APPLIED CRYPTOGRAPHY

Lab 4 : Secret Key Encryption

Name: Siri N Shetty SRN: PES2UG22CS556

(use SEEDVM)

Task 1: Frequency Analysis

Step 1 : Generation of key from python code.

Step 2 : Generation of plaintextcustom.txt from below article.txt

There are two main types of cryptosystems: symmetric and asymmetric. In symmetric systems, the only ones known until the 1970s, the same secret key encrypts and decrypts a message. Data manipulation in symmetric systems is significantly faster than in asymmetric systems. Asymmetric systems use a "public key" to encrypt a message and a related "private key" to decrypt it. The advantage of asymmetric systems is that the public key can be freely published, allowing parties to establish secure communication without having a shared secret key. In practice, asymmetric systems are used to first exchange a secret key, and then secure communication proceeds via a more efficient symmetric system using that key.

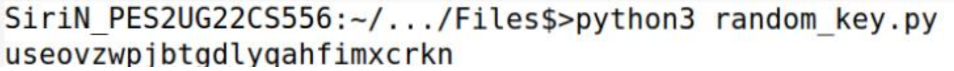
Step 3 : Generate ciphertextcuston.txt from plaintextcustom.txt

Step 4 : Frequency Analysis on given ciphertext.txt (run freq.py)

Step 5 : Decrypted text from given ciphertext.txt

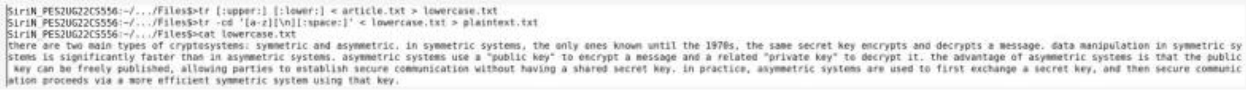
Expected Deliverables -

i) Output Screenshot (should have SRN) for step 1



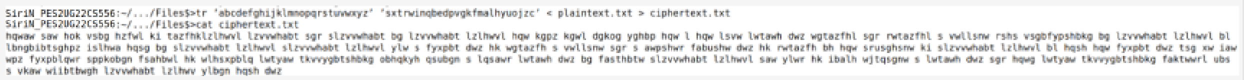
```
SiriN_PES2UG22CS556:~/.../Files$>python3 random_key.py
useovzwpjbtgdlyqahfimxcrkn
```

ii) Output Screenshot (should have SRN) for step 2



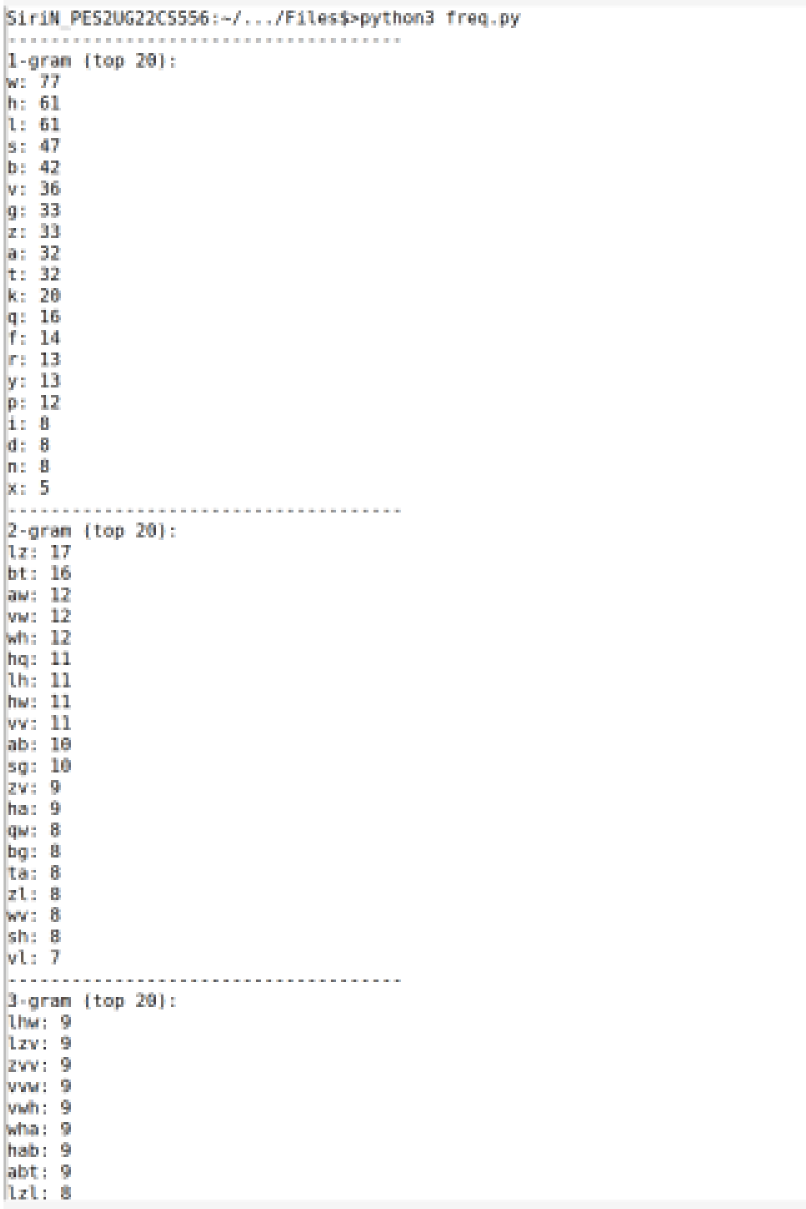
```
SiriN_PES2UG22CS556:~/.../Files$>tr [:upper:] [:lower:] < article.txt > lowercase.txt
SiriN_PES2UG22CS556:~/.../Files$>tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
SiriN_PES2UG22CS556:~/.../Files$>cat lowercase.txt
there are two main types of cryptosystems: symmetric and asymmetric. in symmetric systems, the only ones known until the 1970s, the same secret key encrypts and decrypts a message. data manipulation in symmetric sy
stems is significantly faster than in asymmetric systems. asymmetric systems use a "public key" to encrypt a message and a related "private key" to decrypt it. the advantage of asymmetric systems is that the public
 key can be freely published, allowing parties to establish secure communication without having a shared secret key. in practice, asymmetric systems are used to first exchange a secret key, and then secure communic
ation proceeds via a more efficient symmetric system using that key.
```

iii) Output Screenshot (should have SRN) for step 3



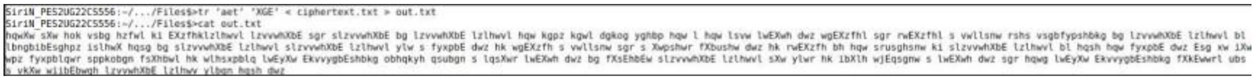
```
SiriN_PES2UG22CS556:~/.../Files$>tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' < plaintext.txt > ciphertext.txt
SiriN_PES2UG22CS556:~/.../Files$>cat ciphertext.txt
hqwaw saw hok vsbg hzfwl ki tazfhklzlhwvl lzvvwhabt sgr slzvvwhabt bg lzvvwhabt lzlhwvl hqw kgpz kgwl dgkog yghbp hqw l hqw lsvw lwtawh dwz wgtazfhl sgr rwtazfhl s vwllsnw rshs vsgbfypshbkg bg lzvvwhabt lzlhwvl bl
lbngbibtsghpz islhwa hqsg bg slzvvwhabt lzlhwvl slzvvwhabt lzlhwvl ylw s fyxpbt dwz hk wgtazfh s vwllsnw sgr s awpshwr fabushw dwz hk rwtazfh bh hqw srusghsnw ki slzvvwhabt lzlhwvl bl hqsh hqw fyxpbt dwz tsg xw iaw
wpz fyxpblqwr sppkobgn fsahbwl hk wlhsxpblq lwtyaw tkvvygbtshbkg obhqkyh qsubgn s lqsawr lwtawh dwz bg fasthbtw slzvvwhabt lzlhwvl saw ylwr hk ibalh wjtqsgnw s lwtawh dwz sgr hqwg lwtyaw tkvvygbtshbkg faktwwrl ubs
s vkaw wiibtbwgh lzvvwhabt lzlhwv ylbgn hqsh dwz
```

iv) Output Screenshot (should have SRN) for step 4



```
SiriN_PES2UG22CS556:~/.../Files$>python3 freq.py
-------------------------------------
1-gram (top 20):
w: 77
h: 61
l: 61
s: 47
b: 42
v: 36
g: 33
z: 33
a: 32
t: 32
k: 20
q: 16
f: 14
r: 13
y: 13
p: 12
i: 8
d: 8
n: 8
x: 5
-------------------------------------
2-gram (top 20):
lz: 17
bt: 16
aw: 12
vw: 12
wh: 12
hq: 11
lh: 11
hw: 11
vv: 11
ab: 10
sg: 10
zv: 9
ha: 9
qw: 8
bg: 8
ta: 8
zl: 8
wv: 8
sh: 8
vl: 7
-------------------------------------
3-gram (top 20):
lhw: 9
lzv: 9
zvv: 9
vvw: 9
vwh: 9
wha: 9
hab: 9
abt: 9
lzl: 8
```

v) Complete Decrypted text Output (should have SRN) for step 5



```
SiriN_PES2UG22CS556:~/.../Files$>tr 'aet' 'XGE' < ciphertext.txt > out.txt
SiriN_PES2UG22CS556:~/.../Files$>cat out.txt
hqwXw sXw hok vsbg hzfwl ki EXzfhklzlhwvl lzvvwhXbE sgr slzvvwhXbE bg lzvvwhXbE lzlhwvl hqw kgpz kgwl dgkog yghbp hqw l hqw lsvw lwEXwh dwz wgEXzfhl sgr rwEXzfhl s vwllsnw rshs vsgbfypshbkg bg lzvvwhXbE lzlhwvl bl
lbngbibEsghpz islhwX hqsg bg slzvvwhXbE lzlhwvl slzvvwhXbE lzlhwvl ylw s fyxpbE dwz hk wgEXzfh s vwllsnw sgr s Xwpshwr fXbushw dwz hk rwEXzfh bh hqw srusghsnw ki slzvvwhXbE lzlhwvl bl hqsh hqw fyxpbE dwz Esg xw iXw
wpz fyxpblqwr sppkobgn fsXhbwl hk wlhsxpblq lwEyXw EkvvygbEshbkg obhqkyh qsubgn s lqsXwr lwEXwh dwz bg fXsEhbEw slzvvwhXbE lzlhwvl sXw ylwr hk ibXlh wjEqsgnw s lwEXwh dwz sgr hqwg lwEyXw EkvvygbEshbkg fXkEwwrl ubs
s vkXw wiibEbwgh lzvvwhXbE lzlhwv ylbgn hqsh dwz
```

Task 2: Encryption using Different Ciphers and Modes

Step 1 : Various ciphers supported using " openssl enc -list "

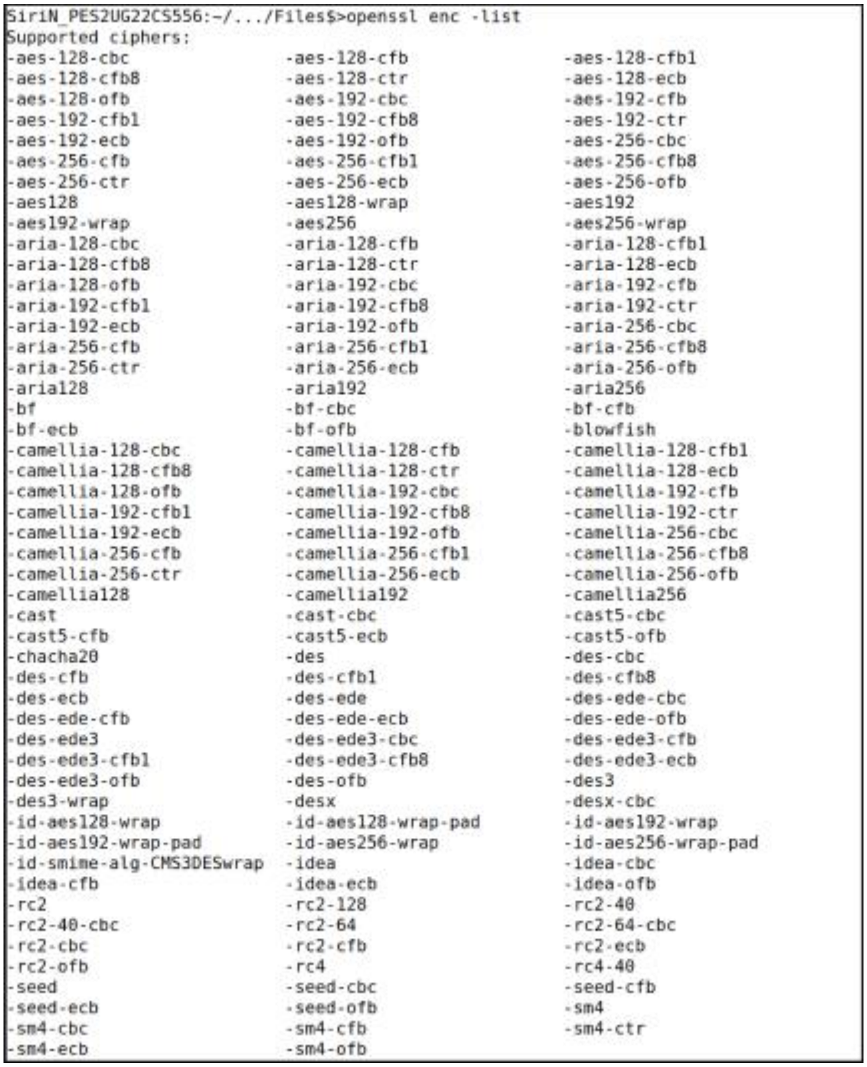
Step 2 : Encrypt (-e flag) and Decrypt (-d flag) for ciphertype1

Step 3 : Encrypt (-e flag) and Decrypt (-d flag) for ciphertype2

Step 4 : Encrypt (-e flag) and Decrypt (-d flag) for ciphertype3

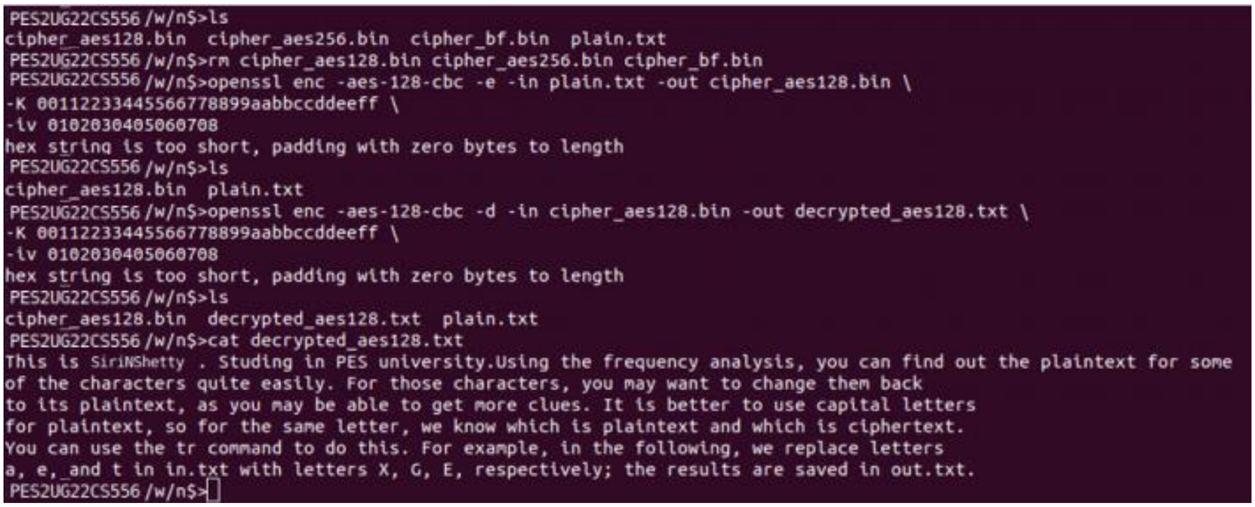
Expected Deliverables -

i) Output Screenshot (should have SRN) for step 1



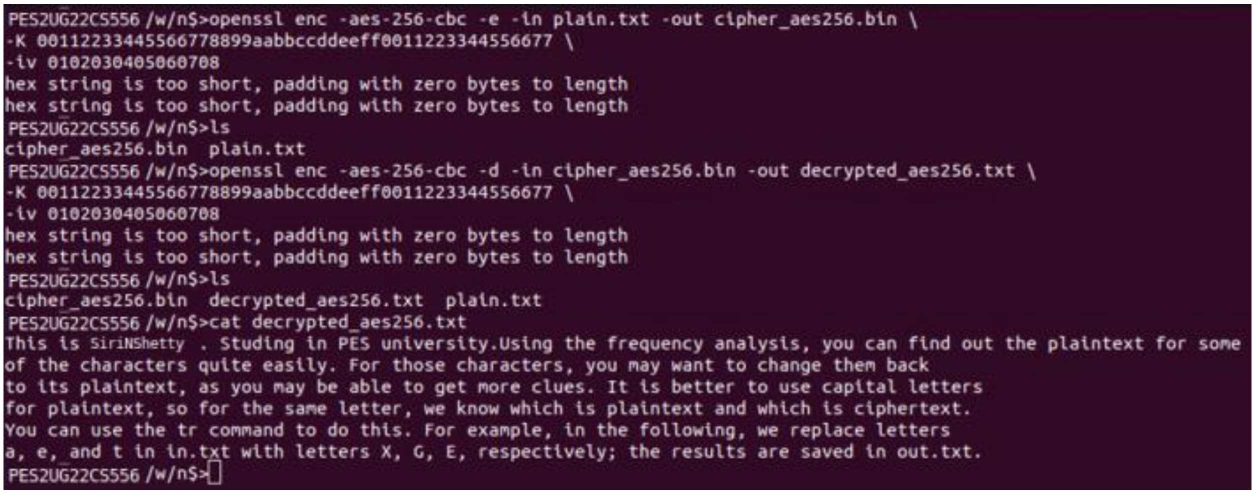
```
SiriN_PES2UG22CS556:~/.../Files$>openssl enc -list
Supported ciphers:
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8              -aes-192-ctr
-aes-192-ecb               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes128-wrap               -aes192
-aes192-wrap               -aes256                    -aes256-wrap
-aria-128-cbc              -aria-128-cfb              -aria-128-cfb1
-aria-128-cfb8             -aria-128-ctr              -aria-128-ecb
-aria-128-ofb              -aria-192-cbc              -aria-192-cfb
-aria-192-cfb1             -aria-192-cfb8             -aria-192-ctr
-aria-192-ecb              -aria-192-ofb              -aria-256-cbc
-aria-256-cfb              -aria-256-cfb1             -aria-256-cfb8
-aria-256-ctr              -aria-256-ecb              -aria-256-ofb
-aria128                   -aria192                   -aria256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
-camellia-128-cbc          -camellia-128-cfb          -camellia-128-cfb1
-camellia-128-cfb8         -camellia-128-ctr          -camellia-128-ecb
-camellia-128-ofb          -camellia-192-cbc          -camellia-192-cfb
-camellia-192-cfb1         -camellia-192-cfb8         -camellia-192-ctr
-camellia-192-ecb          -camellia-192-ofb          -camellia-256-cbc
-camellia-256-cfb          -camellia-256-cfb1         -camellia-256-cfb8
-camellia-256-ctr          -camellia-256-ecb          -camellia-256-ofb
-camellia128               -camellia192               -camellia256
-cast                      -cast-cbc                  -cast5-cbc
-cast5-cfb                 -cast5-ecb                 -cast5-ofb
-chacha20                  -des                       -des-cbc
-des-cfb                   -des-cfb1                  -des-cfb8
-des-ecb                   -des-ede                   -des-ede-cbc
-des-ede-cfb               -des-ede-ecb               -des-ede-ofb
-des-ede3                  -des-ede3-cbc              -des-ede3-cfb
-des-ede3-cfb1             -des-ede3-cfb8             -des-ede3-ecb
-des-ede3-ofb              -des-ofb                   -des3
-des3-wrap                 -desx                      -desx-cbc
-id-aes128-wrap            -id-aes128-wrap-pad        -id-aes192-wrap
-id-aes192-wrap-pad        -id-aes256-wrap            -id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap  -idea                      -idea-cbc
-idea-cfb                  -idea-ecb                  -idea-ofb
-rc2                       -rc2-128                   -rc2-40
-rc2-40-cbc                -rc2-64                    -rc2-64-cbc
-rc2-cbc                   -rc2-cfb                   -rc2-ecb
-rc2-ofb                   -rc4                       -rc4-40
-seed                      -seed-cbc                  -seed-cfb
-seed-ecb                  -seed-ofb                  -sm4
-sm4-cbc                   -sm4-cfb                   -sm4-ctr
-sm4-ecb                   -sm4-ofb
```

ii) Output Screenshot (should have SRN) for step 2 with Brief description of ciphertype-1



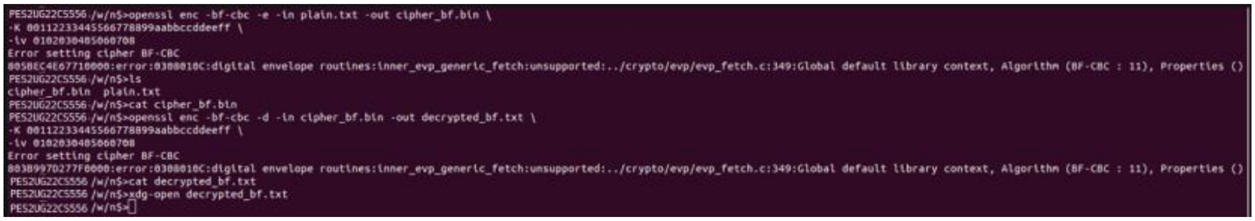
```
PES2UG22CS556 /w/n$>ls
cipher_aes128.bin  cipher_aes256.bin  cipher_bf.bin  plain.txt
PES2UG22CS556 /w/n$>rm cipher_aes128.bin cipher_aes256.bin cipher_bf.bin
PES2UG22CS556 /w/n$>openssl enc -aes-128-cbc -e -in plain.txt -out cipher_aes128.bin \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
cipher_aes128.bin  plain.txt
PES2UG22CS556 /w/n$>openssl enc -aes-128-cbc -d -in cipher_aes128.bin -out decrypted_aes128.txt \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
cipher_aes128.bin  decrypted_aes128.txt  plain.txt
PES2UG22CS556 /w/n$>cat decrypted_aes128.txt
This is SiriNShetty . Studing in PES university.Using the frequency analysis, you can find out the plaintext for some
of the characters quite easily. For those characters, you may want to change them back
to its plaintext, as you may be able to get more clues. It is better to use capital letters
for plaintext, so for the same letter, we know which is plaintext and which is ciphertext.
You can use the tr command to do this. For example, in the following, we replace letters
a, e, and t in in.txt with letters X, G, E, respectively; the results are saved in out.txt.
PES2UG22CS556 /w/n$>
```

iii) Output Screenshot (should have SRN) for step 3 with Brief description of ciphertype-2



```
PES2UG22CS556 /w/n$>openssl enc -aes-256-cbc -e -in plain.txt -out cipher_aes256.bin \
-K 00112233445566778899aabbccddeeff0011223344556677 \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
cipher_aes256.bin  plain.txt
PES2UG22CS556 /w/n$>openssl enc -aes-256-cbc -d -in cipher_aes256.bin -out decrypted_aes256.txt \
-K 00112233445566778899aabbccddeeff0011223344556677 \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
cipher_aes256.bin  decrypted_aes256.txt  plain.txt
PES2UG22CS556 /w/n$>cat decrypted_aes256.txt
This is SiriNShetty . Studing in PES university.Using the frequency analysis, you can find out the plaintext for some
of the characters quite easily. For those characters, you may want to change them back
to its plaintext, as you may be able to get more clues. It is better to use capital letters
for plaintext, so for the same letter, we know which is plaintext and which is ciphertext.
You can use the tr command to do this. For example, in the following, we replace letters
a, e, and t in in.txt with letters X, G, E, respectively; the results are saved in out.txt.
PES2UG22CS556 /w/n$>
```

iv) Output Screenshot (should have SRN) for step 4 with Brief description of ciphertype-3



```
PES2UG22CS556 /w/n$>openssl enc -bf-cbc -e -in plain.txt -out cipher_bf.bin \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
Error setting cipher BF-CBC
8058EC4E67710000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:../crypto/evp/evp_fetch.c:349:Global default library context, Algorithm (BF-CBC : 11), Properties ()
PES2UG22CS556 /w/n$>ls
cipher_bf.bin  plain.txt
PES2UG22CS556 /w/n$>cat cipher_bf.bin
PES2UG22CS556 /w/n$>openssl enc -bf-cbc -d -in cipher_bf.bin -out decrypted_bf.txt \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
Error setting cipher BF-CBC
8038997D277F0000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:../crypto/evp/evp_fetch.c:349:Global default library context, Algorithm (BF-CBC : 11), Properties ()
PES2UG22CS556 /w/n$>cat decrypted_bf.txt
PES2UG22CS556 /w/n$>xdg-open decrypted_bf.txt
PES2UG22CS556 /w/n$>
```

Task 3 : Encryption Mode – ECB vs. CBC

Step 1 : Encrypt picture using ecb

Step 2 : Encrypt picture using cbc

Step 3 : Attach headers to both the encrypted pictures

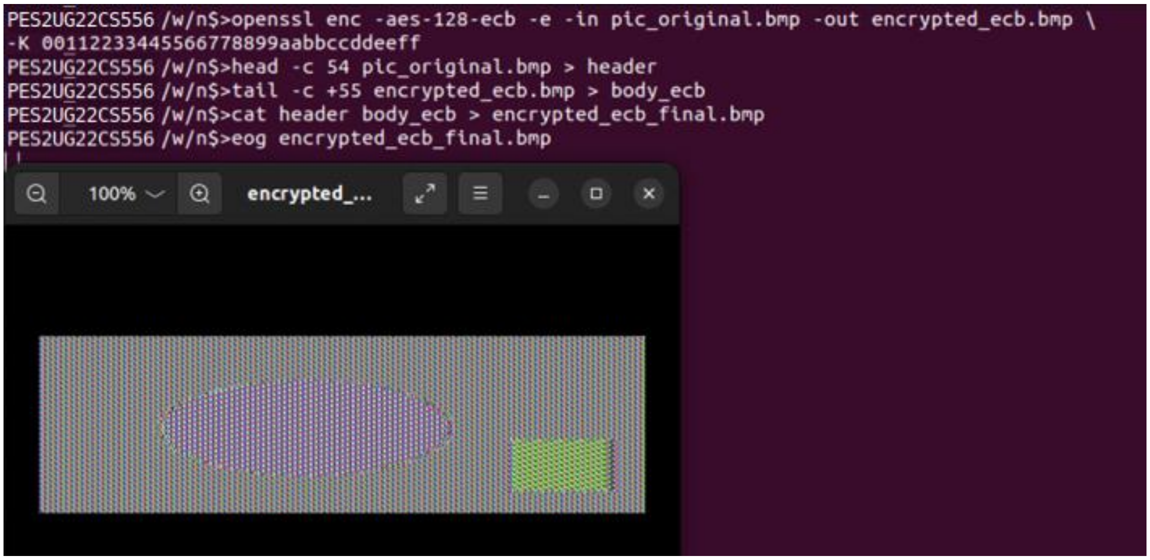
(p1 refers to original_pic ; p2 refers to encrypted_pic ;
new refers to header_attached_encrypted_pic)

Step 4 : Display the encrypted picture using a picture viewing program

Step 5 : Repeat for custom selected picture.

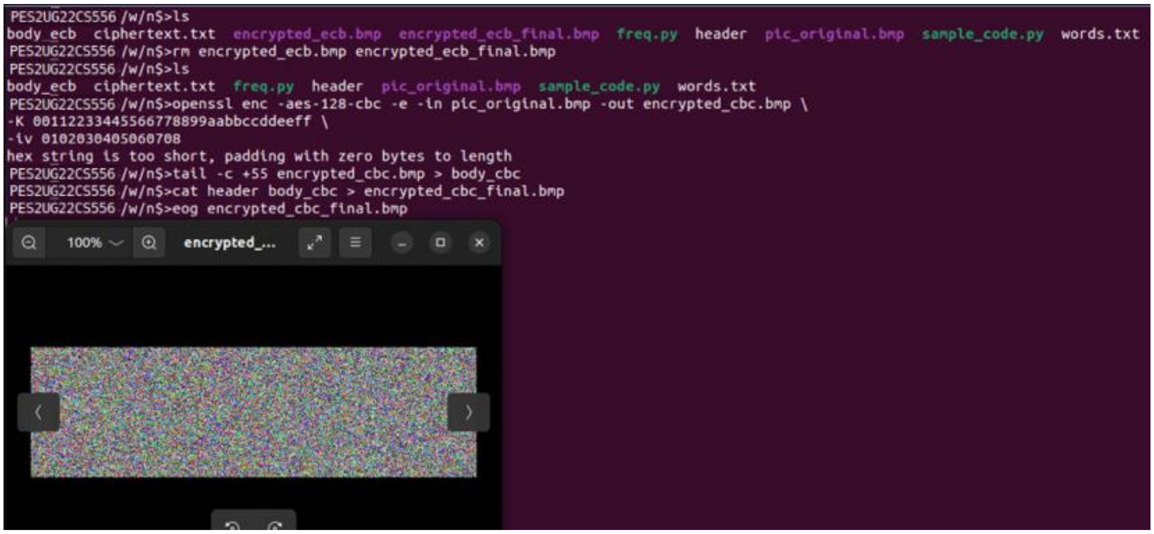
Expected Deliverables -

i) Output Screenshot (should have SRN) for step 1



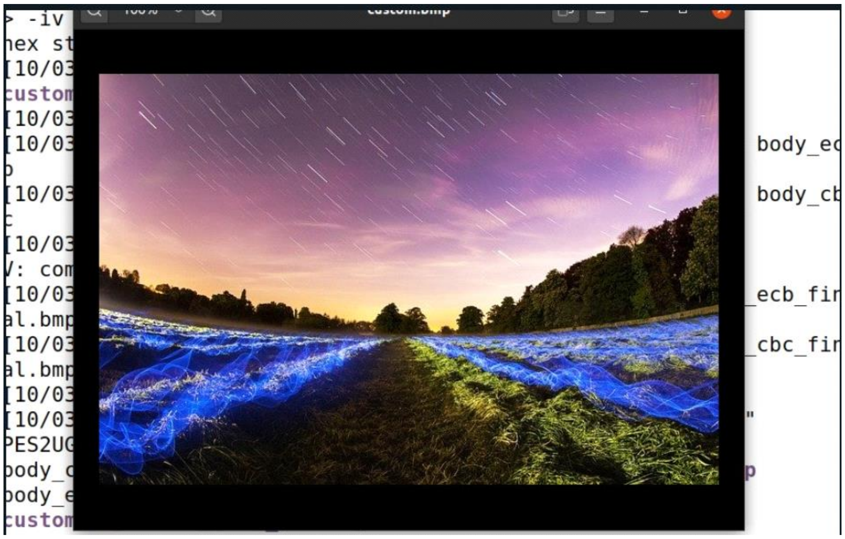
```
PES2UG22CS556 /w/n$>openssl enc -aes-128-ecb -e -in pic_original.bmp -out encrypted_ecb.bmp \
-K 00112233445566778899aabbccddeeff
PES2UG22CS556 /w/n$>head -c 54 pic_original.bmp > header
PES2UG22CS556 /w/n$>tail -c +55 encrypted_ecb.bmp > body_ecb
PES2UG22CS556 /w/n$>cat header body_ecb > encrypted_ecb_final.bmp
PES2UG22CS556 /w/n$>eog encrypted_ecb_final.bmp
```

ii) Output Screenshot (should have SRN) for step 2



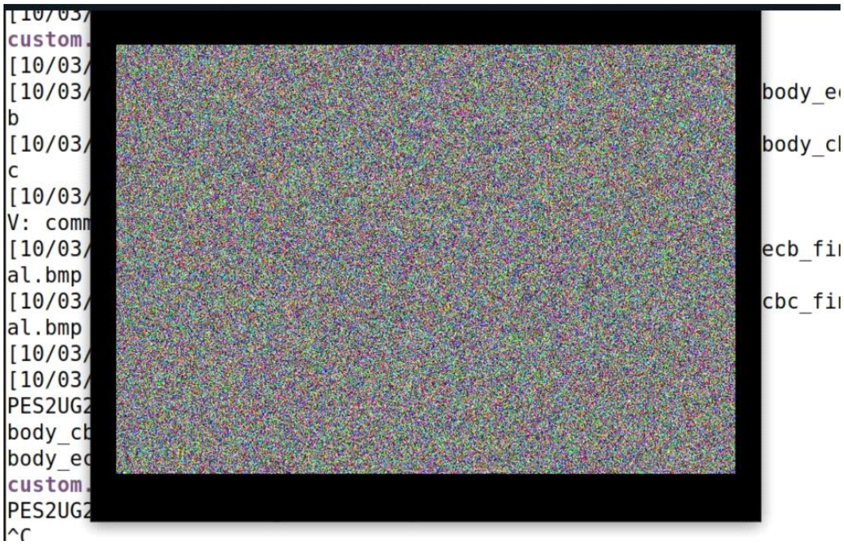
```
PES2UG22CS556 /w/n$>ls
body_ecb  ciphertext.txt  encrypted_ecb.bmp  encrypted_ecb_final.bmp  freq.py  header  pic_original.bmp  sample_code.py  words.txt
PES2UG22CS556 /w/n$>rm encrypted_ecb.bmp encrypted_ecb_final.bmp
PES2UG22CS556 /w/n$>ls
body_ecb  ciphertext.txt  freq.py  header  pic_original.bmp  sample_code.py  words.txt
PES2UG22CS556 /w/n$>openssl enc -aes-128-cbc -e -in pic_original.bmp -out encrypted_cbc.bmp \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>tail -c +55 encrypted_cbc.bmp > body_cbc
PES2UG22CS556 /w/n$>cat header body_cbc > encrypted_cbc_final.bmp
PES2UG22CS556 /w/n$>eog encrypted_cbc_final.bmp
```

v) Output Screenshots for step 5 (include deliverables i to iv)



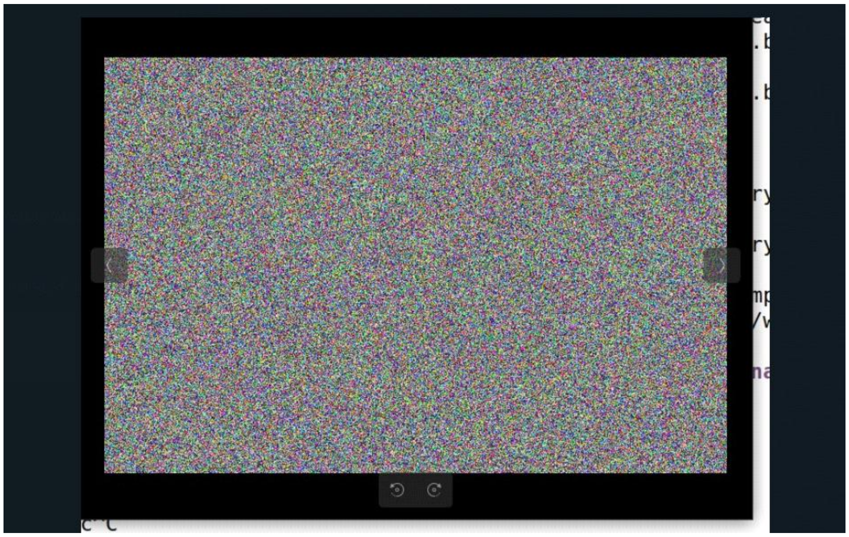
```
> -iv
hex st
[10/03
custom
[10/03
[10/03
b
[10/03
c
[10/03
V: com
[10/03
al.bmp
[10/03
al.bmp
[10/03
[10/03
PES2UG
body_c
body_e
custom
```

```
body_ec
body_cb
_ecb_fin
_cbc_fin
"
p
```



```
[10/03/
custom.
[10/03/
[10/03/
b
[10/03/
c
[10/03/
V: comm
[10/03/
al.bmp
[10/03/
al.bmp
[10/03/
[10/03/
PES2UG2
body_cb
body_ec
custom.
PES2UG2
^C
```

```
body_e
body_cl
ecb_fi
cbc_fi
```



```
.b
.b
ry
ry
mp
/w
na
^C
```

vi) Image encrypted by which mode of encryption has greater similarity to the original? Give reason. Explain your observations.

In ECB mode, identical plaintext blocks are transformed into identical ciphertext blocks, thereby exposing patterns within the data. ECB exhibits a closer resemblance to the original data. This absence of diffusion renders ECB inappropriate for images, as repeated patterns are maintained in the encrypted output.

Task 4 : Error Propagation – Corrupted Cipher Text

Step 1 : Create task4.txt, atleast 1000 bytes long.
Run " wc -c task4.txt " to show size.

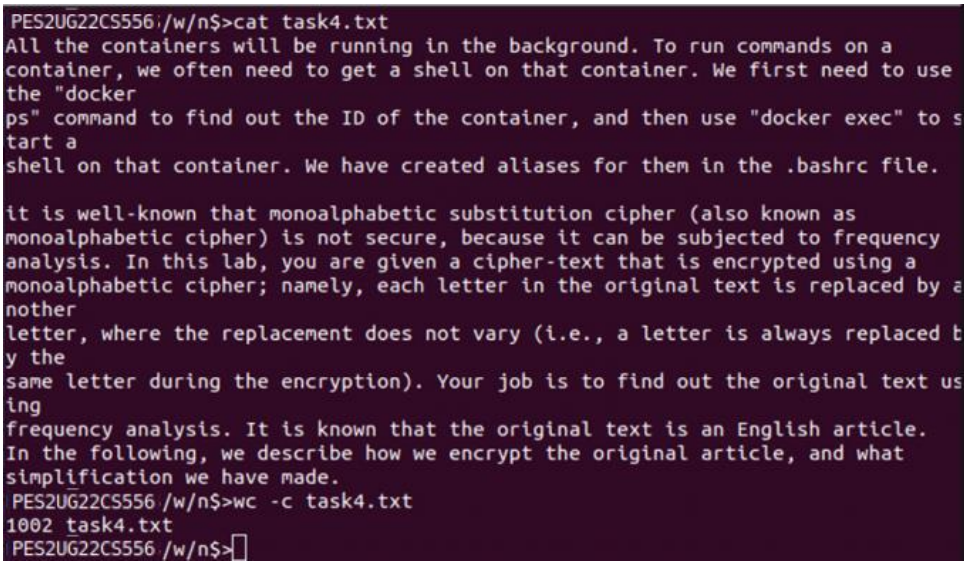
Step 2 : Encrypt the file using the AES-128 (ECB, CBC, CFB, OFB)

Step 3 : Corrupt the 55th bit in the generated .bin using " bless <file.bin> "

Step 4 : Decrypt all 4 modes files using openssl command.

Expected Deliverables -

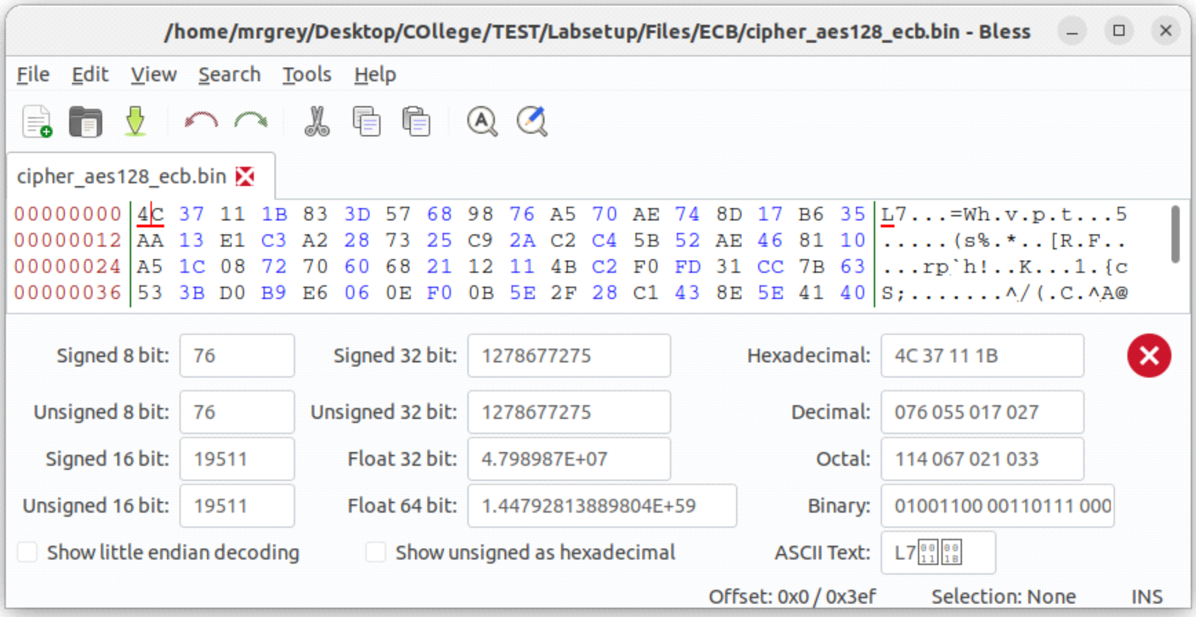
i) Output Screenshot (should have SRN) for step 1



```
PES2UG22CS556 /w/n$>cat task4.txt
All the containers will be running in the background. To run commands on a
container, we often need to get a shell on that container. We first need to use
the "docker
ps" command to find out the ID of the container, and then use "docker exec" to s
tart a
shell on that container. We have created aliases for them in the .bashrc file.

it is well-known that monoalphabetic substitution cipher (also known as
monoalphabetic cipher) is not secure, because it can be subjected to frequency
analysis. In this lab, you are given a cipher-text that is encrypted using a
monoalphabetic cipher; namely, each letter in the original text is replaced by a
nother
letter, where the replacement does not vary (i.e., a letter is always replaced b
y the
same letter during the encryption). Your job is to find out the original text us
ing
frequency analysis. It is known that the original text is an English article.
In the following, we describe how we encrypt the original article, and what
simplification we have made.
PES2UG22CS556 /w/n$>wc -c task4.txt
1002 task4.txt
PES2UG22CS556 /w/n$>
```

ii) Output Screenshot (should have SRN) for step 2,3,4 – ecb

Before Corrupting 55th Bit


/home/mrgrey/Desktop/COllege/TEST/Labsetup/Files/ECB/cipher_aes128_ecb.bin - Bless
File Edit View Search Tools Help
cipher_aes128_ecb.bin

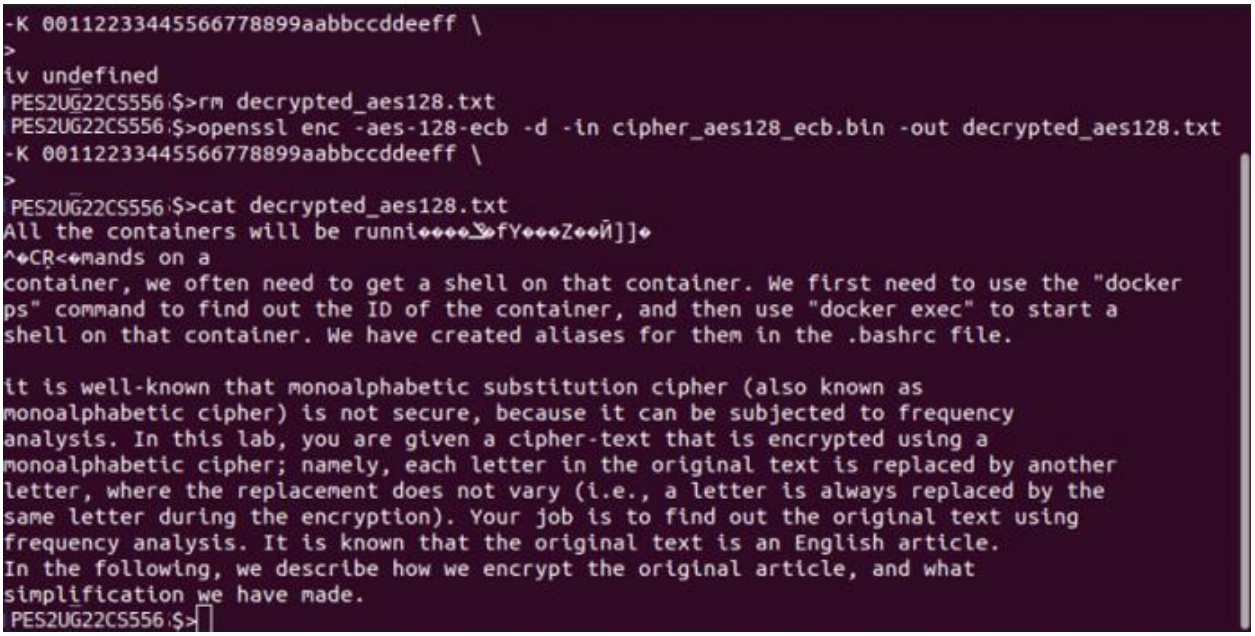
```
00000000 4C 37 11 1B 83 3D 57 68 98 76 A5 70 AE 74 8D 17 B6 35 L7...=Wh.v.p.t...5
00000012 AA 13 E1 C3 A2 28 73 25 C9 2A C2 C4 5B 52 AE 46 81 10 .....(s%.*..[R.F..
00000024 A5 1C 08 72 70 60 68 21 12 11 4B C2 F0 FD 31 CC 7B 63 ...rp`h!..K...1.{c
00000036 53 3B D0 B9 E6 06 0E F0 0B 5E 2F C1 43 8E 5E 41 40   S;.......^/(.C.^A@
```

Signed 8 bit:	76	Signed 32 bit:	1278677275	Hexadecimal:	4C 37 11 1B
Unsigned 8 bit:	76	Unsigned 32 bit:	1278677275	Decimal:	076 055 017 027
Signed 16 bit:	19511	Float 32 bit:	4.798987E+07	Octal:	114 067 021 033
Unsigned 16 bit:	19511	Float 64 bit:	1.44792813889804E+59	Binary:	01001100 00110111 000

□ Show little endian decoding □ Show unsigned as hexadecimal ASCII Text: L7

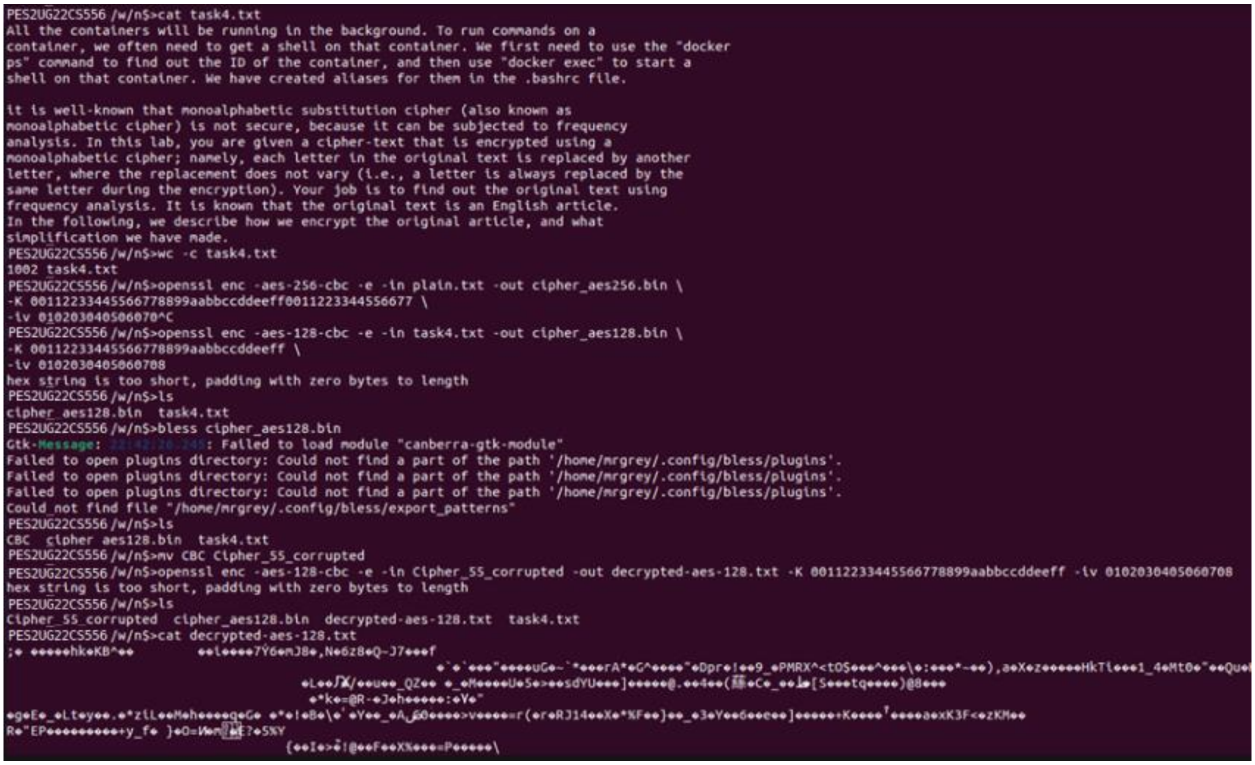
Offset: 0x0 / 0x3ef Selection: None INS

After Corrupting && Decrypting the bit we get



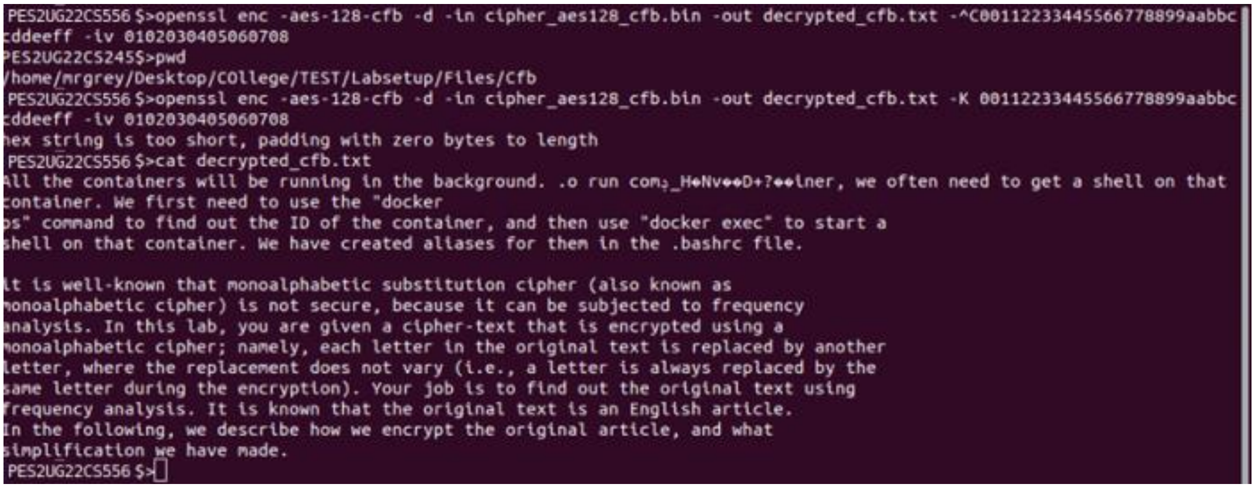
```
-K 00112233445566778899aabbccddeeff \
>
iv undefined
PES2UG22CS556 $>rm decrypted_aes128.txt
PES2UG22CS556 $>openssl enc -aes-128-ecb -d -in cipher_aes128_ecb.bin -out decrypted_aes128.txt
-K 00112233445566778899aabbccddeeff \
>
PES2UG22CS556 $>cat decrypted_aes128.txt
All the containers will be runni����Ѷ�fY���Z��Й]]�
^�CR<�mands on a
container, we often need to get a shell on that container. We first need to use the "docker
ps" command to find out the ID of the container, and then use "docker exec" to start a
shell on that container. We have created aliases for them in the .bashrc file.

it is well-known that monoalphabetic substitution cipher (also known as
monoalphabetic cipher) is not secure, because it can be subjected to frequency
analysis. In this lab, you are given a cipher-text that is encrypted using a
monoalphabetic cipher; namely, each letter in the original text is replaced by another
letter, where the replacement does not vary (i.e., a letter is always replaced by the
same letter during the encryption). Your job is to find out the original text using
frequency analysis. It is known that the original text is an English article.
In the following, we describe how we encrypt the original article, and what
simplification we have made.
PES2UG22CS556 $>
```

iii) Output Screenshot (should have SRN) for step 2,3,4 – cbc



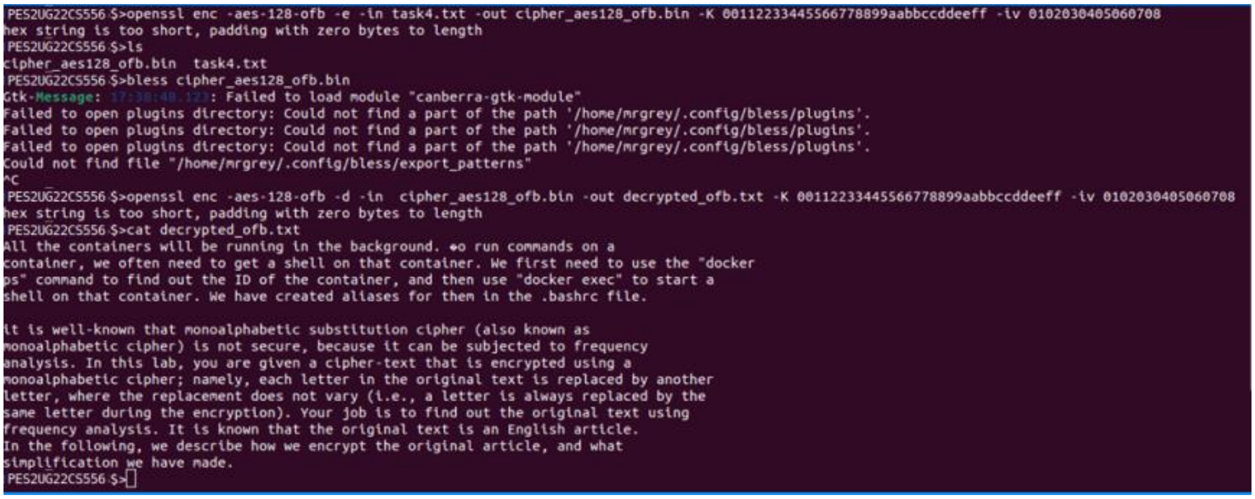
```
PES2UG22CS556 /w/n$>cat task4.txt
All the containers will be running in the background. To run commands on a
container, we often need to get a shell on that container. We first need to use the "docker
ps" command to find out the ID of the container, and then use "docker exec" to start a
shell on that container. We have created aliases for them in the .bashrc file.

it is well-known that monoalphabetic substitution cipher (also known as
monoalphabetic cipher) is not secure, because it can be subjected to frequency
analysis. In this lab, you are given a cipher-text that is encrypted using a
monoalphabetic cipher; namely, each letter in the original text is replaced by another
letter, where the replacement does not vary (i.e., a letter is always replaced by the
same letter during the encryption). Your job is to find out the original text using
frequency analysis. It is known that the original text is an English article.
In the following, we describe how we encrypt the original article, and what
simplification we have made.
PES2UG22CS556 /w/n$>wc -c task4.txt
1002 task4.txt
PES2UG22CS556 /w/n$>openssl enc -aes-256-cbc -e -in plain.txt -out cipher_aes256.bin \
-K 00112233445566778899aabbccddeeff00112233445566677 \
-iv 0102030405060708^C
PES2UG22CS556 /w/n$>openssl enc -aes-128-cbc -e -in task4.txt -out cipher_aes128.bin \
-K 00112233445566778899aabbccddeeff \
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
cipher_aes128.bin  task4.txt
PES2UG22CS556 /w/n$>bless cipher_aes128.bin
Gtk-Message: 22:42:26.245: Failed to load module "canberra-gtk-module"
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Could not find file "/home/mrgrey/.config/bless/export_patterns"
PES2UG22CS556 /w/n$>ls
CBC  cipher_aes128.bin  task4.txt
PES2UG22CS556 /w/n$>mv CBC Cipher_55_corrupted
PES2UG22CS556 /w/n$>openssl enc -aes-128-cbc -e -in Cipher_55_corrupted -out decrypted-aes-128.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 /w/n$>ls
Cipher_55_corrupted  cipher_aes128.bin  decrypted-aes-128.txt  task4.txt
PES2UG22CS556 /w/n$>cat decrypted-aes-128.txt
;◆ ◆◆◆◆◆hk◆KB^◆◆        ◆◆l◆◆◆◆7Y6◆mJ8◆,N◆6z8◆Q-J7◆◆◆f
                                                            ◆`◆`◆◆◆"◆◆◆◆uG◆~`*◆◆◆rA*◆G^◆◆◆◆"◆Dpr◆!◆◆9_◆PMRX^<tO$◆◆◆^◆◆◆\◆:◆◆◆*~◆◆),a◆X◆z◆◆◆◆◆HkTl◆◆◆1_4◆Mt0◆"◆◆Qu◆
                                       ◆L◆◆/X/◆◆u◆◆_QZ◆◆ ◆_◆M◆◆◆◆U◆5◆>◆◆sdYU◆◆◆]◆◆◆◆◆@.◆◆4◆◆(◆◆C◆_◆◆◆[S◆◆◆tq◆◆◆◆)@8◆◆◆
                                        ◆*k◆=@R-◆J◆h◆◆◆◆◆:◆V◆"
◆g◆E◆_◆Lt◆y◆◆.◆*zlL◆◆M◆h◆◆◆◆q◆G◆ ◆*◆!◆B◆\◆'◆Y◆◆_◆A_60◆◆◆◆>v◆◆◆◆=r(◆r◆RJ14◆◆X◆*%F◆◆)◆◆_◆3◆Y◆◆6◆◆◆◆◆]◆◆◆◆◆+K◆◆◆◆'◆◆◆◆a◆xK3F<◆zKM◆◆
R◆"EP◆◆◆◆◆◆◆◆◆◆+y_f◆ }◆O=W◆m◆◆E?◆5%Y
                                    {◆◆I◆>◆!@◆◆F◆◆X%◆◆◆=P◆◆◆◆◆\
```

iv) Output Screenshot (should have SRN) for step 2,3,4 – cfb



```
PES2UG22CS556 $>openssl enc -aes-128-cfb -d -in cipher_aes128_cfb.bin -out decrypted_cfb.txt -^C00112233445566778899aabbc
cddeeff -iv 0102030405060708
PES2UG22CS245$>pwd
/home/mrgrey/Desktop/COllege/TEST/Labsetup/Files/Cfb
PES2UG22CS556 $>openssl enc -aes-128-cfb -d -in cipher_aes128_cfb.bin -out decrypted_cfb.txt -K 00112233445566778899aabbc
cddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 $>cat decrypted_cfb.txt
All the containers will be running in the background. .o run com◆_H◆Nv◆◆D+?◆◆iner, we often need to get a shell on that
container. We first need to use the "docker
ps" command to find out the ID of the container, and then use "docker exec" to start a
shell on that container. We have created aliases for them in the .bashrc file.

it is well-known that monoalphabetic substitution cipher (also known as
monoalphabetic cipher) is not secure, because it can be subjected to frequency
analysis. In this lab, you are given a cipher-text that is encrypted using a
monoalphabetic cipher; namely, each letter in the original text is replaced by another
letter, where the replacement does not vary (i.e., a letter is always replaced by the
same letter during the encryption). Your job is to find out the original text using
frequency analysis. It is known that the original text is an English article.
In the following, we describe how we encrypt the original article, and what
simplification we have made.
PES2UG22CS556 $>
```

v) Output Screenshot (should have SRN) for step 2,3,4 - ofb



```
PES2UG22CS556 $>openssl enc -aes-128-ofb -e -in task4.txt -out cipher_aes128_ofb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 $>ls
cipher_aes128_ofb.bin  task4.txt
PES2UG22CS556 $>bless cipher_aes128_ofb.bin
Gtk-Message: 17:38:48.123: Failed to load module "canberra-gtk-module"
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/mrgrey/.config/bless/plugins'.
Could not find file "/home/mrgrey/.config/bless/export_patterns"
^C
PES2UG22CS556 $>openssl enc -aes-128-ofb -d -in  cipher_aes128_ofb.bin -out decrypted_ofb.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
PES2UG22CS556 $>cat decrypted_ofb.txt
All the containers will be running in the background. ◆o run commands on a
container, we often need to get a shell on that container. We first need to use the "docker
ps" command to find out the ID of the container, and then use "docker exec" to start a
shell on that container. We have created aliases for them in the .bashrc file.

it is well-known that monoalphabetic substitution cipher (also known as
monoalphabetic cipher) is not secure, because it can be subjected to frequency
analysis. In this lab, you are given a cipher-text that is encrypted using a
monoalphabetic cipher; namely, each letter in the original text is replaced by another
letter, where the replacement does not vary (i.e., a letter is always replaced by the
same letter during the encryption). Your job is to find out the original text using
frequency analysis. It is known that the original text is an English article.
In the following, we describe how we encrypt the original article, and what
simplification we have made.
PES2UG22CS556 $>
```

vi) In which mode is the information least recoverable? Give reason. Explain your observations.

In CBC and CFB the information is least recoverable because successive cipher blocks are dependent on the previous cipher blocks which maximally scramble the output on subtle changes of the cipher text.

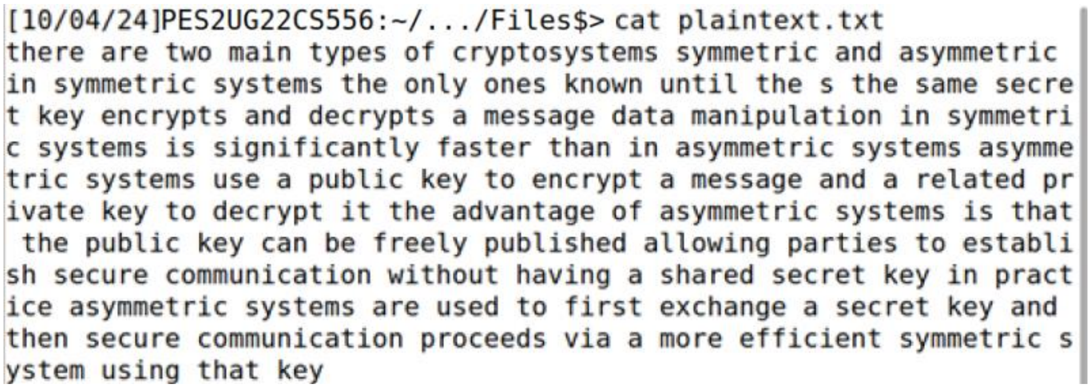
Task 5 : Initial Vector (IV) and Common Mistakes

Task 5.1. IV Experiment

Step 1 : Create a plaintext.txt file
Step 2 : Encrypt using a key and IV.
Step 3 : Encrypt using same key and same IV as in step 2.
Step 4 : Encrypt using same key and different IV as in step 2.

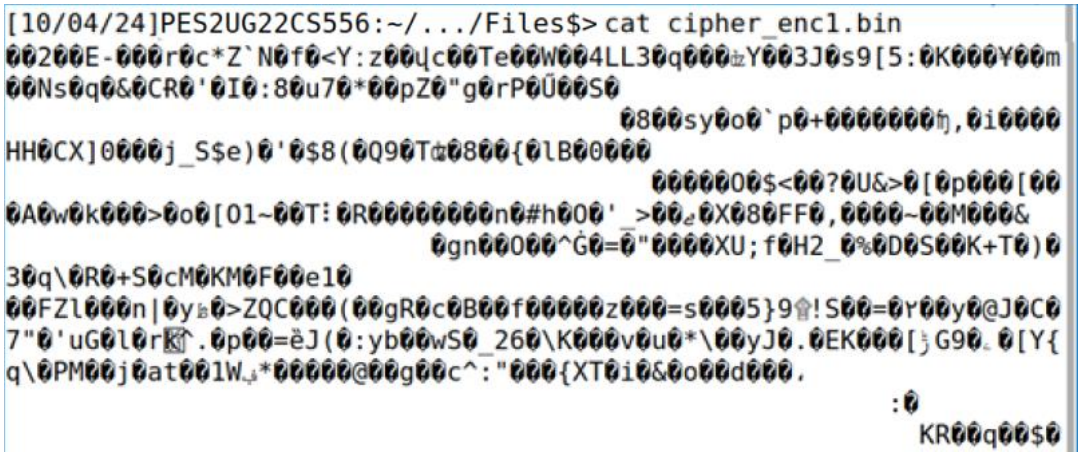
Expected Deliverables -

i) Output Screenshot (should have SRN) for step 1



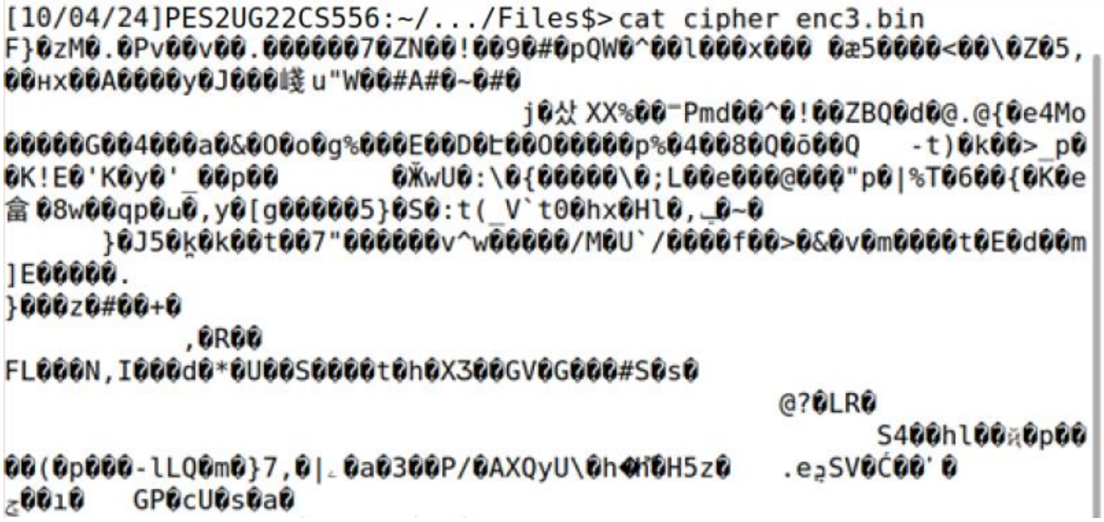
```
[10/04/24]PES2UG22CS556:~/.../Files$> cat plaintext.txt
there are two main types of cryptosystems symmetric and asymmetric
in symmetric systems the only ones known until the s the same secre
t key encrypts and decrypts a message data manipulation in symmetri
c systems is significantly faster than in asymmetric systems asymme
tric systems use a public key to encrypt a message and a related pr
ivate key to decrypt it the advantage of asymmetric systems is that
 the public key can be freely published allowing parties to establi
sh secure communication without having a shared secret key in pract
ice asymmetric systems are used to first exchange a secret key and
then secure communication proceeds via a more efficient symmetric s
ystem using that key
```

ii) Output Screenshot (should have SRN) for step 2



```
[10/04/24]PES2UG22CS556:~/.../Files$> cat cipher_enc1.bin
��2��E-���r�c*Z`N�f�<Y:z��ɥc��Te��W��4LL3�q���㉼Y��3J�s9[5:�K���Y��m
��Ns�q�&�CR�'�I�:8�u7�*��pZ�"g�rP�Ű��S�
                                                  �8��sy�o�`p�+�������ђ,�i����
HH�CX]0���j_S$e)�'�$8(�Q9�T㉾�8��{�lB�0���
                                                    ������0�$<��?�U&>�[�p���[��
�A�w�k���>�o�[01~��T⁝�R���������n�#h�O�'_>��ⅇ�X�8�FF�,����~��M���&
                                        �gn��O��^Ġ�=�"����XU;f�H2_�%�D�S��K+T�)�
3�q\�R�+S�cM�KM�F��e1�
��FZl���n|�yㅌ�>ZQC���(��gR�c�B��f�����z���=s���5}9㊤!S��=�Ɏ��y�@J�C�
7"�'uG�l�r㎅.�p��=ëJ(�:yb��wS�_26�\K���v�u�*\��yJ�.�EK���[ꠗG9�꜀�[Y{
q\�PM��j�at��1Wꜗ*�����@��g��c^:"���{XT�i�&�o��d���.
                                                                      :�
                                                                        KR��q��$�
```

iii) Output Screenshot (should have SRN) for step 3



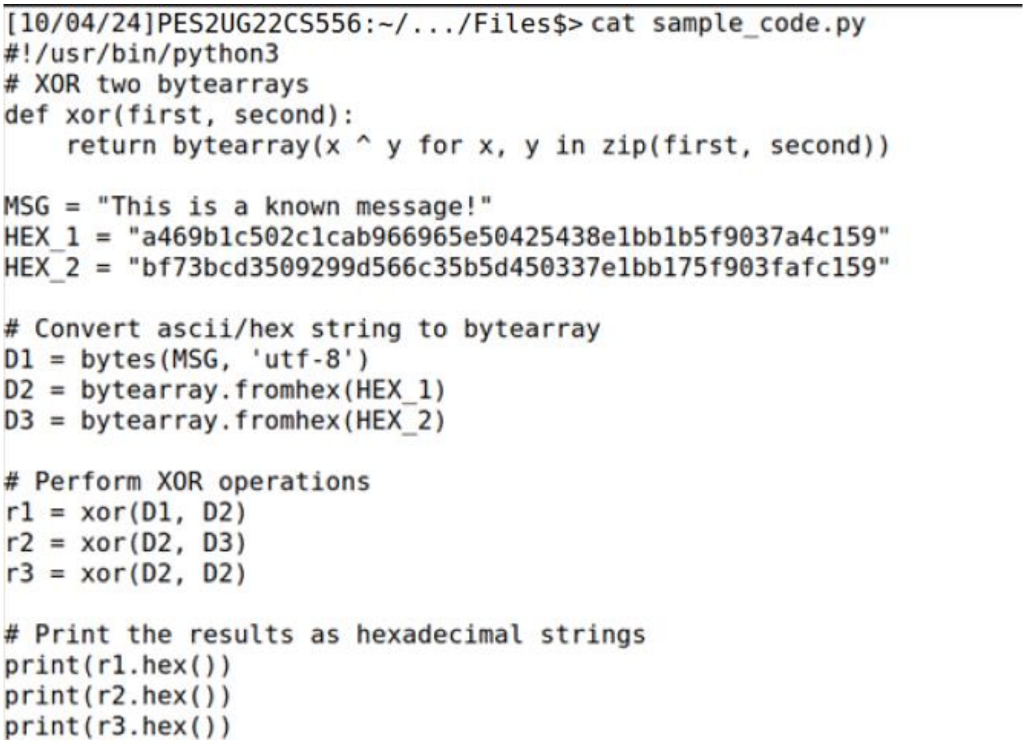
```
[10/04/24]PES2UG22CS556:~/.../Files$>cat cipher enc3.bin
F}�zM�.�Pv��v��.������7�ZN��!��9�#�pQW�^��l���x��� �æ5����<��\�Z�5,
��ʜx��A����y�J���峻u"W��#A#�~�#�
                                             j�䨯 XX%��⁻Pmd��^�!��ZBQ�d�@.@{�e4Mo
�����G��4���a�&�O�o�g%���E��D�Ŀ��O������p%�4��8�Q�ō��Q    -t)�k��>_p�
�K!E�'K�y�'_��p��        �ӁwU�:\�{�����\�;L��e���@���"p�|%T�6��{�K�e
畣�8w��qp�ㅟ�,y�[g�����5}�S�:t(_V`t0�hx�Hl�,_�~�
      }�J5�ḵ�k��t��7"������v^w�����/M�U`/����f��>�&�v�m����t�E�d��m
]E�����.
}���z�#��+�
                        ,�R��
FL���N,I���d�*�U��S����t�h�X3��GV�G���#S�s�
                                                            @?�LR�
                                                                  S4��hl��ꞔ�p��
��(�p���-lLQ�m�}7,�|ﮮ�a�3��P/�AXQyU\�h�ň�H5z�    .eꞔSV�Ć��'�
ج��1�    GP�cU�s�a�
```

Task 5.2. Common Mistake: Use the Same IV

Step 1 : Modify the sample_code.py file msg,hex1,hex2 with given values of plaintext, ciphertext1, ciphertext2.
Step 2 : Run sample_code.py

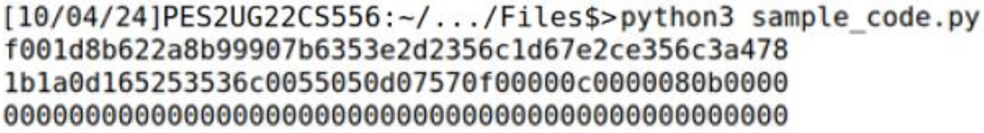
Expected Deliverables -

i) Output Screenshot for step 1



```
[10/04/24]PES2UG22CS556:~/.../Files$> cat sample_code.py
#!/usr/bin/python3
# XOR two bytearrays
def xor(first, second):
    return bytearray(x ^ y for x, y in zip(first, second))

MSG = "This is a known message!"
HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

# Convert ascii/hex string to bytearray
D1 = bytes(MSG, 'utf-8')
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

# Perform XOR operations
r1 = xor(D1, D2)
r2 = xor(D2, D3)
r3 = xor(D2, D2)

# Print the results as hexadecimal strings
print(r1.hex())
print(r2.hex())
print(r3.hex())
```

ii) Output Screenshot (should have SRN) for step 2



```
[10/04/24]PES2UG22CS556:~/.../Files$>python3 sample_code.py
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
1b1a0d165253536c0055050d07570f00000c0000080b0000
000000000000000000000000000000000000000000000000
```

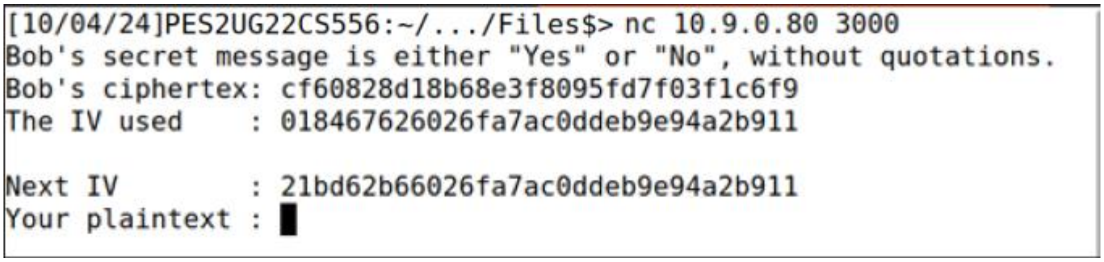
iii) Explain your observations.

Since the IV's are same across both the cipher texts, we can xor the given plaintext with the known cipher text in-order to obtain the IV and then xor the IV with the ciphertext2 to obtain the message

Task 5.3. Common Mistake: Use a Predictable IV

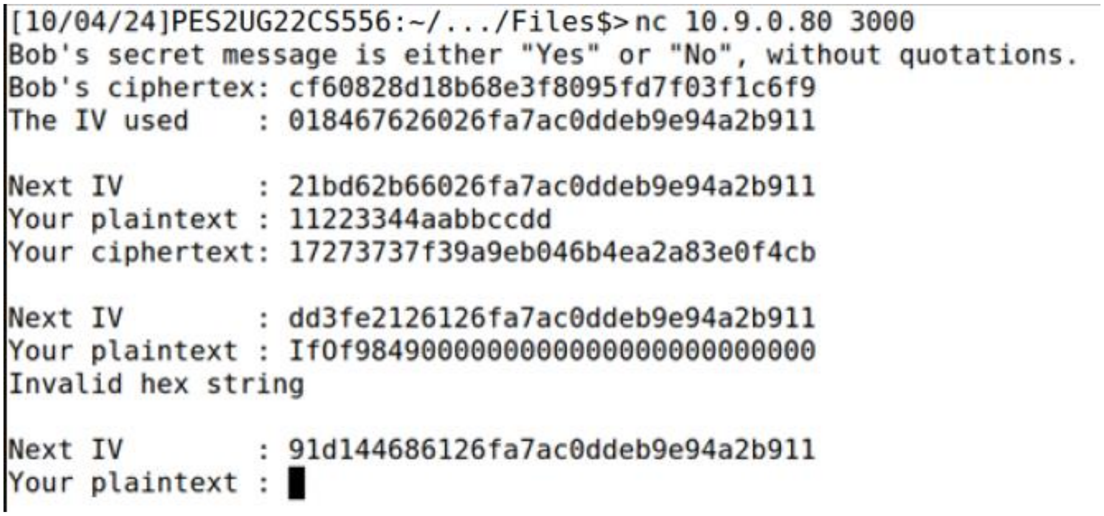
Step 1 : Run the nc cmd
Step 2 : Find the plaintext to enter (hint use sample_code.py as reference to construct script for this ; (plaintext is generated from the message where message can be 'yes'/'no'))
(to know plaintext entered is right match , check that the ciphertext generated from it should be same as bob's ciphertext given.

Expected Deliverables -
i) Output Screenshot (should have SRN) for step 1



```
[10/04/24]PES2UG22CS556:~/.../Files$> nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: cf60828d18b68e3f8095fd7f03f1c6f9
The IV used     : 018467626026fa7ac0ddeb9e94a2b911

Next IV         : 21bd62b66026fa7ac0ddeb9e94a2b911
Your plaintext : █
```

ii) Output Screenshot (should have SRN) for step 2



```
[10/04/24]PES2UG22CS556:~/.../Files$>nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: cf60828d18b68e3f8095fd7f03f1c6f9
The IV used     : 018467626026fa7ac0ddeb9e94a2b911

Next IV         : 21bd62b66026fa7ac0ddeb9e94a2b911
Your plaintext : 11223344aabbccdd
Your ciphertext: 17273737f39a9eb046b4ea2a83e0f4cb

Next IV         : dd3fe2126126fa7ac0ddeb9e94a2b911
Your plaintext : If0f98490000000000000000000000000
Invalid hex string

Next IV         : 91d144686126fa7ac0ddeb9e94a2b911
Your plaintext : █
```

iii) What is Bob's message ? (yes/no)
YES

iv) Explain your observations for deciphering above.

Given the initialization vector (kIV), plaintext (kP), and ciphertext (kC), we know that the ciphertext was generated using AES:

C=AES(kIV⊕kP)C = AES(kIV \oplus kP)C=AES(kIV⊕kP)

Now, when we have a specific IV (mIV) for encrypting a plaintext (mP), the ciphertext is given by:

mC=AES(mIV⊕mP)mC = AES(mIV \oplus mP)mC=AES(mIV⊕mP)

If we select mPmPmP as mP=kIV⊕mIV⊕mPmP = kIV \oplus mIV \oplus mPmP=kIV⊕mIV⊕mP, then mCmCmC simplifies to:

mC=AES(kIV⊕mP)mC = AES(kIV \oplus mP)mC=AES(kIV⊕mP)

Thus, mCmCmC will equal kCkCkC if and only if mPmPmP matches kPkPkP, enabling us to successfully decipher the plaintext.

Overall Submission in SRN_Lab4_AC : pdf