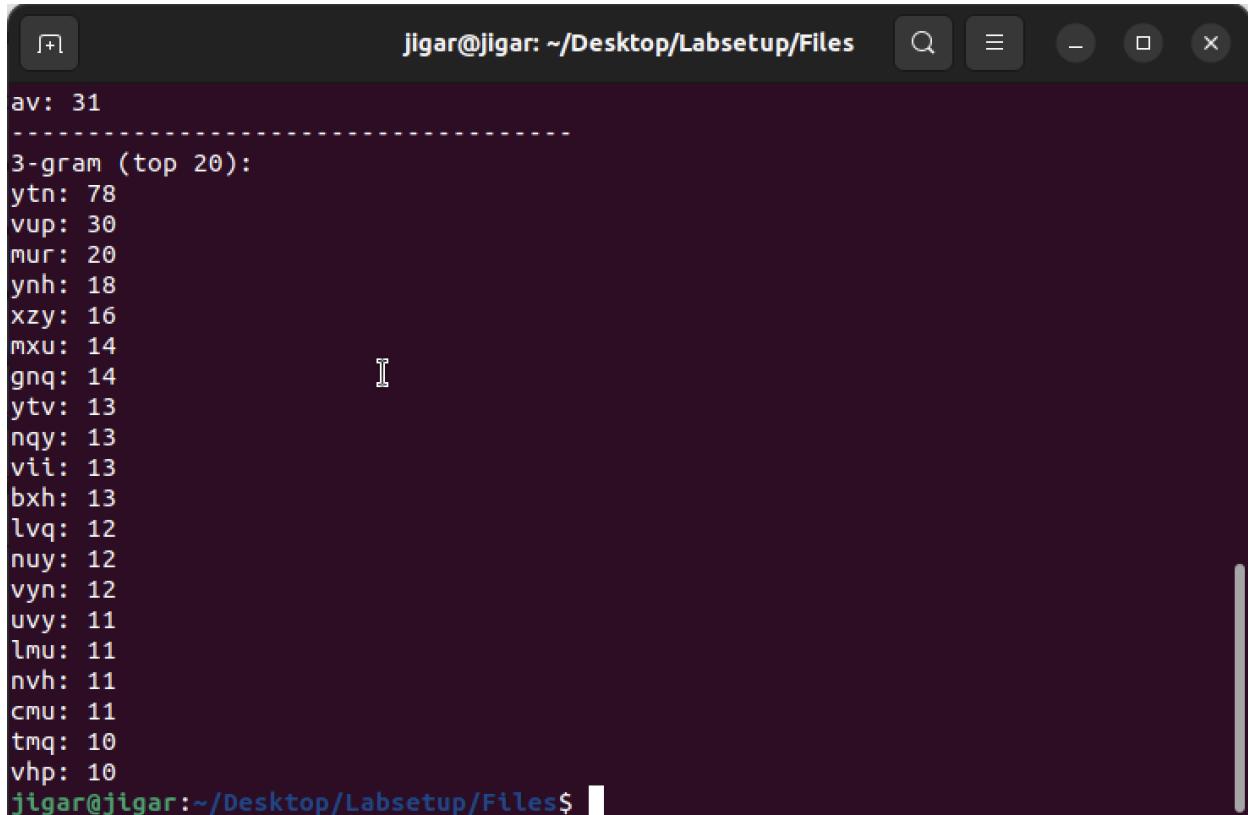


Task 1: Frequency Analysis

In this task monoalphabetic cipher is used for encryption and this cipher is not so secure. it can be easily subjected to frequency analysis.

We got single letter frequencies, bigram frequencies and trigram frequencies using freq.py file. With the help of difference resources on internet we can guess the most frequent letters and then we can guess other words.

Ex.:



A screenshot of a terminal window titled "jigar@jigar: ~/Desktop/Labsetup/Files". The window displays a list of trigrams and their frequencies. The output starts with "av: 31" followed by a dashed line, then "3-gram (top 20)". Below this, a list of trigrams and their counts is shown, starting with "ytn: 78" and ending with "vhp: 10". The terminal prompt "jigar@jigar:~/Desktop/Labsetup/Files \$" is visible at the bottom.

```
av: 31
-----
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10
jigar@jigar:~/Desktop/Labsetup/Files$
```

In ciphertext most frequent trigram letters are *ytn* and most frequent english word is *the* so **y** will be **t**, **t** will be **h** and **n** wil be **e**. Second most freq is *vup* and in english it's *and*. So that's how you can guess the correct letters.

- Key for ciphertext is : cfmypybirlqxwiejdsgkhnazotu
- Replace this key with abcdefghijklmnopqrstuvwxyz with the use of command ‘tr’.
- \$ tr ‘abcdefghijklmnopqrstuvwxyz’ ‘cfmypybirlqxwiejdsgkhnazotu’ \
 <ciphertext.txt> plaintext.txt

This command will create a new text file with name plaintext.txt and in that file it will replace all letter with key letter and we can get decrypted message back with the help of frequency analysis.

```
● ● ● ciphertext.txt
ytn xqavhq yzhu xu qzupvd ltmat gnncg vgxxzy hmrtv vbynh ytmg ixur qyhvurn
vlvhpg yhme ytn qvrrnh bnnig imsn v uxuvrnuvhmvu yxx
ytn vlvhpg hvan lvg gxxsnupnp gd ytn pncmqn xb tvhfnd lnmuqynmu vy myq xzyqny
vup ytn veevhnuv mceixqmxu xb tmq bmic axcevud vy ytn nup vup my lvg qtvenp gd
ytn ncncrnuan xb cnyxx ymcng ze givasrxlu eximymag vhacavupd vayfmqac vup
v uvymxuv3 axufnhqyyymu vg ghmnbg vup cvp vg v bnfh phnvc vgxxzy ltnytvh ytnhn
xzrty yx gn v ehngmpnuy lmubhnd ytn qnvqgxu pmpuy ozqy qnnc nkyhv ixur my lvg
nkyhv ixur gnavzqn ytn xqavhq lnhn cxfnp yx ytr bmhqv lnnsnup mu cvhat yx
vfxmp axubimaymur lmyt ytn aixqmur anhncxud xb ytn lmuynh xidcemag ytvusq
ednxuratvur
xun gmr jznayymxu qzhbxzupmru ytmq dnvhg vavpcnd vlvhpg mq txl xh mb ytn
anhncxud lmi vpphngq cnyxx ngenamviid vbynh ytn rxipnu rixqng ltmat gnacn
v ozqimivuy axcmurxzy evhyd bxh ymcng ze ytn cxfnctuy genvhtnvpnp gd
exlnhbzi txidlxp lxncu ltx tnienp hvmqn cmiimxug xb pxiivhq yx brmrtv qnkzvi
tvhvqacnuy vhxuzp ytn axzuyhd
omruvimir ytnmh qzeexhy rxipnu rixqng vyyupnng qlvytvp ytncqnfq mu givas
gexhny iveni emug vup qxzupnp xbb vgxxzy qnmqy exlnh mcgvivuang bhxc ytn hnp
avheny vup ytn qvvrn xu ytn vmb n lvq aviinp xzy vgxxzy evd munjzmyd vbynh
myg bxhcmb vuatxh avyy gvpinh jzmy xuan qtn invhunp ytv ytn lvg cysmur bvh
inqq ytvu v cvin axtxqy vup pzhmru ytn anhncxud uvyyvimm exhycvu yxxs v gizuy
vup qvymqbdmru pmr v ytn viicvin hxqynh xb uxcmuvyvp pmhnayxqg txl axzip
ytvy gn yxeenp
vq my yzhuq xzy vy invqy mu ynhcq xb ytn xqavhq my ehxgvqid lxuy gn
lxncu mufxifnp mu ymcng ze qvmp ytvv viytxrt ytn rixqng qmrumbmnp ytn
mumymvymfng ivzuat ytn unfnh muynupnp my yx gn ozqy vu vlvhpg qnvqgxu
```

This was ciphertext

```
● ● ● plaintext.txt
the oscars turn on sunday which seems about right after this long strange
awards trip the bagger feels like a nonagenarian too
the awards race was bookended by the demise of harvey weinstein at its outset
and the apparent implosion of his film company at the end and it was shaped by
the emergence of metoo times up blackgown politics armcandy activism and
a national conversation as brief and mad as a fever dream about whether there
ought to be a president winfrey the season didnt just seem extra long it was
extra long because the oscars were moved to the first weekend in march to
avoid conflicting with the closing ceremony of the winter olympics thanks
pyeongchang
one big question surrounding this years academy awards is how or if the
ceremony will address metoo especially after the golden globes which became
a jubilant comingout party for times up the movement spearheaded by
powerful hollywood women who helped raise millions of dollars to fight sexual
harassment around the country
signaling their support golden globes attendees swathed themselves in black
sported lapel pins and sounded off about sexist power imbalances from the red
carpet and the stage on the air e was called out about pay inequity after
its former anchor catt sadler quit once she learned that she was making far
less than a male cohost and during the ceremony natalie portman took a blunt
and satisfying dig at the allmale roster of nominated directors how could
that be topped
as it turns out at least in terms of the oscars it probably wont be
women involved in times up said that although the globes signified the
initiatives launch they never intended it to be just an awards season
```

And this is plain text that we got from deciphering cipher text.

Task 2: Encryption using Different Ciphers and Modes

– For this task we used openssl command for encrypt and decrypt files.

```
[06/18/22] seed@VM:~/.../Task4$ openssl enc -aes-128-ecb -e -in ex.txt -out ecb.txt
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/18/22] seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in ex.txt -out cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/18/22] seed@VM:~/.../Task4$ openssl enc -aes-128-cfb -e -in ex.txt -out cfb.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/18/22] seed@VM:~/.../Task4$ openssl enc -aes-128-ofb -e -in ex.txt -out ofb.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
```



The terminal window shows the encrypted output of the 'ex.txt' file using the 'aes-128-ofb' cipher. The text is heavily redacted with a large red marker, obscuring most of the content. The redaction starts with the first few lines of the encrypted text and continues through the entire block of data shown in the terminal.

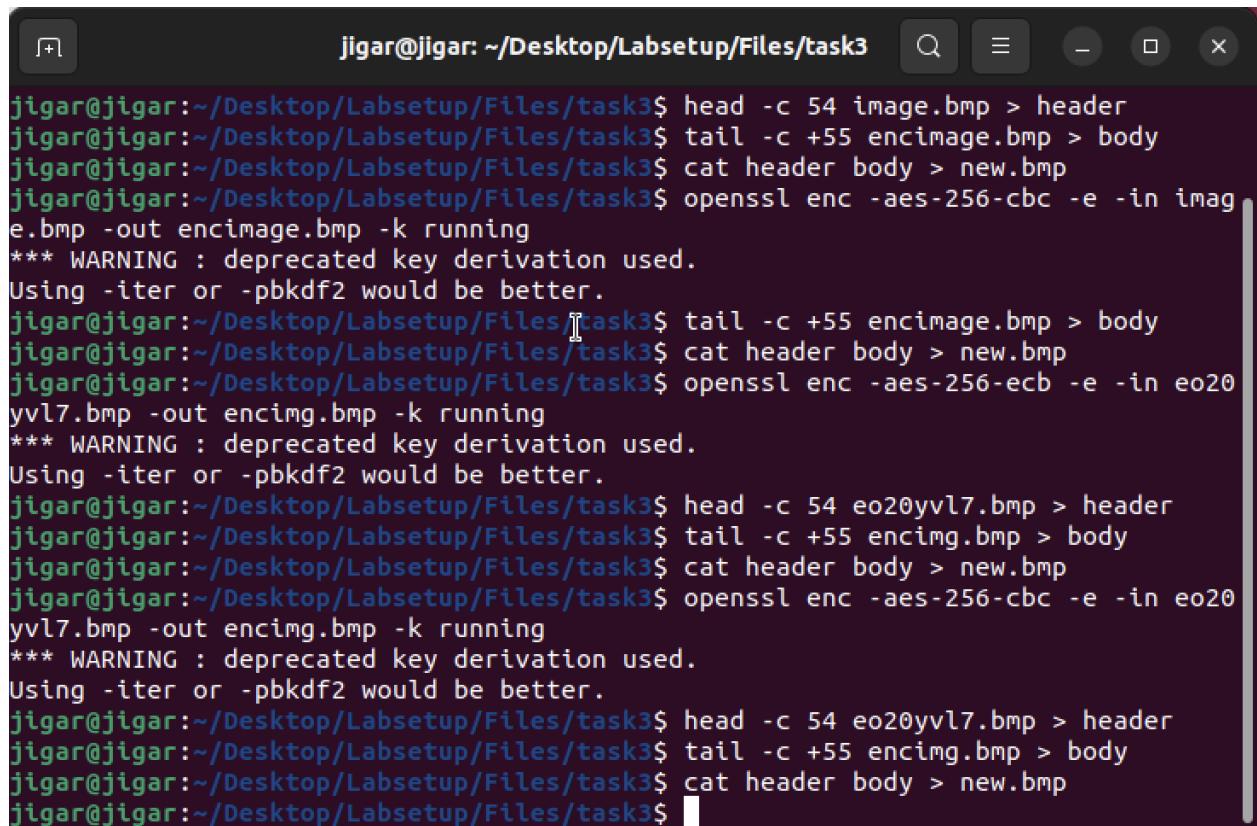
Encrypted cipher text.

(we used previous task's plaintext for this task)

- We used -aes-128-ecb, -aes-128-cbc, -aes-128-cfb, -aes-128-ofb, -aes-256, -aria-128-cbc modes for encryption.
 - In this process we used several commands for encryption and decryption..
 - **-in <file>** for input file
 - **-out <file>** for output file
 - **-k/-iv** for key/iv
 - **-e** for encryption
 - **-d** for decryption.
-

Task 3: Encryption Mode – ECB vs. CBC

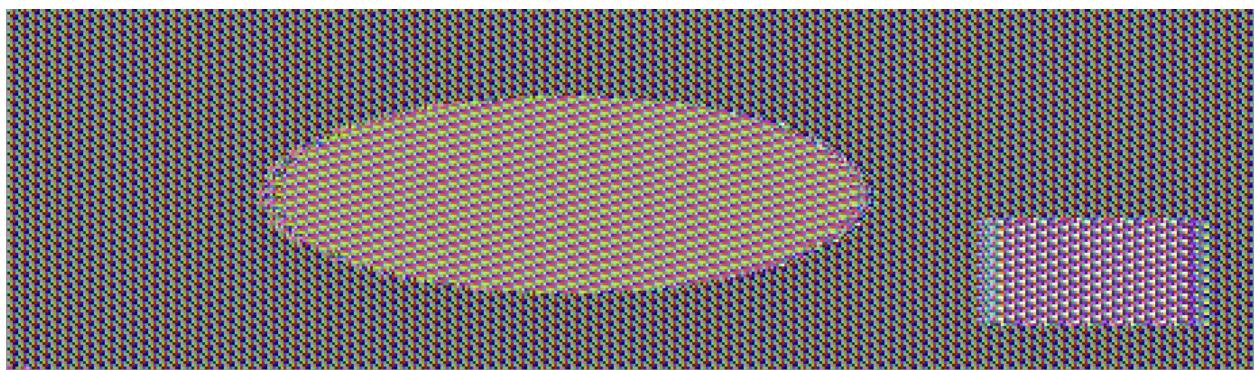
In this task we encrypt images using ecb and cbc modes with the help of openssl command



```
jigar@jigar:~/Desktop/Labsetup/Files/task3$ head -c 54 image.bmp > header
jigar@jigar:~/Desktop/Labsetup/Files/task3$ tail -c +55 encimage.bmp > body
jigar@jigar:~/Desktop/Labsetup/Files/task3$ cat header body > new.bmp
jigar@jigar:~/Desktop/Labsetup/Files/task3$ openssl enc -aes-256-cbc -e -in image.bmp -out encimage.bmp -k running
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jigar@jigar:~/Desktop/Labsetup/Files/task3$ tail -c +55 encimage.bmp > body
jigar@jigar:~/Desktop/Labsetup/Files/task3$ cat header body > new.bmp
jigar@jigar:~/Desktop/Labsetup/Files/task3$ openssl enc -aes-256-ecb -e -in eo20yvl7.bmp -out encimg.bmp -k running
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jigar@jigar:~/Desktop/Labsetup/Files/task3$ head -c 54 eo20yvl7.bmp > header
jigar@jigar:~/Desktop/Labsetup/Files/task3$ tail -c +55 encimg.bmp > body
jigar@jigar:~/Desktop/Labsetup/Files/task3$ cat header body > new.bmp
jigar@jigar:~/Desktop/Labsetup/Files/task3$ openssl enc -aes-256-cbc -e -in eo20yvl7.bmp -out encimg.bmp -k running
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jigar@jigar:~/Desktop/Labsetup/Files/task3$ head -c 54 eo20yvl7.bmp > header
jigar@jigar:~/Desktop/Labsetup/Files/task3$ tail -c +55 encimg.bmp > body
jigar@jigar:~/Desktop/Labsetup/Files/task3$ cat header body > new.bmp
jigar@jigar:~/Desktop/Labsetup/Files/task3$
```



This is the image before encryption



This is the image after encryption with ECB mode

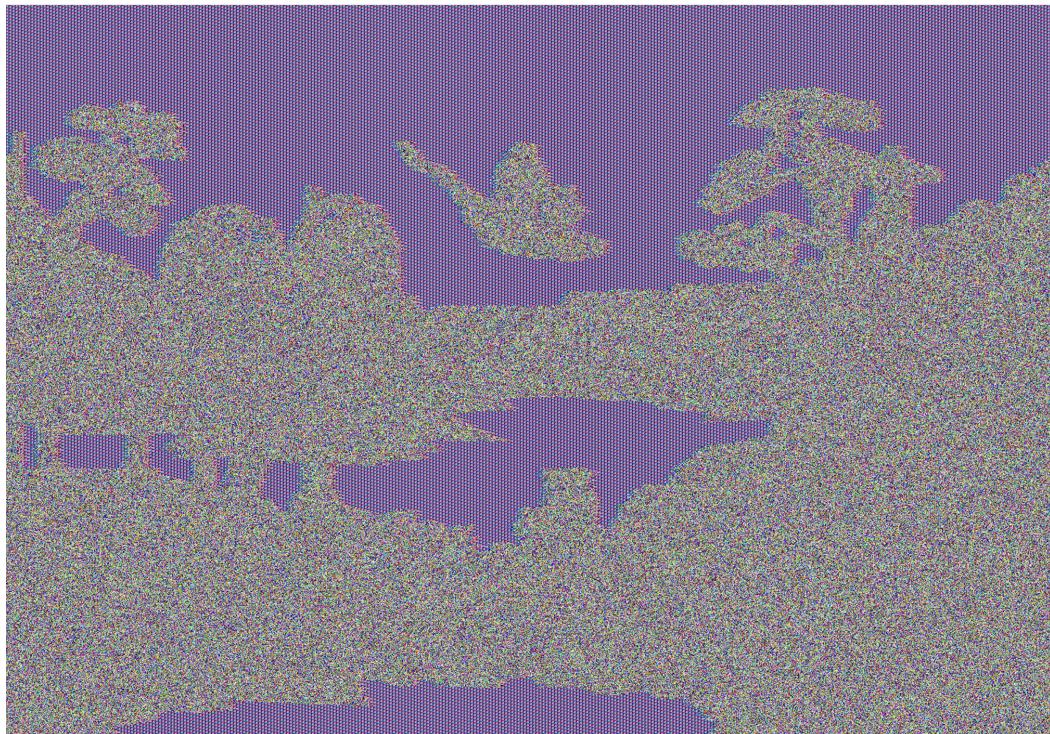


This is the image after encryption with CBC mode

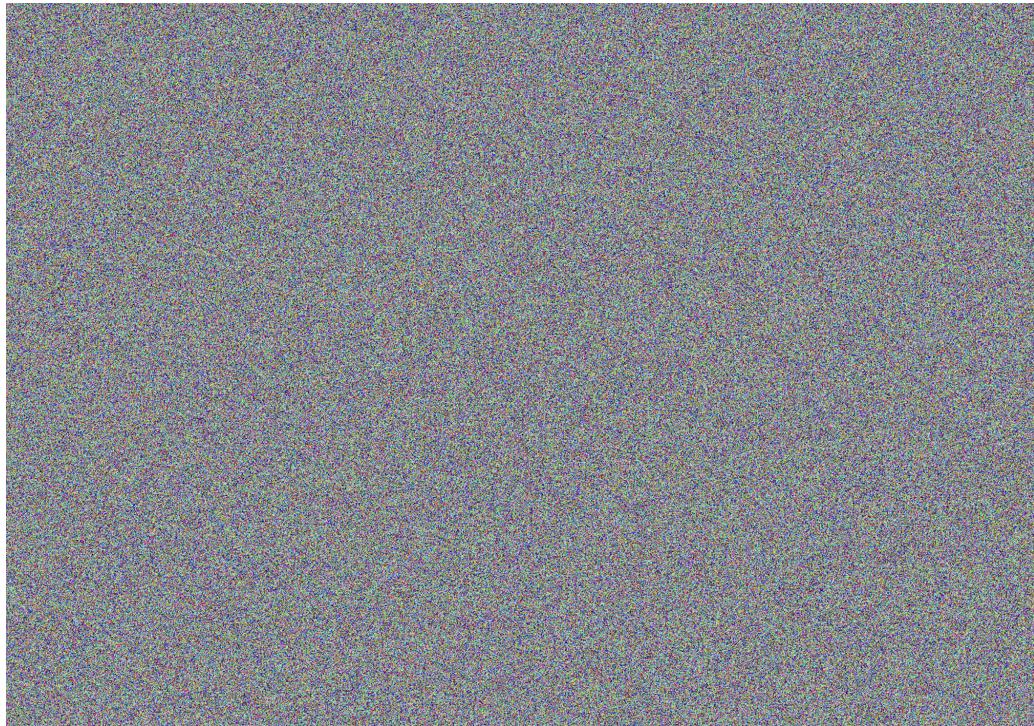
– After observing encrypted images we can tell that with ecb mode we can still get some information about the real image and with cbc we can not get any information about the real image. This same thing we observe with another image.



This is the image before encryption



This is the image after encryption with ECB mode



This is the image after encryption with CBC mode

- In this image also we can guess the shape of real image from encrypted image with ecb and can not get any information about real image with cbc.
 - In conclusion we can say that using CBC mode for encryption is more safe and make it hard to derive any information about real image than ECB mode.
-

Task 4: Padding

(1)

- We encrypted a 5 byte file with cbc, cfb, ecb,ofb modes

```
jigar@jigar: ~/Desktop/Labsetup/Files/task4.2$ openssl enc -aes-128-ecb -e -in text1.txt -out encecb.txt -k 00112233445566778889aabcccddeff
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
jigar@jigar:~/Desktop/Labsetup/Files/task4.2$ openssl enc -aes-128-cfb -e -in text1.txt -out enccfb.txt -k 00112233445566778889aabcccddeff -iv 1122334455667788
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
jigar@jigar:~/Desktop/Labsetup/Files/task4.2$ openssl enc -aes-128-ofb -e -in text1.txt -out encofb.txt -k 00112233445566778889aabcccddeff -iv 1122334455667788
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
jigar@jigar:~/Desktop/Labsetup/Files/task4.2$ ls -l
total 20
-rw-rw-r-- 1 jigar jigar 32 Jun 20 11:56 enccbc.txt
-rw-rw-r-- 1 jigar jigar 21 Jun 20 11:59 enccfb.txt
-rw-rw-r-- 1 jigar jigar 32 Jun 20 11:58 encecb.txt
-rw-rw-r-- 1 jigar jigar 21 Jun 20 11:59 encofb.txt
-rw-rw-r-- 1 jigar jigar 5 Jun 18 17:10 text1.txt
jigar@jigar:~/Desktop/Labsetup/Files/task4.2$
```

```
jigar@jigar: ~/Desktop/Labsetup/Files/task4$ hexdump -C enccfb.txt
00000000  53 61 6c 74 65 64 5f 5f  75 c9 12 16 4d b4 7c 40  |Salted_u...M.|@|
00000010  ef 5d 9f df 17                               [...]...
00000015
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C enccbc.txt
00000000  53 61 6c 74 65 64 5f 5f  5f b2 10 19 7c 91 77 da  |Salted...|.w.|
00000010  cf f5 29 a8 ed e9 15 44  94 0f 47 0c 82 14 fc 99  |...)....D..G....|
00000020
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C encecb.txt
00000000  53 61 6c 74 65 64 5f 5f  dc 9f 36 20 c6 d9 31 35  |Salted...6 ..15|
00000010  4d 45 fe 32 e6 1f ae 9e  e1 d1 49 35 dc 1c f2 82  |ME.2.....I5....|
00000020
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C encofb.txt
00000000  53 61 6c 74 65 64 5f 5f  6b d4 23 88 4e 89 22 7b  |Salted_k.#.N."{|
00000010  26 57 47 99 e5                               |&WG..|
00000015
jigar@jigar:~/Desktop/Labsetup/Files/task4$
```

– Padding is used for filling the space that are remaining in the block after filling it with plaintext. After observing HAX format of text file we can say that **ECB and CBC modes need padding. CFB and OFB do not need padding** because the encryption and decryption functions are identical, and it doesn't require the plaintext data to be padded (i.e. the ciphertext is the same length as the plaintext).

(2)

- created 3 text files with 5 bytes, 10 bytes, 16 bytes

```
jigar@jigar:~/Desktop/Labsetup/Files/task4$ echo -n "1234567890" > text2.txt
jigar@jigar:~/Desktop/Labsetup/Files/task4$ echo -n "1234567890123456" > text3.txt

-rw-rw-r-- 1 jigar jigar 5 Jun 18 17:10 text1.txt
-rw-rw-r-- 1 jigar jigar 10 Jun 20 12:11 text2.txt
-rw-rw-r-- 1 jigar jigar 16 Jun 20 12:12 text3.txt
```

- After encryption of all files with -aes-128-cbc

```
-rw-rw-r-- 1 jigar jigar 32 Jun 20 12:40 encbct1.txt
-rw-rw-r-- 1 jigar jigar 32 Jun 20 12:14 encbct2.txt
-rw-rw-r-- 1 jigar jigar 48 Jun 20 12:14 encbct3.txt
```

– Size of files after encryption is

Size of plaintext	Size of ciphertext
5 bytes	32 bytes
10 bytes	32 bytes
16 bytes	48 bytes

- Decryption with -nopad command

```
jigar@jigar:~/Desktop/Labsetup/Files/task4$ openssl enc -aria-128-cbc -d -in encbct1.txt -out decbct1.txt -nopad -k 00112233445566778889aabcccddeeff -iv 0102030405060708
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
jigar@jigar:~/Desktop/Labsetup/Files/task4$ openssl enc -aria-128-cbc -d -in encbct2.txt -out decbct2.txt -nopad -k 00112233445566778889aabcccddeeff -iv 0102030405060708
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
jigar@jigar:~/Desktop/Labsetup/Files/task4$ openssl enc -aria-128-cbc -d -in encbct3.txt -out decbct3.txt -nopad -k 00112233445566778889aabcccddeeff -iv 0102030405060708
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
```

```
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C deccbct1.txt  
00000000  31 32 33 34 35 0b |12345.....|  
00000010  
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C deccbct2.txt  
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 |1234567890.....|  
00000010  
jigar@jigar:~/Desktop/Labsetup/Files/task4$ hexdump -C deccbct3.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  
jigar@jigar:~/Desktop/Labsetup/Files/task4$ █
```

- In 5 bytes plaintext, 11 byte of padding was added and padding was 0B.
 - In 10 bytes plaintext, 6 bytes of padding was added and padding was 06.
 - In 16 bytes plaintext, 16 bytes of padding was added and padding was 10.
-

Task 5: Error Propagation – Corrupted Cipher Text

The file contains a series of "1234567890".(100 times)

Our Initial assumptions were:-

1. CBC: We expected that 1 entire block of 8 bytes would be affected and a single byte in next block will be affected
2. CFB: We expected only a single bit error.
3. OFB: We expected only a single bit error just like CFB.
4. ECB: We expect an entire block of error, making the corrupted byte's block completely useless.

After conducting the practical we gain that:



Files :

- decb - deciphered ECB
- dcfb - deciphered EFB
- dofbc - deciphered OFB
- dcdbc - deciphered CBC

- Final Observations drawn:

1. CBC: Entire block of corrupted bytes
2. CFB: Entire block of corrupted bytes with 8bytes of usable data in between
3. OFB: One corrupted byte
4. ECB: Entire block of corrupted bytes

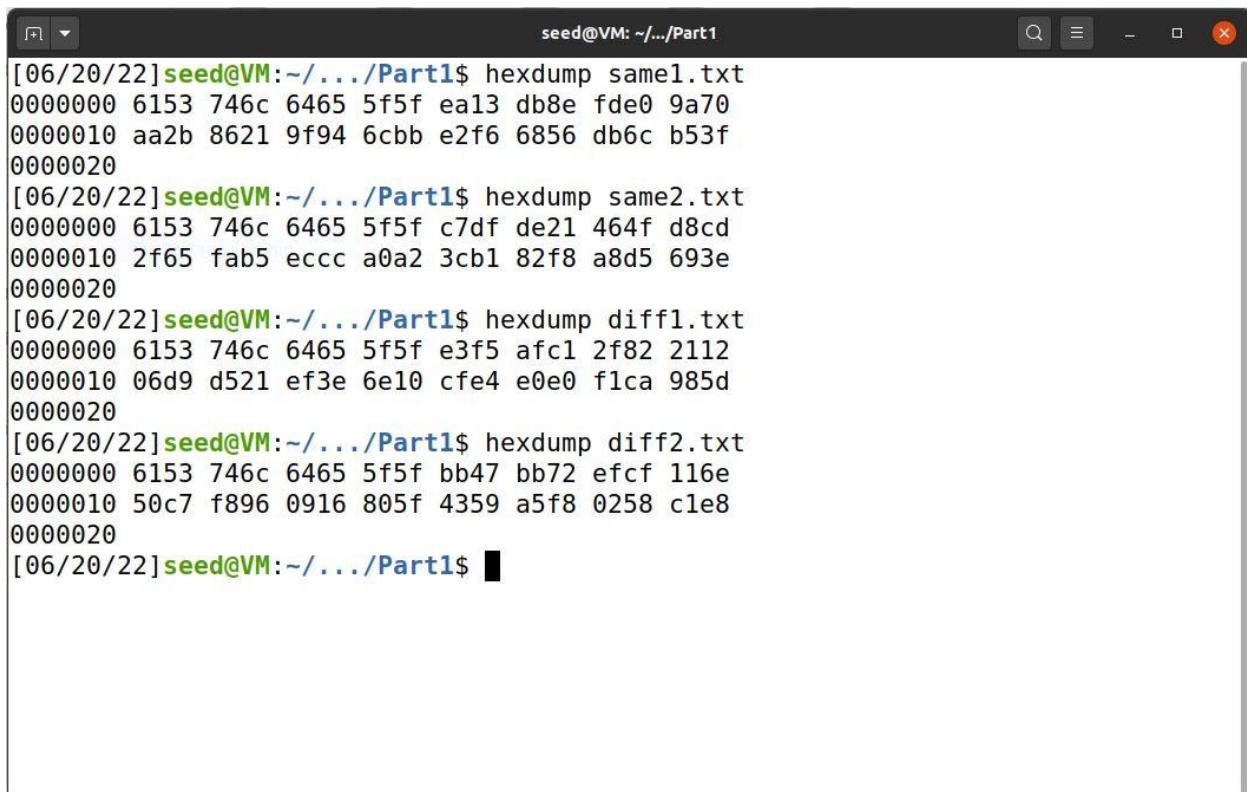
Justification:

1. CBC : the way CBC deciphers data, by XORing previous ciphertext and block deciphering using key and block, it in turn makes the block completely useless with random bits.
 2. CFB : the way CFB deciphers data, the affected bit in our case 55th will be corrupted as well as the next block of data because of XORing and key block deciphering.
 3. OFB : the least affected deciphering method, as 1byte stream decipher method provides lowest loss of usable data.
 4. ECB : As one by one byte of data is deciphered using individual keys, we get an entire block of corrupted data.
-

Task 6:Initial Vector (IV) and Common Mistakes

– 6.1 IV Experiment:

The image we obtain after using same IV's and different IV's:



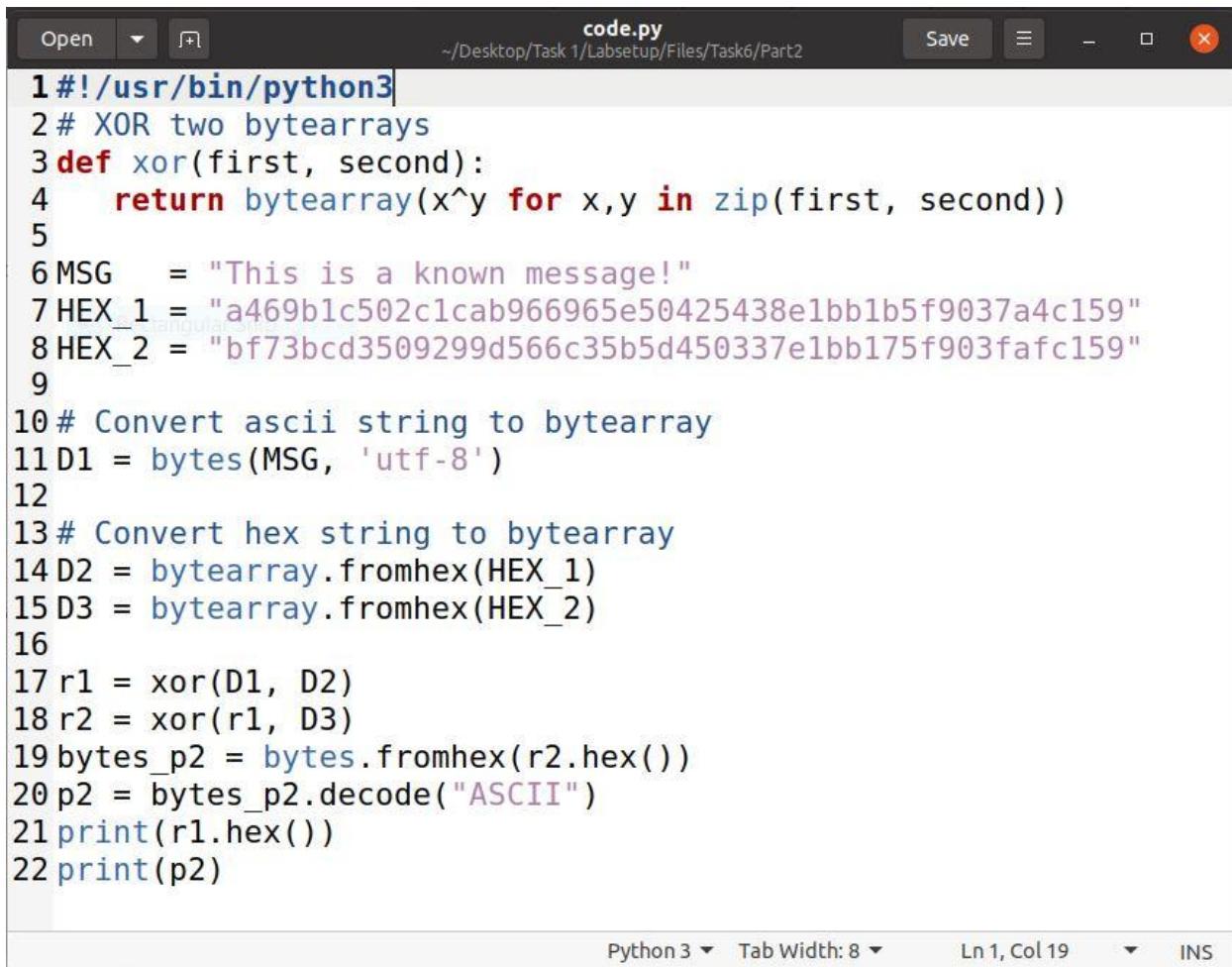
```
seed@VM: ~/.../Part1$ hexdump same1.txt
00000000 6153 746c 6465 5f5f ea13 db8e fde0 9a70
00000010 aa2b 8621 9f94 6ccb e2f6 6856 db6c b53f
00000020
[06/20/22]seed@VM:~/.../Part1$ hexdump same2.txt
00000000 6153 746c 6465 5f5f c7df de21 464f d8cd
00000010 2f65 fab5 eccc a0a2 3cb1 82f8 a8d5 693e
00000020
[06/20/22]seed@VM:~/.../Part1$ hexdump diff1.txt
00000000 6153 746c 6465 5f5f e3f5 afc1 2f82 2112
00000010 06d9 d521 ef3e 6e10 cfe4 e0e0 f1ca 985d
00000020
[06/20/22]seed@VM:~/.../Part1$ hexdump diff2.txt
00000000 6153 746c 6465 5f5f bb47 bb72 efcf 116e
00000010 50c7 f896 0916 805f 4359 a5f8 0258 c1e8
00000020
[06/20/22]seed@VM:~/.../Part1$
```

Justification :

If the same key and IV are used, then you get the same stream, so you have the conditions of the (in)famous two-times pad. **Without knowing the key, you can still compute the XOR of any two messages, which is often enough to crack them, by exploiting their internal structure.**

– 6.2 Common Mistake: Use the Same IV:

After readjusting the code to gain our intended output:



The screenshot shows a code editor window titled "code.py" with the following Python script:

```
1 #!/usr/bin/python3
2 # XOR two bytearrays
3 def xor(first, second):
4     return bytearray(x^y for x,y in zip(first, second))
5
6 MSG    = "This is a known message!"
7 HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
8 HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"
9
10 # Convert ascii string to bytearray
11 D1 = bytes(MSG, 'utf-8')
12
13 # Convert hex string to bytearray
14 D2 = bytearray.fromhex(HEX_1)
15 D3 = bytearray.fromhex(HEX_2)
16
17 r1 = xor(D1, D2)
18 r2 = xor(r1, D3)
19 bytes_p2 = bytes.fromhex(r2.hex())
20 p2 = bytes_p2.decode("ASCII")
21 print(r1.hex())
22 print(p2)
```

The status bar at the bottom indicates "Python 3" and "Tab Width: 8".

The obtained output:

```
[06/19/22] seed@VM:~/.../Task6$ python3 code.py
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
Order: Launch a missile!
```

The hidden plaintext2 is : **Lanch a missile!**

Part2:

OFB is an improved version of CFB. The basic difference among the two is that in OFB, the output of block cipher encryption is fed as a feedback to calculate the next block. The first block is calculated using an initialization vector (normally a string of random digits and alphabets). **If the IV is reused, the whole key stream will be reproduced again.**

– 6.3 Common Mistake: Use a Predictable IV :

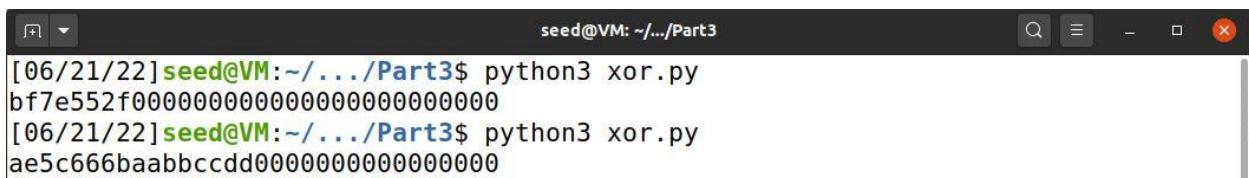
Our initial nc 10.9.0.80 3000 :

```
[06/21/22]seed@VM:~/.../encryption_oracle$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertext: acad1c3e77e0e64589b89c95d387ac75
The IV used      : 5af0b23d5c9d5edf09fa20b3fbbeeff74

Next IV          : b6cca05e5c9d5edf09fa20b3fbbeeff74
Your plaintext   : 11223344aabbcdd
Your ciphertext  : 596892191b41f50d654ddc8c3a737ae4

Next IV          : 09b2f5715c9d5edf09fa20b3fbbeeff74
Your plaintext   : ae5c666baabbccdd
```

With the provided IV's we applied Xor's and got the right answer 2nd Your Plaintext.



A terminal window titled 'seed@VM: ~/.../Part3'. It contains two lines of command-line output. The first line shows the result of running 'python3 xor.py' with input 'bf7e552f00000000000000000000000000000000'. The second line shows the result of running 'python3 xor.py' with input 'ae5c666baabbccdd00000000000000000000000000000000'. The terminal has a dark theme with light-colored text.

```
[06/21/22]seed@VM:~/.../Part3$ python3 xor.py
bf7e552f00000000000000000000000000000000
[06/21/22]seed@VM:~/.../Part3$ python3 xor.py
ae5c666baabbccdd00000000000000000000000000000000
```

Excluding all the padding, we were on right track.

To find out Bob's initial plaintext we carried on with same method to give us:

```
[06/21/22]seed@VM:~/.../Part3$ python3 xor.py
b559616300000000000000000000000000000000
[06/21/22]seed@VM:~/.../Part3$ python3 xor.py
a253126300000000000000000000000000000000
```

1. XOR result for "Yes", which does not match with Bob's first ciphertext
2. XOR result for "No", which does match with Bob's first ciphertext, excluding all the padding and salt values.

Task 7:Programming using the Crypto Library

We didn't manage to find the key using below provided code

```
seed@VM: ~/.../Task7
b'7ded330eae386eb83e4b7e1a7471690c517fa18c74db0d12ceb2498c7b8c6283'
b'zoo#000000000000'
b'147b91defbc78bf252cd9d5c53e594a990876fcf95fcda71d15d8e6d5'
b'zoology#00000000'
b'743d9ab2bd15adda7d375bb44eea70010246696c38615517d38af4dff4df343'
b'zoom#000000000000'
b'ffa5e86d8f6b6caf37c6a31180095d297ba902cefea58aa62dcfad328b8f71'
b'Zorn#000000000000'
b'f7843cdfe3246ef613710f989f06aac9d7e111886a7623600b7a46e614a40075'
b'Zoroaster#000000'
b'99de3b11856f9763a34bf563f1303d25841924a16ea9e5563c4b6f212a822f97'
b'Zoroastrian#0000'
b'a2665cd121a663ab0724a8d1cf18667c8db1aa42d8d77141f2e2b573d32e1ebf'
b'ounds#0000000000'
b'3c24cc6140585205f41fb20ee136aace058449af1231621b8370d0f963f52db'
b"z's#000000000000"
b'6734116f9c2087cd55d36f3ebaf61406cd4c667d93c8ab8179cd0ab26b67d6cc'
b'zucchini#00000000'
b'b206587631f3c90d4d52df44257f00f0a1cecf85a22f0870bb7a8a2fe84f7824'
b'Zurich#0000000000'
b'70cc1dc19f0cb7f4c9a006496e57a5f3acc323d47e7b6d0bcde819fe5c3936ac'
b'zygote#0000000000'
b'8ef5a5879854579bb6c937efbd8f6d2390a0f9f1941147f7fc805ca48e5d26c'
[06/21/22] seed@VM:~/.../Task7$
```

Code of python file:

```
from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from binascii import hexlify, unhexlify

fpx = open("/home/seed/Desktop/words.txt", "r")

for fp in fpx:

    if(len(fp) <= 16):

        x = 16 - len(fp)

        a = ""

        for _ in range(0,x):

            a += "0"

        key_str = fp+a

        key = str.encode(key_str)
```

```
key_str = key_str.replace("\n","0")
print(key)

IV_hex = "aabbccddeeff00998877665544332211"
IV_hex_bytes = bytes.fromhex(IV_hex)

text_str = "This is a secret message."
text = str.encode(text_str)

mode = AES.MODE_CBC

encryptor = AES.new(key, mode, IV = IV_hex_bytes)

ciphertext = encryptor.encrypt(pad(text,AES.block_size))

match_hex =
b'764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2'

guess_hex = hexlify(ciphertext)

print(guess_hex)

if(guess_hex == match_hex):

    print("Key found :" + fp)

    decryptor = AES.new(key, mode, IV=IV_from_hex_bytes)

    plain = unpad(decryptor.decrypt(ciphertext),AES.block_size)

    print(plain)

    break
```