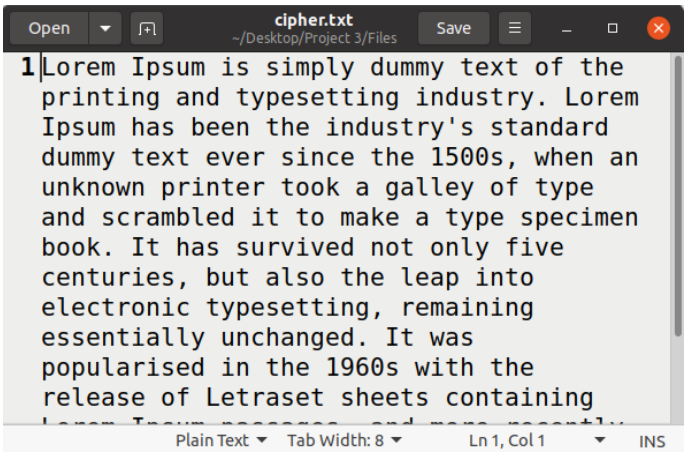
Task 2: Encryption using Different Ciphers and Modes

```
[11/07/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc  -e -in pla
in.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102
030405060708
hex string is too short, padding with zero bytes to length
[11/07/22]seed@VM:~/.../Files$ openssl enc -bf-cbc  -e -in plain.tx
t -out cipher2.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[11/07/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb  -e -in pla
in.txt -out cipher3.bin -K 00112233445566778889aabbccddeeff -iv 010
2030405060708
hex string is too short, padding with zero bytes to length
[11/07/22]seed@VM:~/.../Files$
```

Encryption of plain.txt into cipher.bin files using CBC, BF-CBC and CFB.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in ciph
er.bin -out cipher.txt -K 00112233445566778889aabbccddeeff -iv 0102
030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -bf-cbc -d -in cipher2.b
in -out cipher2.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in ciph
er3.bin -out cipher3.txt -K 00112233445566778889aabbccddeeff -iv 01
02030405060708
hex string is too short, padding with zero bytes to length
```

Decryption of cipher.bin files into cipher.txt files.



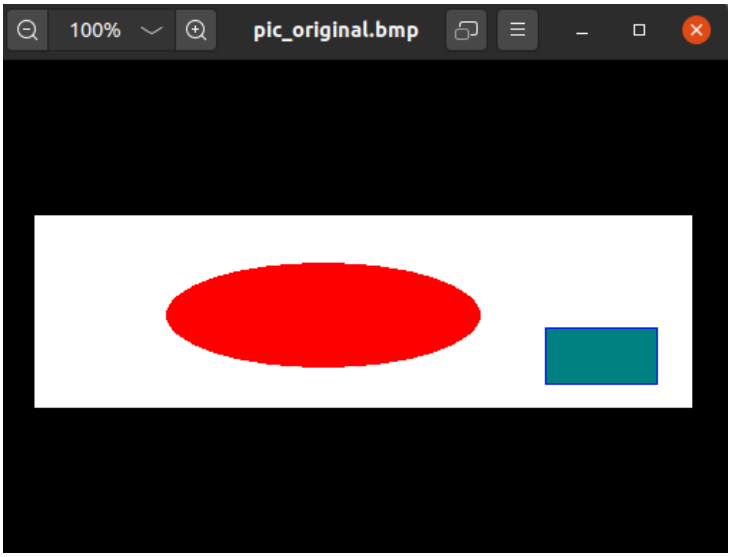
Successful decryption

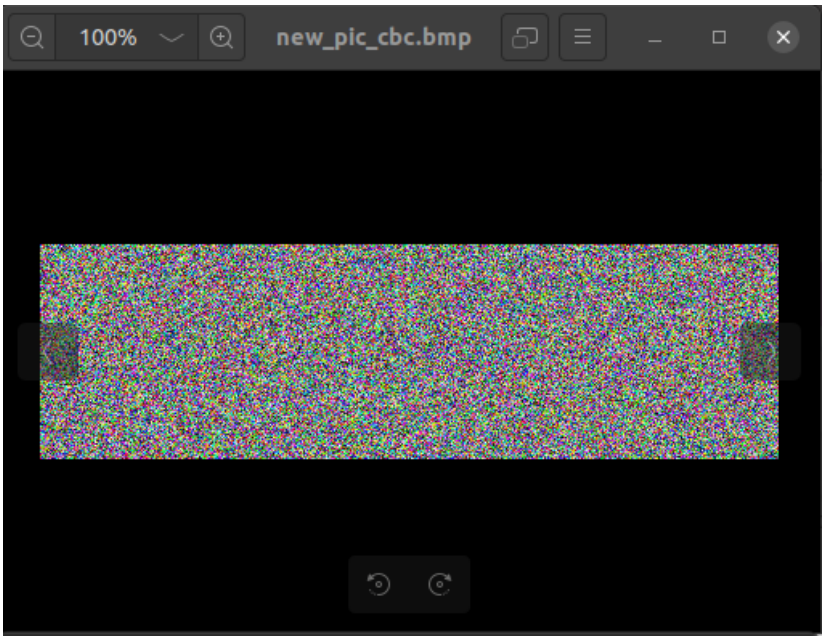
Task 3: Encryption Mode – ECB vs. CBC

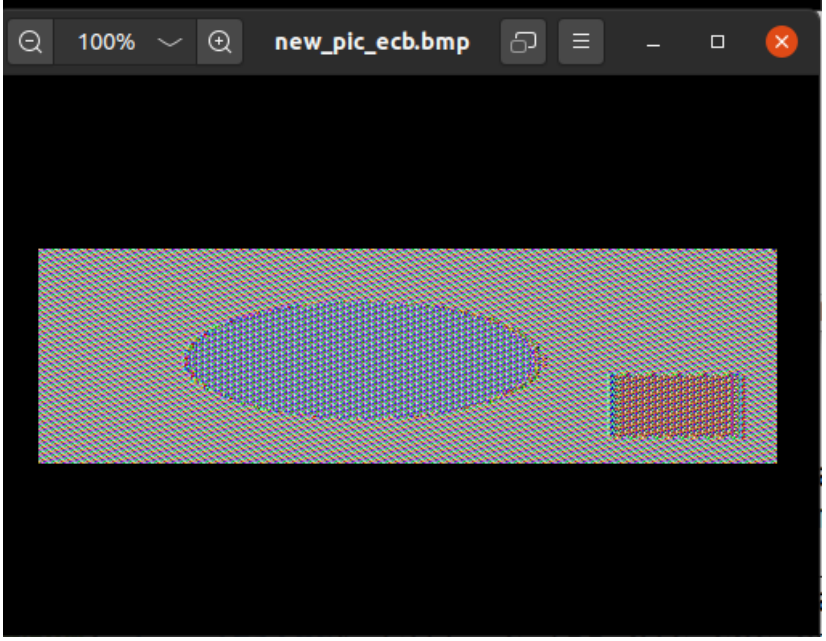
```
[11/07/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb  -e -in pic
_original.bmp -out pic_ecb.bmp -K 00112233445566778889aabbccddeeff
-iv 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc  -e -in pic
_original.bmp -out pic_cbc.bmp -K 00112233445566778889aabbccddeeff
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$
```

```
[11/13/22]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 pic_ecb.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_pic_ecb.bmp
[11/13/22]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 pic_cbc.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_pic_cbc.bmp
```

The above 2 images show the process of using ECB and CBC encryptions on the pic_original.bmp image and the conversion of the header information of the generated files.

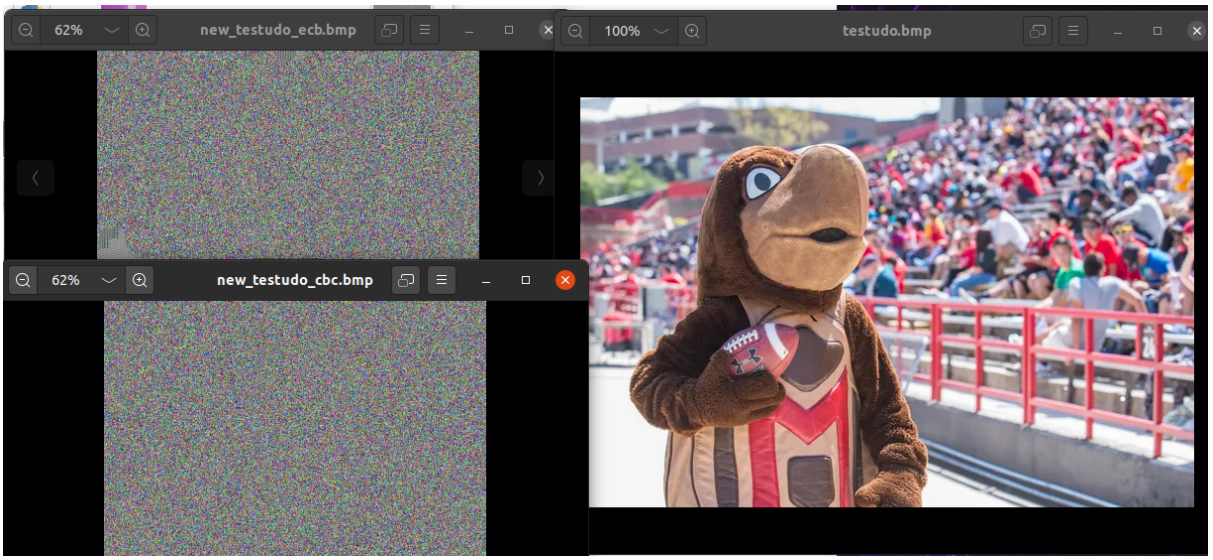

100%
pic_original.bmp


100%
new_pic_cbc.bmp


100%
new_pic_ecb.bmp

As shown in the 3 images above, ECB is the more insecure encryption scheme as the image still resembles the original image. This is because when identical plaintext blocks are encrypted into identical ciphertext blocks, a pattern among the data generated can be observed. In contrast, CBC is more secure as the image is wholly converted into random white noise.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in test
udo.bmp -out testudo_ecb.bmp -K 00112233445566778889aabbccddeeff -i
v 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in test
udo.bmp -out testudo_cbc.bmp -K 00112233445566778889aabbccddeeff -i
v 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ head -c 54 testudo.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 testudo_ecb.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_testudo_ecb.bm
p
[11/13/22]seed@VM:~/.../Files$ head -c 54 testudo.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 testudo_cbc.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_testudo_cbc.bm
p
[11/13/22]seed@VM:~/   /Files$
```



For the 2nd image, I used the above image of Testudo. The observed results here are different from pic_original.bmp's. Both pictures generated using ECB and CBC show that the original image has been converted to random white noise and is unrecognizable. A plausible explanation is that the testudo.bmp image is more complicated than pic_original.bmp.

Task 4: Padding

1.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plai
n.txt -out cipher_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0
102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plai
n.txt -out cipher_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0
102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in plai
n.txt -out cipher_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0
102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in plai
n.txt -out cipher_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0
102030405060708
warning: iv not used by this cipher
```

```
-rw-rw-r-- 1 seed seed    576 Nov 13 08:17 cipher_cbc.txt
-rw-rw-r-- 1 seed seed    575 Nov 13 08:17 cipher_cfb.txt
-rw-rw-r-- 1 seed seed    576 Nov 13 08:18 cipher_ecb.txt
-rw-rw-r-- 1 seed seed    575 Nov 13 08:17 cipher_ofb.txt
```

CBC and ECB require padding while CFB and OFB do not require padding. CFB and OFB do not require padding as they are stream ciphers.

2.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.t
xt -out f1_cbc.txt -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.t
xt -out f2_cbc.txt -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.t
xt -out f3_cbc.txt -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
hex string is too short, padding with zero bytes to length
```

```
-rw-rw-r-- 1 seed seed     16 Nov 13 08:28 f1_cbc.txt
-rw-rw-r-- 1 seed seed      5 Nov 13 02:45 f1.txt
-rw-rw-r-- 1 seed seed     16 Nov 13 08:28 f2_cbc.txt
-rw-rw-r-- 1 seed seed     10 Nov 13 02:44 f2.txt
-rw-rw-r-- 1 seed seed     32 Nov 13 08:28 f3_cbc.txt
-rw-rw-r-- 1 seed seed     16 Nov 13 02:45 f3.txt
```

f1.txt and f2.txt created outputs of 16 bytes while f3.txt generated an output of 32 bytes. It shows the extent of padding in a CBC cipher. CBC uses a block size of 16 bytes - the generated ciphertext has to be in multiples of 16. Therefore, f1.txt and f2.txt are extended to 16 bytes and f3.txt is extended to 32 bytes. f3.txt is padded even though it is already 16 bytes as it is clearer for the receiver to know whether the file is 16 bytes or 1-15 bytes with padding.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f1_cbc.txt -out f1_d.txt -K 00112233445566778889aabbccddeeff -iv
 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f2_cbc.txt -out f2_d.txt -K 00112233445566778889aabbccddeeff -iv
 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f3_cbc.txt -out f3_d.txt -K 00112233445566778889aabbccddeeff -iv
 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$
```

```
[11/13/22]seed@VM:~/.../Files$ hexdump -C f1_d.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345.
..........|
00000010
[11/13/22]seed@VM:~/.../Files$ hexdump -C f2_d.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |123456
7890......|
00000010
[11/13/22]seed@VM:~/.../Files$ hexdump -C f3_d.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |123456
7890123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |......
..........|
00000020
[11/13/22]seed@VM:~/.../Files$
```

f1.txt: 0x0b is used to pad the file
f2.txt: 0x06 is used to pad the file
f3.txt: 0x10 is used to pad the file

Task 5:

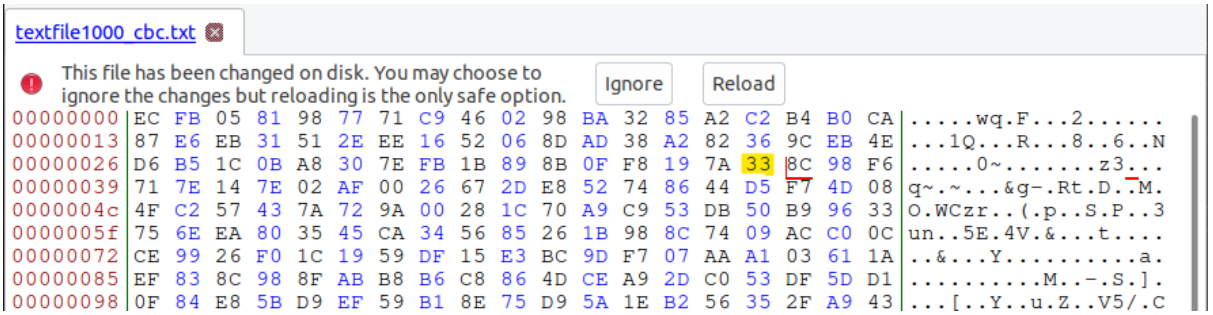
```
[11/13/22]seed@VM:~/.../Files$ python3 -c "print('1234567890'*100)"
 > textfile1000.txt
```

```
-rw-rw-r-- 1 seed seed   1001 Nov 13 08:58 textfile1000.txt
```

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in text
file1000.txt -out textfile1000_ecb.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in text
file1000.txt -out textfile1000_cbc.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in text
file1000.txt -out textfile1000_cfb.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in text
file1000.txt -out textfile1000_ofb.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

Creating a file and encrypting it using ECB, CBC, CFB and OFB.

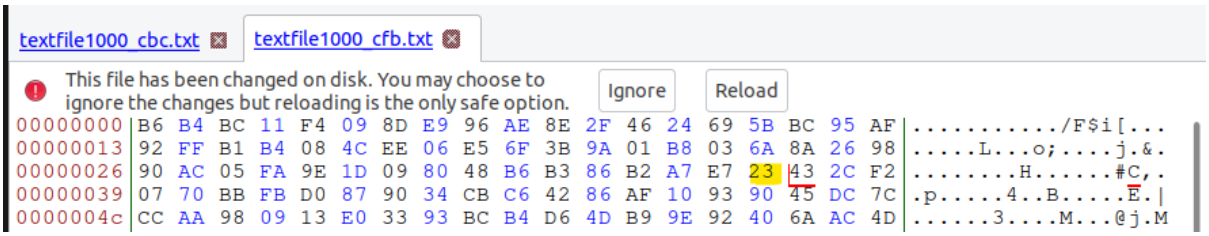
CBC:



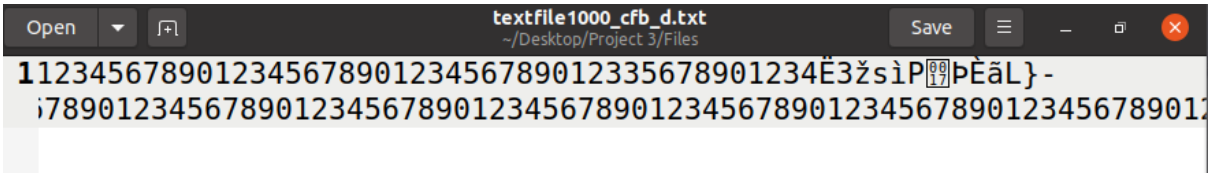
The 50th byte was corrupted by changing it from 32 to 33.

In CBC, 2 blocks in total were corrupted.

CFB:

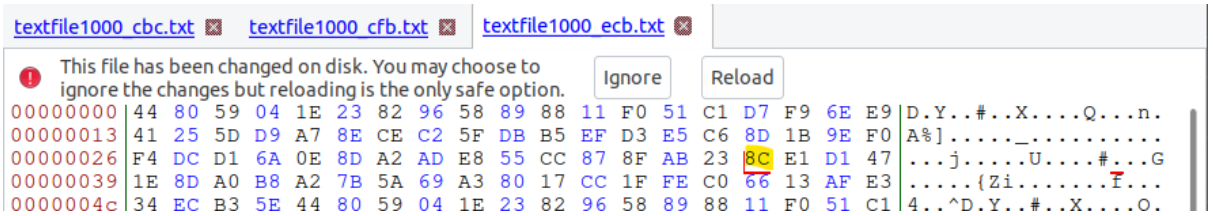


The 50th byte was corrupted by changing it from 22 to 23.

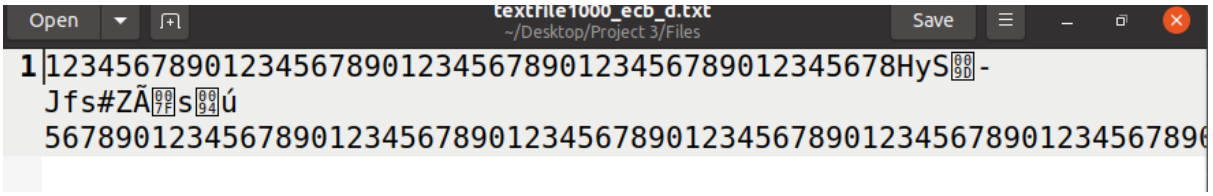


In CFB, 2 blocks in total were corrupted.

ECB:

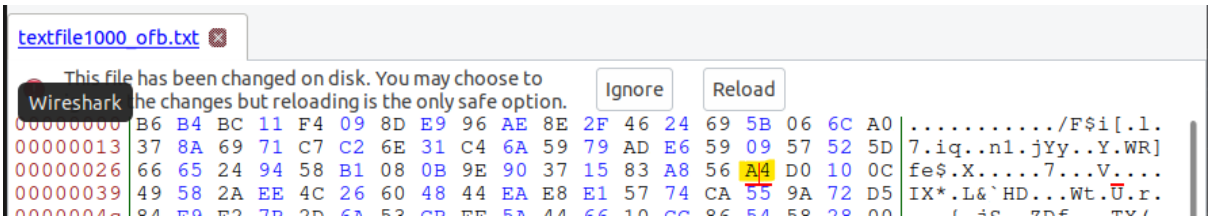


The 50th byte was corrupted by changing it from 8B to 8C.

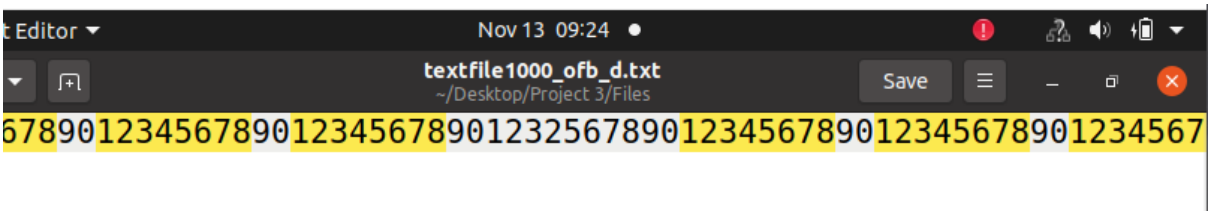


In ECB, 1 block was corrupted.

OFB:



The 50th byte was corrupted by changing it from A3 to A4.

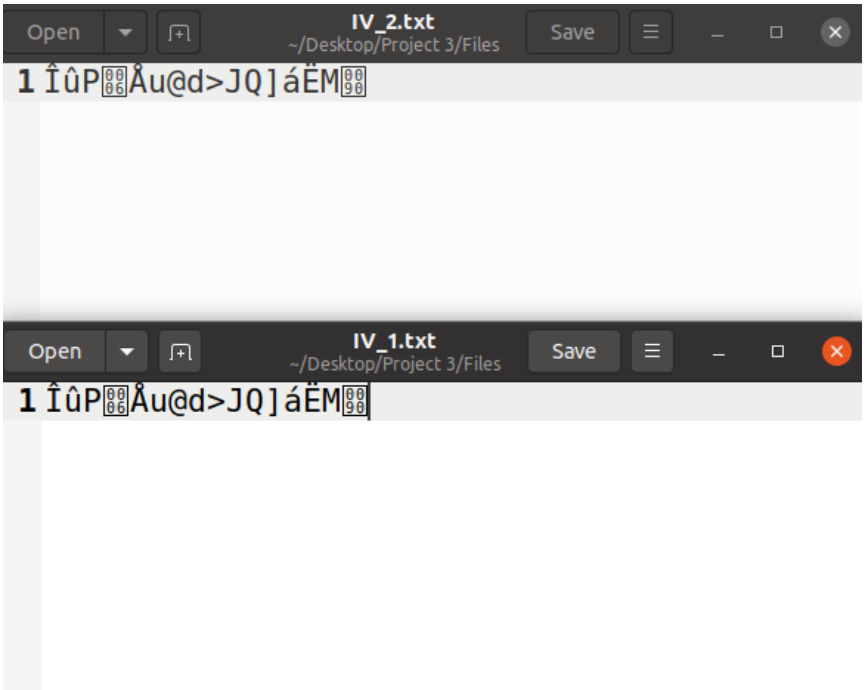


In OFB, only the corresponding byte is affected. Therefore, OFB is able to recover the most information as it affects the least number of bytes.

Task 6:

6.1:

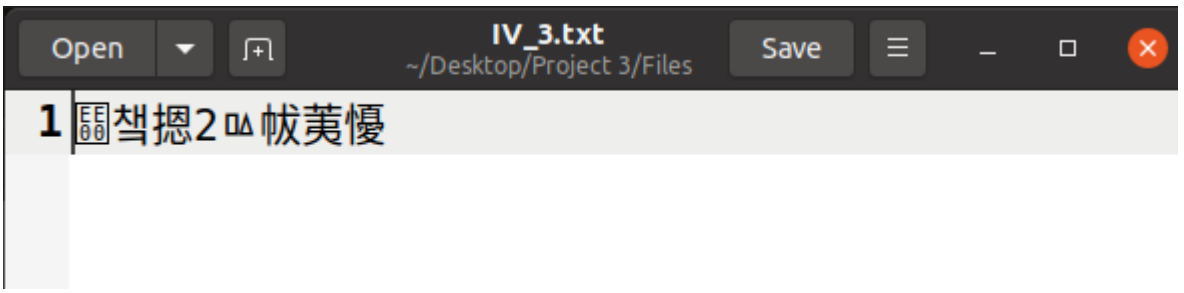
```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plai
n.txt -out IV_1.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plai
n.txt -out IV_2.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
```



IV_1.txt and IV_2.txt are encrypted with an iv of 0102030405060708. When the same plaintexts are used with the same IVs, the same ciphertexts are produced, thus after decryption, they produce the same plaintext again.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plai
n.txt -out IV_3.txt -K 00112233445566778889aabbccddeeff -iv 1122334
45566
hex string is too short, padding with zero bytes to length
```

IV_3.txt is encrypted with a different iv of 112233445566.



This generates a completely different ciphertext.

Using unique IVs is important as it helps defend against known-plaintext attacks. By using a different IV every time, it ensures that an adversary generates a different ciphertext each time, making it harder for the adversary to get clues to decrypt and interpret the original values.

6.2

Using known-plaintext-attack.py

```python
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
        return bytearray(x^y for x,y in zip(first, second))

MSG = "This is a known message!"
HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"
# Convert ascii/hex string to bytearray
D1 = bytes(MSG, "utf-8")
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)
r1 = xor(D1, D2)
r2 = xor(D2, D3)
r3 = xor(D2, D2)

r4 = xor(r2,D1)
print(r4)
print(r1.hex())
print(r2.hex())
print(r3.hex())
```

```
[11/13/22]seed@VM:~/.../Files$ python3 known-plaintext-attack.py
bytearray(b'Order: Launch a missile!')
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
1b1a0d165253536c0055050d07570f00000c0000080b0000
0000000000000000000000000000000000000000000000000
[11/13/22]seed@VM:~/.../Files$
```

P2 is equal to ‘Order: Launch a missile!’

For CFB: If the same IV is used, similarities might show up in the ciphertext. It reveals information about the common blocks shared by the 2 plaintexts.

6.3

Bob's plaintext is known to have two cases: (1) "Yes" and (2) "No".
We know that "Yes" will be padded to 16 bytes in CBC encryption: 59 65 73 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D. "No" will be completed as 4E 6F 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E.

Because Bob's plaintext has only two cases, we can first assume that the previous plaintext is “Yes’, and construct P2 to verify that C2 and ‘Yes’ have the same result after encryption.

Task 7: