



# 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*

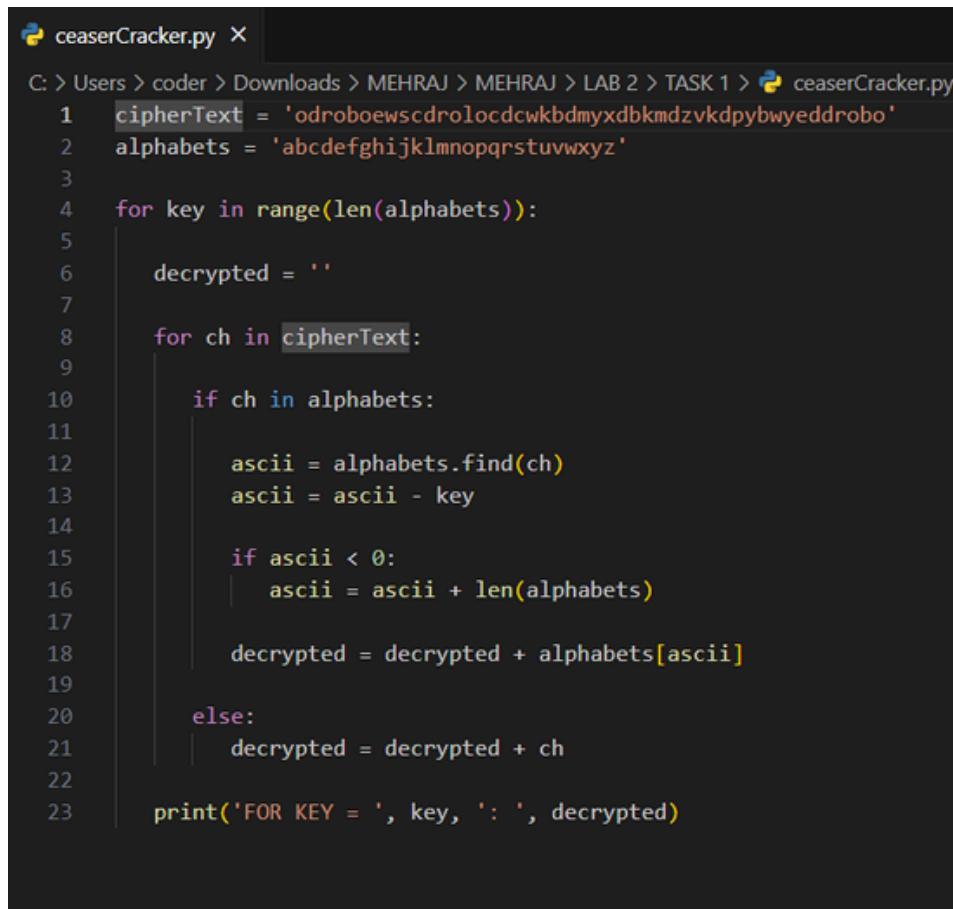
**2019831074**

# LAB 2

## Attacking Classic Crypto Systems

### Checkpoint 1: Breaking Caesar Cipher

**Given Cipher:** odroboewscdroloccwkbmymxdbkmdzvkdpybwyyeddrobo



```
ceaserCracker.py ×
C: > Users > coder > Downloads > MEHRAJ > MEHRAJ > LAB 2 > TASK 1 > ceaserCracker.py
1 cipherText = 'odroboewscdroloccwkbmymxdbkmdzvkdpybwyyeddrobo'
2 alphabets = 'abcdefghijklmnopqrstuvwxyz'
3
4 for key in range(len(alphabets)):
5
6     decrypted = ''
7
8     for ch in cipherText:
9
10         if ch in alphabets:
11
12             ascii = alphabets.find(ch)
13             ascii = ascii - key
14
15             if ascii < 0:
16                 ascii = ascii + len(alphabets)
17
18             decrypted = decrypted + alphabets[ascii]
19
20         else:
21             decrypted = decrypted + ch
22
23     print('FOR KEY = ', key, ': ', decrypted)
```

#### Algorithm:

- There can be possible 25 keys. So first a loop with **key 0 to 25**
- In each iteration for each character in ciphertext get its ASCII and deduct key value from that
- Add the character in the final string and print them after each iteration
- Lastly go through all of them and find the meaningful one

# OUTPUT

```
● PS C:\Users\coder> & C:/Python312/python.exe "c:/Users/coder/Downloads/decipher.py"
FOR KEY =  0 : odroboewscdrolocdcwkdmyxdbkmdzvkdpybwedrobo
FOR KEY =  1 : ncqnandvrbcqnknbcvjaclxwcajlcuyjcoxavxdccqnan
FOR KEY =  2 : mbpmzmcuqabpmjmabauizbkwbzikbxtbnwzuwcbbpmzm
FOR KEY =  3 : laolylbtptaolilzazthyajvuayhjawshamvytvbaolyl
FOR KEY =  4 : kznkxkasoyznkhkyzsgxziutzxgizvrgzluxsuazznkxk
FOR KEY =  5 : jymjwjzrnxymjgjxyxrwyhtsywfhyuqfyktwrtymjwj
FOR KEY =  6 : ixliviyqmwxlifiwxwqevxgsrxvegxtpejsvqsyxxlivi
FOR KEY =  7 : hwhuhxp1vwkhehvwpduwfrquwfsodwiruprxwkhu
FOR KEY =  8 : gvjgtgwokujgdguvuoctveqptcevrncvhqtoqwwvjgtg
FOR KEY =  9 : fuifsfvnjtuifcftutnbsudpousbduqmbugpsnpvuui
FOR KEY = 10 : ethereumis the best smart contract platform out there
FOR KEY = 11 : dsgdqdltlhrsgdadrsrlzqsbnmsqzsokzsenqlntssgdqd
FOR KEY = 12 : crfcpcskgqrfczcrqkyprramlrpyarnjyrdmpkmsrrfcpc
FOR KEY = 13 : bqebobjrfpqebypqpxoqzlkoqxzqmixqclojlrqqebob
FOR KEY = 14 : apdanaqieopdaxaopoiwnpykjpnwyp1hwpbknikqppdana
FOR KEY = 15 : zoczmzphdnoczwznonhvmoxiomvxokgvoajmhjpooczmz
FOR KEY = 16 : ynbylyogcmnbyvymnmngulnwihnluwnjfunzilgionnbyly
FOR KEY = 17 : xmaxkxnfblmaxuxlmlftkmvhgmktvmietmyhkfhnmmaxkx
FOR KEY = 18 : wlzwjwmeaklzwtklkesjlugfljsulhdslxgjegmllzwjw
FOR KEY = 19 : vkyvivldzjkyvsvjkjdriktfekirtkgcrkwfidflkkyviv
FOR KEY = 20 : ujxuhukcyijxuruijicqhjsedjhqsjfbqjvehcekjjxuhu
FOR KEY = 21 : tiwtgtjbxhiwtqthihbpgirdcigpriapiudgbdjiwtgt
FOR KEY = 22 : shvsfsiawghvpspsghgaofhqcbhfoqhdzohtcfacihhvsfs
FOR KEY = 23 : rgurerhzvfgurorfgfznegpbagenpgcngsbezbhggurer
FOR KEY = 24 : qftqdqgyueftqnqefeymdfoazfdmofbxmfradyagfftqdq
FOR KEY = 25 : pespcpfxtdesmpdedxlcenzyeclneawleqzcxzfeespccp
```

HERE FOR KEY = 10 WE GET:  
*ethereum is the best smart contract platform out there*

SO DECRYPTED MESSAGE IS:  
*ethereum is the best smart contract platform out there*

## Checkpoint 2: Breaking Substitution Cipher

**Cipher 1:** af p xpkcaqvnpk pfg, af ipqe qpri, gauuikifc tpw, ceiri udvk tiki  
afgarxitfrphni cd eaowvmd popkwn, hiqpvi du ear jvaql vfgikrcpfgafm du cei  
xkafqaxnir du xrwqedearcdkw pfg du ear aopmafpcasi xkdhafmr afcd fit  
pkipr. ac tpr qdoudkcafcd lfdt cepc au pfwceafm epxxifig cd ringdf  
eaorinu hiudki cei opceiopcaqr du cei uaing qdvng hi qdoxnicinw tdklig  
dvc pfg edt rndtnw ac xkdqiigig, pfg edt odvfcpafdrv cei dhrcpqnir--ceiki  
tdvng pc niprc kiopafdfi mddg oafg cepc tdvng qdfcafvi cei kiripkqe

# SUBSTITUTION CYPHER CODE

```
map < char , int > cipherTextFrequency;
int numberofReplaceableCharacters = 0;

string ciphertext = "af p xpkcaqvnk pfg, af ipqe qpri, gauuikifc tpw, ceiri udvk tiki
afgarxifrphni cd eaowvmd popkwn, hiqpvr du ear jvaql vfgikrcpfcafdu cei xkafqaxnir du
xrwqedearcdkw pfg du ear aopmafcasi xkdhafmr afcd fit pkopr. ac tpr qdoudkcafcd lfdt cepc
au pfwceafm epxxifig cd ringdf eaorinu hiudki cei opceiopcaqr du cei uaing qdvng hi
qdoxxnicinw tdklig dvc pfg edt rndtnw ac xkdqiigig, pfg edt odvfcpfadvr cei dhrcpqnir--ceiki
tdvng pc nprc kiopadf mddg oafg cepc tdvng qdfcafvi cei kiripkqe";

for(auto ch: ciphertext){
    if('a' <= ch && 'z' >= ch) cipherTextFrequency[ch]++;
    numberofReplaceableCharacters++;
}

vector < pair < double , char > > cipherFreqList;
string englishLanguageFreqOrder = "etaonhisrdluwmgcfybpkjxqz";
string key = "abcdefghijklmnopqrstuvwxyz";
string ciphertextFrequencyOrder = "";

for(auto freqData: cipherTextFrequency) {
    cipherFreqList.push_back({(freqData.second * 100.00) / numberofReplaceableCharacters, freqData.first});
}

sort(cipherFreqList.begin(), cipherFreqList.end());

cout << "CIPHERTEXT FREQUENCY ORDER : " << endl;
for(auto freq: cipherFreqList){
    cout << freq.second << " >> " << freq.first << "%" << endl;
    ciphertextFrequencyOrder += freq.second;
}

reverse(ciphertextFrequencyOrder.begin(), ciphertextFrequencyOrder.end());

cout << endl << "ciphertextFrequencyOrder : " << ciphertextFrequencyOrder << endl;
cout << "englishLanguageFrequencyOrder : " << englishLanguageFreqOrder << endl << endl;

int indx = 0;
for(auto ch: ciphertextFrequencyOrder){
    cout << key[(int)englishLanguageFreqOrder[indx] - (int)'a'] << " will be replaced as " << ch
        << endl;
    key[(int)englishLanguageFreqOrder[indx] - (int)'a'] = ch;
    indx++;
}

cout << endl << " >>> KEY FOUND : " << key << endl << endl;

string decrypted = ciphertext;

int i = 0;
for(auto lol: ciphertext) {
    if('a' <= ciphertext[i] && 'z' >= ciphertext[i]) decrypted[i] = key[(int)lol - (int)'a'];
    i++;
}

cout << " >>> AFTER DECRYPTION : " << endl;
cout << decrypted << endl;

return 0;
}
```

# ALGORITHM FOR CRACKING SUBSTITUTION CIPHER

## ALGORITHM:

- Calculate the percentage of the frequency for the cipher text
- Sort them according to their appearance
- Compare them with the English language frequency
- Replace the cipher text according to the frequency of the cipher text and English language
- Check the result

## OUTPUT FOR CIPHER 1

### CIPHERTEXT FREQUENCY ORDER :

j >> 0.203666%  
s >> 0.203666%  
l >> 0.610998%  
h >> 1.222%  
m >> 1.42566%  
w >> 1.62933%  
x >> 2.03666%  
o >> 2.24033%  
t >> 2.24033%  
u >> 2.64766%  
v >> 2.64766%  
q >> 3.05499%  
n >> 3.25866%  
g >> 3.86965%  
k >> 3.86965%  
e >> 4.48065%  
r >> 4.68432%  
f >> 6.10998%  
a >> 6.31365%  
p >> 6.51731%  
c >> 6.72098%  
d >> 7.33198%  
i >> 9.36864%

### ciphertextFrequencyOrder :

idcpafrekgnqvutoxwmhlsj

### englishLanguageFrequencyOrder :

etaonhisrdluwmgcfybpkjxzq

### KEY FOUND : cmogixtfjrlnuaphqedqsvxwz

### AFTER DECRYPTION :

cx h xhlocqsahl hxt, cx rhqi qhkr, tcqqrlrxo  
dhv, oirkr qgsl drlr cxtckrxkhfar og icpvsug  
hphlva, frqhskr gq ick jscqn sxtrlkohxtcxu gq  
oir xlcxqcxark gq xkvqigickoglv hxt gq ick  
cphucxhocer xlxfcruk cxog xrd hlrhk. co dhk  
qgpqglocxu og nxgd oiho cq hxvoicxu  
ihxxrxt og kratgx icpkraq frqqlr oir  
phoirphocqk gq oir qcrat qgsat fr qgpxarorav  
dglnrt gso hxt igd kagdav co xlgqrtrt, hxt igd  
pgsxohcxgsk oir gfkohqark--oirlr dgsat ho  
arhko lrphcxgxr uggt pcxt oiho dgsat  
qgxocxs oir lrkrhlqi

**Cipher 2:** aceah toz puvg vcdl omj puvg yudqecov, omj loj auum klu thmjuv hs klu zlcu shv zcbkg guovz, upuv zcmdu lcz vuwovoaeu jczooyuovomdu omj qmubyudkuj vukqvm. klu vcdluz lu loj avhqnlk aodr svhw lcz kvopuez loj mht audhwu o ehdoe eunumj, omj ck toz yhyqeoveg auecupuj, tlokupuv klu hej sher wcnlk zog, klok klu lcee ok aon umj toz sqee hs kqmmuez zkqssuj tckl kvozqv. omj cs klok toz mhk umhqnl shv sowu, kluvu toz oezh lcz yvhehmnuj pcnhqv kh wovpue ok. kcwu thvu hm, aqk ck zuuwuj kh lopu eckkeu ussudk hm wv. aonncmz. ok mcmukg lu toz wqdl klu zowu oz ok scskg. ok mcmukg-mcmu klug aunom kh doee lcw tuee-yvuzuvpuj; aqk qmdlomnuj thqej lopu auum muovuv klu wovr. kluvu tuvu zhuw klok zlhhr klucv luojz omj klhqnlk klcz toz khh wqdl hs o nhhj klcmn; ck zuuwuj qmsocv klokomghmu zlhqej yhzzuzz (oyyovumkeg) yuvyukqoe ghqkl oz tuee oz (vuyqkujeg) cmubloqzkcaeu tuoekl. ck tcee lopu kh au yocj shv, klug zocj. ck czm'k mokqvoe, omj kvhqaeu tcee dhwu hs ck! aqk zh sov kvhqaeu loj mhk dhwu; omj oz wv. aonncmz toz numuvhqz tckl lcz whmug, whzk yuhyeu tuvu tceecmn kh shvncpu lcw lcz hijkcuz omj lcz nhhj shvkqmu. lu vuwocmuj hm pczckcmn kuwwz tckl lcz vueokcpuz (ubduyk, hs dhqvzu, klu zodrpceeu- aonncmzuz), omj lu loj womg juphkuj ojwcvuvz owhmn klu lhaackz hs yhhv omj qmcwyhvkomk sowcecz. aqk lu loj mh dehzu svcumjz, qmkce zhwu hs lcz ghqmnuv dhqzcmz aunom kh nvht qy. klu uejuzk hs klu, omj aceah'z sophqvcku, toz ghqmnn svhjh aonncmz. tlum aceah toz mcmukg-mcmu lu ojhykuj svhjh oz lcz lucv, omj avhqnlk lcw kh ecpu ok aon umj; omj klu lhyuz hs klu zodrpceeu- aonncmzuz tuvu scmoeeg jozluj. aceah omj svhjh loyyumuj kh lopu klu zowu acvkljog, zuykuwaav 22mj. ghq loj aukkuv dhwu omj ecpu luvu, svhjh wg eo, zocj aceah hmu jog; omj klum tu dom dueuavoku hqv acvkljog-yovkucz dhwshvkoag khnukluv. ok klok kcwu svhjh toz zkcee cm lcz ktuumz, oz klu lhaackz doeeuj klu cvvuzyhmzcaeau ktumkcuz auktuum dlcejhhj omj dhwcmn hs onu ok klcvkg-kluuu

## OUTPUT FOR CIPHER 2

### CIPHERTEXT FREQUENCY ORDER:

b >> 0.205339%  
 r >> 0.359343%  
 p >> 1.12936%  
 g >> 1.43737%  
 y >> 1.43737%  
 d >> 1.48871%  
 t >> 1.74538%  
 n >> 1.89938%  
 s >> 1.95072%  
 w >> 1.95072%  
 q >> 2.15606%  
 a >> 2.41273%  
 e >> 3.64476%  
 j >> 3.79877%  
 v >> 4.36345%  
 m >> 4.8768%  
 z >> 4.8768%  
 l >> 4.97947%  
 c >> 5.23614%  
 h >> 5.80082%  
 o >> 6.72485%  
 k >> 6.77618%  
 u >> 10.1643%

**ciphertextFrequencyOrder :**  
 ukohclzmvjeaqwsntdygprb

**englishLanguageFrequencyOrder :**  
 etaonhisrdluwmgcfybpkjxqz

**KEY FOUND :** oynjutslzbpewchgqvmkarqxdz

## AFTER DECRYPTION :

onuol khz gars rnje hwb gars dajqunhr, hwb ehb oaaw pea klwbar lm pea zenra mlr znyps sahrz, **evar** znwja enz raqhrvhoua bnhzzddahrhwja hwb qwaydajpab rapqrw. pea rnjeaz ea ehb orlcep ohjv mrlq enz prhgauz ehb wlk oajlqa h uljhu uacawb, hwb np khz dldquhrus oaunagab, kehpagar pea lub mluv qncep zhs, pehp pea enuu hp ohc awb khz mquu lm pqwwauz zpqmmab knpe prahzqra. hwb nm pehp khz wlp awlqce mlr mhqa, peara khz huzl enz drlulwcab gnclqr pl qhrgau hp. pnqa klra lw, oqp np zaaqab pl ehga unppua ammajp lw qr. ohccnwz. hp wnwaps ea khz qqje pea zhqa hz hp mnmps. hp wnwaps-wnwa peas oachw pl jhnu enq kauu-drazargab; oqp qwjehwcab klqub ehga oaaw wahrar pea qhrv. peara kara zlqa pehp zellv peanr eahbz hwb pelqcep penz khz pll qqje lm h cllb penwc; np zaaqab qwmhn r pehphws lwa zelqub **plssass** (hddhrawpus) dardapqhu slqpe hz kauu hz (radqpabus) nwayehqzpnoua kahupe. np knuu ehga pl oa dhnb mlr, peas zhnb. np nzw'p whpqrhu, hwb prlqoua knuu jlqa lm np! oqp zl mhr prlqoua ehb wlp jlqa; hwb hz qr. ohccnwz khz cawarlqz knpe enz qlwas, qlzp daldua kara knuunwc pl mrcnga enq enz lbbnpnaz hwb enz cllb mlpqwa. ea raqhnwab lw gnznpnwc parqz knpe enz rauhpngaz (**expcdt**, lm jlqrza, pea zhjvgnuua- ohccnwzaz), hwb ea ehb qhws baglpab hbqnrraz hqlwc pea eloopz lm dllr hwb qwnqdrlphwp mhqnunaz. oqp ea ehb wl julza mrnawbz, wpnu zlqa lm enz slqwcar jlqznwz oachw pl crlk qd. pea aubazp lm peaza, hwb onuol'z mhglqrnpa, khz slqwc mrlbl ohccnwz. keaw onuol khz wnwaps-wnwa ea hblpdab mrlbl hz enz eanr, hwb orlcep enq pl unga hp ohc awb; hwb pea eldaz lm pea zhjvgnuua- ohccnwzaz kara mnwhuu bhzeab. onuol hwb mrlbl ehddawab pl ehga pea zhqa onrpebhs, zadpaqoar 22wb. slq ehb oappar jlqa hwb unga eara, mrlbl qs ubh, zhnb onuol lwa bhs; hwb peaw ka jhw jauaorhpa lqr onrpebhs-dhrpnaz jlqmlrphous plcapear. hp pehp pnqa mrlbl khz zpnuu nw enz pkaawz, hz pea eloopz jhuuab pea nrrazdlwznoua pkawpnaz oapkaaw jenubellb hwb jlqnwc lm hca hp penrps-peraa

## CONCLUSION :

Here in the both ciphertext I tried to solve it using the English language frequency. But still it was unable to crack it. Because the text was too small. As we know that if the length of the ciphertext is bigger the breaking of substitution cipher becomes very easy. As the first ciphertext is smaller that is why it cannot be decoded at all but the second one is relatively bigger which makes it a little bit better suit for decoding. So, in the second one we can see some words like "evar"(ever), "plssass"(possess) and "expcdt"(expect).

# LAB 3

## Symmetric encryption & hashing

### TASK 3.1: AES encryption using different modes

Firstly, I openned a text file names **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 &

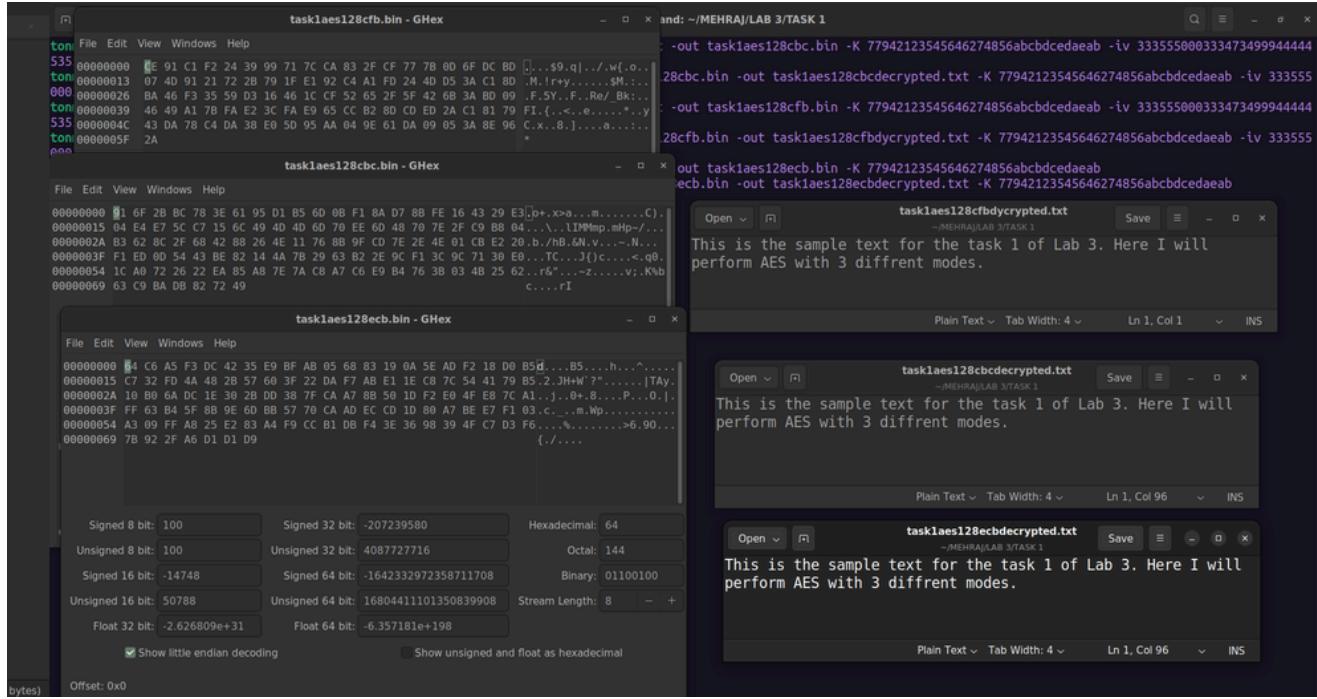
ghex task1aes128cfb.bin &

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

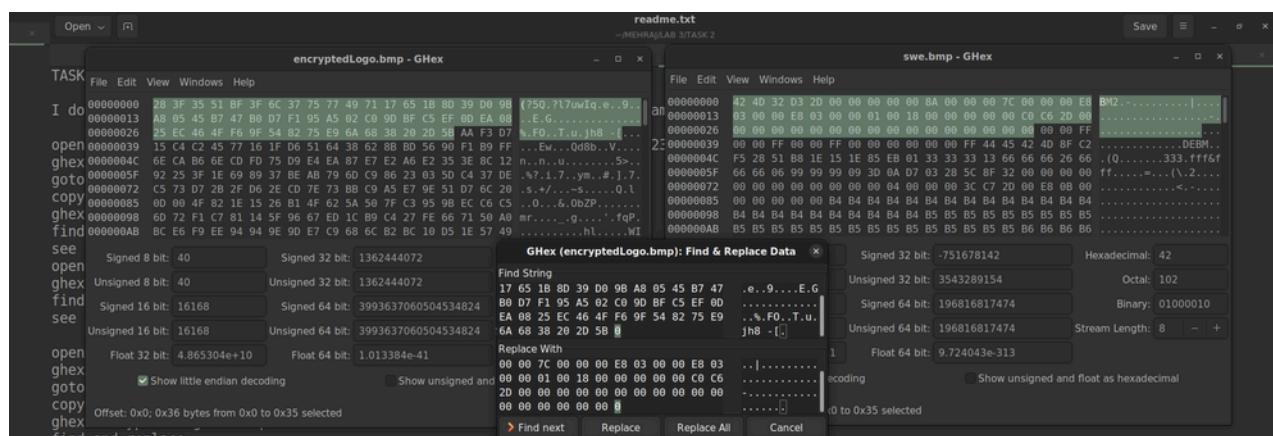
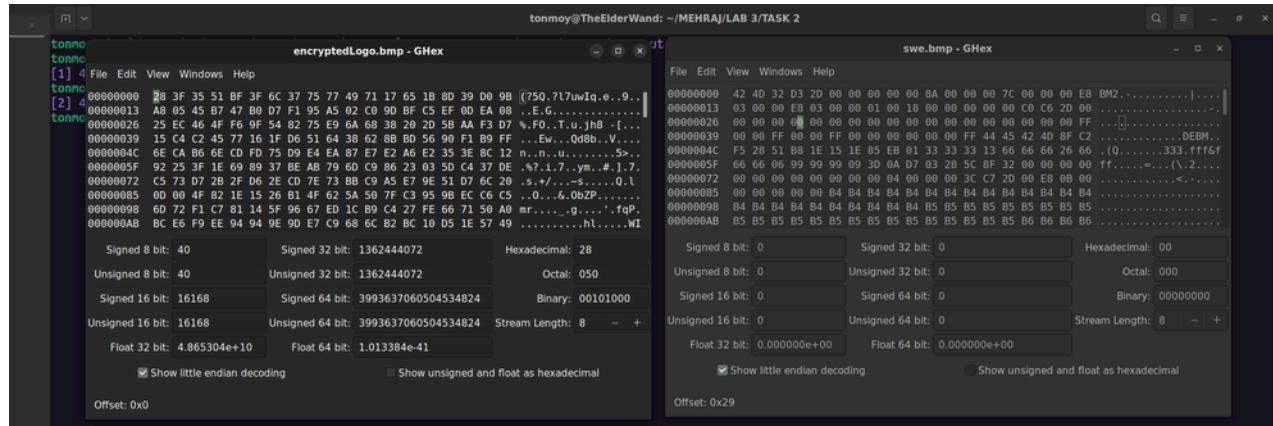
```
openssl enc -aes-128-ecb -e -in swe.bmp -out encryptedLogo.bmp  
-K 77942123545646274856abcbdcdaeaab  
ghex swe.bmp &
```

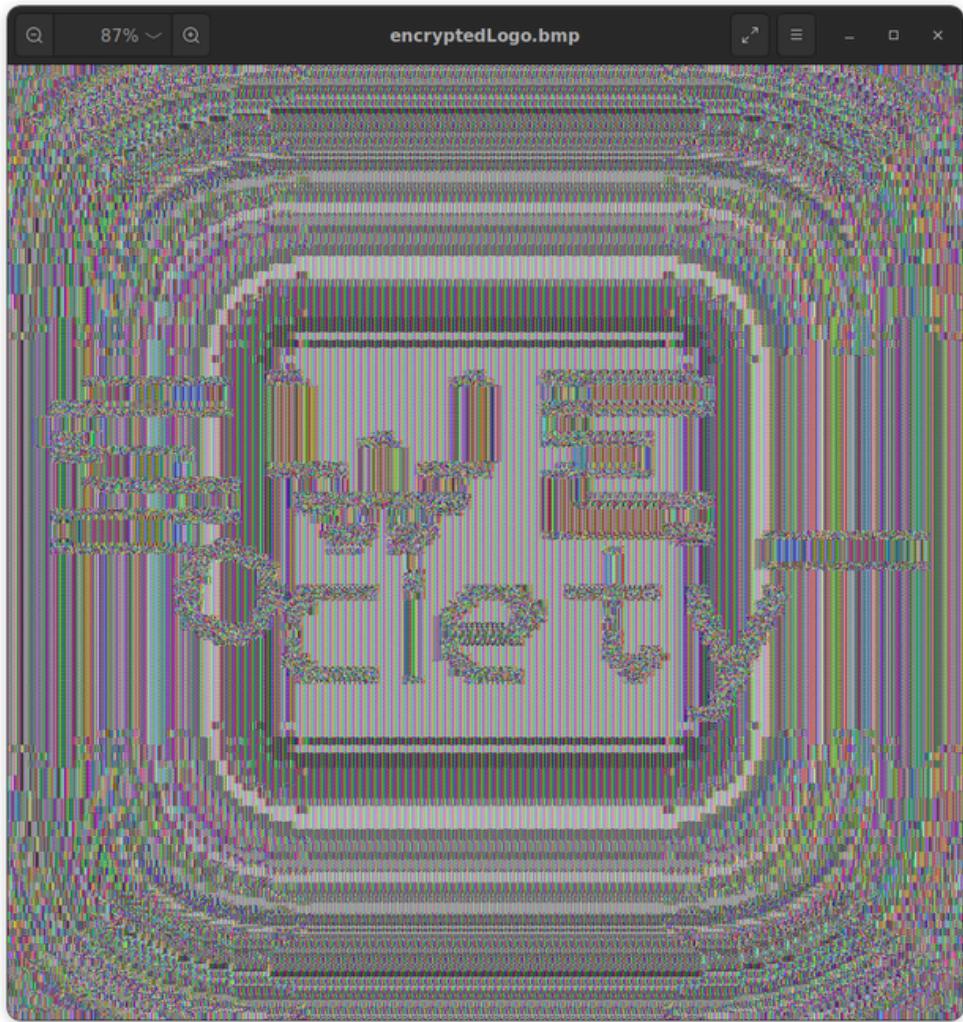
```
ॐ श 6 03:29
tonmoy@TheElderWand: ~/MEHRAJ/LAB 3/TASK 2
EHRAJ/LAB 3/TASK 2$ openssl enc -aes-128-ecb -e -in swe.bmp -out encryptedLogo.bmp -K 77942123545646274856abcbcdcedaeab
EHRAJ/LAB 3/TASK 2$ ghex swe.bmp &
EHRAJ/LAB 3/TASK 2$ ghex encryptedLogo.bmp &
EHRAJ/LAB 3/TASK 2$
```

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
```



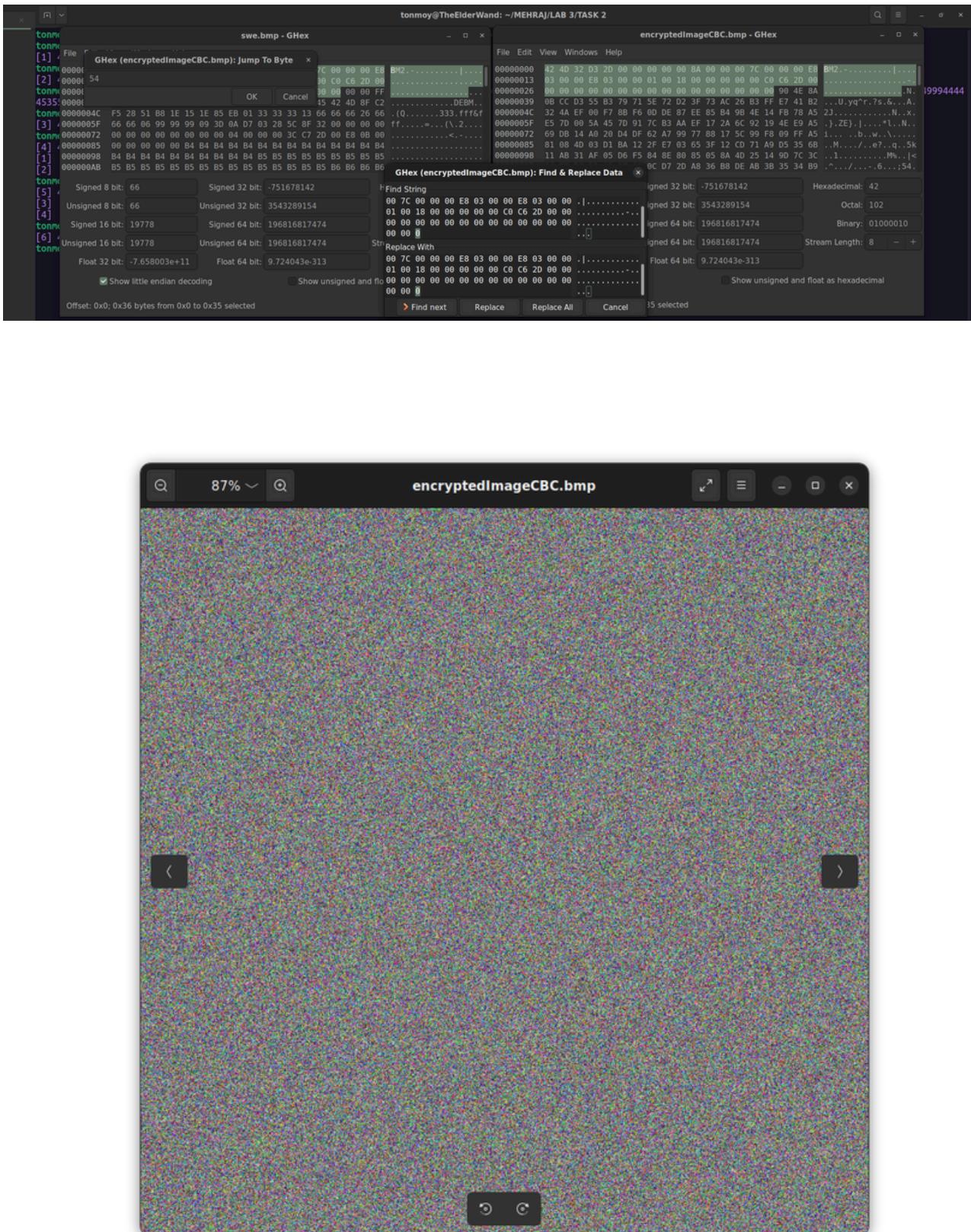


This is the output of the encryption for AES128-ECB

## 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$
```



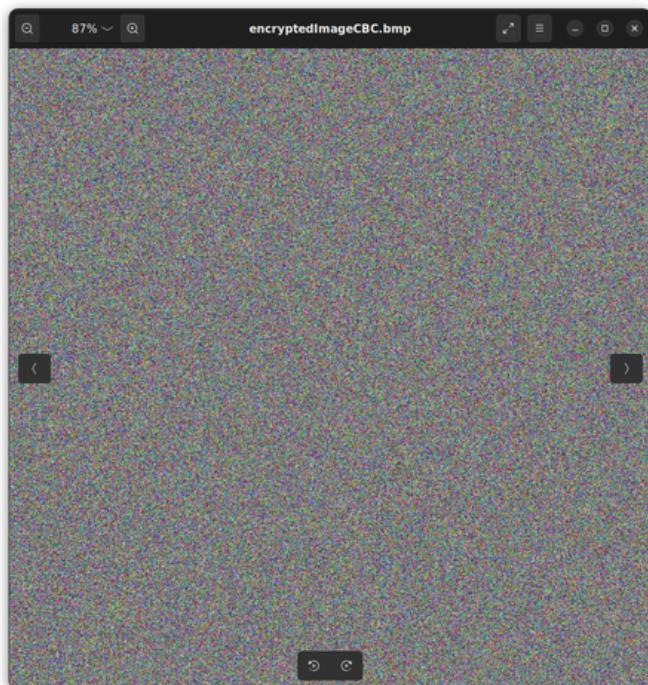
This is the output of the encryption for AES128-CBC

## ECB & CBC Mode Analysis

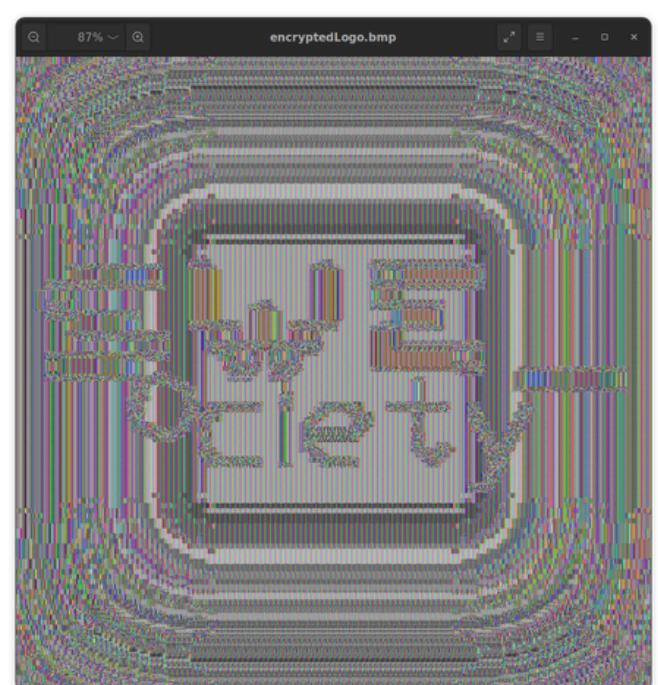
CBC (Cipher Block Chaining) and ECB (Electronic Codebook) are both block cipher encryption. where each block of plaintext is encrypted independently with the same key in ECB mode. It lacks diffusion, meaning identical blocks of plaintext will result in identical blocks of ciphertext. This can lead to 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.



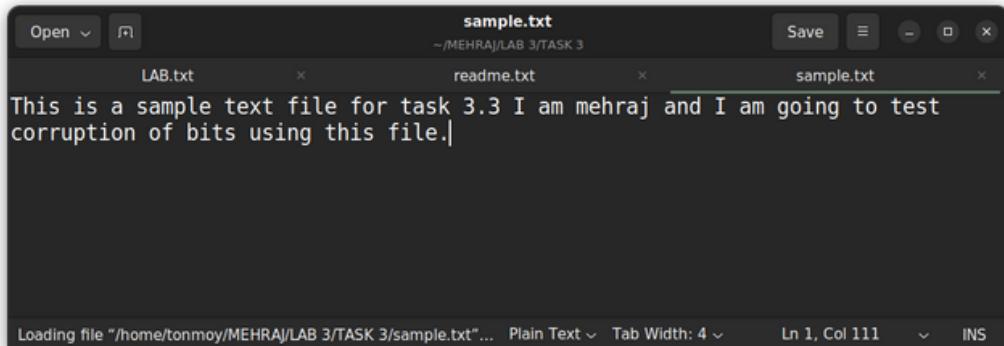
CBC



ECB

# TASK 3.3: ENCRYPTION MODE : CORRUPTED CIPHERTEXT

Firstly, I created a text file sample.txt

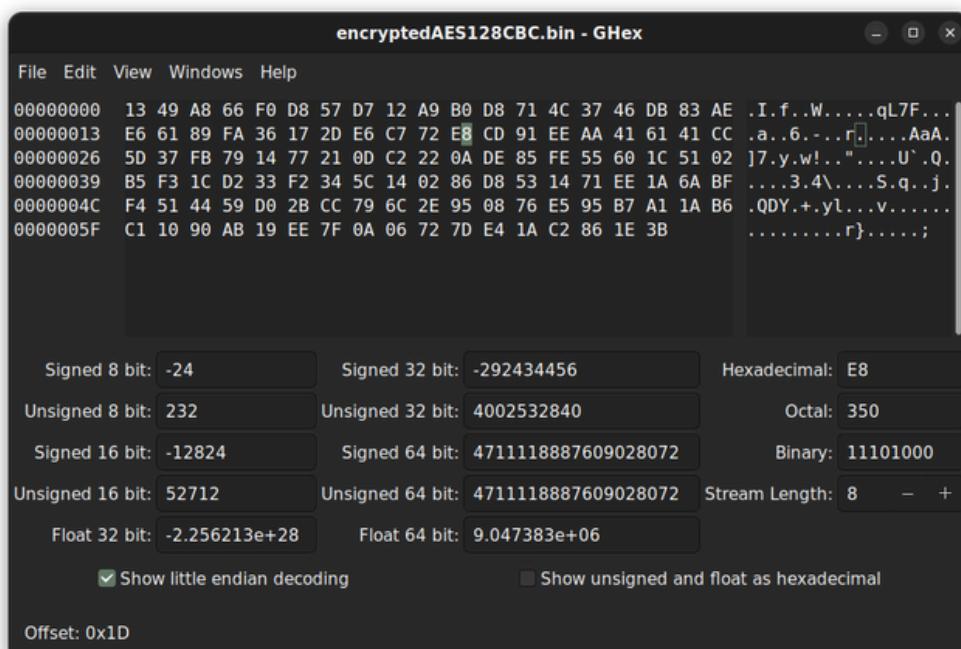


## FOR CBC MODE:

```
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
```



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

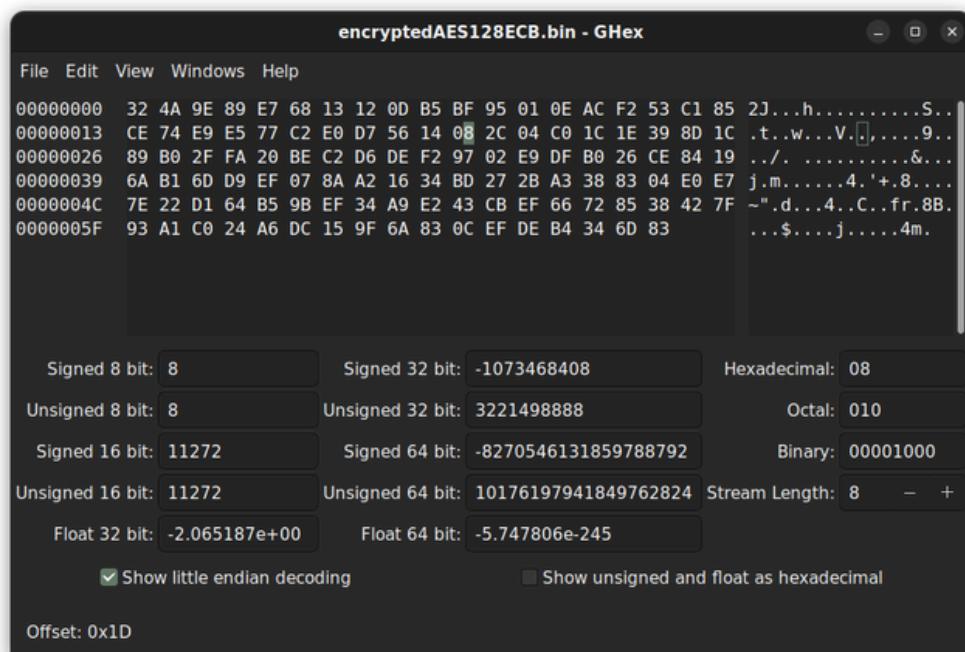
Plain Text Tab Width: 4 Ln 1, Col 111 INS

## 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
```

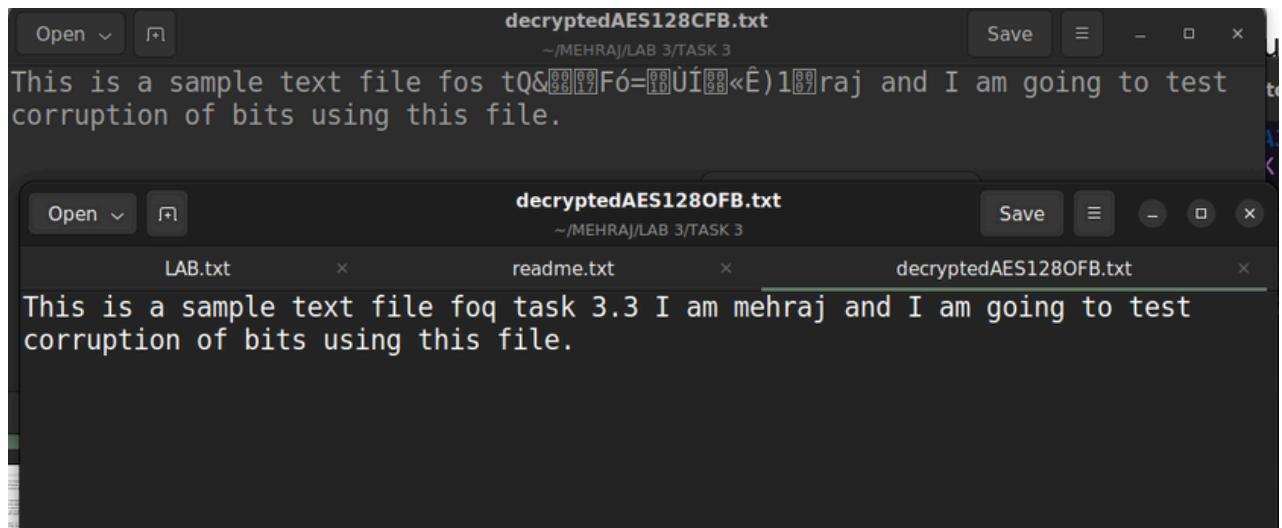
### FOR OFB MODE

```
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.bin &
[1]+ Done ghex encryptedAES128CFB.bin
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 -K 77942123545646274856abcbdcedaeab -iv 33355500033347349994444453555222
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$ ghex encryptedAES128OFB.bin &
[1]+ Done ghex encryptedAES128OFB.bin
tonmoy@TheElderWand:~/MEHRAJ/LAB 3/TASK 3$
```



Here these are the results of CFB & OFB

# ANALYSIS

## **ECB Mode**

**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÷Vg¤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:**

**Corrupted Information:** "Q&-ØFó=ØÙÍ»«Ê)1Ø" (16 bytes)

**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:

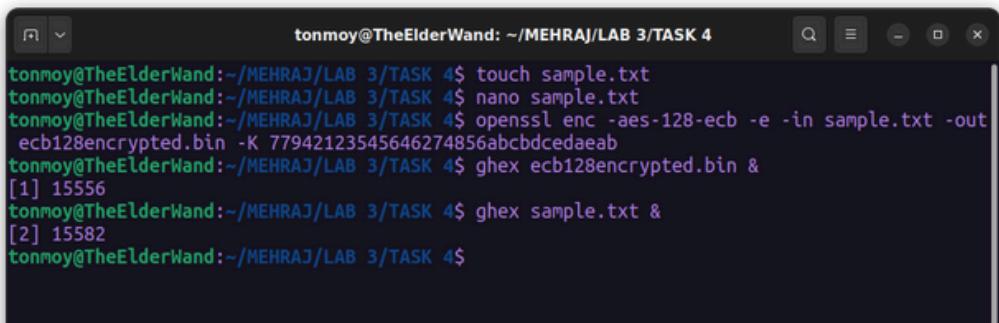
- ECB mode offers the least security and diffusion, as evident from the limited recoverable information.
- CBC mode provides better security compared to ECB but still suffers from partial corruption propagation.
- CFB offers improved diffusion, allowing for more recoverable information compared to CBC 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

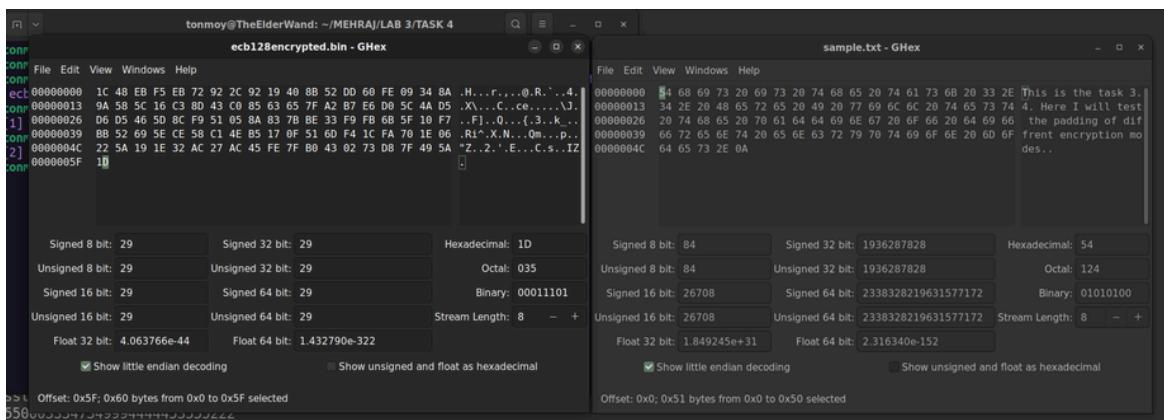
Firstly I created a text file sample.txt

### 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$
```

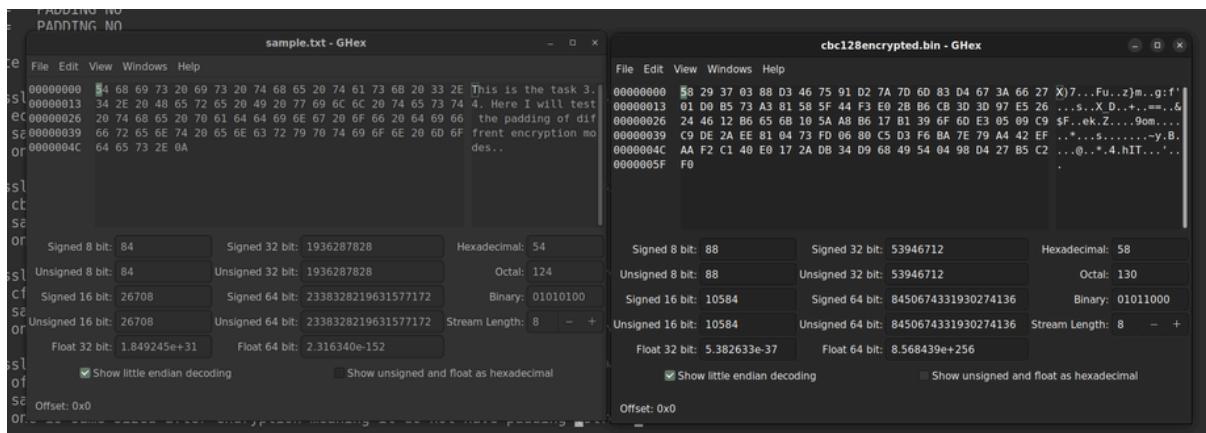


ECB generates larger output after encryption meaning it has padding

## 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$
```

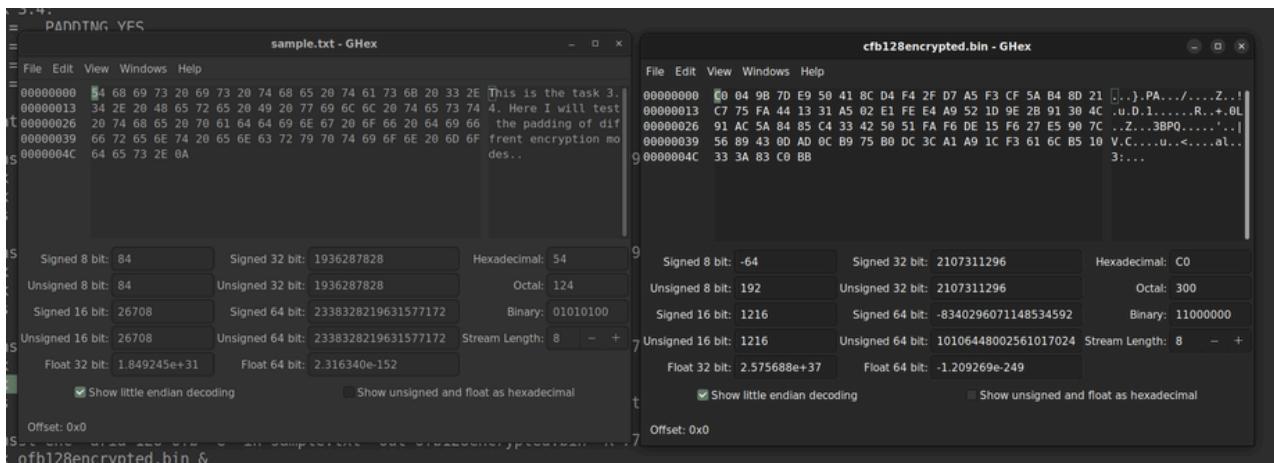


CBC generates larger output after encryption meaning it has padding

## 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$
```

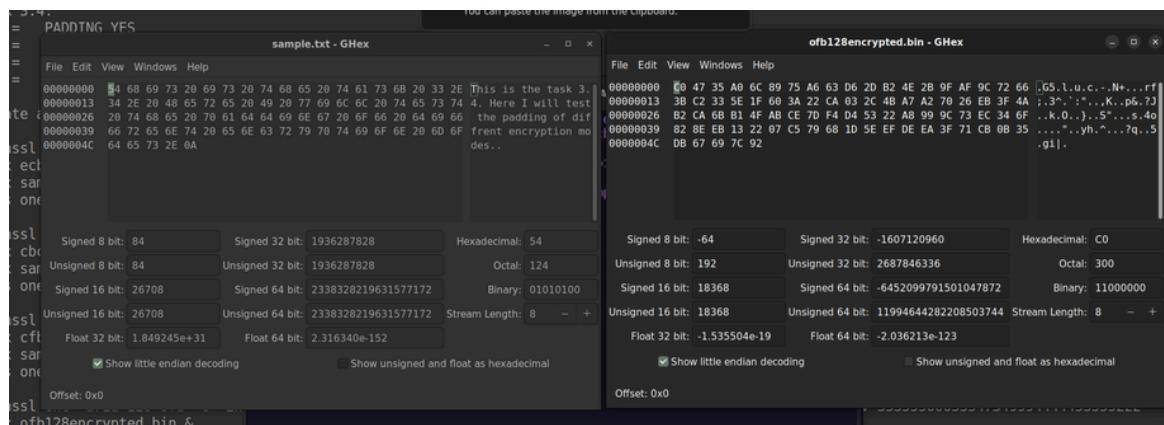


**CFB do not generate larger output after encryption meaning it do not have padding**

### 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**

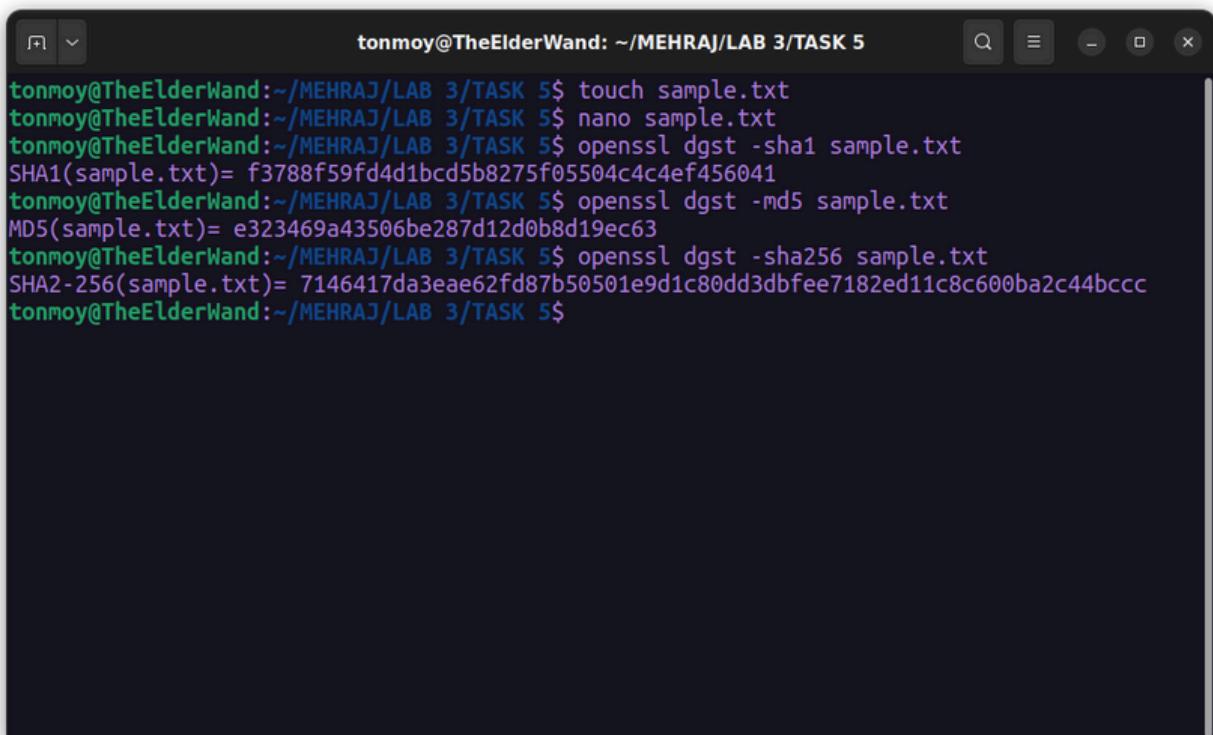
## **TASK 3.5: GENERATING MESSAGE DIGEST**

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: 7146417da3eae62fd87b50501e9d1c80dd3dbfee7182ed11c8c600ba2c44bcc
```



```
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)= 7146417da3eae62fd87b50501e9d1c80dd3dbfee7182ed11c8c600ba2c44bcc  
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
```

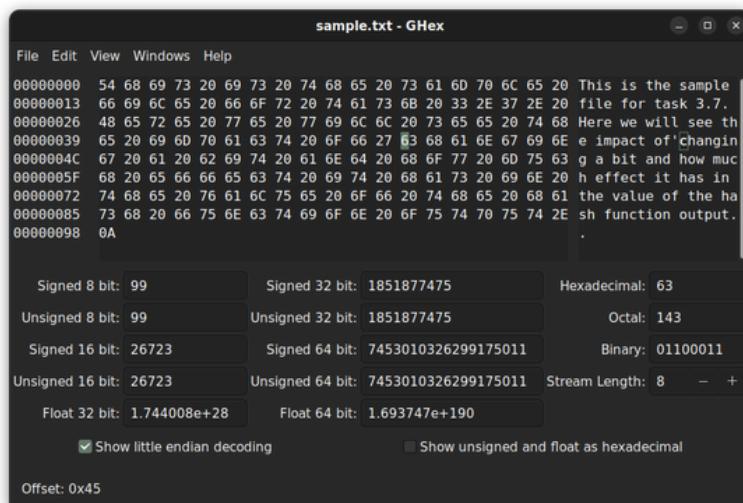
## FOR MD5

```
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:

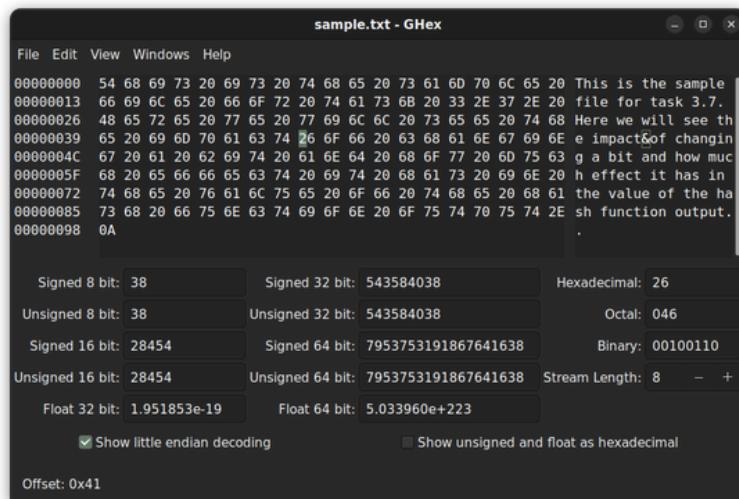
**923b829a244a667348611ffe93016ddf798224b71a0ad6c650e68f9e10cd1c6e**  
ghex text.txt &

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