

LAB- 01

By :- Faraz Ahmed

Task 1- Frequency Analysis

- First, we will download Labsetup.zip and unzip it to get Labsetup file.
 - Then run the python file freq.py using “./freq.py” command as highlighted in screenshot 1. It is used to analyze the frequency of letters and n-grams in a ciphertext file and can count how often certain letter is present in that ciphertext. (as shown in Screenshot 1)

```
[09/10/25] seed@VM:~/.../Files$ ls
seed.txt freq.py ngram1.bmp sample_code.py words.txt
[09/10/25] seed@VM:~/.../Files$ ./freq.py
-----  
1-gram (top 20):  
n: 489  
e: 348  
v: 348  
u: 280  
q: 264  
h: 235  
t: 183  
l: 166  
p: 153  
a: 116  
c: 104  
f: 95  
i: 90  
s: 83  
b: 83  
r: 76  
d: 59  
-----  
2-gram (top 20):  
vn: 85  
th: 85  
nh: 84  
mh: 58  
vh: 57  
ht: 57  
vu: 56  
hv: 53  
ku: 52  
up: 46  
ns: 44  
yn: 44  
ny: 44  
vy: 44  
nv: 37  
yv: 39  
vq: 33  
vz: 32  
gn: 32  
av: 3  
-----  
3-gram (top 20):  
vnu: 78  
vup: 30  
vns: 20  
vnh: 18
```

Screenshot 1: frequency of letters in freq.py file.

- Now by using “`tr 'nyv...' 'ETI...' <ciphertext.txt> partial_decrypt_v2.txt`” command as highlighted in Screenshot 2, we can decrypt the ciphertext by replacing different letters in ciphertext till we get a fully readable plaintext as seen in Screenshot 4.

Screenshot 2: Entering command "tr 'nyv...' 'ETI...' <ciphertext.txt> partial decrypt v2.txt" to decrypt ciphertext.

- Screenshot 3 is about the original ciphertext which will be converted into fully readable plaintext as shown in Screenshot 4.

```

1 ytn xqavhq yzhu xu qzupvd ltmat qmncq vxqzy hmrtv byvhny ytmq ixur qyhvurn
2 vlvhpq yhme ytn gvrnh bnniq imsn v uxuvrnvhvnm yx
3
4 ytn xlvhqg huan lvg pxqncnup gd ytn pncnq xb tvtfrd lnsqncnup vy mqk xzyqny
5 vpxp vtn weehnvuy mcaixqnpn xb tmc basic axcevud vy vtn nup vlp my lvg qzvnpn gd
6 ytn ncnnhrnian xb cnvxx ymcnc qz qivsrxlu eximymaq vhcavupd vaymfmc vup
7 v uymxuvi auxfhnqaymxu vg ghlmh vup cyp vq v bnfnh phvc vxqzy ltntnhy ytnhn
8 xzrty yx gn ur hqnmnpuy lmubnd ytn qnqaxi pmppu ozgy qnnc khyv ixur my lvg
9 nkyhv ixur gnazqn ytn xqavhnq lnhn cxfnp yx ytn bmhgy lnnsnup mu cvhat yx
10 vtfxm xubimaymuy leyt ytn aixqmur ahnncxud xb ytn lmuyhn xidcemeq ytvusq
11 ednxuratvur
12
13 xun gmr jzqnymxu qzhhxzupmnr ytmq dnvhq vavpnco vlvhqg mq tlx xb mb ytn
14 ahnnccxud lmii vpphnnq cnvxx qgenamvliid vbyhn ytn rxipnu rxqng ltmat ghavn
15 v oqgnimuy axcmurxzy evhdh bxh ymcnc ze ytn cxfnctuy qenvhntvnpng gd
16 exlnhbz1 tx1d1lxcs lxcnu ltx trienp hmgnq cmilmxuq xb pxlihng yx bmrtv qnkzv1
17 tvhqqncnuy vhxuzp ytn axuyhd
18
19 qmruviaur ytmh qzeexhy rxipnu rxqng vynupnng qlyvtnp ytnconing mu givas
20 qexhynp lveni emuq vup qzxpmpn bbb vxqzy qnkmgq exlnh mcqivuand bhx ytn hnp
21 avhny vup ytn qyvrm xu ytn vhm n lvp avihp xzy vxqzy evd munjzmd vbyhn
22 myq bxhcnh vuatxh avyy qvpmh jzmy xuan qtn invhnpn ytv ytn lva cysmnr bv
23 inqq ytvu v cvin axtxqy vup pzhmr ytn ahnnccxu uyyvilm exhycvuy xxs v gizu
24 vup qvymqbdmnr pmr v ytn vilcvin hqxyh xb ucxcmuvynp pmhnyhxq txl azxip
25 ytv ygn xseepn
26
27 vq yzhuad xzy v ynvqy mu ynhcq xb ytn xqavhq my ehxgvqld lxuy gn
28
29 lxcnu mufxclp mu yeqng ze qmp ytvv vlybzcrt ytn rxqng qmrrnbng ytn
30 umaymymvfdi lizuat vtrnd unfnh mynuvne my yx on ozay v ylvhpn enqqu
31 avcevrmr xh xun ytvv gnacnv qvqamvnp xuid lmt ytnpahvney vasmxuq myqnyvnp
32 v qexsqnlxvcu qvmp ytn rxhzx mg lkhsmr gntmpu aixnpn pxxhq vup tqg qmuan
33 vcvqgnp cmilmxu bxh myq irnvi pnbnqng bzup ltmat vbyhn ytn rxqng lvq
34 bixxnpn lmyt ytzxqvpdq xb pxhuyxmq xb yh inqq bhx enxein mu qxcn
35 axzuyhmq
36
37
38 ux avli yx lnhv givas rlxq lnyu xzy mu vpfvuan xb ytn xqavhq ytzxrt ytn
39 cxfnuy lmii vlcixy fxavi anhyymuid qn hnbnhnuanp gnbxhn vup zphmr ytn ahnnccxud
40 emenamvld qmuan vphqf cnvxx qzeexhyhq imsn vqtdn opzzp izvhy phnu vup
41 umaxin sepcu vhn qatpnzng ehqnyhnu
42
43 vxuyxhn bnvyzhn xb ytnq qnqauu ux un hnvidi sxlq ltx mq rxmru yx lmu qnqy
44 emayzhn vrhrzgld ytmq tveenug v lxy bch ytn ymcn muvhrzgld ytn umigymh
45 uvhyyvnm xuid qmhfq ytn vlvhpq tden cvatmn gzy xbyhn ytn enxein bxhnavqymur
46 ytn hvan qxavaiinp xqavhixmymq avv cuvn xuid npzavynp rznqnd
47
48 ytn lvd ytn wavpnco yvqzvnyh ytn gar lmuuh pxnqy trae mu nfnid yxtn
49 avayrxhd ytn uxmcmn lmyt ytn cxqy fxyng lmuq gzy ytn qnqy emayzhn
50 avayrxhd ytn cxqy vhn vqspn yx imqy ytnmh yxe cxfmng mu ehnbnhnuvnyi xhpnh mb v
51 cxfmnm rnyq cxhn ytvu enhanuy xb ytn bmhgeyivan fxyng my lmuq ltnu ux
52 cxfmnm cvuerng ytv ytn xun lmyt ytn blnqy bmhgeyivan fxyng mq nimcmuvynp vup

```

Screenshot 3: Ciphertext with all random letters which is unreadable for anyone.

```

1 THE OSCARS TURN ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
2 AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO
3
4 THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET
5 AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY
6 THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMANDY ACTIVISM AND
7 A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER THERE
8 OUGHT TO BE A PRESIDENT WINFREY THE SEASON DIDNT JUST SEEM EXTRA LONG IT WAS
9 EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEKEND IN MARCH TO
10 AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS THANKS
11 PYEONGCHANG
12
13 ONE BIG QUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE
14 CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME
15 A JUBILANT COMINGOUT PARTY FOR TIMES UP THE MOVEMENT SPEARHEADED BY
16 POWERFUL HOLLYWOOD WOMEN WHO HELPED RAISE MILLIONS OF DOLLARS TO FIGHT SEXUAL
17 HARASSMENT AROUND THE COUNTRY
18
19 SIGNALING THEIR SUPPORT GOLDEN GLOBES ATTENDEES SWATHED THEMSELVES IN BLACK
20 SPORDED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED
21 CARPET AND THE STAGE ON THE AIR E WAS CALLED OUT ABOUT PAY INEQUITY AFTER
22 ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR
23 LESS THAN A MALE COHOST AND DURING THE CEREMONY NATALIE PORTMAN TOOK A BLUNT
24 AND SATISFYING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD
25 THAT BE TOPPED
26
27 AS IT TURNS OUT AT LEAST IN TERMS OF THE OSCARS IT PROBABLY WONT BE
28
29 WOMEN INVOLVED IN TIMES UP SAID THAT ALTHOUGH THE GLOBES SIGNIFIED THE
30 INITIATIVES LAUNCH THEY NEVER INTENDED IT TO BE JUST AN AWARDS SEASON
31 CEREMONY AND THE CAMPAGNE AGAINST HARVEY WEINSTEIN WASNT OVER
32 A SPOKESWOMAN SAID THE GROUP IS WORKING BEHIND CLOSED DOORS AND HAS SINCE
33 AMASSED MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS
34 FLOODED WITH THOUSANDS OF DONATIONS OF OR LESS FROM PEOPLE IN SOME
35 COUNTRIES
36
37
38 NO CALL TO WEAR BLACK GOWNS WENT OUT IN ADVANCE OF THE OSCARS THOUGH THE
39 MOVEMENT WILL ALMOST CERTAINLY BE REFERENCED BEFORE AND DURING THE CEREMONY
40 ESPECIALLY SINCE VOCAL METOO SUPPORTERS LIKE ASHLEY JUDD LAURA DERN AND
41 NICOLE KIDMAN ARE SCHEDULED PRESENTERS
42
43 ANOTHER FEATURE OF THIS SEASON NO ONE REALLY KNOWS WHO IS GOING TO WIN BEST
44 PICTURE ARGUABLY THIS HAPPENS A LOT OF THE TIME INARGUABLY THE NAILBITER
45 NARRATIVE ONLY SERVES THE AWARDS HYPE MACHINE BUT OFTEN THE PEOPLE FORECASTING
46 THE RACE SOCALLED OSCARLOGISTS CAN MAKE ONLY EDUCATED GUESSES
47
48 THE WAY THE ACADEMY TABULATES THE BIG WINNER DOESNT HELP IN EVERY OTHER
49 CATEGORY THE NOMINEE WITH THE MOST VOTES WINS BUT IN THE BEST PICTURE
50 CATEGORY VOTERS ARE ASKED TO LIST THEIR TOP MOVIES IN PREFERENTIAL ORDER IF A
51 MOVIE GETS MORE THAN PERCENT OF THE FIRSTPLACE VOTES IT WINS WHEN NO

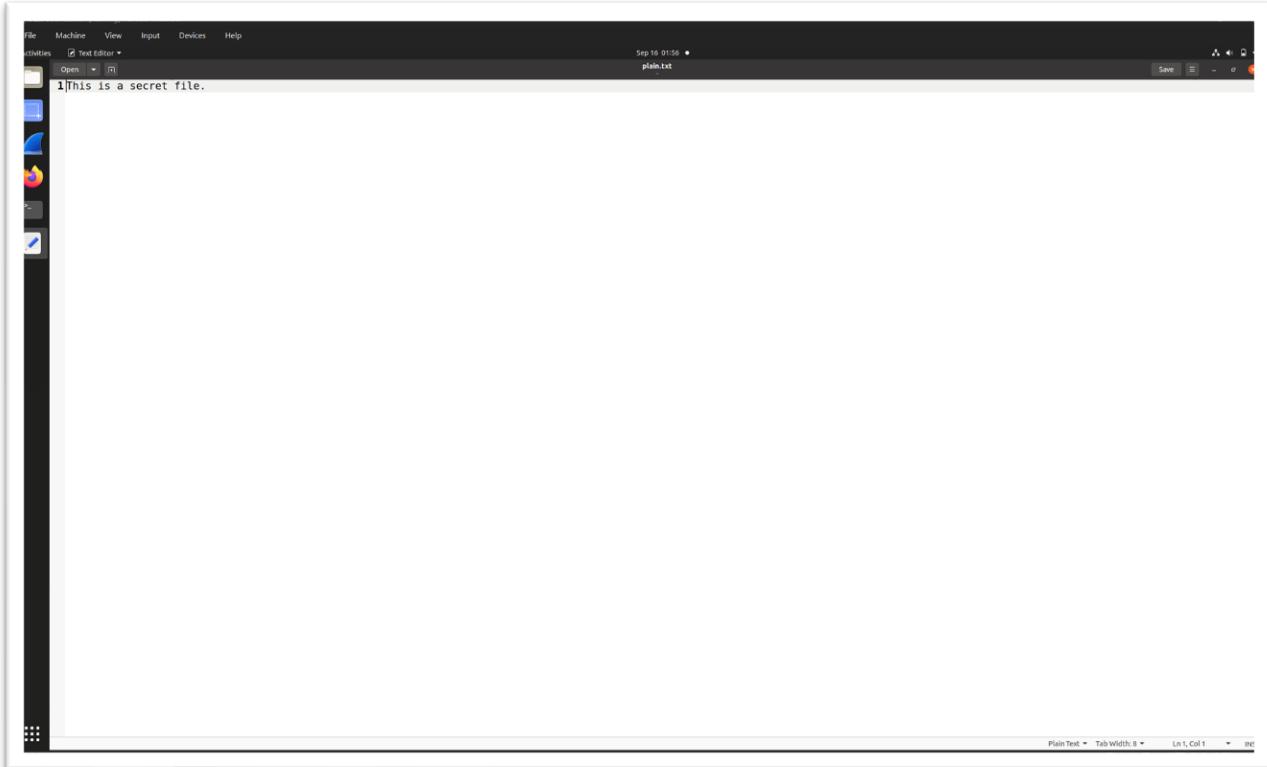
```

Screenshot 4: By using command mentioned above, we can convert ciphertext into readable plaintext.

Task 2- Encryption using Different Ciphers and Modes.

- So first we can create a new plain file with command “echo -n “This is a secret file.” > plain.txt” as highlighted in Screenshot 5. And we can observe the message inside that plain file as observed in Screenshot 6. Then we enter 32 hex digits as key (128 bits) and 16 hex digits as IV (for CBC modes). Finally, we can encrypt that plain file using “openssl enc -aes-128-cbc -e -in plain.txt -out aes.txt -K \$KEY -iv \$IV” command (repeat same command for rest DES and Blowfish) and see the result by entering “xxd aes.txt” (same for rest DES and Blowfish) as observed in Screenshot 5. We can also observe the encrypted plaintext in Screenshots 7, 8 and 9 for AES, DES and Blowfish respectively.

Screenshot 5: Different commands in order to encrypt plaintext into ciphertext using AES, DES and Blowfish.



Screenshot 6: Message inside plain.txt file which we created to encrypt.



Screenshot 7: Encrypted message inside aes.txt which was encrypted using AES.

A screenshot of a terminal window titled "Text Editor". The file "des.txt" is open, containing the following encrypted text:
19f6Ez#0Y&0{2RØE
2x6x±hI

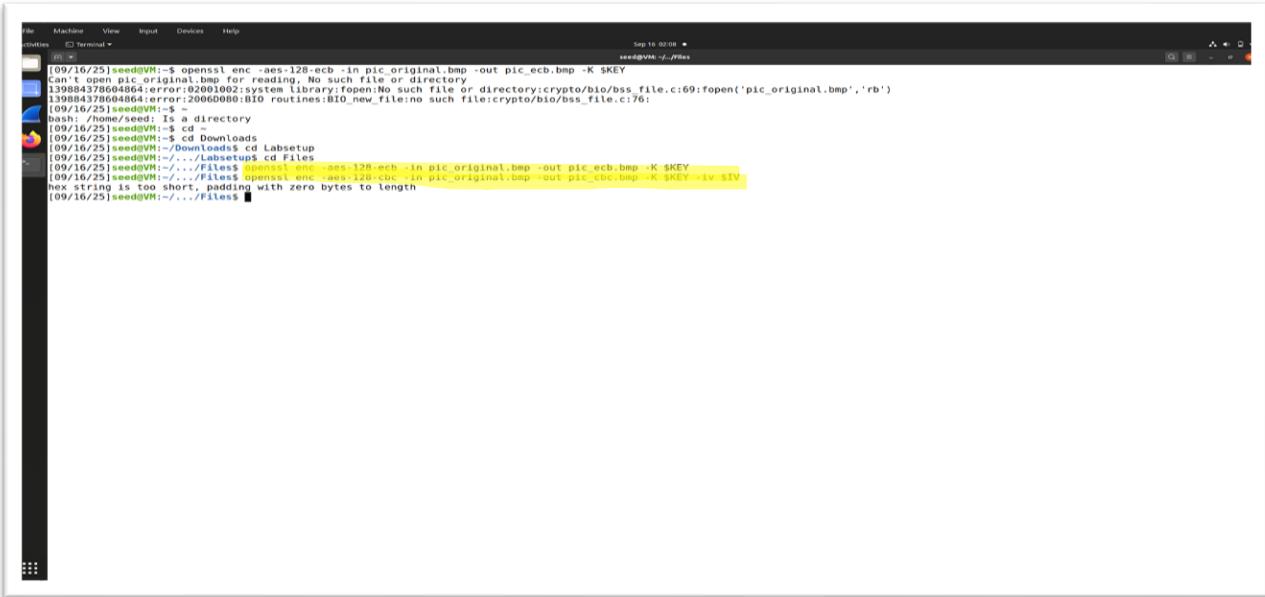
Screenshot 8: Encrypted message inside des.txt which was encrypted using DES.

A screenshot of a terminal window titled "Text Editor". The file "bf.txt" is open, containing the following encrypted text:
1A0dE1E5I
2x6x±hI

Screenshot 9: Encrypted message inside bf.txt which was encrypted using Blowfish.

Task 3- Encryption Mode– ECB vs. CBC.

- Now we will encrypt pic_original.bmp using ECB and CBC by inputting “openssl enc -aes-128-cbc -in pic_original.bmp -out pic_cbc.bmp -K \$KEY -iv \$IV” command and enter same for ECB (Note- ECB algorithm doesn’t need IV so we don’t need to enter it in command) as highlighted in Screenshot 10.



```
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -in pic_original.bmp -out pic_ecb.bmp -K $KEY
can't open pic_original.bmp for reading. No such file or directory
139884378664864:error:02001002:system library:fopen:No such file or directory:crypto/bio/bss_file.c:69:fopen('pic_original.bmp','rb')
139884378664864:error:20000000:BIO routines:BIO_new_file:no such file:crypto/bio/bss_file.c:76:
bash: /home/seed: Is a directory
[09/16/25]seed@VM:~$ cd ..
[09/16/25]seed@VM:~/Downloads$ cd Labsetup
[09/16/25]seed@VM:~/.../Labsetup$ cd Files
[09/16/25]seed@VM:~/.../Labsetup/Files$ ./Fileless openssl enc -aes-128-ecb -in pic_original.bmp -out pic_ecb.bmp -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~/.../Files$
```

Screenshot 10: Using command to encrypt “pic_original.bmp” using ECB and CBC.

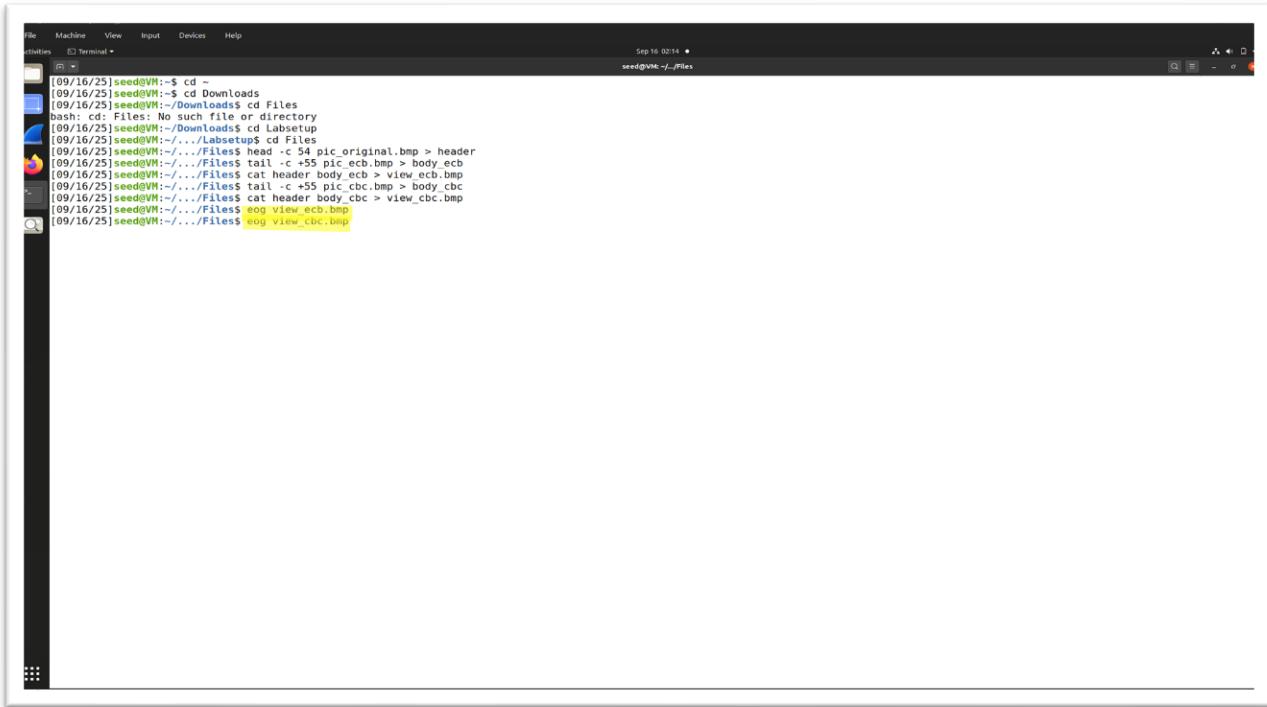
- All .bmp files have 54- byte headers so in order to make encrypted files viewable, we enter commands as highlighted in Screenshot 11.



```
[09/16/25]seed@VM:~$ cd ..
[09/16/25]seed@VM:~$ cd Downloads
[09/16/25]seed@VM:~/Downloads$ cd Files
hash: cd: Files: No such file or directory
[09/16/25]seed@VM:~/Downloads$ cd Labsetup
[09/16/25]seed@VM:~/.../Labsetup$ ./Fileless head -c 54 pic_original.bmp > header
[09/16/25]seed@VM:~/.../Fileless tail -c +55 pic_ecb.bmp > body_ecb
[09/16/25]seed@VM:~/.../Fileless tail -c +55 pic_cbc.bmp > body_cbc
[09/16/25]seed@VM:~/.../Fileless cat header body_ecb > view_ecb.bmp
[09/16/25]seed@VM:~/.../Fileless cat header body_cbc > view_CBC.bmp
[09/16/25]seed@VM:~/.../Fileless cmp pic_original.bmp view_ecb.bmp
[09/16/25]seed@VM:~/.../Fileless cmp pic_original.bmp view_CBC.bmp
[09/16/25]seed@VM:~/.../Fileless
```

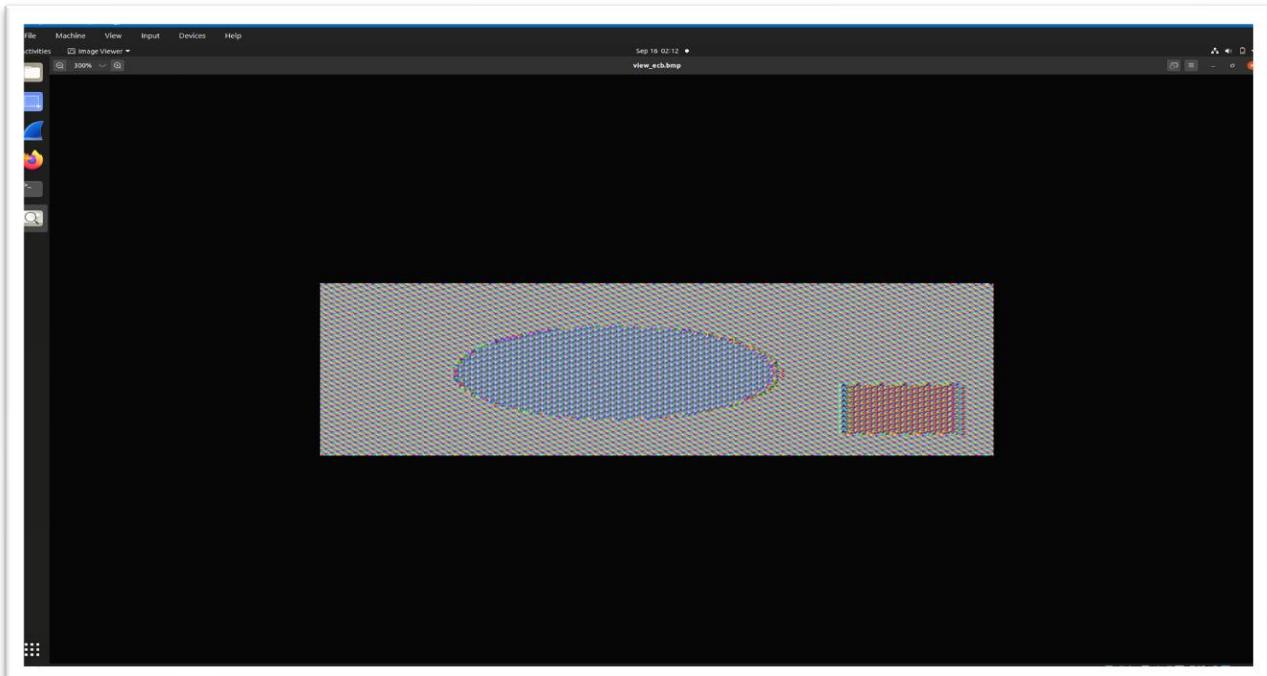
Screenshot 11: Using different commands to make .bmp file viewable.

- After that, we will run image viewer using “eog view_ecb.bmp” command and do the same for CBC file which is highlighted in Screenshot 12. The result of this command is shown in Screenshot 13 and 14 for ECB and EBC respectively.

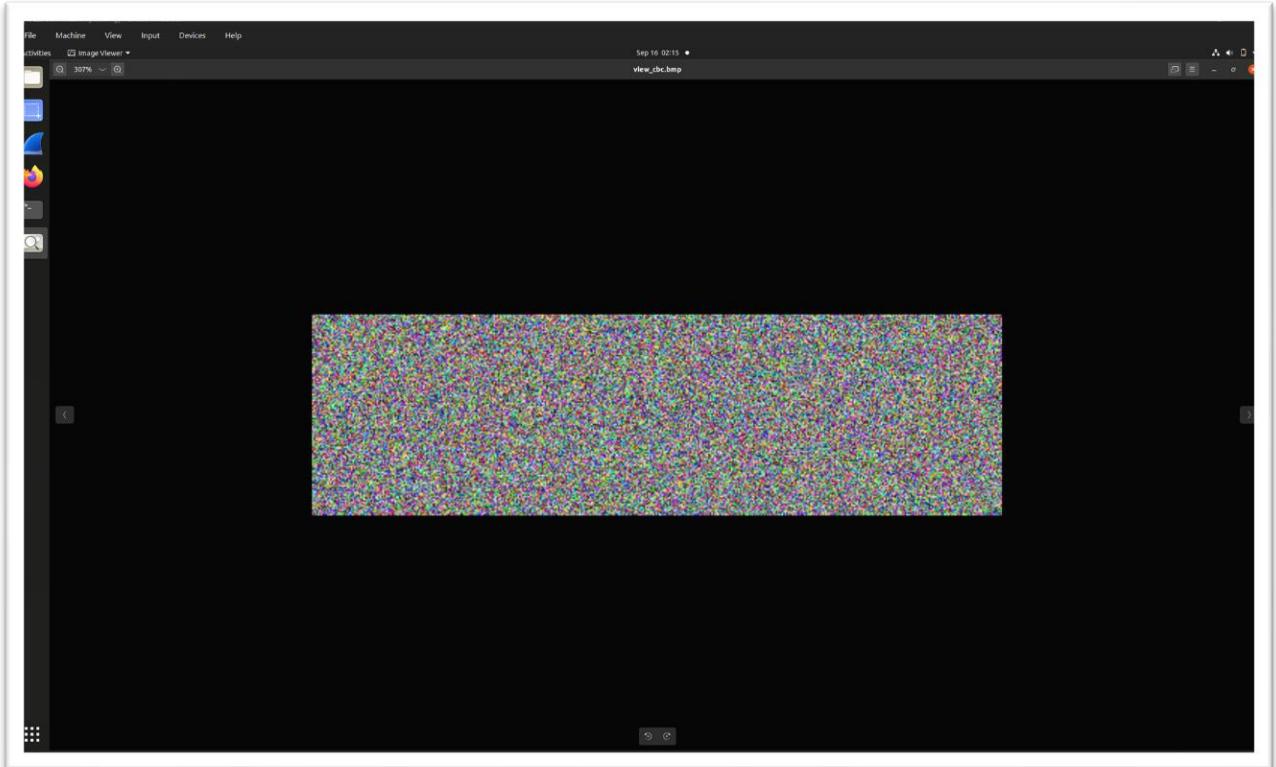


```
[09/16/25]seed@VM:~$ cd ~
[09/16/25]seed@VM:~$ cd Downloads
[09/16/25]seed@VM:~/Downloads$ cd Files
bash: cd: Files: No such file or directory
[09/16/25]seed@VM:~/Downloads$ cd Labsetup
[09/16/25]seed@VM:~/.../Labsetup$ cd Files
[09/16/25]seed@VM:~/.../Files$ tail -c +55 pic_original.bmp > header
[09/16/25]seed@VM:~/.../Files$ cat header body_ecb > view_ecb.bmp
[09/16/25]seed@VM:~/.../Files$ cat header body_cbc > view_cbc.bmp
[09/16/25]seed@VM:~/.../Files$ cat header body_cbc > view_cbc.bmp
[09/16/25]seed@VM:~/.../Files$ eog view_ecb.bmp
[09/16/25]seed@VM:~/.../Files$ eog view_cbc.bmp
```

Screenshot 12: Using commands to run image viewer for both encrypted files.



Screenshot 13: Image observed after encrypting “pic_original.bmp” using ECB.



Screenshot 14: Image observed after encrypting “pic_original.bmp” using CBC.

- So, when we observe the result image in Screenshot 13 of ECB, we can notice some visible structures (blue oval and red rectangle) which is revealing the pattern but Screenshot 14 of CBC, there is no visible patterns. Hence, ECB encryption still reveals plaintext information while CBC doesn't.

Task 4- Padding.

- Now, we will encrypt each file by using “openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K \$KEY” (include -iv \$IV for other three encryptions). So, we will enter commands as highlighted in Screenshot 15 and 16. Then we enter “xxd ecb.txt” command (and same for rest) which will show us the padding. We can observe from Screenshot 15 and 16 that ECB and CBC has some extra padding while CFB and OFB (which are stream ciphers) lacks it.

```
[09/16/25]seed@VM:~$ KEY="00112233445566778899aabbcdddeeff"
[09/16/25]seed@VM:~$ IV=0102030405060708090a0b0c0d0e0f00
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out cbc.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out cfb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-ofb -e -in plain.txt -out ofb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ xxd ecb.txt
00000000: 8857 85c9 4ac7 944b d03a ba04 c676 f6cf .W..J..K:...v.
00000008: d03a 85c9 a27c 0743 41b8 e832 5270 6a91 ...|A..2Rpj.
[09/16/25]seed@VM:~$ xxd ofb.txt
00000000: ab22 63cf ef60 582c f0dc ebbf f06c a674 ..c..`X.....l.t
00000008: f584 d7c4 e39c 75ch 99bc 7ed4 ef11 4f7d .....U..-=...0)
[09/16/25]seed@VM:~$ xxd cfb.txt
00000000: d3ee e656 e156 c0f1 cebe cc78 1662 3919 ...V.V.....x.b9.
00000010: d879 2da4 8c61 .y...a
[09/16/25]seed@VM:~$
```

Screenshot 15: To check padding in all four different modes (ECB, CBC and CFB).

```
[09/16/25]seed@VM:~$ openssl enc -aes-128-ofb -e -in plain.txt -out ofb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ xxd ofb.txt
00000000: 8857 85c9 4ac7 944b d03a ba04 c676 f6cf ...V.V.....x.b9.
00000008: 1132 f06b det5 .2.k.u
[09/16/25]seed@VM:~$
```

Screenshot 16: To check padding in all four different modes (OFB).

- Now, first we will create three new different-length plaintext using “echo -n "12345" > f1.txt” as highlighted in Screenshot 17. Then we will encrypt those three new files using CBC as observed in Screenshot 17.

```
[09/16/25]seed@VM:~$ KEY="00112233445566778899aabbcdddeeff"
[09/16/25]seed@VM:~$ IV="0102030405060708"
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cfb -e -in plain.txt -out cfb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cfb -e -in plain.txt -out ofb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ xxd ecb.txt
00000000: 8857 85c9 4ac7 9d4b d03a ba06 c676 f6cf .W..J..K:...v..
00000008: ab22 63cf ef68 582c f0dc ebbf f06c a674 .c..X.....l.t
00000010: r5b8 07c7 e3ef 75cb 99ba 7ed4 e1ff 4ff7d .....u..-.0)
00000018: 00000000: d3ee e656 e156 c9f1 cebe c78 1662 3919 ...V.V....x.b9.
00000010: d879 2dae 8c61 .y...a
[09/16/25]seed@VM:~$ echo -n "12345" > f1.txt
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f1.txt -out encf1.txt -K $KEY -iv $IV
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f2.txt -out encf2.txt -K $KEY -iv $IV
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f3.txt -out encf3.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f1.txt -out encf1.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f2.txt -out encf2.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f3.txt -out encf3.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$
```

Screenshot 17: Different commands to create three files and encrypt them using CBC.

- Then we will decrypt with no padding removal process using “openssl enc -aes-128-cbc -d -nopad -in encf1.txt -out decf1.txt -K \$KEY -iv \$IV” (do same for other two encryptions) as highlighted in Screenshot 18.

```
[09/16/25]seed@VM:~$ KEY="00112233445566778899aabbcdddeeff"
[09/16/25]seed@VM:~$ IV="0102030405060708"
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in plain.txt -out ecb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cfb -e -in plain.txt -out cfb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cfb -e -in plain.txt -out ofb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ xxd ecb.txt
00000000: 8857 85c9 4ac7 9d4b d03a ba06 c676 f6cf .W..J..K:...v..
00000008: ab22 63cf ef68 582c f0dc ebbf f06c a674 .c..X.....l.t
00000010: r5b8 07c7 e3ef 75cb 99ba 7ed4 e1ff 4ff7d .....u..-.0)
00000018: 00000000: d3ee e656 e156 c9f1 cebe c78 1662 3919 ...V.V....x.b9.
00000010: d879 2dae 8c61 .y...a
[09/16/25]seed@VM:~$ echo -n "12345" > f1.txt
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in f1.txt -out encf1.txt -K $KEY -iv $IV
[09/16/25]seed@VM:~$ echo -n "1234567890" > f2.txt
[09/16/25]seed@VM:~$ echo -n "1234567890123456" > f3.txt
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf1.txt -out decf1.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf2.txt -out decf2.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf3.txt -out decf3.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf1.txt -out decf1.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf2.txt -out decf2.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in encf3.txt -out decf3.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$
```

Screenshot 18: Command “openssl enc -aes-128-cbc -d -nopad -in encf1.txt -out decf1.txt -K \$KEY -iv \$IV” to decrypt with no padded removal.

- Then finally, we can use “hexdump -C decf1.txt” command (do the same for other two files) as highlighted in Screenshot 19. So, we can observe that there are some padding bytes like 0b for 11-byte padding and decf3 doesn't have any as length is maximum to fill padding in it as we can observe in Screenshot 19.

```

[09/16/25]seed@VM:~$ hexdump -C decf1.txt
00000000 31 32 33 34 35 0b |12345.....|
00000010
[09/16/25]seed@VM:~$ hexdump -C decf2.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |1234567890.....|
00000010
[09/16/25]seed@VM:~$ hexdump -C decf3.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|
00000020
[09/16/25]seed@VM:~$
```

Screenshot 19: Using command “hexdump -C decf1.txt” to check for present padding bytes “0b” in them.

Task 5- Error Propagation– Corrupted Cipher Text.

- First, we will create a 1000 bytes long text file using “head -c 1000 /dev/urandom > longfile.txt” command as highlighted in Screenshot 20. Then we will encrypt file with AES for all four modes (ECB, CBC, CFB, OFB) using “openssl enc -aes-128-ecb -e -in longfile.txt -out enc_longecb.txt -K \$KEY” (put -iv -IV on other three modes) as observed in Screenshot 20. Then we need to corrupt 55th byte inside all encrypted files using “bless enc_longecb.txt” and changing value of 55th byte as observed in Screenshot 21.

```
[09/16/25]seed@VM:~$ head -c 1000 /dev/urandom > longfile.txt
[09/16/25]seed@VM:~$ openssl enc -aes-128-ecb -e -in longfile.txt -out enc_longcbc.txt -K $KEY -iv $IV
warning: iv not used by this cipher
[09/16/25]seed@VM:~$ openssl enc -aes-128-cbc -e -in longfile.txt -out enc_longcbc.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-cfb -e -in longfile.txt -out enc_longcfb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ openssl enc -aes-128-ofb -e -in longfile.txt -out enc_longofb.txt -K $KEY -iv $IV
hex string is too short, padding with zero bytes to length
[09/16/25]seed@VM:~$ bless enc_longcbc.txt
Gtk-Messages: 03:33:05.730: Failed to load module "cumberba-gtk-module"
Could not find file "/home/seed/.config/bless/plugins".
Could not find a part of the path "/home/seed/.config/bless/plugins".
Could not find a part of the path "/home/seed/.config/bless/plugins".
Could not find a part of the path "/home/seed/.config/bless/export_patterns".
Could not find file "/home/seed/.config/bless/history.xml".
Marshaling delete event signal
Exception in Gtk# callback delegate
Note: Applications can use Glib.ExceptionManager.UnhandledException to handle the exception.
System.Runtime.InteropServices.MarshalDirectiveException: An attempt was made to invoke a method or an interface.
at Glib.Gui.FileService.SaveFileInternal (Glib.Gui.FileView, System.String filename, System.Boolean synchronous) [0x0014e] in <36541bb0d43e468db07d5b1914b5d1bf>:0
at Glib.Gui.FileService.SaveFile (Glib.Gui.FileView, System.String filename, System.Boolean forceSaveAs4, System.Boolean synchronous) [0x0005f] in <36541bb0d43e468db07d5b1914b5d1bf>:0
at Glib.Gui.FileService.AskForSaveIfFileChanged (Glib.Gui.FileView args) [0x0041] in <36541bb0d43e468db07d5b1914b5d1bf>:0
at Glib.Gui.FileService.AskForSaveIfFileChanged (Glib.Gui.FileView args) [0x00041] in <36541bb0d43e468db07d5b1914b5d1bf>:0
at Glib.Gui.FileService.TryQuit () [0x00000] in <36541bb0d43e468db07d5b1914b5d1bf>:0
at Bless.Main.MainWindowDeleteEvent (System.Object, Gtk.WidgetEventArgs args) [0x00026] in <c2904a26fd0348a996849b2f4e8293c>:0
at (wrapper managed-to-native) System.Reflection.RuntimeMethodInfo.InternalInvoke(System.Reflection.RuntimeMethodInfo,object,object[])
at System.Reflection.RuntimeMethodInfo.InternalInvoke (System.Object obj, System.Reflection.BindingFlags invokeAttr, System.Reflection.Binder binder, System.Object[] parameters, System.Globalization.CultureInfo culture) [0x0007c] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at System.Reflection.RuntimeMethodInfo.InternalInvoke (System.Object obj, System.Reflection.BindingFlags invokeAttr, System.Reflection.Binder binder, System.Object[] parameters, System.Globalization.CultureInfo culture) [0x00000] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at System.Reflection.MethodBase.Invoke (System.Object obj, System.Object[] parameters) [0x00000] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at System.Delegate.DynamicInvokeImpl (System.Object[] args) [0x0010d] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at System.MulticastDelegate.DynamicInvokeImpl (System.Object[] args) [0x0000b] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at System.Delegate.DynamicInvoke (System.Object[] args) [0x00000] in <2b418a781c4ca0993feeaaf67f1e7f>:0
at Glib.Signal.ClosureInvokeD (Glib.ClosureInvokeD args) [0x0006d] in <918680a31aa84cb9cfafa7cab56ea29bb>:0
at Glib.SignalClosure.Invoke (Glib.ClosureInvokeD args) [0x0000c] in <918680a31aa84cb9cfafa7cab56ea29bb>:0
at Glib.SignalClosure.MarshalCallback (System.IntPtr return_val, System.UInt32 n_params_vals, System.IntPtr invocation_hint, System.IntPtr marshaled_data) [0x00000] in <918680a31aa84cb9cfafa7cab56ea29bb>:0
at Glib.ExceptionManager.RaiseUnhandledException (System.Exception e, System.Boolean is_terminal) [0x00000] in <918680a31aa84cb9cfafa7cab56ea29bb>:0
at Glib.SignalClosure.MarshalCallback (System.IntPtr return_val, System.UInt32 n_params_vals, System.IntPtr invocation_hint, System.IntPtr marshaled_data) [0x00000] in <918680a31aa84cb9cfafa7cab56ea29bb>:0
at Glib.Application.gtk_main () [0x00000] in <9147213b1394fa4ea87012594abed2f>:0
at Gtk.Application.Run () [0x00000] in <9147213b1394fa4ea87012594abed2f>:0
at Bless.Main.(ctor) (System.String[] args) [0x00000] in <29904a26fd0348a996849b2f4e8293c>:0
at Bless.Main.Main (System.String[] args) [0x00000] in <29904a26fd0348a996849b2f4e8293c>:0
[09/16/25]seed@VM:~$ bless enc_longcbc.txt
Gtk-Messages: 03:49:50.666: Failed to load module "cumberba-gtk-module"
Could not find a part of the path "/home/seed/.config/bless/plugins".
Could not find a part of the path "/home/seed/.config/bless/plugins".
```

Screenshot 20: Using various commands to create 1000 bytes file, encrypt them with AES and then corrupt 55th byte inside it using “bless” command.



Screenshot 21: Corrupting the 55th byte in encrypted files using “bless” command.

- Then after this, we will decrypt all encrypted files using “`openssl enc -aes-128-ecb -d -in enc_longecb.txt -out dec_longecb.txt -K $KEY`” for all other modes (include `-iv $IV` in all other modes) as highlighted in Screenshot 22.



Screenshot 22: Decrypted all encrypted files after corrupting it.

- Now, after that we have to observe any significant changes caused due to corrupting files from original files. So, in Screenshot 23, it shows the original file's content and in Screenshots 24,25,26 and 27.

Screenshot 23: Content inside original file “longfile.txt”.

Screenshot 24: Changed content inside decrypted file “dec_lonecb.txt”.

Screenshot 25: Changed content inside decrypted file “dec_longcbc.txt”.

Screenshot 26: Changed content inside decrypted file “dec_longcfb.txt”.

Screenshot 27: Changed content inside decrypted file “dec_longofb.txt”.

- So, finally after all, we can observe and report from Screenshots 24,25,26 and 27 that:-
 - ECB- Only the block which was corrupted are changed.
 - CBC- The block with error and the next block so total two blocks are changed.
 - CFB/OFB- Only the byte which was corrupted is altered.

Task 6- Initial Vector (IV) and Common Mistakes.

- Given:- P1= “This is a known message!”

C1= a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

C2= bf73bcd3509299d566c35b5d450337e1bb175f903fafc159

- Find:- P2
- So, we will use $K = P1 \oplus C1$
 $P2 = C2 \oplus K$ (K is same for OFB mode)
- Now, P1= “This is a known message!”, we need to find Hex(ASCII) value

Character	Decimal	Hex
T	84	54
h	104	68
i	105	69
s	115	73
(space)	32	20
i	105	69
s	115	73
(space)	32	20
a	97	61
(space)	32	20
k	107	6b
n	110	6e
o	111	6f
w	119	77
n	110	6e
(space)	32	20
m	109	6d
e	101	65
s	115	73
s	115	73
a	97	61
g	103	67
e	101	65
!	33	21

So, whole Hex value = 54 68 69 73 20 69 73 20 61 20 6b 6e 6f 77 6e 20 6d 65 73 73 61 67 65 21
with length= 24 bytes.

- $K = P1 \oplus C1$

Now, XOR byte by byte:

- $54 \oplus a4 = f0$
- $68 \oplus 69 = 01$
- $69 \oplus b1 = d8$
- $73 \oplus c5 = b6$
- $20 \oplus 02 = 22$
- $69 \oplus c1 = a8$
- $73 \oplus ca = b9$
- $20 \oplus b9 = 99$
- $61 \oplus 66 = 07$
- $20 \oplus 96 = b6$
- $6b \oplus 5e = 35$
- $6e \oplus 50 = 3e$
- $6f \oplus 42 = 2d$
- $77 \oplus 54 = 23$
- $6e \oplus 38 = 56$
- $20 \oplus e1 = c1$
- $6d \oplus bb = d6$
- $65 \oplus 1b = 7e$
- $73 \oplus 5f = 2c$
- $73 \oplus 90 = e3$
- $61 \oplus 37 = 56$
- $67 \oplus a4 = c3$
- $65 \oplus c1 = a4$
- $21 \oplus 59 = 78$

So, $K = f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478$

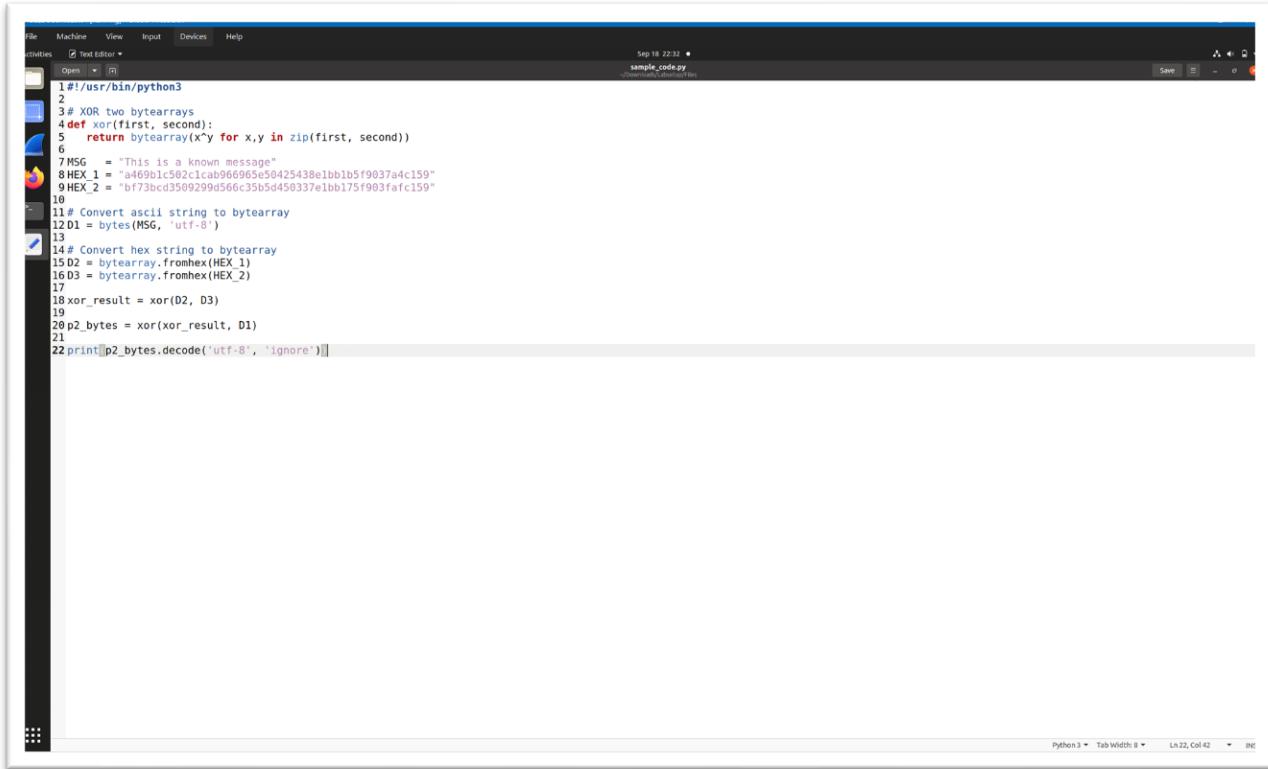
- $P2 = C2 \oplus K$

Now XOR byte by byte:

- $bf \oplus f0 = 4f$ (ASCII O)
- $73 \oplus 01 = 72$ (ASCII r)
- $bc \oplus d8 = 64$ (ASCII d)
- $d3 \oplus b6 = 65$ (ASCII e)
- $50 \oplus 22 = 72$ (ASCII r)

- $92 \oplus a8 = 3a$ (ASCII :)
• $99 \oplus b9 = 20$ (space)
• $d5 \oplus 99 = 4c$ (ASCII L)
• $66 \oplus 07 = 61$ (ASCII a)
• $c3 \oplus b6 = 75$ (ASCII u)
• $5b \oplus 35 = 6e$ (ASCII n)
• $5d \oplus 3e = 63$ (ASCII c)
• $45 \oplus 2d = 68$ (ASCII h)
• $03 \oplus 23 = 20$ (space)
• $37 \oplus 56 = 61$ (ASCII a)
• $e1 \oplus c1 = 20$ (space)
• $bb \oplus d6 = 6d$ (ASCII m)
• $17 \oplus 7e = 69$ (ASCII i)
• $5f \oplus 2c = 73$ (ASCII s)
• $90 \oplus e3 = 73$ (ASCII s)
• $3f \oplus 56 = 69$ (ASCII i)
• $af \oplus c3 = 6c$ (ASCII l)
• $c1 \oplus a4 = 65$ (ASCII e)
• $59 \oplus 78 = 21$ (ASCII !)
- So, P2= "Order: Launch a missile!"

- Here are some changes which I did to “sample_code.py” in order to get P2 as seen in Screenshot 28.



```

File Machine View Input Devices Help
Activities Text Editor Sep 18 22:32 sample_code.py
Open Save
1#!/usr/bin/python3
2
3# XOR two bytearrays
4def xor(first, second):
5    return bytearray(x^y for x,y in zip(first, second))
6
7MSG = "This is a known message"
8HEX_1 = "a469b1c5021cab966965e50425438elbb1b5f9037a4c159"
9HEX_2 = "bf73bcd3509299d566c35b5d450337elbb175f903fafc159"
10
11# Convert ascii string to bytearray
12D1 = bytes(MSG, 'utf-8')
13
14# Convert hex string to bytearray
15D2 = bytearray.fromhex(HEX_1)
16D3 = bytearray.fromhex(HEX_2)
17
18xor_result = xor(D2, D3)
19
20p2_bytes = xor(xor_result, D1)
21
22print(p2_bytes.decode("utf-8", 'ignore'))

```

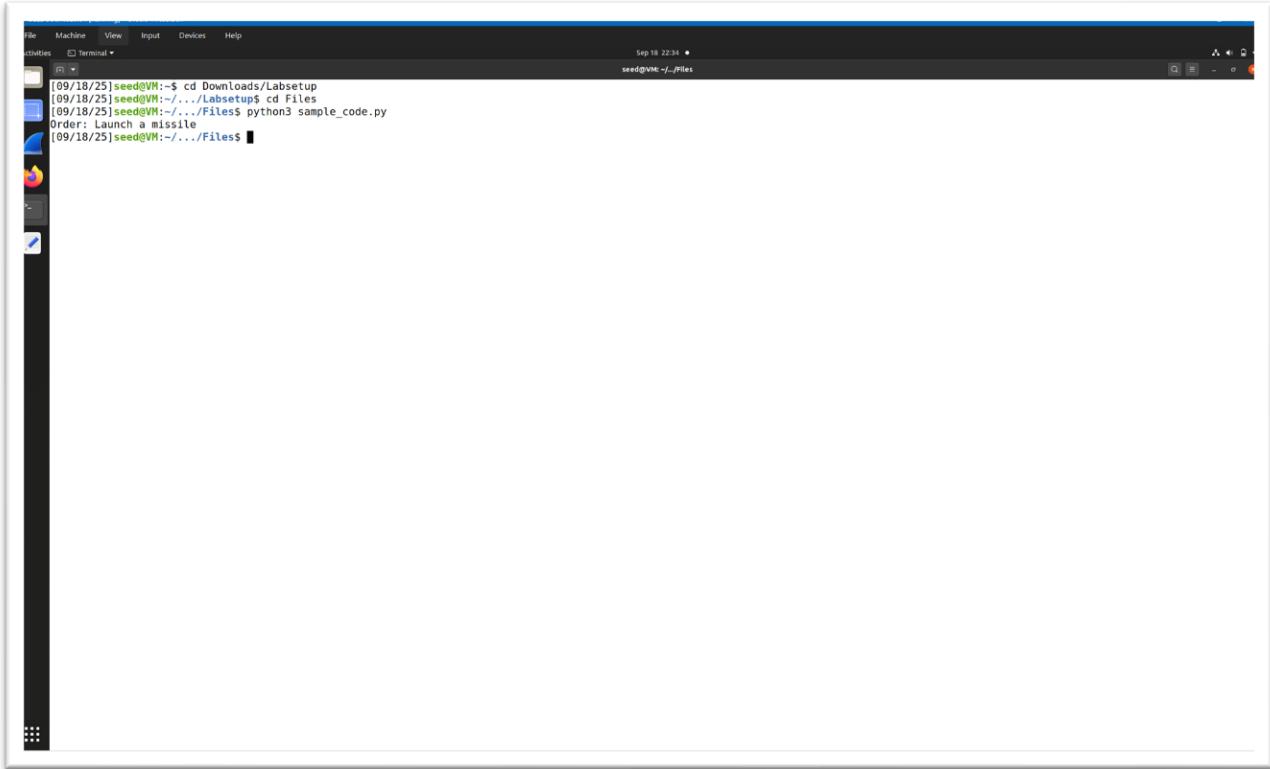
The screenshot shows a terminal window with the title "sample_code.py". The code in the terminal is as follows:

```

1#!/usr/bin/python3
2
3# XOR two bytearrays
4def xor(first, second):
5    return bytearray(x^y for x,y in zip(first, second))
6
7MSG = "This is a known message"
8HEX_1 = "a469b1c5021cab966965e50425438elbb1b5f9037a4c159"
9HEX_2 = "bf73bcd3509299d566c35b5d450337elbb175f903fafc159"
10
11# Convert ascii string to bytearray
12D1 = bytes(MSG, 'utf-8')
13
14# Convert hex string to bytearray
15D2 = bytearray.fromhex(HEX_1)
16D3 = bytearray.fromhex(HEX_2)
17
18xor_result = xor(D2, D3)
19
20p2_bytes = xor(xor_result, D1)
21
22print(p2_bytes.decode("utf-8", 'ignore'))

```

Screenshot 28: Newly updated “sample_code.py” to find P2.



```
[09/18/25]seed@VM:~$ cd Downloads/Labsetup  
[09/18/25]seed@VM:~/Downloads/Labsetup$ cd Files  
[09/18/25]seed@VM:~/Downloads/Labsetup/Files$ python3 sample_code.py  
Order: Launch a missile  
[09/18/25]seed@VM:~/Downloads/Labsetup/Files$
```

Screenshot 29: Found P2 which is “Order: Launch a missile”.


```
[09/18/25]seed@VM:-$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's Ciphertext: b18f53209fdec01f81a395e34aa9cf86b
The IV used      : a81b2117244d3c85e766b1836e9cf94
Next IV          : 772de66f244d3c85e766b1836e9cf94
Your plaintext :
```

Screenshot 30: Ciphertext received from Oracle by entering “nc 10.9.0.80 3000”.

```
[File Machine View Input Devices Help]
[Activities Terminal Sep 18 19:45]
[09/18/25] seed@VM:~$ nc 10.9.0.88 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertext: bf8f5399fdec1f81a395e34aa9c786b
The IV used : a81b2117244d3c85e766b1836e9c4f94
Next IV : 772de65f244d3c85e766b1836e9c4f94
Your plaintext : 8653b4750d0c0d0d0d09dc3dd0d0d0d0d0
Your ciphertext: 1f67ba2a85ec95a1139204a3007e4ef57b364abe23fde8a73bbd969ffc92ca4
Next IV : 7f60lab9244d3c85e766b1836e9c4f94
Your plaintext : █
```

Screenshot 31: Result ciphertext which we will match with Bob's ciphertext to check for secret message.

References:-

- [ChatGPT](#) for some commands recommendations.
- [Ask Ubuntu](#) for some doubt during encryption.
- <https://www.google.com> to search steps for setup.