



LAB REPORT

SWE430: Information and Network Security

Submitted to

Partha Protim Paul

Lecturer

Dept. of Software Engineering

Shahjalal University of Science and Technology

Submitted by

Md. Mehrajul Islam

2019831074

LAB 3

Symmetric encryption & hashing

TASK 3.1: AES encryption using different modes

Firstly, I opened a text file named **task1.txt**

Then I gave the following commands:

Using AES128 CBC

```
openssl enc -aes-128-cbc -e -in task1.txt -out task1aes128cbc.bin  
-K 77942123545646274856abcdcedaeab  
-iv 33355500033347349994444453555222
```

```
openssl enc -aes-128-cbc -d -in task1aes128cbc.bin  
-out task1aes128cbcdecrypted.txt  
-K 77942123545646274856abcdcedaeab -iv  
33355500033347349994444453555222
```

Using AES128 CFB

```
openssl enc -aes-128-cfb -e -in task1.txt -out task1aes128cfb.bin  
-K 77942123545646274856abcdcedaeab  
-iv 33355500033347349994444453555222
```

```
openssl enc -aes-128-cfb -d -in task1aes128cfb.bin  
-out task1aes128cfbdycrypted.txt  
-K 77942123545646274856abcdcedaeab  
-iv 33355500033347349994444453555222
```

Using AES128 ECB

```
openssl enc -aes-128-ecb -e -in task1.txt -out task1aes128ecb.bin  
-K 77942123545646274856abcdcedaeab
```

```
openssl enc -aes-128-ecb -d -in task1aes128ecb.bin  
-out task1aes128ecbdecrypted.txt  
-K 77942123545646274856abcdcedaeab
```

To read the encrypted binary files I used ghex:

ghex task1aes128cbc.bin &

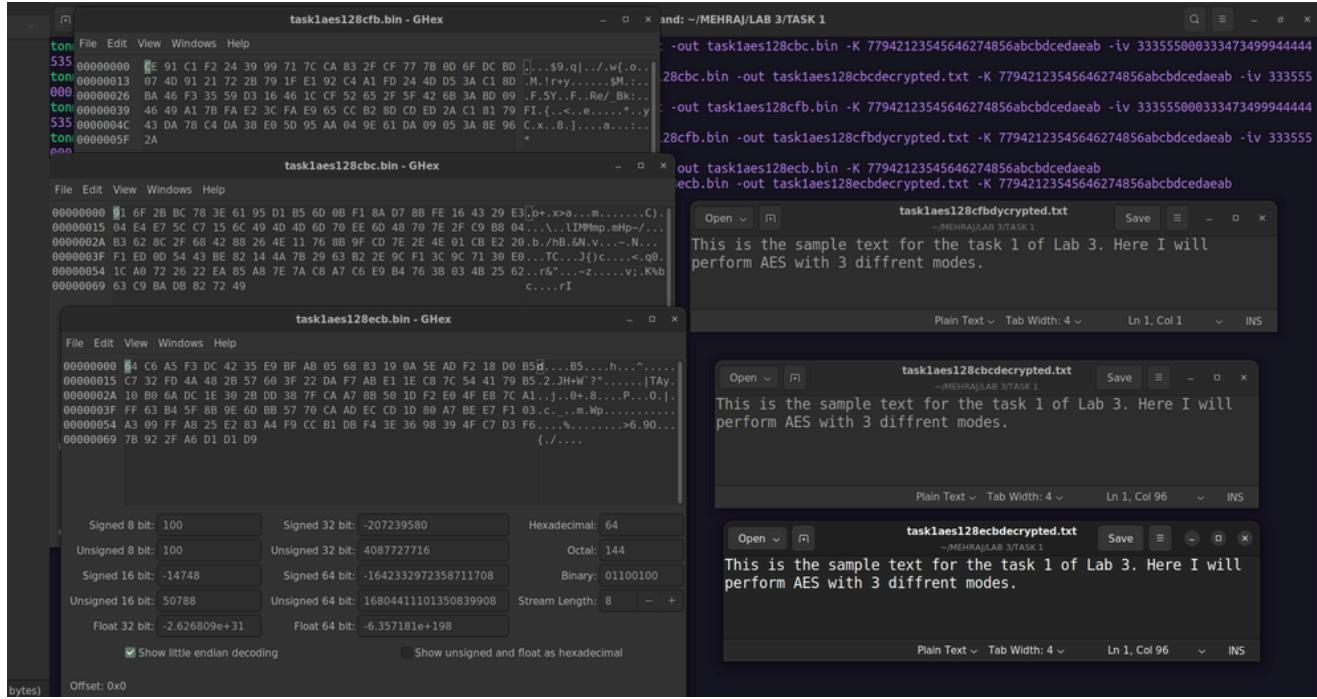
ghex task1aes128cfb.bin &

ghex task1aes128ecb.bin &

These are the commands that I gave

```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-cbc -e -in task1.txt -out task1aes128cbc.bin -K 77942123545646274856abcbcdcedaeab -iv 333555000333473499944444  
53555222  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-cbc -d -in task1aes128cbc.bin -out task1aes128cbcdecrypted.txt -K 77942123545646274856abcbcdcedaeab -lv 333555  
0003334734999444453555222  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-cfb -e -in task1.txt -out task1aes128cfb.bin -K 77942123545646274856abcbcdcedaeab -iv 333555000333473499944444  
53555222  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-cfb -d -in task1aes128cfb.bin -out task1aes128cfbdecrypted.txt -K 77942123545646274856abcbcdcedaeab -iv 333555  
0003334734999444453555222  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-ecb -e -in task1.txt -out task1aes128ecb.bin -K 77942123545646274856abcbcdcedaeab  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ openssl enc -aes-128-ecb -d -in task1aes128ecb.bin -out task1aes128ecbdecrypted.txt -K 77942123545646274856abcbcdcedaeab  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ ghex task1aes128cbc.bin &  
[1] 41838  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ ghex task1aes128cfb.bin &  
[2] 41860  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$ ghex task1aes128ecb.bin &  
[3] 41882  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 1$
```

This are the encrypted and decrypted files



TASK 3.2: ENCRYPTION MODE - ECB VS CBC

I downloaded the logo of SWE Society and converted it to bmp file.
The name of the file is swe.bmp

ENCRYPT USING AES128-ECB

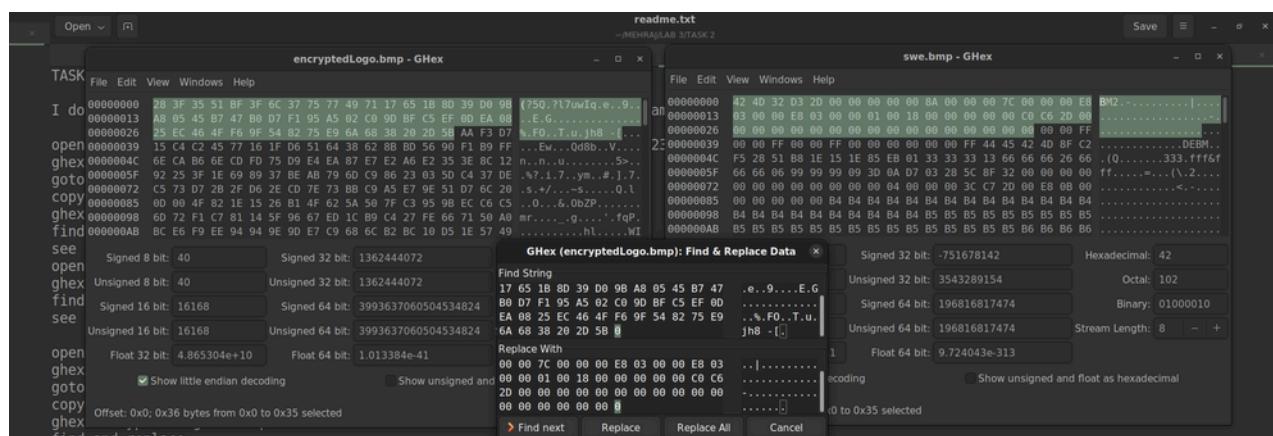
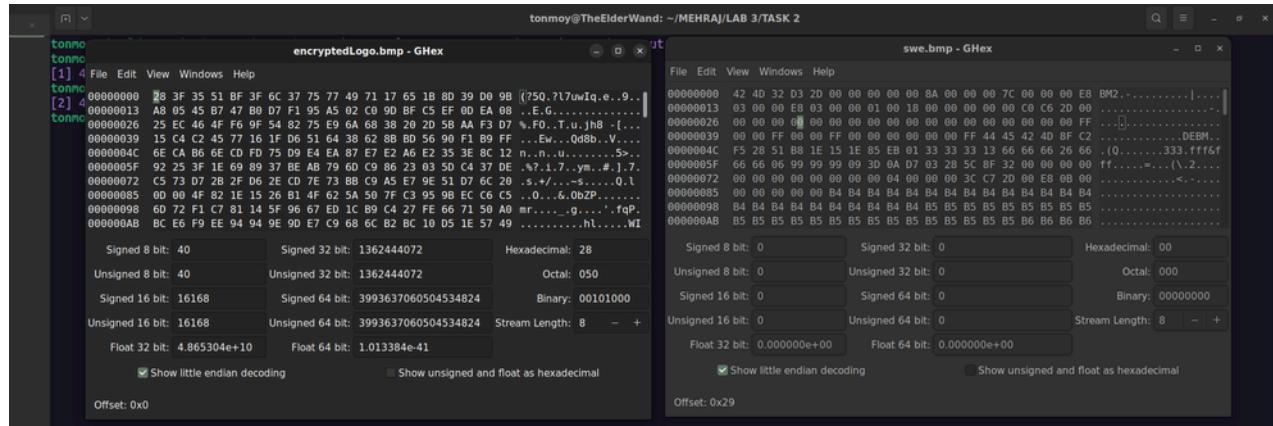
```
openssl enc -aes-128-ecb -e -in swe.bmp -out encryptedLogo.bmp  
-K 77942123545646274856abcbdcdaeaab  
ghex swe.bmp &
```

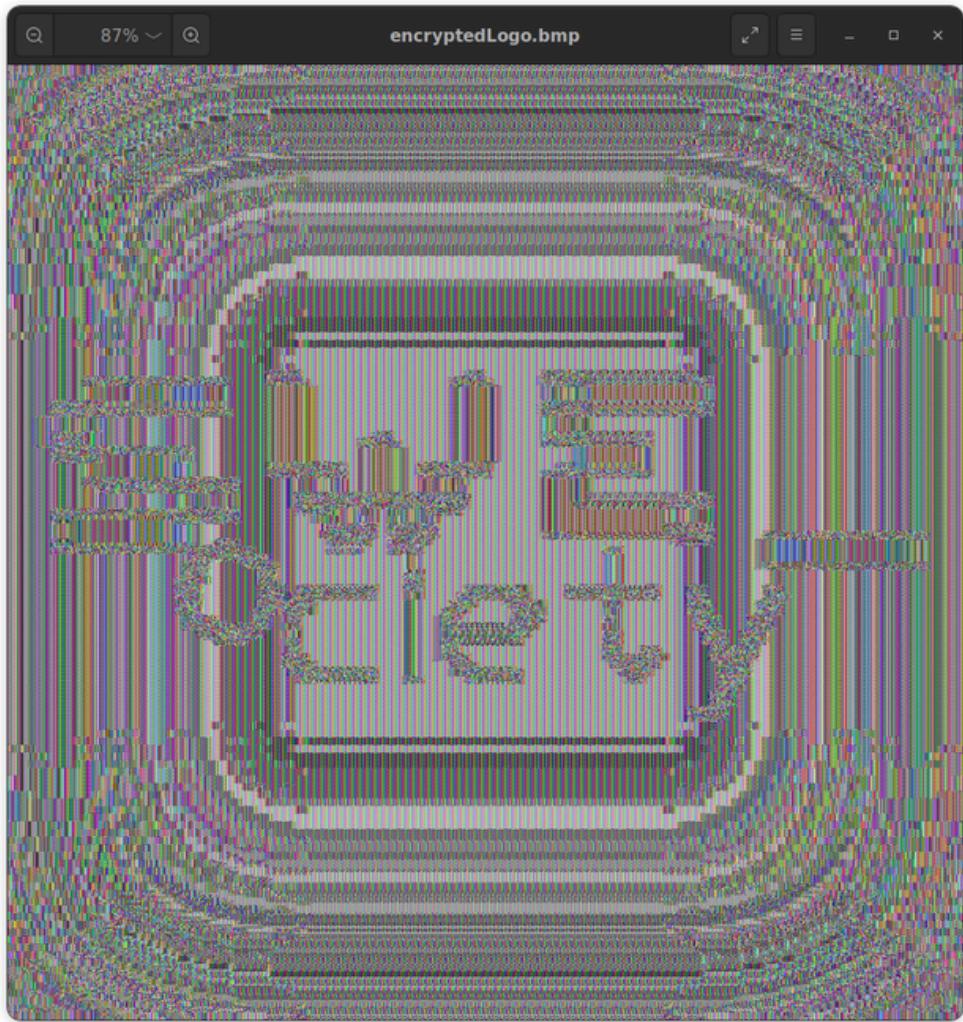
```
ॐ श 6 03:29
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 2
EHRAJ/LAB 3/TASK 2$ openssl enc -aes-128-ecb -e -in swe.bmp -out encryptedLogo.bmp -K 77942123545646274856abcbcdcedaeab
EHRAJ/LAB 3/TASK 2$ ghex swe.bmp &
EHRAJ/LAB 3/TASK 2$ ghex encryptedLogo.bmp &
EHRAJ/LAB 3/TASK 2$
```

Now the encrypted image cannot be read due to the header encryption so now I will take the first 52 bits from swe.bmp and replace those bits in the encrypted image

PROCEDURE:

```
    goto byte 54 of swe.bmp  
        copy 54 bits  
ghex encryptedLogo.bmp &  
        goto byte 54  
        find and replace
```



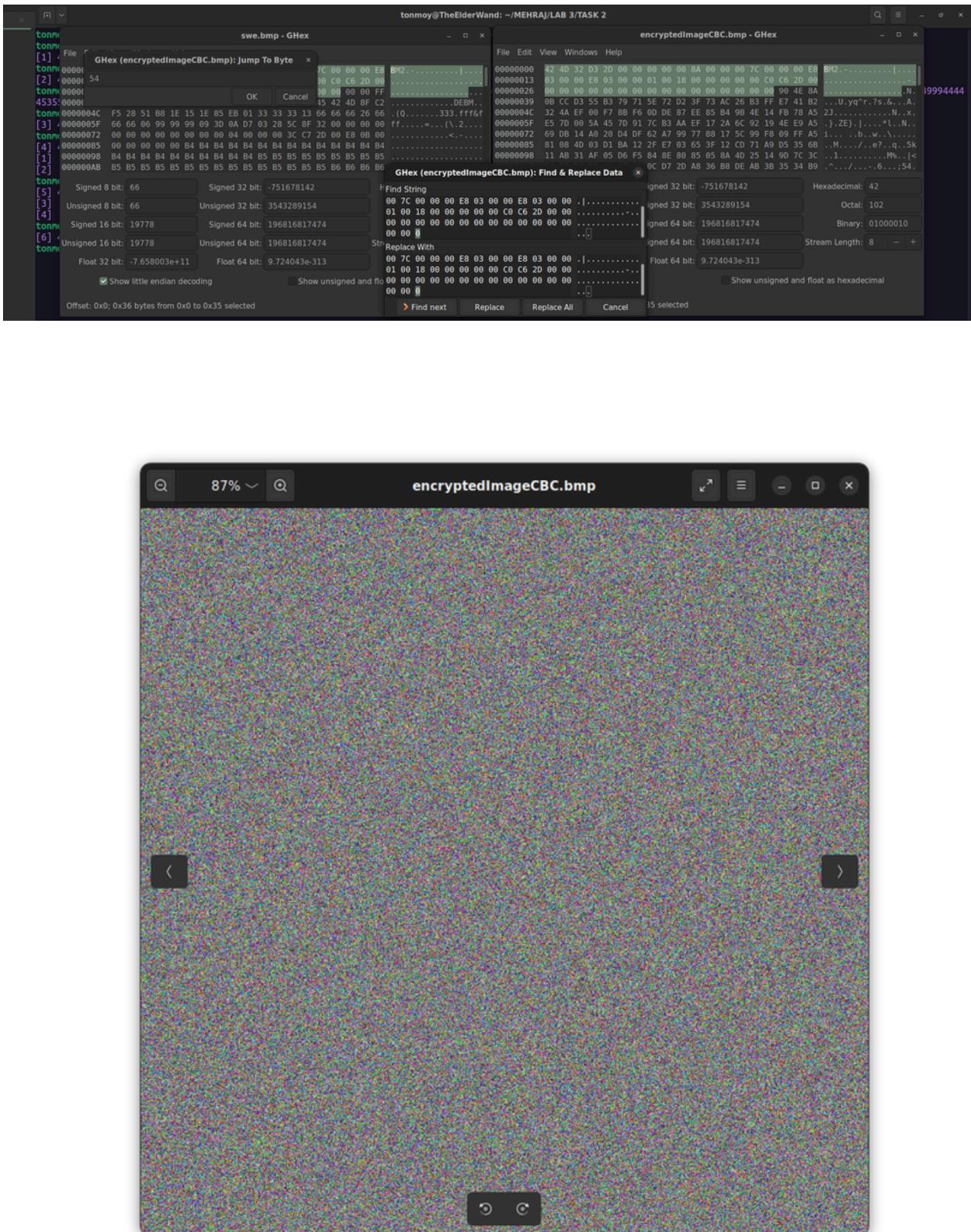


This is the output of the encryption for AES128-ECB

USING AES128-CBC

```
openssl enc -aes-128-cbc -e -in swe.bmp -out encryptedImageCBC.bmp  
-K 77942123545646274856abcbcdcedaeab  
-iv 33355500033347349994444453555222  
ghex swe.bmp &
```

```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ openssl enc -aes-128-ecb -e -in swe.bmp -out encryptedLogo.bmp -K 77942123545646274856abcbcdcedaeab  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ ghex swe.bmp &  
[1] 42415  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ ghex encryptedLogo.bmp &  
[2] 42444  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ openssl enc -aes-128-cbc -e -in swe.bmp -out encryptedImageCBC.bmp -K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
453555222  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ ghex swe.bmp &  
[3] 42718  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$ ghex encryptedImageCBC.bmp &  
[4] 42749  
[1] Done ghex swe.bmp  
[2] Done ghex encryptedLogo.bmp  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 2$
```



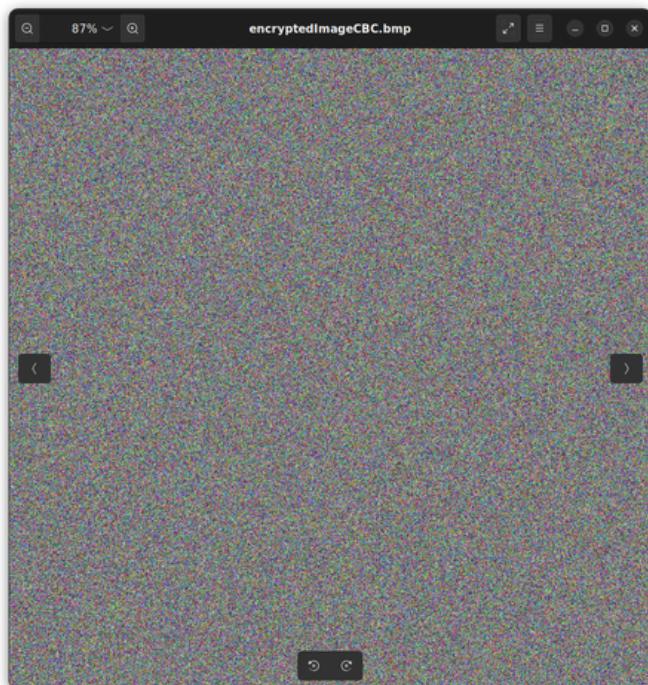
This is the output of the encryption for AES128-CBC

ECB & CBC Mode Analysis

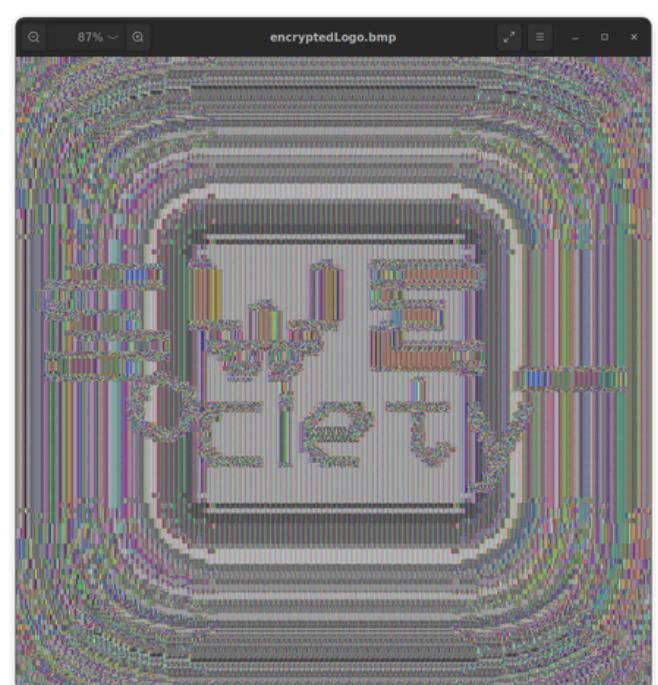
CBC (Cipher Block Chaining) and ECB (Electronic Codebook) are both block cipher encryption. where each block of plaintext is encrypted independently with the same key in ECB mode. It lacks diffusion, meaning identical blocks of plaintext will result in identical blocks of ciphertext. This can lead to security vulnerabilities, especially in image encryption, where patterns might be preserved. On the other hand, In CBC mode, each plaintext block is XORed with the previous ciphertext block before encryption. This adds diffusion, making it more resistant to patterns and repetition in the plaintext. CBC mode requires an Initialization Vector (IV) for the first block to start the chaining process.

COMPAISON:

ECB is simpler and faster, but less secure, especially for images. CBC is more secure due to its chaining mechanism, but slightly slower and requires an IV. For image encryption, CBC is generally preferred over ECB due to its better resistance to pattern preservation and higher security.



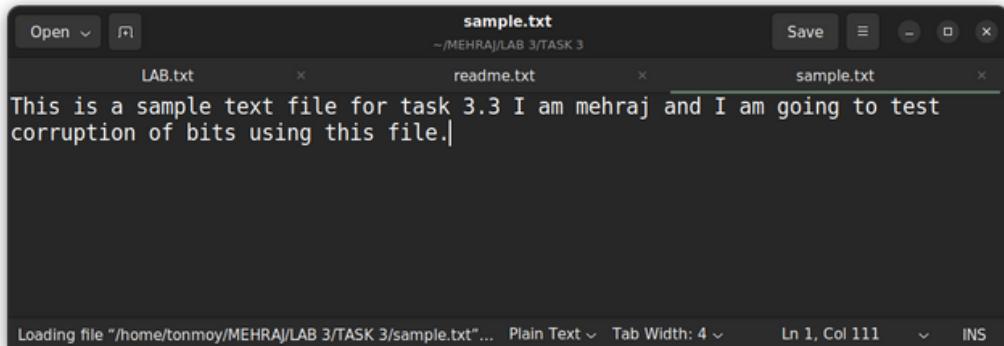
CBC



ECB

TASK 3.3: ENCRYPTION MODE : CORRUPTED CIPHERTEXT

Firstly, I created a text file sample.txt

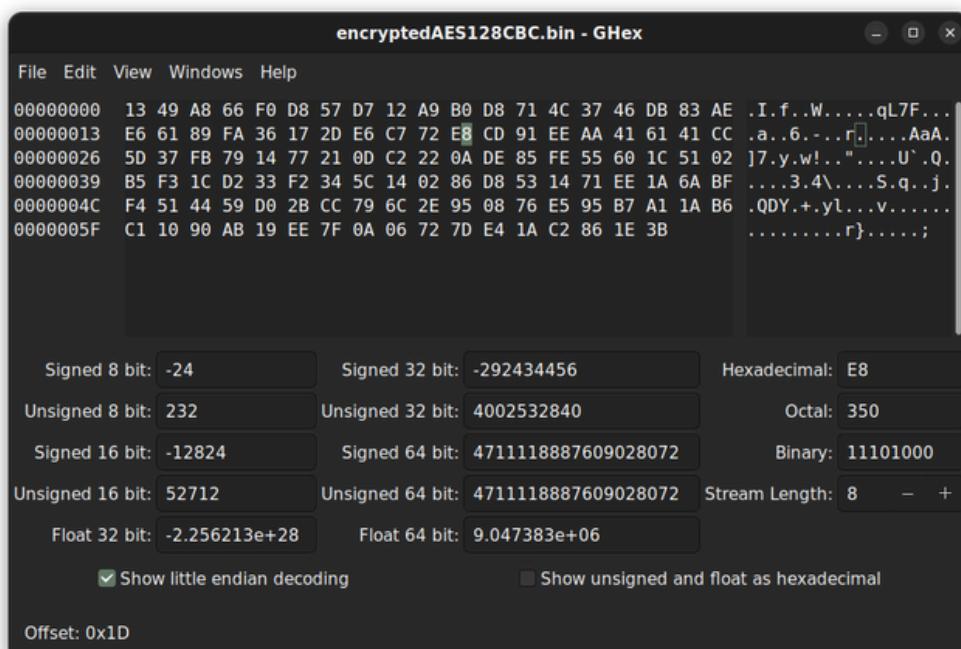


FOR CBC MODE:

```
openssl enc -aes-128-cbc -e -in sample.txt -out encryptedAES128CBC.bin  
-K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
ghex encryptedAES128CBC.bin &
```

Then I went to 30th bit and changed the HEX value

```
openssl enc -aes-128-cbc -d -in encryptedAES128CBC.bin -out decryptedAES128CBC.txt  
-K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222
```



```
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ touch sample.txt  
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ nano sample.txt  
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aes-128-cbc -e -in sample.txt -out encryptedAES128CBC.bin -K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128CBC.bin &  
[1] 12142  
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aes-128-cbc -d -in encryptedAES128CBC.bin -out decryptedAES128CBC.txt -K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
[1]+ Done ghex encryptedAES128CBC.bin  
tomoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$
```

```
Open decryptedAES128CBC.txt Save
LAB.txt x readme.txt x decryptedAES128CBC.txt x
This is a sampletextqvy$Eq>Vg4ask 3.3 I am behraj and I am going to test
corruption of bits using this file.|
```

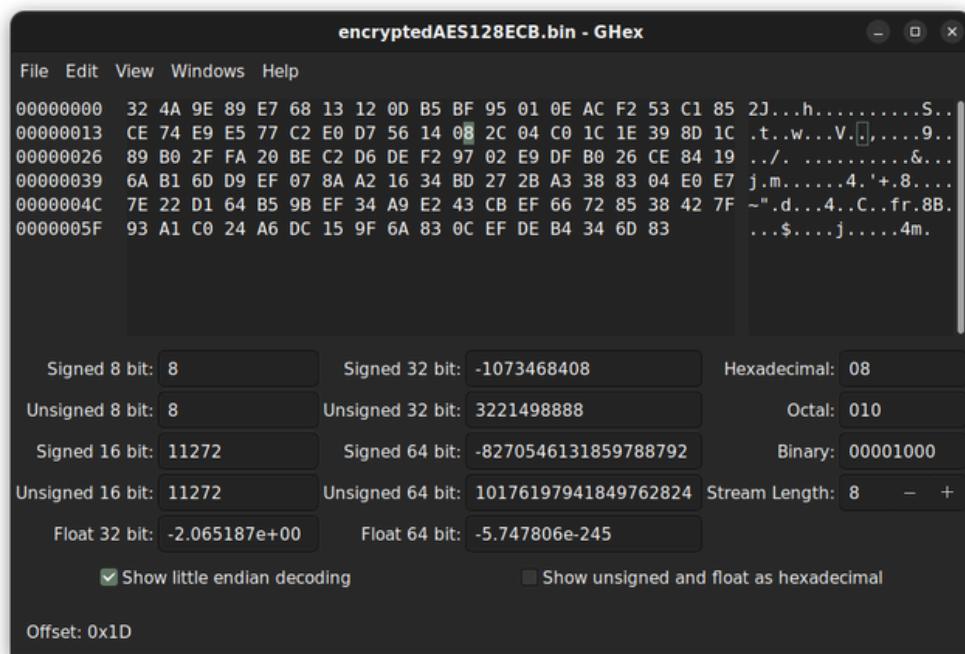
Plain Text Tab Width: 4 Ln 1, Col 111 INS

FOR ECB MODE

```
openssl enc -aes-128-ecb -e -in sample.txt -out encryptedAES128ECB.bin
-K 77942123545646274856abcbcdcedaeab
ghex encryptedAES128ECB.bin &
```

Then I went to 30th bit and changed the HEX value

```
openssl enc -aes-128-ecb -d -in encryptedAES128ECB.bin -out decryptedAES128ECB.txt
-K 77942123545646274856abcbcdcedaeab
```



```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aes-128-ecb -e -in sample.txt -out encryptedAES128ECB.bin -K 77942123545646274856abcbcdcedaeab
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128ECB.bin &
[1] 12482
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aes-128-ecb -d -in encryptedAES128ECB.bin -out decryptedAES128ECB.txt -K 77942123545646274856abcbcdcedaeab
[1]+ Done ghex encryptedAES128ECB.bin
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$
```

Open Save

decryptedAES128ECB.txt

~/MEHRAJ/LAB 3/TASK 3

LAB.txt readme.txt decryptedAES128ECB.txt

This is a sample|éüH+tn|||||I*Ãask 3.3 I am mehraj and I am going to test corruption of bits using this file.|

Plain Text Tab Width: 4 Ln 1, Col 111 INS

FOR CFB MODE

```
openssl enc -aria-128-cfb -e -in sample.txt -out encryptedAES128CFB.bin
-K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
ghex encryptedAES128CFB.bin &
```

Then I went to 30th bit and changed the HEX value

```
openssl enc -aria-128-cfb -d -in encryptedAES128CFB.bin -out decryptedAES128CFB.txt
-K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
```

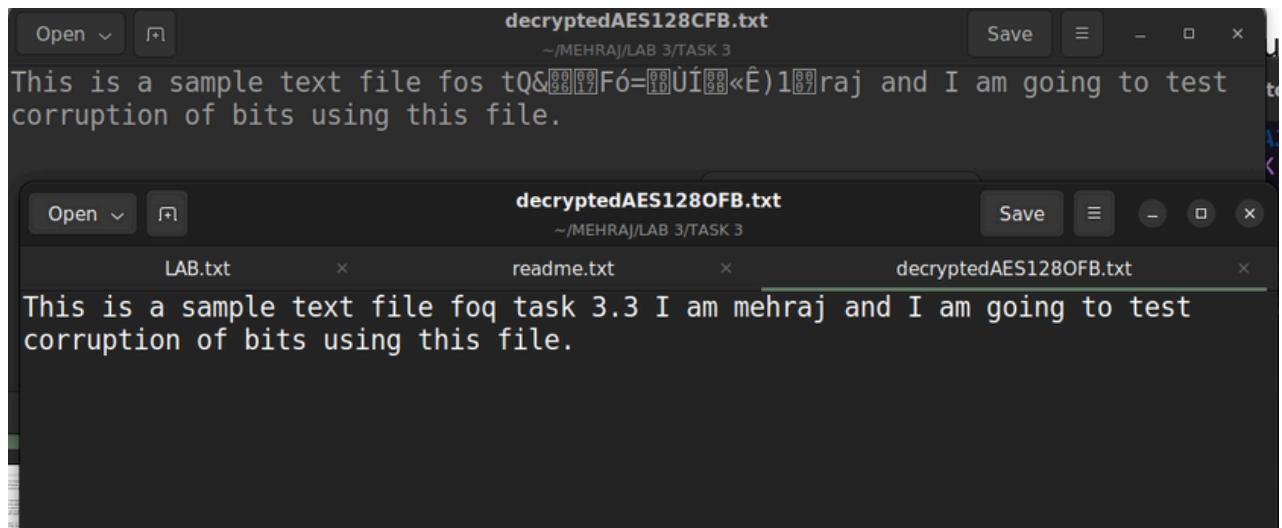
FOR OFB MODE

```
openssl enc -aria-128-ofb -e -in sample.txt -out encryptedAES128OFB.bin
-K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
ghex encryptedAES128OFB.bin &
```

Then I went to 30th bit and changed the HEX value

```
openssl enc -aria-128-ofb -d -in encryptedAES128OFB.bin -out decryptedAES128OFB.txt
-K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
```

```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aria-128-cfb -e -in sample.txt -out encryptedAES128CFB.bin -K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128CFB.bin &
[1] 12860
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aria-128-cfb -d -in encryptedAES128CFB.bin -out decryptedAES128CFB.txt -K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128CFB.txt &
[1]+ Done ghex encryptedAES128CFB.txt
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aria-128-ofb -e -in sample.txt -out encryptedAES128OFB.bin -K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128OFB.bin &
[1] 13078
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ openssl enc -aria-128-ofb -d -in encryptedAES128OFB.bin -out decryptedAES128OFB.txt -K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex decryptedAES128OFB.txt &
```



Here these are the results of CFB & OFB

ANALYSIS

ECB Mode

Corrupted Information: "|éüH÷tn\x\x\x\x\¿*\x{f}" (16 bytes)

Explanation: In ECB mode, each block is encrypted independently. Therefore, the corruption only affects the specific block where it occurred. The rest of the plaintext remains intact.

CBC Mode:

Corrupted Information: "tÈ3«qvy\$Æq÷Vq¤4" (16 bytes)

Explanation: While CBC mode encrypts blocks in a chained manner, the corruption in one block affects the decryption of subsequent blocks. However, the first block can still be recovered as it only depends on the IV and the first block of ciphertext.

CFB Mode:

Explanation: In CFB mode, the corruption in one ciphertext block affects the decryption of subsequent blocks. However, due to the feedback mechanism, only a part of the subsequent blocks is corrupted. As a result, more information can be recovered compared to CBC mode.

OFB Mode:

Corrupted Information: "q" (1 bytes)

Explanation: Similar to CFB mode, the corruption in one ciphertext block affects the decryption of subsequent blocks. However, in OFB mode, the feedback mechanism is independent of the plaintext, resulting in a similar recovery as in CFB mode.

IMPLICATIONS:

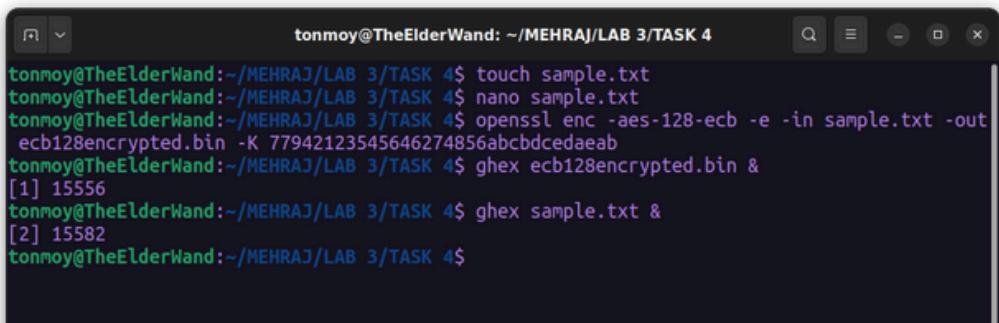
- ECB mode offers the least security and diffusion, as evident from the limited recoverable information.
- CBC mode provides better security compared to ECB but still suffers from partial corruption propagation.
- CFB offers improved diffusion, allowing for more recoverable information compared to CBC mode, although still susceptible to partial corruption propagation.
- OFB gives the best result among them, as it has only 1 corrupted bit and all other bits have been recovered being significant improvements.

TASK 3.4: PADDING

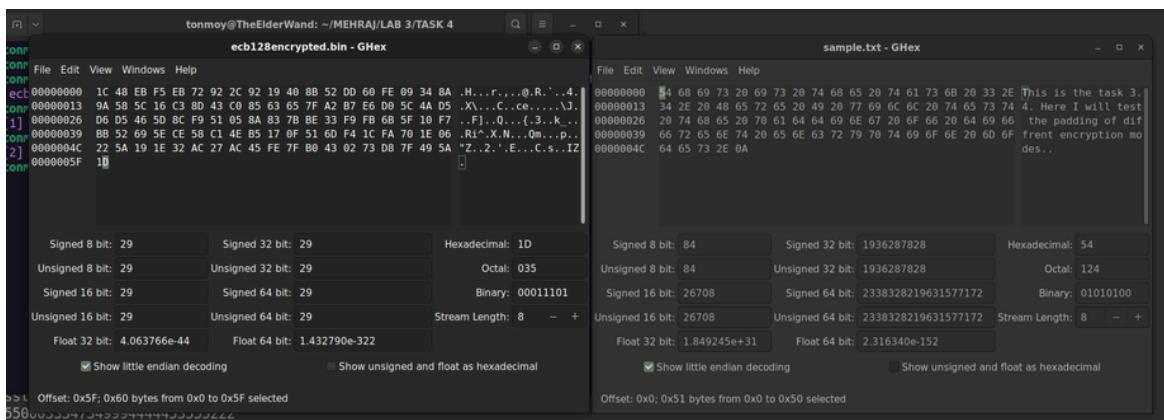
Firstly I created a text file sample.txt

FOR ECB

```
openssl enc -aes-128-ecb -e -in sample.txt -out ecb128encrypted.bin  
-K 77942123545646274856abcbcdcedaeab  
ghex ecb128encrypted.bin &  
ghex sample.txt &
```



```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ touch sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ nano sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ openssl enc -aes-128-ecb -e -in sample.txt -out  
ecb128encrypted.bin -K 77942123545646274856abcbcdcedaeab  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex ecb128encrypted.bin &  
[1] 15556  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex sample.txt &  
[2] 15582  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$
```

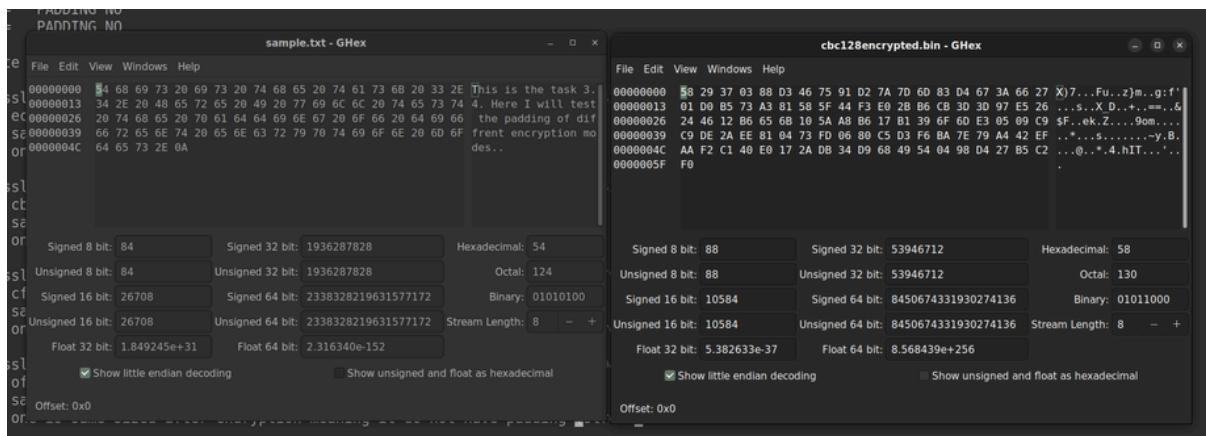


ECB generates larger output after encryption meaning it has padding

FOR CBC

```
openssl enc -aes-128-cbc -e -in sample.txt -out cbc128encrypted.bin  
-K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
ghex cbc128encrypted.bin &  
ghex sample.txt &
```

```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ openssl enc -aes-128-cbc -e -in sample.txt -out cbc128encrypted.bin -K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
44453555222  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex cbc128encrypted.bin &  
[1] 15740  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex sample.txt &  
[2] 15764  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$
```

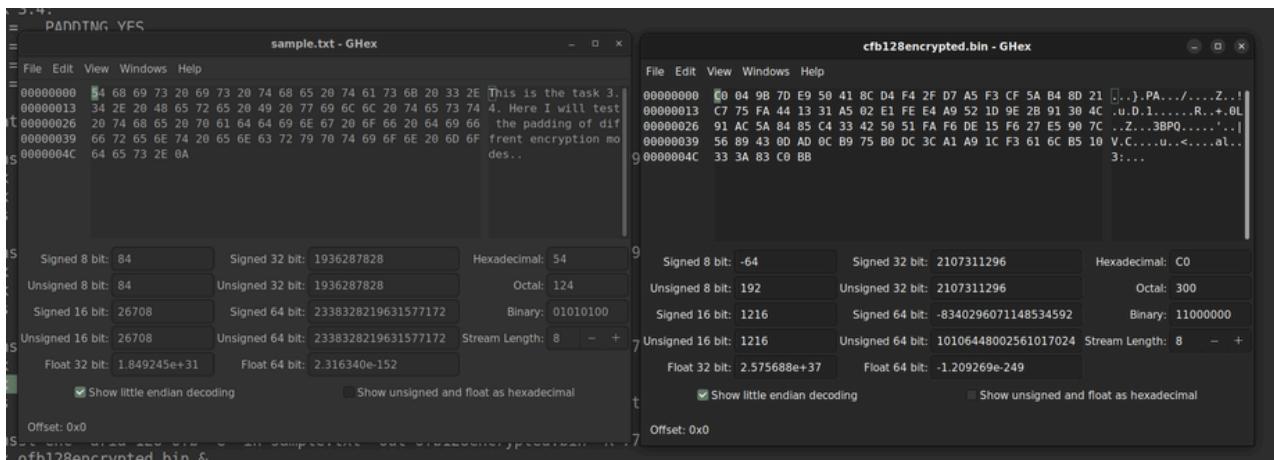


CBC generates larger output after encryption meaning it has padding

FOR CFB

```
openssl enc -aria-128-cfb8 -e -in sample.txt -out cfb128encrypted.bin  
-K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
ghex cfb128encrypted.bin &  
ghex sample.txt &
```

```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ openssl enc -aria-128-cfb8 -e -in sample.txt -out cfb128encrypted.bin -K 77942123545646274856abcbcdcedaeab -iv 33355500033347349994444453555222  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex cfb128encrypted.bin &  
[1] 17252  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$ ghex sample.txt &  
[2] 17273  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4$
```

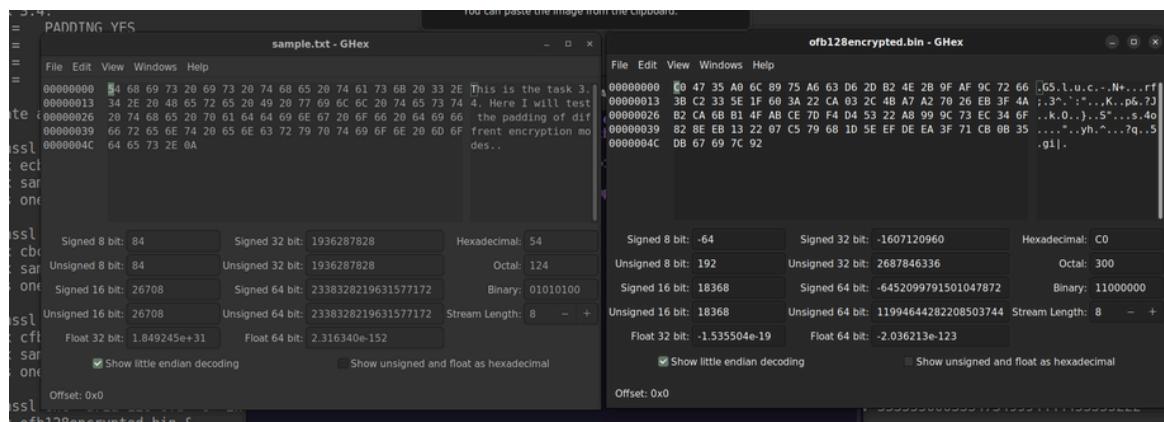


CFB do not generate larger output after encryption meaning it do not have padding

FOR OFB

```
openssl enc -aria-128-ofb -e -in sample.txt -out ofb128encrypted.bin
-K 77942123545646274856abcbcdcedaeab -iv 3335550003334734999444453555222
ghex ofb128encrypted.bin &
ghex sample.txt &
```

```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 4
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 4$ openssl enc -aria-128-ofb -e -in sample.txt
-out ofb128encrypted.bin -K 77942123545646274856abcbcdcedaeab -iv 3335550003334734999444
453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 4$ ghex ofb128encrypted.bin &
[1] 17374
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 4$ ghex sample.txt &
[2] 17396
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 4$ 
```



OFB do not generate larger output after encryption meaning it do not have padding

SO MY FINDINGS ARE:

ECB HAS PADDING
CBC HAS PADDING

CFB DON'T HAVE PADDING
OFB DON'T HAVE PADDING

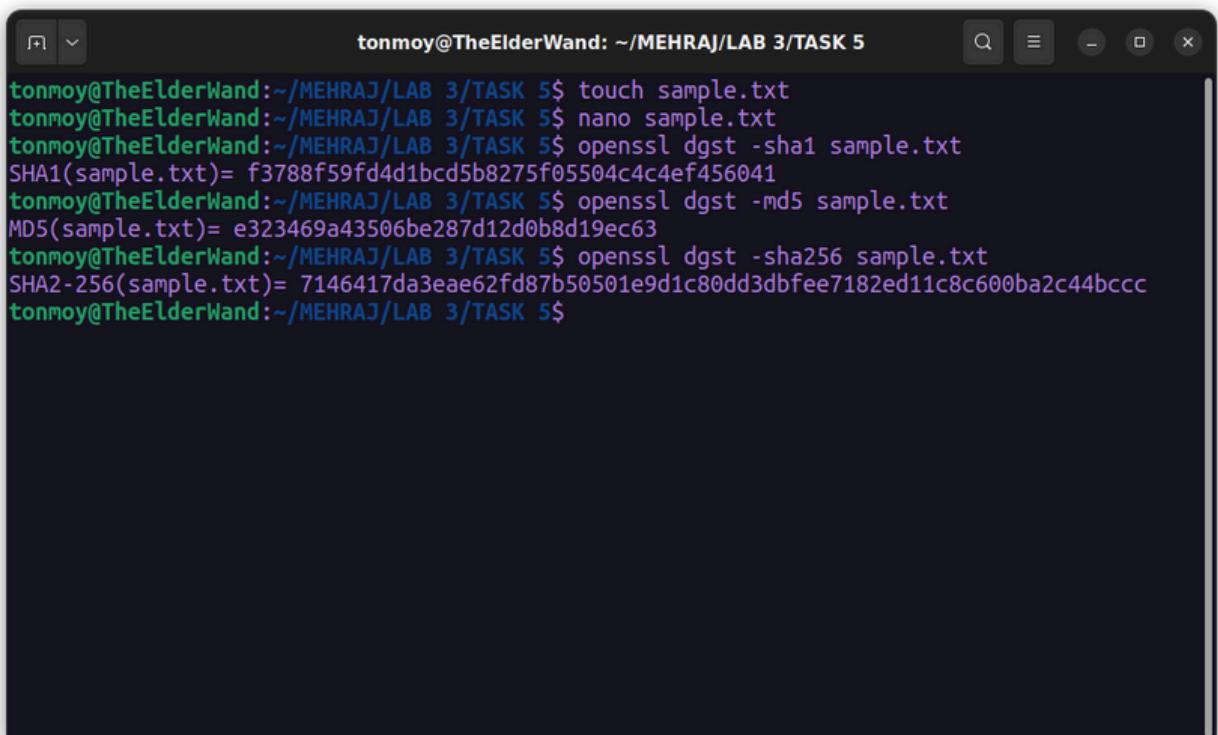
TASK 3.5: GENERATING MESSAGE DIGEST

I created a text file named sample.txt

```
openssl dgst -sha1 sample.txt  
output: f3788f59fd4d1bcd5b8275f05504c4c4ef456041
```

```
openssl dgst -md5 sample.txt  
output: e323469a43506be287d12d0b8d19ec63
```

```
openssl dgst -sha256 sample.txt  
output: 7146417da3eae62fd87b50501e9d1c80dd3dbfee7182ed11c8c600ba2c44bccc
```



```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$ touch sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$ nano sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$ openssl dgst -sha1 sample.txt  
SHA1(sample.txt)= f3788f59fd4d1bcd5b8275f05504c4c4ef456041  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$ openssl dgst -md5 sample.txt  
MD5(sample.txt)= e323469a43506be287d12d0b8d19ec63  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$ openssl dgst -sha256 sample.txt  
SHA2-256(sample.txt)= 7146417da3eae62fd87b50501e9d1c80dd3dbfee7182ed11c8c600ba2c44bccc  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 5$
```

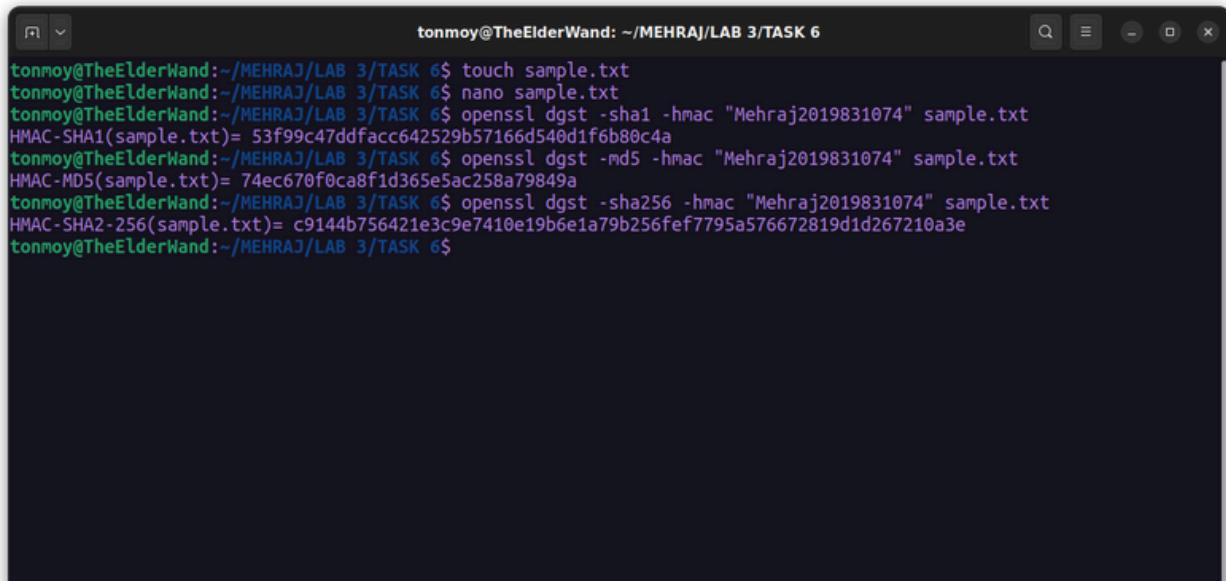
TASK 3.6: KEYED HASH AND HMAC

I created a text file named sample.txt

```
openssl dgst -sha1 -hmac "Mehraj2019831074" sample.txt  
output: 53f99c47ddfacc642529b57166d540d1f6b80c4a
```

```
openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
output: 74ec670f0ca8f1d365e5ac258a79849a
```

```
openssl dgst -sha256 -hmac "Mehraj2019831074" sample.txt  
output: c9144b756421e3c9e7410e19b6e1a79b256fef7795a576672819d1d267210a3e
```



```
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$ touch sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$ nano sample.txt  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$ openssl dgst -sha1 -hmac "Mehraj2019831074" sample.txt  
HMAC-SHA1(sample.txt)= 53f99c47ddfacc642529b57166d540d1f6b80c4a  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$ openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
HMAC-MD5(sample.txt)= 74ec670f0ca8f1d365e5ac258a79849a  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$ openssl dgst -sha256 -hmac "Mehraj2019831074" sample.txt  
HMAC-SHA2-256(sample.txt)= c9144b756421e3c9e7410e19b6e1a79b256fef7795a576672819d1d267210a3e  
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 6$
```

TASK 3.7: KEYED HASH AND HMAC

I created a text file named sample.txt

Then I created a program for counting hash match named hashMatchCount.py



```
hashMatchCount.py  
C: > Users > coder > Downloads > MEHRAJ > MEHRAJ > LAB 3 > TASK 7 > hashMatchCount.py  
1  hash1 = "f965345a146ee5f09984afc05f98b3f25a5086ea6ffbf6efb9a603ff8f2c44aa"  
2  hash2 = "0be68ac4be378c41126836087667298f69b1915f80309e6c142ea4b7d15d5296"  
3  
4  pointer = 0  
5  numOfMatched = 0  
6  for digit in hash1:  
7      if digit==hash2[pointer]:  
8          numOfMatched = numOfMatched + 1  
9      pointer = pointer + 1  
10  
11 print("Number of Matched : ", numOfMatched)  
12
```

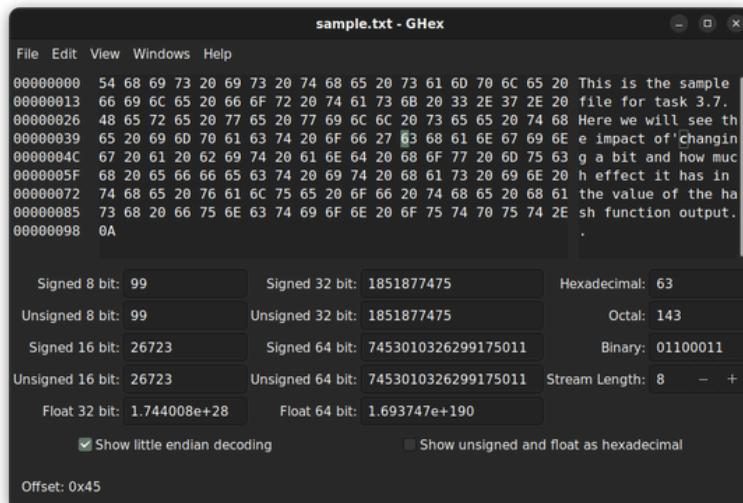
FOR MD5

```
openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
output: 23566c525240bb80668939455bdd21ae  
ghex sample.txt &
```

Then I changed a random bit

```
openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
output: c0ce651211c472a3549bbb8ae5b34529
```

Then updated the python code (replaced hash values)



```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ touch hashMatchCount.py  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ nano hashMatchCount.py  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ touch sample.txt  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ nano sample.txt  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
HMAC-MD5(sample.txt)= 23566c525240bb80668939455bdd21ae  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ ghex sample.txt &  
[1] 20173  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$  
[1]+ Done ghex sample.txt  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ openssl dgst -md5 -hmac "Mehraj2019831074" sample.txt  
HMAC-MD5(sample.txt)= c0ce651211c472a3549bbb8ae5b34529  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ python3 hashMatchCount.py  
Number of Matched : 2  
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$
```

Here number of matched characters is: 2

FOR SHA256

update the text file text.txt to the previous main content

openssl dgst -sha256 -hmac "Mehraj2019831074" text.txt
output:

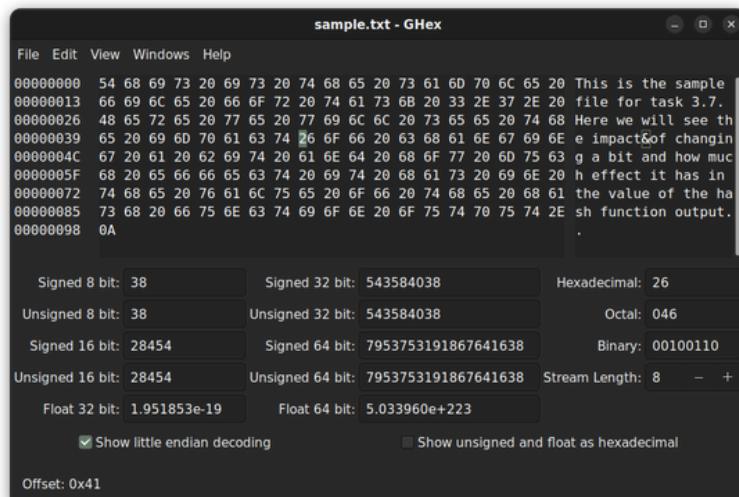
923b829a244a667348611ffe93016ddf798224b71a0ad6c650e68f9e10cd1c6e
ghex text.txt &

Then I changed a random bit

openssl dgst -sha256 -hmac "Mehraj2019831074" text.txt
output:

5f4a04044aa5e14da6887551c00326cac4abba24355c076c38fb9f8dd6151e8f

Then updated the python code (replaced hash values)



```
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ openssl dgst -sha256 -hmac "Mehraj2019831074" sample.txt
HMAC-SHA2-256(sample.txt)= f965345a146ee5f09984afc05f98b3f25a5086ea6ffbf6efb9a603ff8f2c44aa
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ ghex sample.txt &
[1] 20303
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ openssl dgst -sha256 -hmac "Mehraj2019831074" sample.txt
HMAC-SHA2-256(sample.txt)= 0be68ac4be378c41126836087667298f69b1915f80309e6c142ea4b7d15d5296
[1]+ Done ghex sample.txt
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$ python3 hashMatchCount.py
Number of Matched : 0
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 7$
```

Here number of matched characters is: 0

So, changing even a bit changes the hash value completely