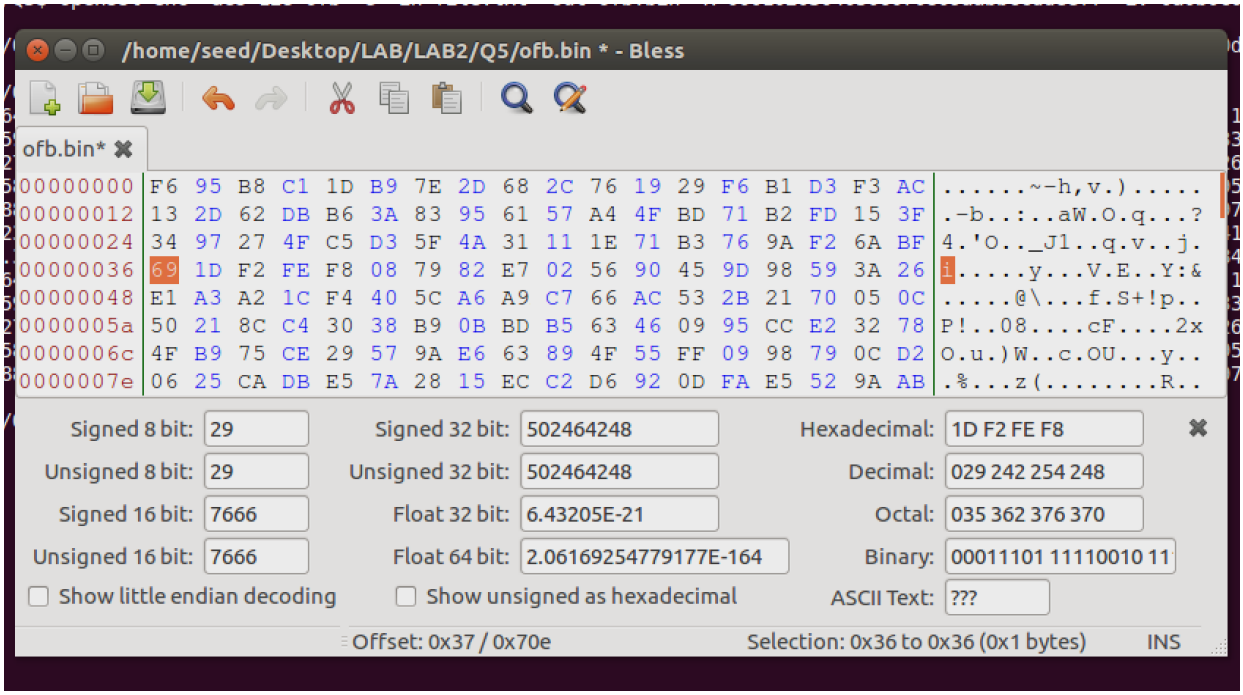

/home/seed/Desktop/LAB/LAB2/Q5/cfb.bin - Bless
cfb.bin
00000000 F6 95 B8 C1 1D B9 7E 2D 68 2C 76 19 29 F6 B1 D3 44 67 ......~-h,v.)...Dg
00000012 05 E2 44 38 64 51 01 84 78 B6 5E 2B EF BC 60 2C 67 4C ..D8dQ..x.^+..`,gL
00000024 CE C0 CB 1B 8E 8F 33 91 10 67 BA A8 49 7C A3 8B 9B C0 ......3..g..I|....
00000036 DC 2E 34 1D A7 1E DD 39 46 F4 C9 23 97 70 A4 FB AC 87 .4....9F..#.p....
00000048 8E EE 33 ED 6F 3E B0 79 39 AC 73 3F 04 A4 37 B3 66 E8 ..3.o>.y9.s?..7.f.
0000005a 41 D7 60 9A 5B 32 21 82 BA A3 71 78 6E 2F 1C CF FD A8 A.`.[2!...qxn/....
0000006c B7 87 DC F3 2A 35 CC 38 AB 04 EB 52 1F 47 4C A1 76 19 ....*5.8...R.GL.v.
0000007e 08 30 29 79 4B E8 42 3E 8E 03 81 96 E2 63 D9 89 D0 F6 .0)yK.B>.....c....
Signed 8 bit: -36
Signed 32 bit: -600951779
Hexadecimal: DC 2E 34 1D
Unsigned 8 bit: 220
Unsigned 32 bit: 3694015517
Decimal: 220 046 052 029
Signed 16 bit: -9170
Float 32 bit: -1.961358E+17
Octal: 334 056 064 035
Unsigned 16 bit: 56366
Float 64 bit: -1.09765648572917E+136
Binary: 11011100 00101110 00
Show little endian decoding
Show unsigned as hexadecimal
ASCII Text: ?.4
Offset: 0x36 / 0x70e
Selection: 0x36 to 0x36 (0x1 bytes)
INS

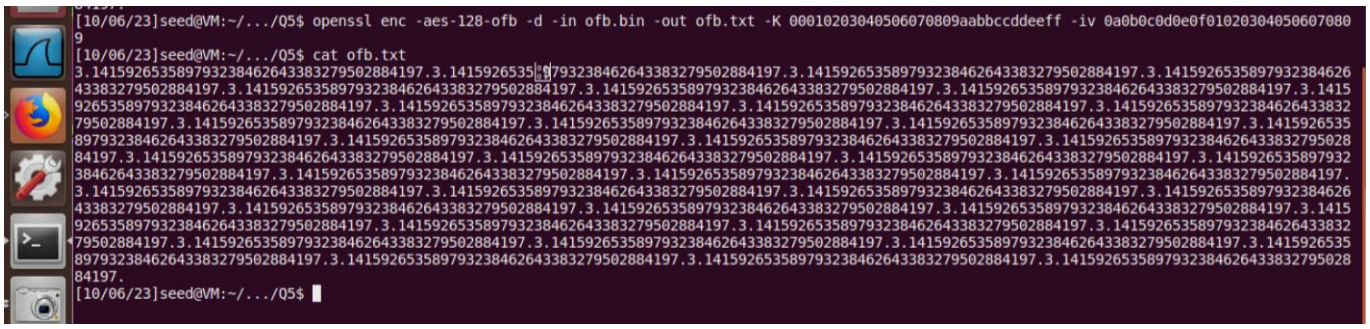

/home/seed/Desktop/LAB/LAB2/Q5/cfb.bin * - Bless
cfb.bin*
00000000 F6 95 B8 C1 1D B9 7E 2D 68 2C 76 19 29 F6 B1 D3 44 67 ......~-h,v.)...Dg
00000012 05 E2 44 38 64 51 01 84 78 B6 5E 2B EF BC 60 2C 67 4C ..D8dQ..x.^+..`,gL
00000024 CE C0 CB 1B 8E 8F 33 91 10 67 BA A8 49 7C A3 8B 9B C0 ......3..g..I|....
00000036 69 2E 34 1D A7 1E DD 39 46 F4 C9 23 97 70 A4 FB AC 87 i.4....9F..#.p....
00000048 8E EE 33 ED 6F 3E B0 79 39 AC 73 3F 04 A4 37 B3 66 E8 ..3.o>.y9.s?..7.f.
0000005a 41 D7 60 9A 5B 32 21 82 BA A3 71 78 6E 2F 1C CF FD A8 A.`.[2!...qxn/....
0000006c B7 87 DC F3 2A 35 CC 38 AB 04 EB 52 1F 47 4C A1 76 19 ....*5.8...R.GL.v.
0000007e 08 30 29 79 4B E8 42 3E 8E 03 81 96 E2 63 D9 89 D0 F6 .0)yK.B>.....c....
Signed 8 bit: 105
Signed 32 bit: 1764635677
Hexadecimal: 69 2E 34 1D
Unsigned 8 bit: 105
Unsigned 32 bit: 1764635677
Decimal: 105 046 052 029
Signed 16 bit: 26926
Float 32 bit: 1.316245E+25
Octal: 151 056 064 035
Unsigned 16 bit: 26926
Float 64 bit: 4.51549687410132E+198
Binary: 01101001 00101110 00
Show little endian decoding
Show unsigned as hexadecimal
ASCII Text: i.4
Offset: 0x36 / 0x70e
Selection: 0x36 to 0x36 (0x1 bytes)
INS


Terminal
[10/06/23]seed@VM:~/.../Q5$ openssl enc -aes-128-cfb -e -in file.txt -out cfb.bin -K 00010203040506070809aabbccddeeff -iv 0a0b0c0d0e0f0102030405060708
09
[10/06/23]seed@VM:~/.../Q5$ openssl enc -aes-128-cfb -d -in cfb.bin -out cfb.txt -K 00010203040506070809aabbccddeeff -iv 0a0b0c0d0e0f0102030405060708
9
[10/06/23]seed@VM:~/.../Q5$ cat cfb.txt
[10/06/23]seed@VM:~/.../Q5$

AES-128-OFB


/home/seed/Desktop/LAB/LAB2/Q5/ofb.bin - Bless
ofb.bin
00000000 F6 95 B8 C1 1D B9 7E 2D 68 2C 76 19 29 F6 B1 D3 F3 AC ......~-h,v.).....
00000012 13 2D 62 DB B6 3A 83 95 61 57 A4 4F BD 71 B2 FD 15 3F .-b..:..aW.O.q...?
00000024 34 97 27 4F C5 D3 5F 4A 31 11 1E 71 B3 76 9A F2 6A BF 4.'O.._J1..q.v..j.
00000036 50 1D F2 FE F8 08 79 82 E7 02 56 90 45 9D 98 59 3A 26 P.....y...V.E..Y:&
00000048 E1 A3 A2 1C F4 40 5C A6 A9 C7 66 AC 53 2B 21 70 05 0C .....@\...f.S+!p..
0000005a 50 21 8C C4 30 38 B9 0B BD B5 63 46 09 95 CC E2 32 78 P!..08....cF....2x
0000006c 4F B9 75 CE 29 57 9A E6 63 89 4F 55 FF 09 98 79 0C D2 O.u.)W..c.OU...y..
0000007e 06 25 CA DB E5 7A 28 15 EC C2 D6 92 0D FA E5 52 9A AB .%...z(........R..
Signed 8 bit: 80
Unsigned 8 bit: 80
Signed 16 bit: 20509
Unsigned 16 bit: 20509
Show little endian decoding
Signed 32 bit: 1344140030
Unsigned 32 bit: 1344140030
Float 32 bit: 1.059979E+10
Float 64 bit: 8.66970197063836E+77
Show unsigned as hexadecimal
Hexadecimal: 50 1D F2 FE
Decimal: 080 029 242 254
Octal: 120 035 362 376
Binary: 01010000 00011101 11
ASCII Text: P??
Offset: 0x36 / 0x70e
Selection: 0x36 to 0x36 (0x1 bytes)
INS

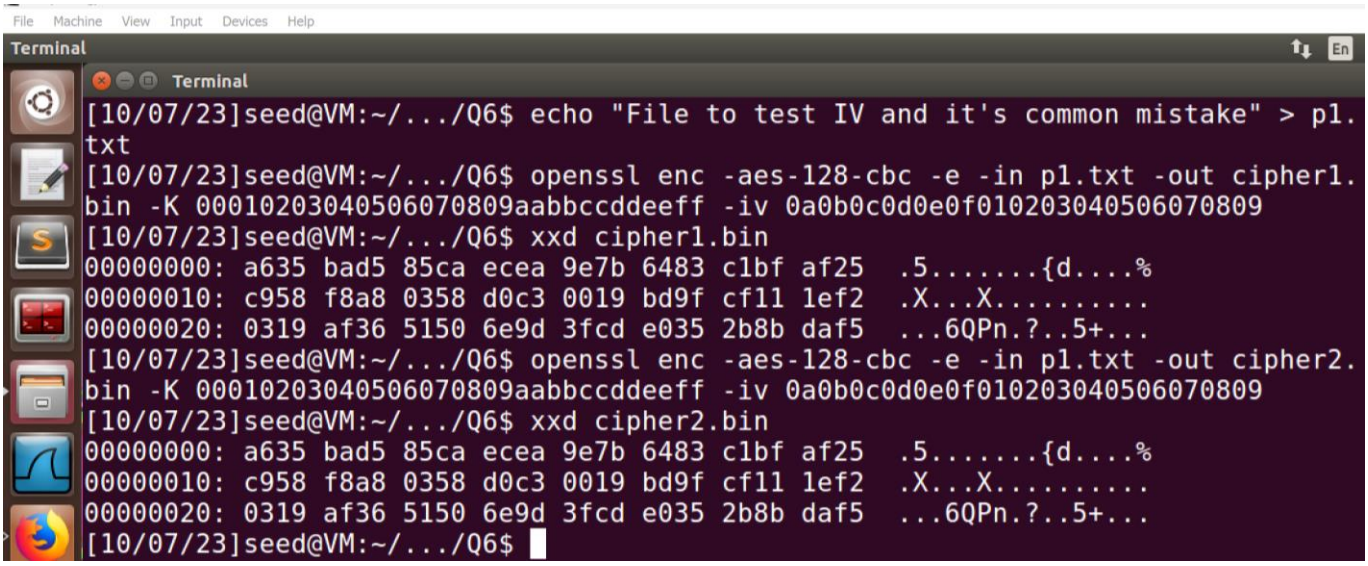

/home/seed/Desktop/LAB/LAB2/Q5/ofb.bin * - Bless
ofb.bin*
00000000 F6 95 B8 C1 1D B9 7E 2D 68 2C 76 19 29 F6 B1 D3 F3 AC ......~-h,v.).....
00000012 13 2D 62 DB B6 3A 83 95 61 57 A4 4F BD 71 B2 FD 15 3F .-b..:..aW.O.q...?
00000024 34 97 27 4F C5 D3 5F 4A 31 11 1E 71 B3 76 9A F2 6A BF 4.'O.._J1..q.v..j.
00000036 69 1D F2 FE F8 08 79 82 E7 02 56 90 45 9D 98 59 3A 26 i.....y...V.E..Y:&
00000048 E1 A3 A2 1C F4 40 5C A6 A9 C7 66 AC 53 2B 21 70 05 0C .....@\...f.S+!p..
0000005a 50 21 8C C4 30 38 B9 0B BD B5 63 46 09 95 CC E2 32 78 P!..08....cF....2x
0000006c 4F B9 75 CE 29 57 9A E6 63 89 4F 55 FF 09 98 79 0C D2 O.u.)W..c.OU...y..
0000007e 06 25 CA DB E5 7A 28 15 EC C2 D6 92 0D FA E5 52 9A AB .%...z(........R..
Signed 8 bit: 29
Unsigned 8 bit: 29
Signed 16 bit: 7666
Unsigned 16 bit: 7666
Show little endian decoding
Signed 32 bit: 502464248
Unsigned 32 bit: 502464248
Float 32 bit: 6.43205E-21
Float 64 bit: 2.06169254779177E-164
Show unsigned as hexadecimal
Hexadecimal: 1D F2 FE F8
Decimal: 029 242 254 248
Octal: 035 362 376 370
Binary: 00011101 11110010 11
ASCII Text: ???
Offset: 0x37 / 0x70e
Selection: 0x36 to 0x36 (0x1 bytes)
INS



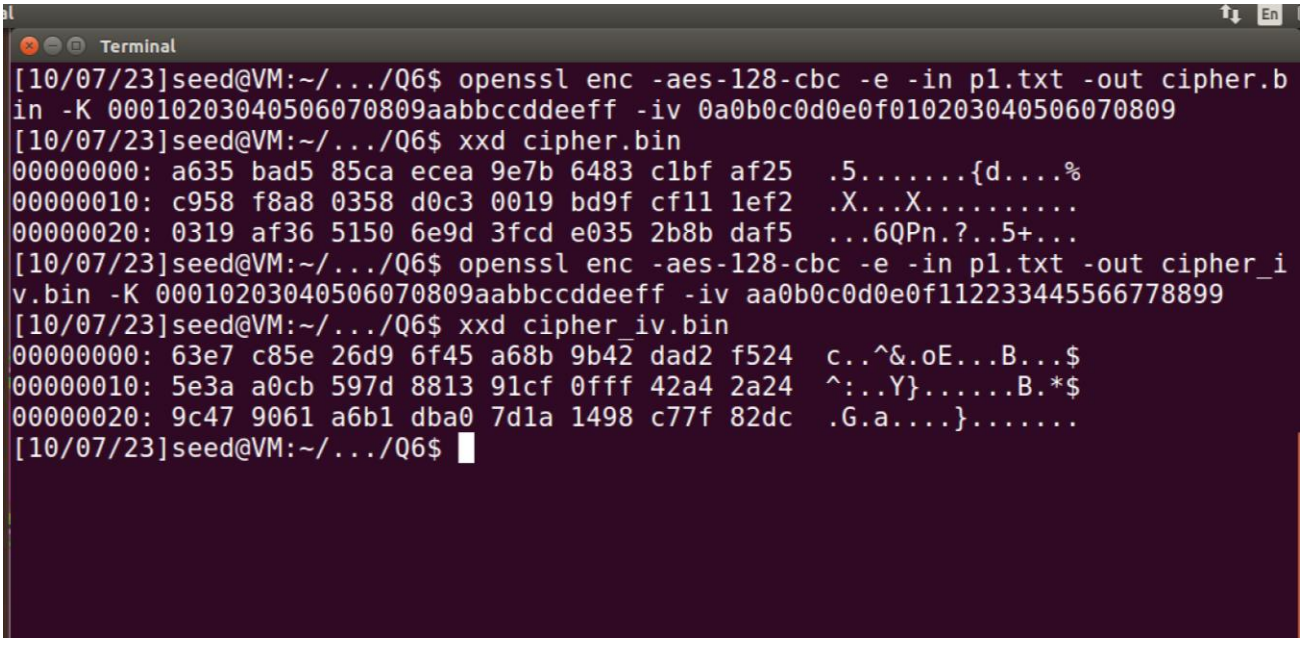
6. Initial Vector

   a. Uniqueness of IV

Same IV



Different IV



Form the above results I observed the following. When using the same Initial Vector (iv) for encryption, we end up with similar encrypted message. I can see this clearly when I used xxd to view the hexadecimal format of the encrypted file, this becomes a vulnerability as someone with enough knowledge can break this encryption. But when I used different ivs, this resulted in formation of different encrypted message. I believe advisable to follow this approach to use different IV to ensure more protection for encryption.

b. Common Mistake: Use of the Same IV

Steps:

P1: This is a known message!

C1: a469b1c502c1cab966965e50425438e1bb1b5f9037a4c15913

C2: bf73bcd3509299d566c35b5d450337e1bb175f903fafc15913

Now I'll convert P1 into hexadecimal

Hex_P1: 546869732069732061206B6E6F776E206D657373616765210A

XORing the Hex_P1 and C1 to obtain k

k: f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a47819

XORing k and C2 to get Hex_P2

Hex_P2: 4f726465723a204c61756e63682061206d697373696c6521

Converting Hex_P2 to letters:

P2: Order: Launch a missile!

If we replace the OFB with CFB mode for this experiment, then neither I nor an attacker could be able to decrypt the ciphertext to get the second plaintext. This is so as, in CFB mode a keystream is generated, which is then combined with the plain text to obtain the encrypted text. Now if I or the attacker knows the plaintext, ciphertext and iv for first message, then we are able to decrypt this message only, but we can't decrypt any other message despite same ivs are used as keystreams are unique, this means each message generates an unique keystream which can't be used to decrypt other messages.

c. Common Mistake: Use a Predictable IV

Steps:

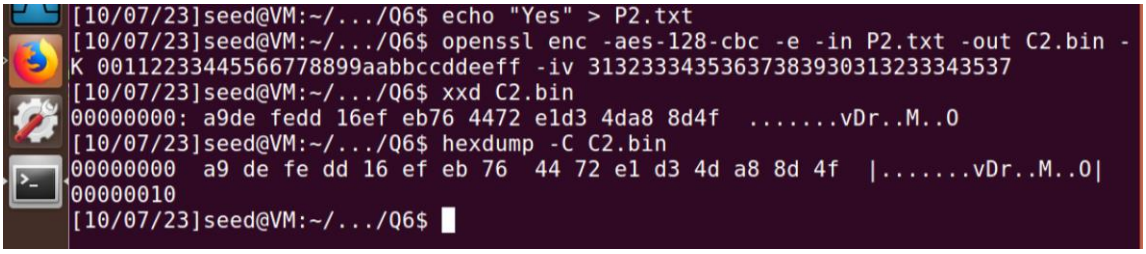
Considering P1 to be yes

Converting the message P1 (yes) to hexadecimal : 59 65 73 0A with the help of : https://www.rapidtables.com/convert/number/ascii-to-hex.html

After doing so, the nest step is to get P2, I'll obtain that by performing XOR operation between P1 and IV1, and performing XOR of this result with IV2

P1 XOR IV1: 5a765039435363738393031323334353 6
That result XOR with P2: 5965730a00000000000000000000000 1
Refence used: https://xor.pw/#

Converting P2 to letters: Yes
Now I calculate C2, by the following formula: C2 = Enc(k,IV2 ⊕ P2) but P2: P2 = P1 ⊕ IV1 ⊕ IV2
So I calculate C2 by using the key Bob has with IV2 which Eve predicted.



C2: a9 de fe dd 16 ef eb 76 44 72 e1 d3 4d a8 8d 4f
Now as I can notice C1!= C2, that mean my assumption on P1 being “Yes” was false. Therefore, P1 was “No”