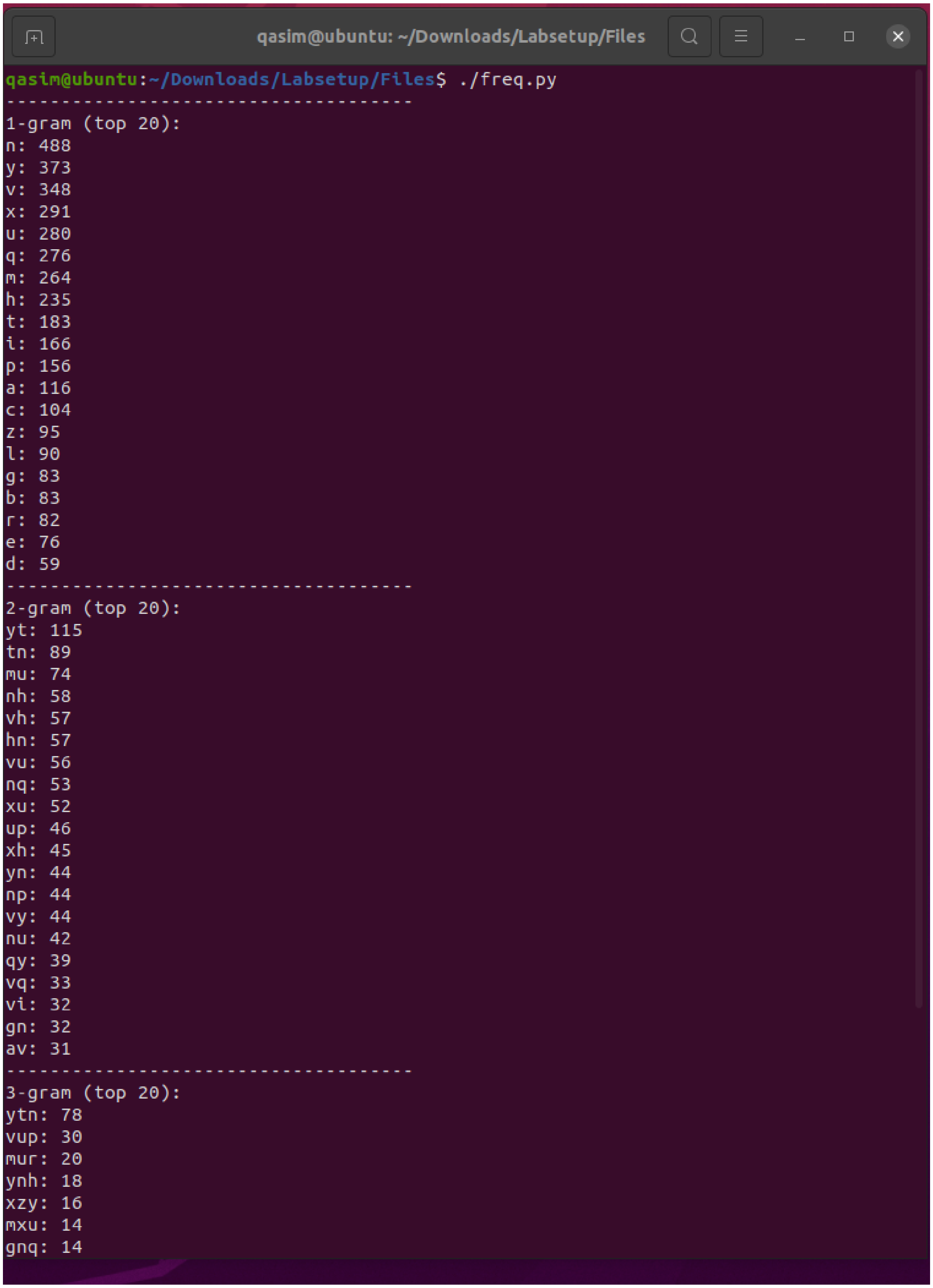
Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

Task 1:

Frequency of letters cipher text:

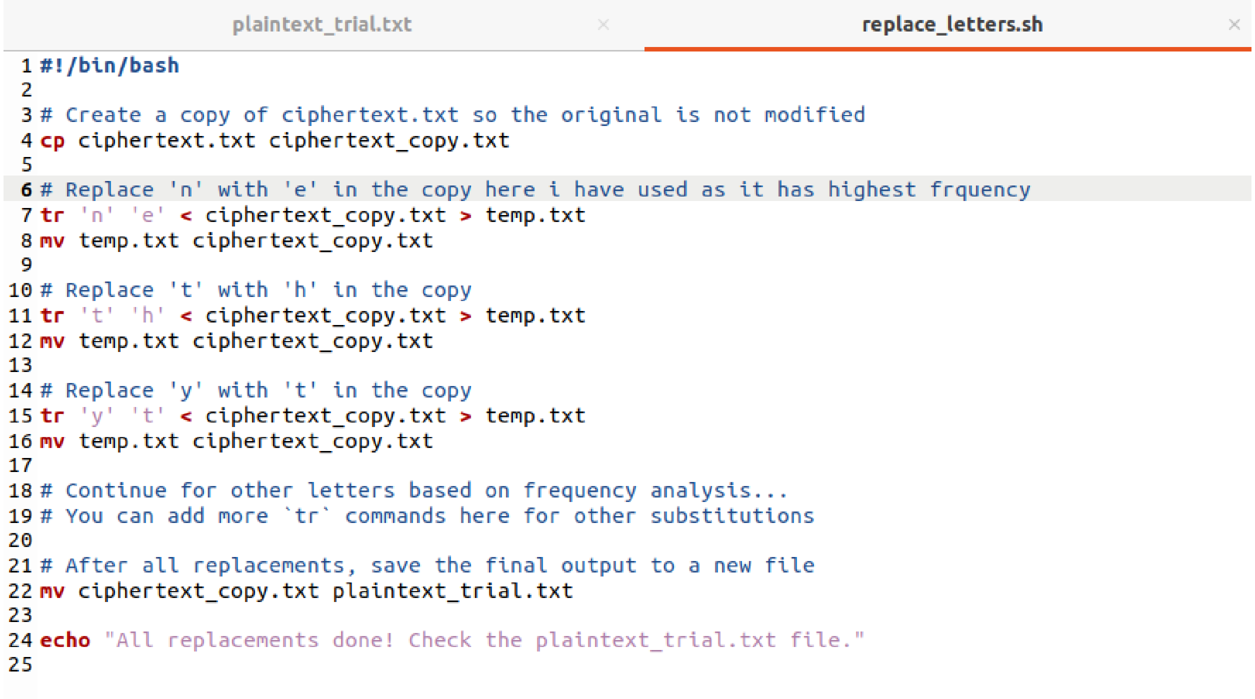


```
qasim@ubuntu:~/Downloads/Labsetup/Files$ ./freq.py
-------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
m: 264
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
-------------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
nh: 58
vh: 57
hn: 57
vu: 56
nq: 53
xu: 52
up: 46
xh: 45
yn: 44
np: 44
vy: 44
nu: 42
qy: 39
vq: 33
vi: 32
gn: 32
av: 31
-------------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
```

Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

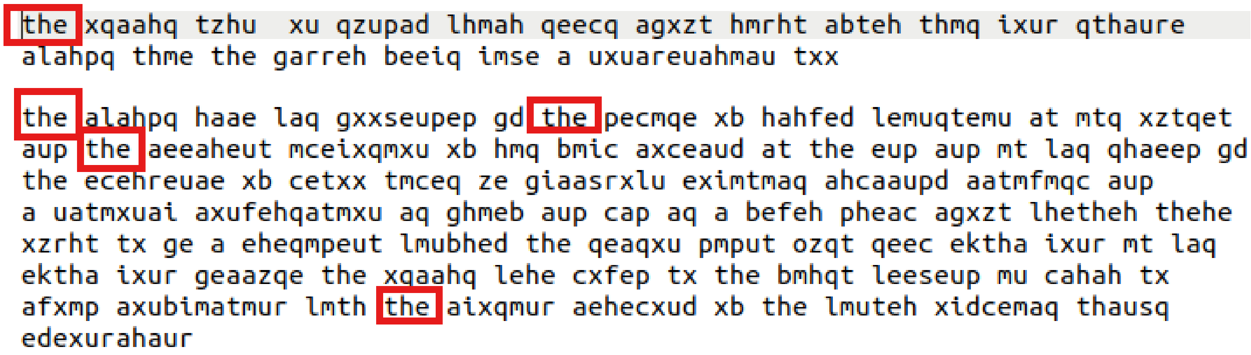
To create a script that decrypts the ciphertext by mapping the highest frequency letters in English to the corresponding most frequent letters in your ciphertext.
Mutiple repeated commands merged in sh file



```bash
#!/bin/bash

# Create a copy of ciphertext.txt so the original is not modified
cp ciphertext.txt ciphertext_copy.txt

# Replace 'n' with 'e' in the copy here i have used as it has highest frquency
tr 'n' 'e' < ciphertext_copy.txt > temp.txt
mv temp.txt ciphertext_copy.txt

# Replace 't' with 'h' in the copy
tr 't' 'h' < ciphertext_copy.txt > temp.txt
mv temp.txt ciphertext_copy.txt

# Replace 'y' with 't' in the copy
tr 'y' 't' < ciphertext_copy.txt > temp.txt
mv temp.txt ciphertext_copy.txt

# Continue for other letters based on frequency analysis...
# You can add more `tr` commands here for other substitutions

# After all replacements, save the final output to a new file
mv ciphertext_copy.txt plaintext_trial.txt

echo "All replacements done! Check the plaintext_trial.txt file."

```

So after using the highest alphabet frequency we could see the word" the" present at many places in cipher text when encrypted.

RESULT:

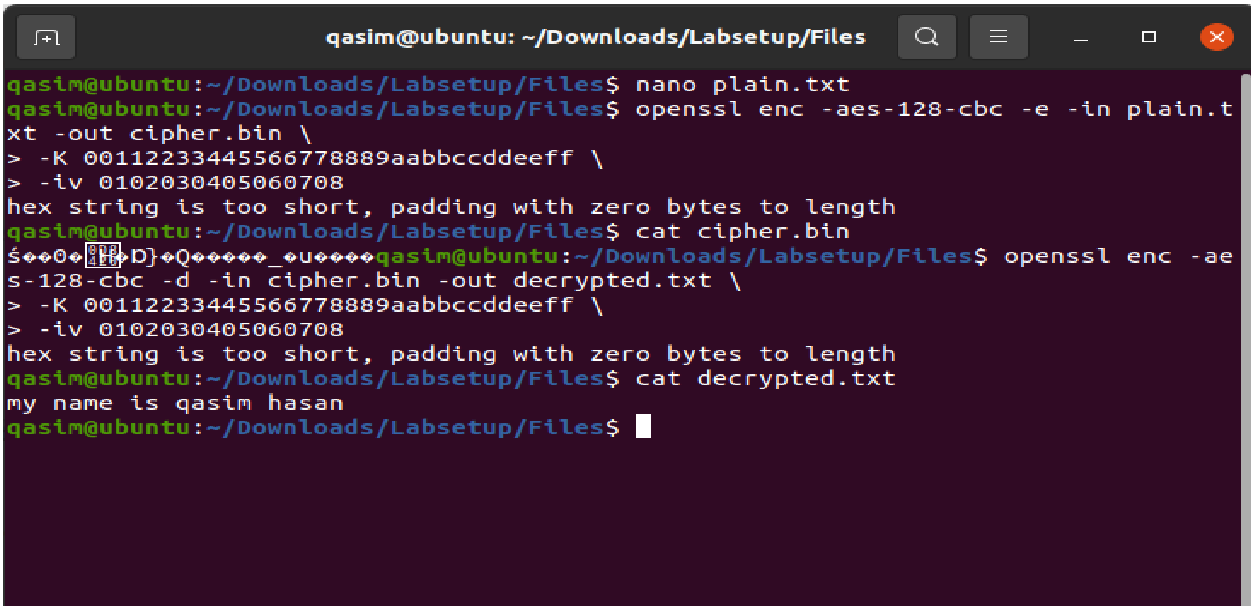


```
the xqaahq tzhu  xu qzupad lhmah qeecq agxzt hmrht abteh thmq ixur qthaure
alahpq thme the garreh beeiq imse a uxuareuahmau txx

the alahpq haae laq gxxseupep gd the pecmqe xb hahfed lemuqtemu at mtq xztqet
aup the aeeaheut mceixqmxu xb hmq bmic axceaud at the eup aup mt laq qhaeep gd
the ecehreuae xb cetxx tmceq ze giaasrxlu eximtmaq ahcaaupd aatmfmqc aup
a uatmxuai axufehqatmxu aq ghmeb aup cap aq a befeh pheac agxzt lhetheh thehe
xzrht tx ge a eheqmpeut lmubhed the qeaqxu pmput ozqt qeec ektha ixur mt laq
ektha ixur geaazqe the xqaahq lehe cxfep tx the bmhqt leeseup mu cahah tx
afxmp axubimatmur lmth the aixqmur aehecxud xb the lmuteh xidcemaq thausq
edexurahaur
```

Task 2:

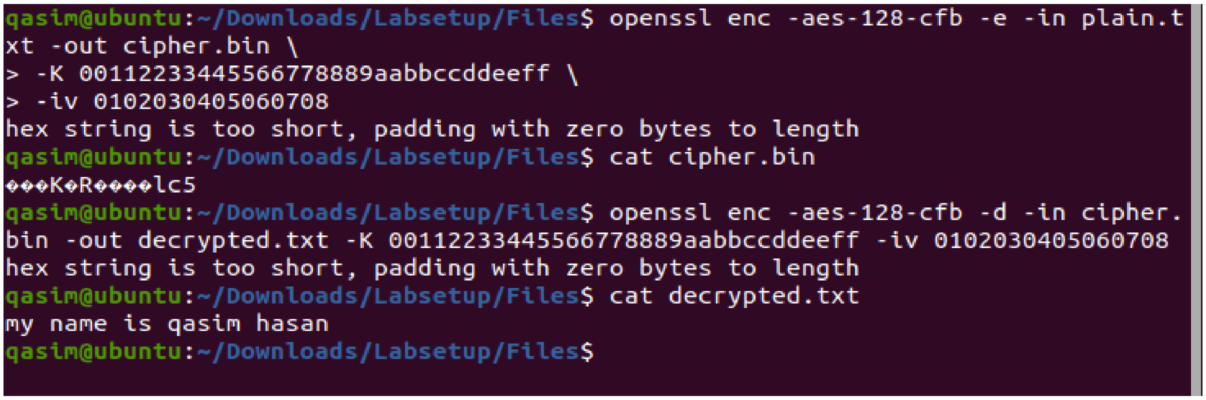
Create a plain.txt with “My name is Qasim hasan” to encrypt with 3 Algorithms.

Algorithm 1: [AES WITH CFC MODE]



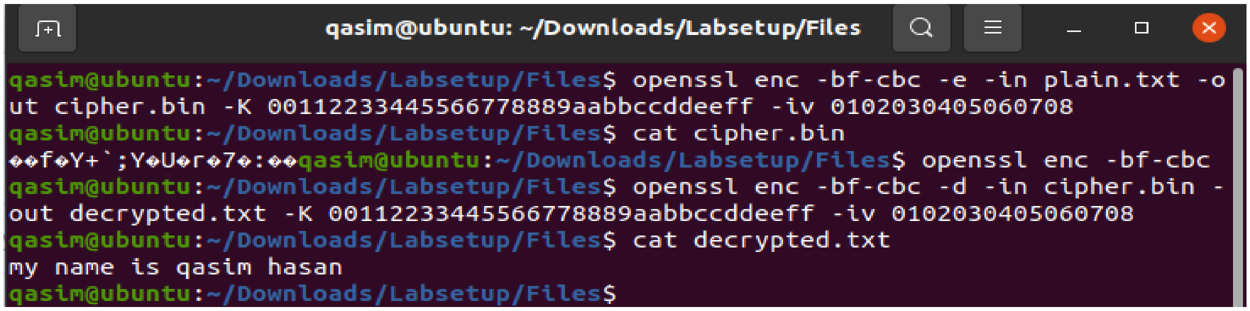
```
qasim@ubuntu:~/Downloads/Labsetup/Files$ nano plain.txt
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-128-cbc -e -in plain.t
xt -out cipher.bin \
> -K 00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat cipher.bin
ś��0�□�D}�Q�����_�u����qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -ae
s-128-cbc -d -in cipher.bin -out decrypted.txt \
> -K 00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat decrypted.txt
my name is qasim hasan
qasim@ubuntu:~/Downloads/Labsetup/Files$ 
```

Algorithm 2: [AES WITH CFB MODE]



```
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-128-cfb -e -in plain.t
xt -out cipher.bin \
> -K 00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat cipher.bin
���K�R����lc5
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-128-cfb -d -in cipher.
bin -out decrypted.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat decrypted.txt
my name is qasim hasan
qasim@ubuntu:~/Downloads/Labsetup/Files$
```

Algorithm 3: [BLOWFISH CIPHER]



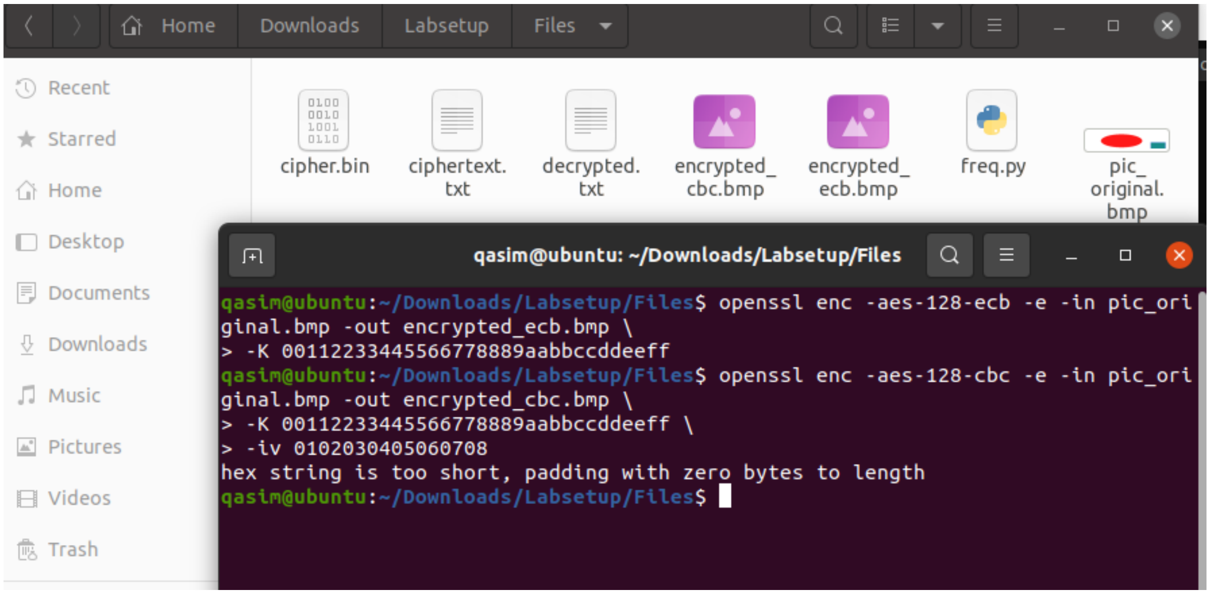
```
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -bf-cbc -e -in plain.txt -o
ut cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat cipher.bin
��f�Y+`;Y�U�r�7�:��qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -bf-cbc
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -bf-cbc -d -in cipher.bin -
out decrypted.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
qasim@ubuntu:~/Downloads/Labsetup/Files$ cat decrypted.txt
my name is qasim hasan
qasim@ubuntu:~/Downloads/Labsetup/Files$
```

Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

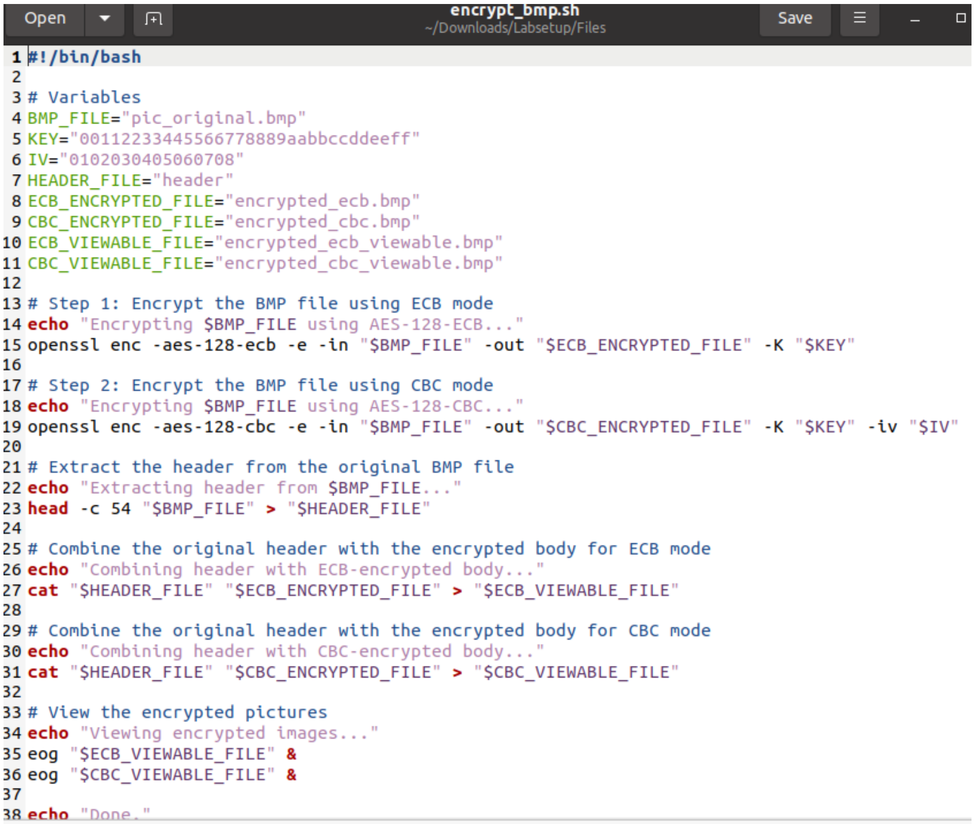
Task 3: Mutiple repeated commands merged in sh file

Image encrypted in both ECB and CBC:

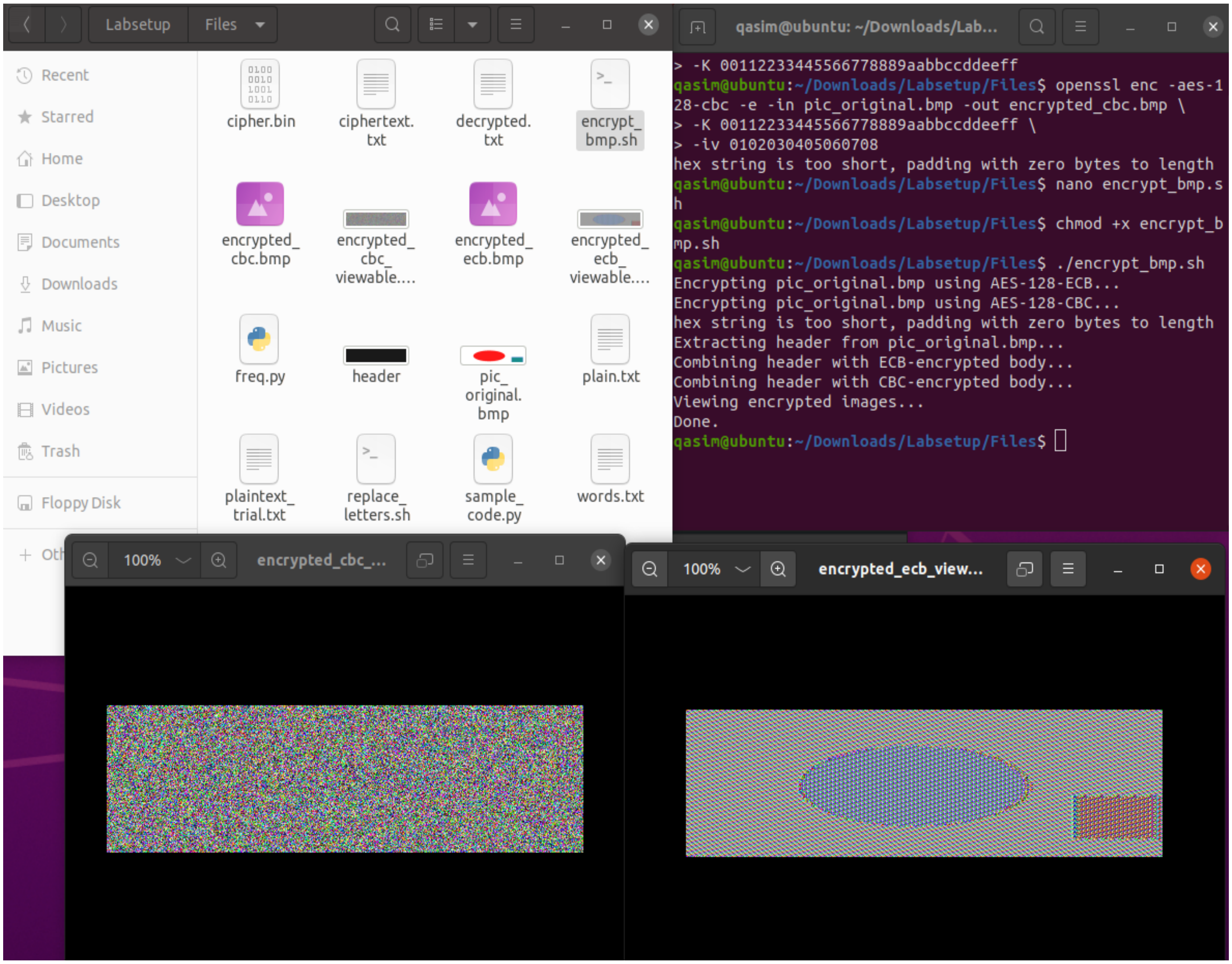


```
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-128-ecb -e -in pic_ori
ginal.bmp -out encrypted_ecb.bmp \
> -K 00112233445566778889aabbccddeeff
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-128-cbc -e -in pic_ori
ginal.bmp -out encrypted_cbc.bmp \
> -K 00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ 
```

Encrypted BMP files are now unusable for viewing because their headers are also encrypted. To view them, you need to keep the original header intact and only replace the data portion with the encrypted content. These are the commands I have merged in .sh file to replace header and show the encrypted images.



```bash
#!/bin/bash

# Variables
BMP_FILE="pic_original.bmp"
KEY="00112233445566778889aabbccddeeff"
IV="0102030405060708"
HEADER_FILE="header"
ECB_ENCRYPTED_FILE="encrypted_ecb.bmp"
CBC_ENCRYPTED_FILE="encrypted_cbc.bmp"
ECB_VIEWABLE_FILE="encrypted_ecb_viewable.bmp"
CBC_VIEWABLE_FILE="encrypted_cbc_viewable.bmp"

# Step 1: Encrypt the BMP file using ECB mode
echo "Encrypting $BMP_FILE using AES-128-ECB..."
openssl enc -aes-128-ecb -e -in "$BMP_FILE" -out "$ECB_ENCRYPTED_FILE" -K "$KEY"

# Step 2: Encrypt the BMP file using CBC mode
echo "Encrypting $BMP_FILE using AES-128-CBC..."
openssl enc -aes-128-cbc -e -in "$BMP_FILE" -out "$CBC_ENCRYPTED_FILE" -K "$KEY" -iv "$IV"

# Extract the header from the original BMP file
echo "Extracting header from $BMP_FILE..."
head -c 54 "$BMP_FILE" > "$HEADER_FILE"

# Combine the original header with the encrypted body for ECB mode
echo "Combining header with ECB-encrypted body..."
cat "$HEADER_FILE" "$ECB_ENCRYPTED_FILE" > "$ECB_VIEWABLE_FILE"

# Combine the original header with the encrypted body for CBC mode
echo "Combining header with CBC-encrypted body..."
cat "$HEADER_FILE" "$CBC_ENCRYPTED_FILE" > "$CBC_VIEWABLE_FILE"

# View the encrypted pictures
echo "Viewing encrypted images..."
eog "$ECB_VIEWABLE_FILE" &
eog "$CBC_VIEWABLE_FILE" &

echo "Done."
```



```
> -K 00112233445566778889aabbccddeeff
qasim@ubuntu:~/Downloads/Labsetup/Files$ openssl enc -aes-1
28-cbc -e -in pic_original.bmp -out encrypted_cbc.bmp \
> -K 00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
qasim@ubuntu:~/Downloads/Labsetup/Files$ nano encrypt_bmp.s
h
qasim@ubuntu:~/Downloads/Labsetup/Files$ chmod +x encrypt_b
mp.sh
qasim@ubuntu:~/Downloads/Labsetup/Files$ ./encrypt_bmp.sh
Encrypting pic_original.bmp using AES-128-ECB...
Encrypting pic_original.bmp using AES-128-CBC...
hex string is too short, padding with zero bytes to length
Extracting header from pic_original.bmp...
Combining header with ECB-encrypted body...
Combining header with CBC-encrypted body...
Viewing encrypted images...
Done.
qasim@ubuntu:~/Downloads/Labsetup/Files$ 
```

Analyzing and Observing the Differences

- **ECB Mode**: ECB-encrypted images may reveal some patterns from the original due to the independent encryption of each block, making repeating patterns visible.
- **CBC Mode**: CBC-encrypted images appear more random with no discernible patterns, as CBC chains blocks together, enhancing security.

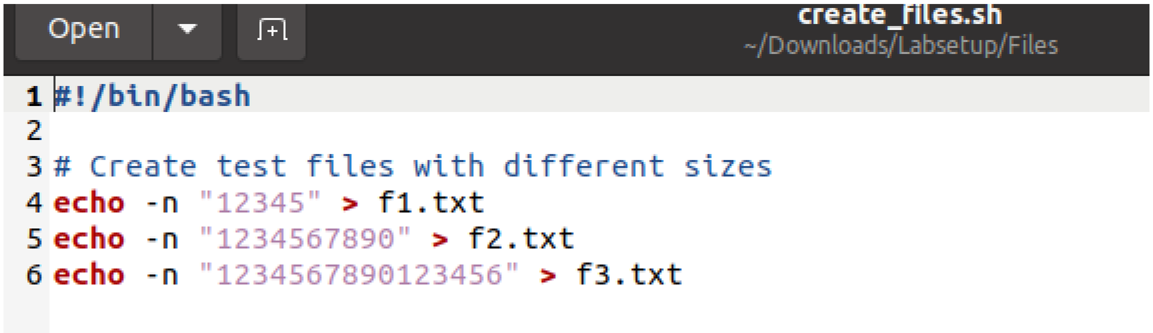
Reporting my Observations

- **Visual Difference**: ECB-encrypted images may retain patterns from the original, while CBC-encrypted images are more uniformly random.
- **Information Derived**: ECB allows partial information recovery due to visible patterns; CBC provides better security by masking patterns.
- **Security Comparison**: CBC is more secure than ECB because it hides repeating patterns by chaining blocks, making encrypted content less predictable.

Task 4:

Creating 3 files to test each algorithm and test if there is padding or not

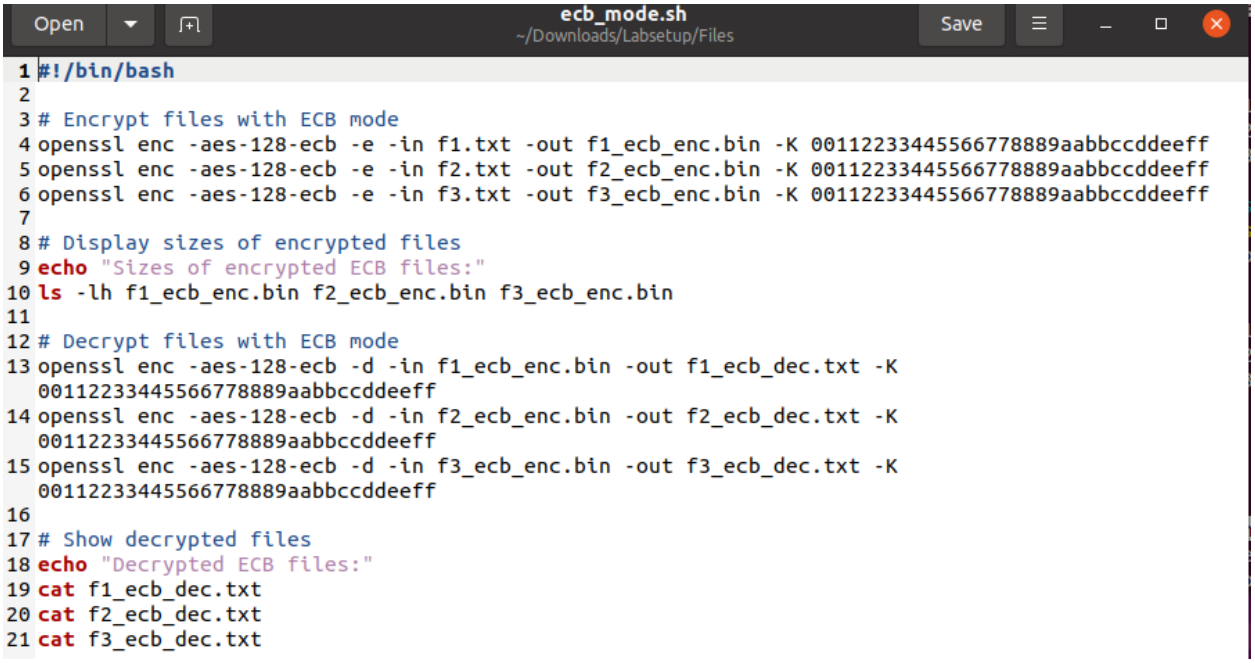
Mutiple repeated commands merged in sh file



```
#!/bin/bash

# Create test files with different sizes
echo -n "12345" > f1.txt
echo -n "1234567890" > f2.txt
echo -n "1234567890123456" > f3.txt
```

shell scripts for each mode, each performing encryption, decryption, padding analysis, and file size comparison **the results are at the end of task 4.**

For ECB mode:

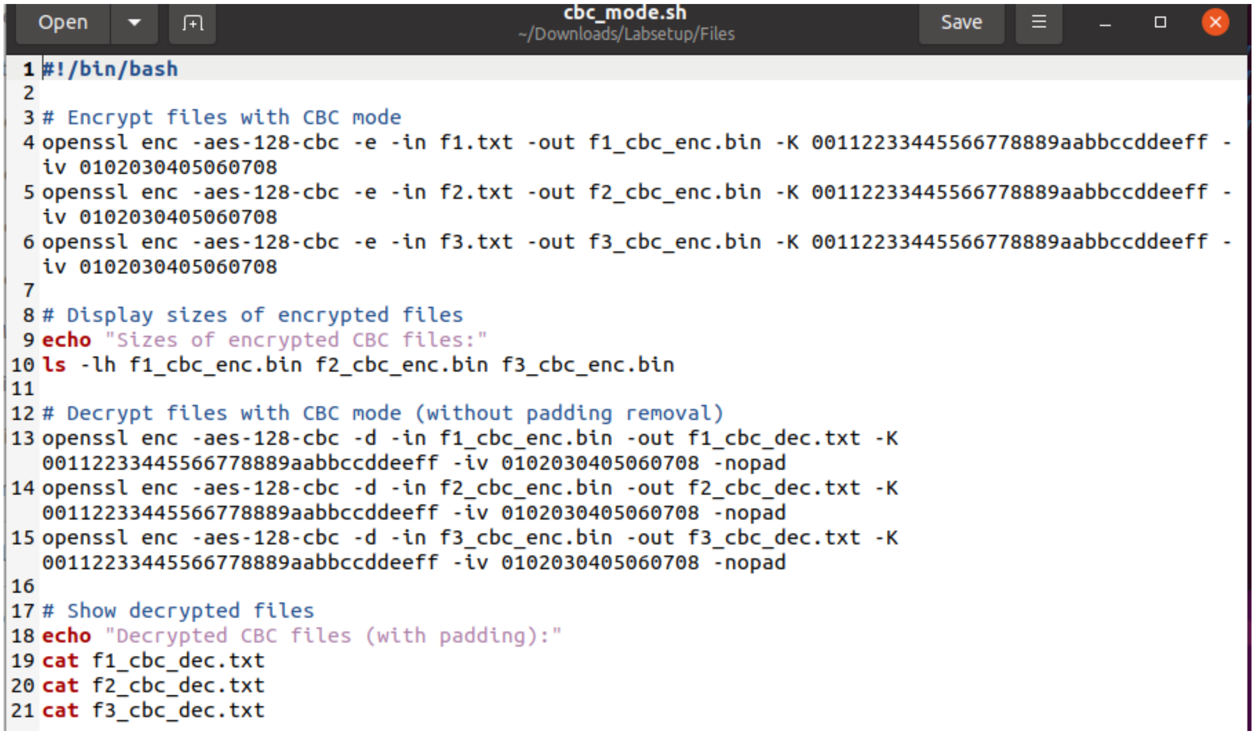


```
#!/bin/bash

# Encrypt files with ECB mode
openssl enc -aes-128-ecb -e -in f1.txt -out f1_ecb_enc.bin -K 00112233445566778889aabbccddeeff
openssl enc -aes-128-ecb -e -in f2.txt -out f2_ecb_enc.bin -K 00112233445566778889aabbccddeeff
openssl enc -aes-128-ecb -e -in f3.txt -out f3_ecb_enc.bin -K 00112233445566778889aabbccddeeff

# Display sizes of encrypted files
echo "Sizes of encrypted ECB files:"
ls -lh f1_ecb_enc.bin f2_ecb_enc.bin f3_ecb_enc.bin

# Decrypt files with ECB mode
openssl enc -aes-128-ecb -d -in f1_ecb_enc.bin -out f1_ecb_dec.txt -K 00112233445566778889aabbccddeeff
openssl enc -aes-128-ecb -d -in f2_ecb_enc.bin -out f2_ecb_dec.txt -K 00112233445566778889aabbccddeeff
openssl enc -aes-128-ecb -d -in f3_ecb_enc.bin -out f3_ecb_dec.txt -K 00112233445566778889aabbccddeeff

# Show decrypted files
echo "Decrypted ECB files:"
cat f1_ecb_dec.txt
cat f2_ecb_dec.txt
cat f3_ecb_dec.txt
```

Information Security LAB 1

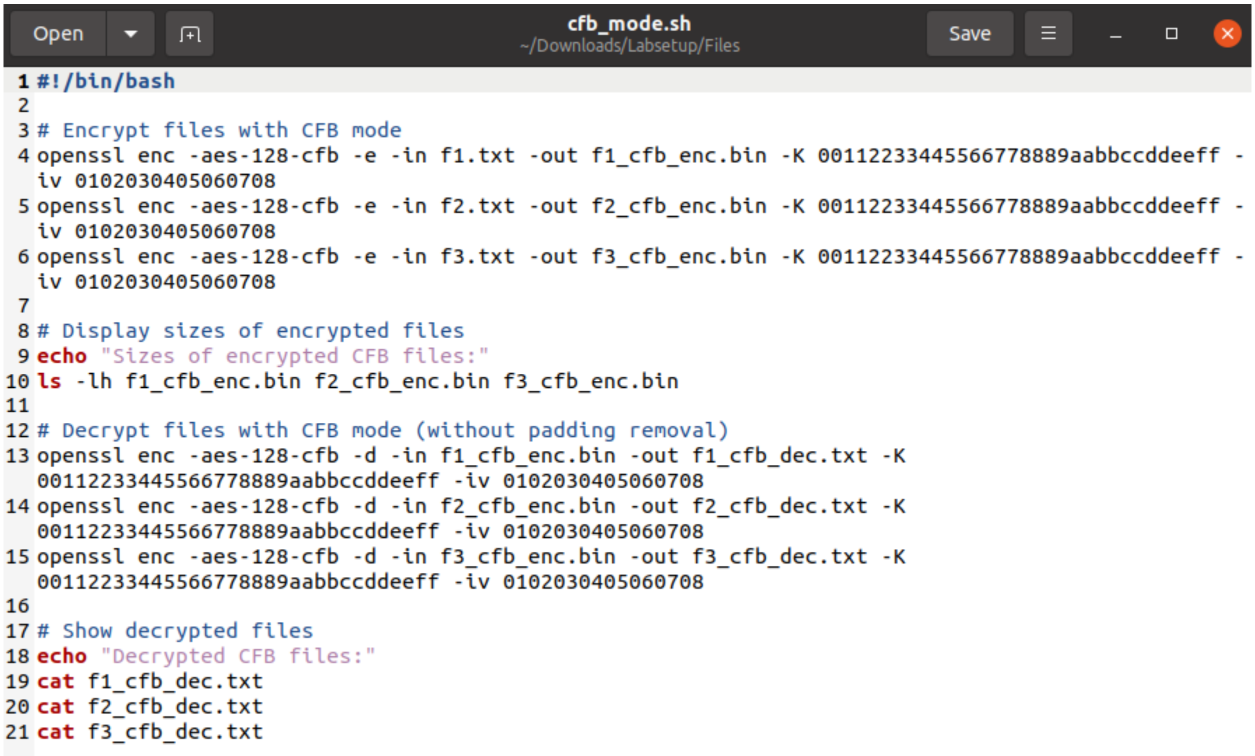
NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

For CBC mode:



```
#!/bin/bash

# Encrypt files with CBC mode
openssl enc -aes-128-cbc -e -in f1.txt -out f1_cbc_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cbc -e -in f2.txt -out f2_cbc_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cbc -e -in f3.txt -out f3_cbc_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

# Display sizes of encrypted files
echo "Sizes of encrypted CBC files:"
ls -lh f1_cbc_enc.bin f2_cbc_enc.bin f3_cbc_enc.bin

# Decrypt files with CBC mode (without padding removal)
openssl enc -aes-128-cbc -d -in f1_cbc_enc.bin -out f1_cbc_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cbc -d -in f2_cbc_enc.bin -out f2_cbc_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cbc -d -in f3_cbc_enc.bin -out f3_cbc_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

# Show decrypted files
echo "Decrypted CBC files (with padding):"
cat f1_cbc_dec.txt
cat f2_cbc_dec.txt
cat f3_cbc_dec.txt
```

For CFB mode:

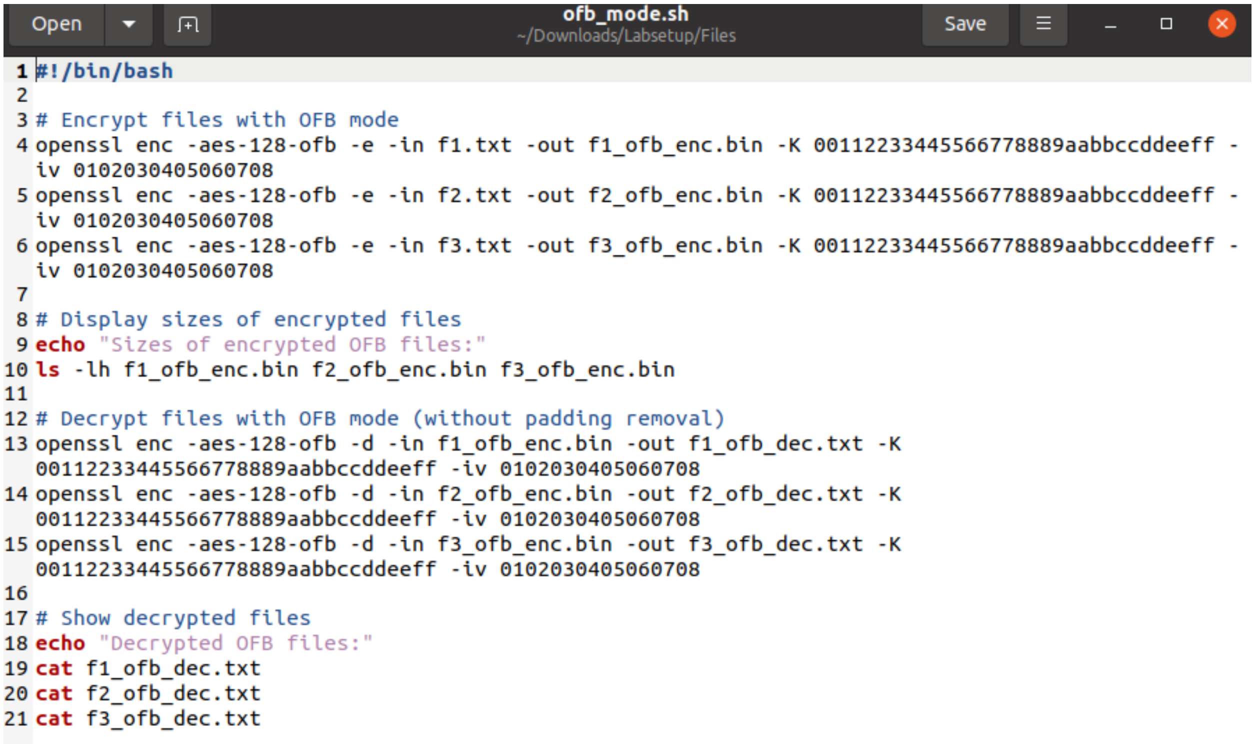


```
#!/bin/bash

# Encrypt files with CFB mode
openssl enc -aes-128-cfb -e -in f1.txt -out f1_cfb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -e -in f2.txt -out f2_cfb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -e -in f3.txt -out f3_cfb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

# Display sizes of encrypted files
echo "Sizes of encrypted CFB files:"
ls -lh f1_cfb_enc.bin f2_cfb_enc.bin f3_cfb_enc.bin

# Decrypt files with CFB mode (without padding removal)
openssl enc -aes-128-cfb -d -in f1_cfb_enc.bin -out f1_cfb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -d -in f2_cfb_enc.bin -out f2_cfb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -d -in f3_cfb_enc.bin -out f3_cfb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

# Show decrypted files
echo "Decrypted CFB files:"
cat f1_cfb_dec.txt
cat f2_cfb_dec.txt
cat f3_cfb_dec.txt
```

Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

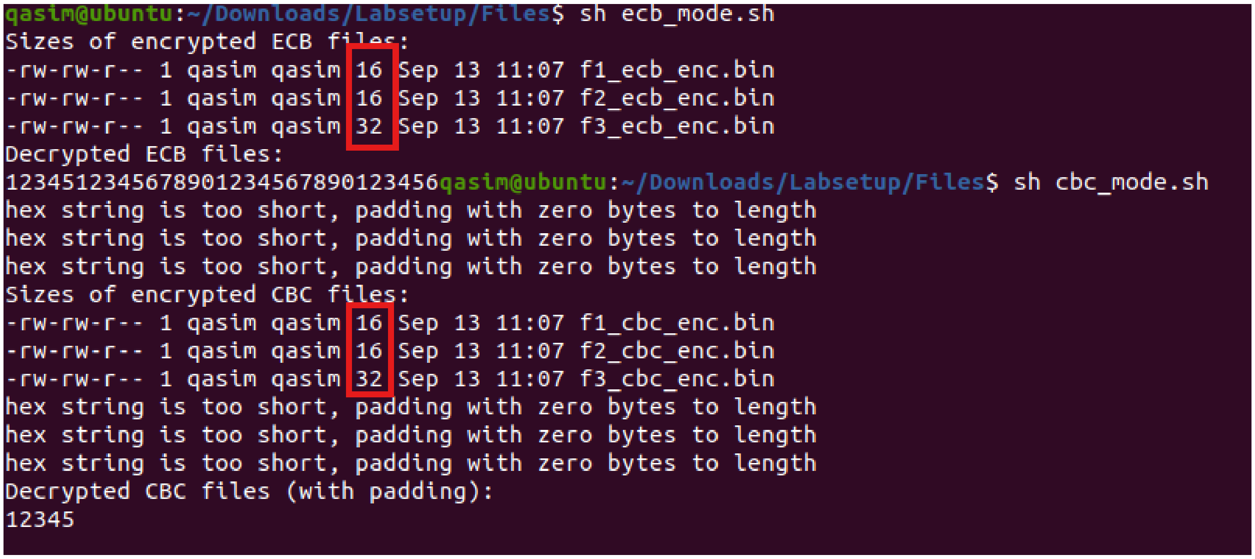
For OFB mode:



```
#!/bin/bash

# Encrypt files with OFB mode
openssl enc -aes-128-ofb -e -in f1.txt -out f1_ofb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -e -in f2.txt -out f2_ofb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -e -in f3.txt -out f3_ofb_enc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

# Display sizes of encrypted files
echo "Sizes of encrypted OFB files:"
ls -lh f1_ofb_enc.bin f2_ofb_enc.bin f3_ofb_enc.bin

# Decrypt files with OFB mode (without padding removal)
openssl enc -aes-128-ofb -d -in f1_ofb_enc.bin -out f1_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -d -in f2_ofb_enc.bin -out f2_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -d -in f3_ofb_enc.bin -out f3_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

# Show decrypted files
echo "Decrypted OFB files:"
cat f1_ofb_dec.txt
cat f2_ofb_dec.txt
cat f3_ofb_dec.txt
```

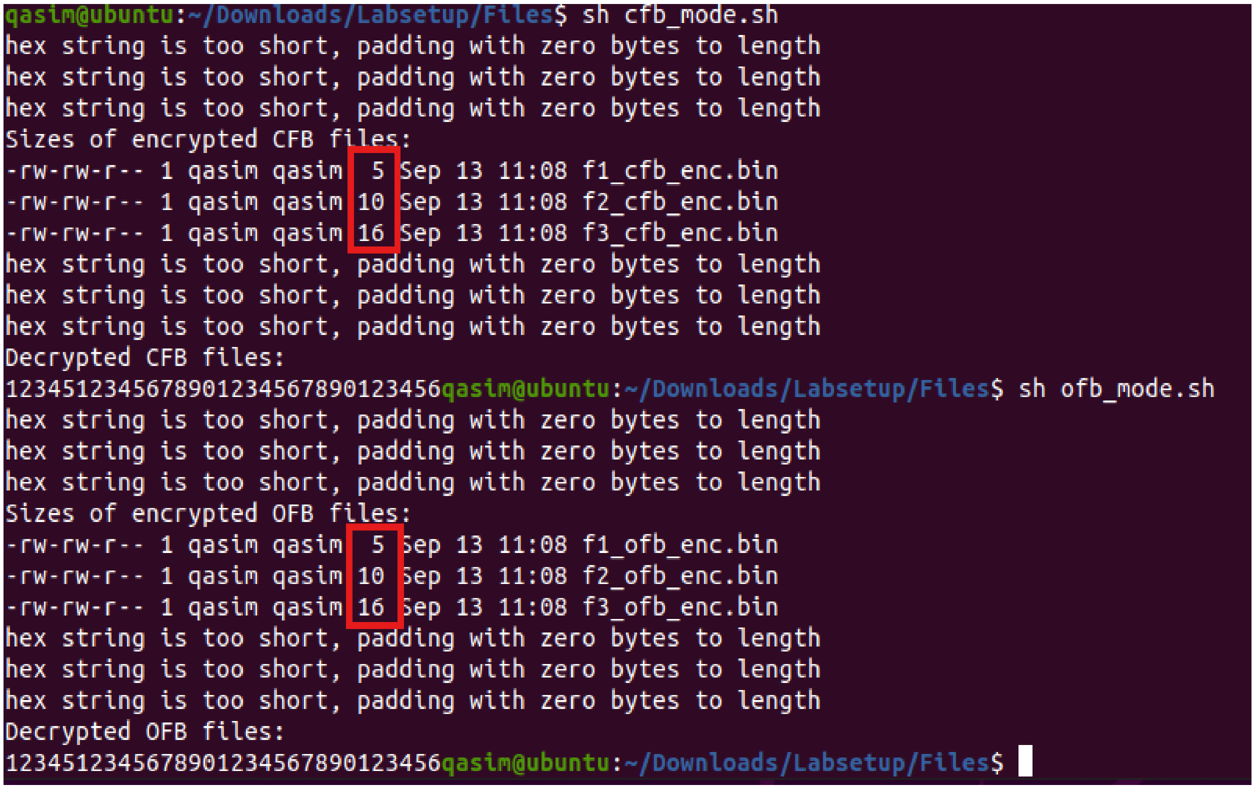
RESULTS:

For First Two Algorithm [ECB AND CBC] Padding is being applied:



```
qasim@ubuntu:~/Downloads/Labsetup/Files$ sh ecb_mode.sh
Sizes of encrypted ECB files:
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:07 f1_ecb_enc.bin
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:07 f2_ecb_enc.bin
-rw-rw-r-- 1 qasim qasim 32 Sep 13 11:07 f3_ecb_enc.bin
Decrypted ECB files:
12345123456789012345678901234 56qasim@ubuntu:~/Downloads/Labsetup/Files$ sh cbc_mode.sh
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Sizes of encrypted CBC files:
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:07 f1_cbc_enc.bin
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:07 f2_cbc_enc.bin
-rw-rw-r-- 1 qasim qasim 32 Sep 13 11:07 f3_cbc_enc.bin
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Decrypted CBC files (with padding):
12345
```

For Last Two Algorithm [CFB AND OFB] Padding is not being applied:



```
qasim@ubuntu:~/Downloads/Labsetup/Files$ sh cfb_mode.sh
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Sizes of encrypted CFB files:
-rw-rw-r-- 1 qasim qasim  5 Sep 13 11:08 f1_cfb_enc.bin
-rw-rw-r-- 1 qasim qasim 10 Sep 13 11:08 f2_cfb_enc.bin
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:08 f3_cfb_enc.bin
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Decrypted CFB files:
1234512345678901234567890123456qasim@ubuntu:~/Downloads/Labsetup/Files$ sh ofb_mode.sh
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Sizes of encrypted OFB files:
-rw-rw-r-- 1 qasim qasim  5 Sep 13 11:08 f1_ofb_enc.bin
-rw-rw-r-- 1 qasim qasim 10 Sep 13 11:08 f2_ofb_enc.bin
-rw-rw-r-- 1 qasim qasim 16 Sep 13 11:08 f3_ofb_enc.bin
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Decrypted OFB files:
1234512345678901234567890123456qasim@ubuntu:~/Downloads/Labsetup/Files$
```

Information Security LAB 1

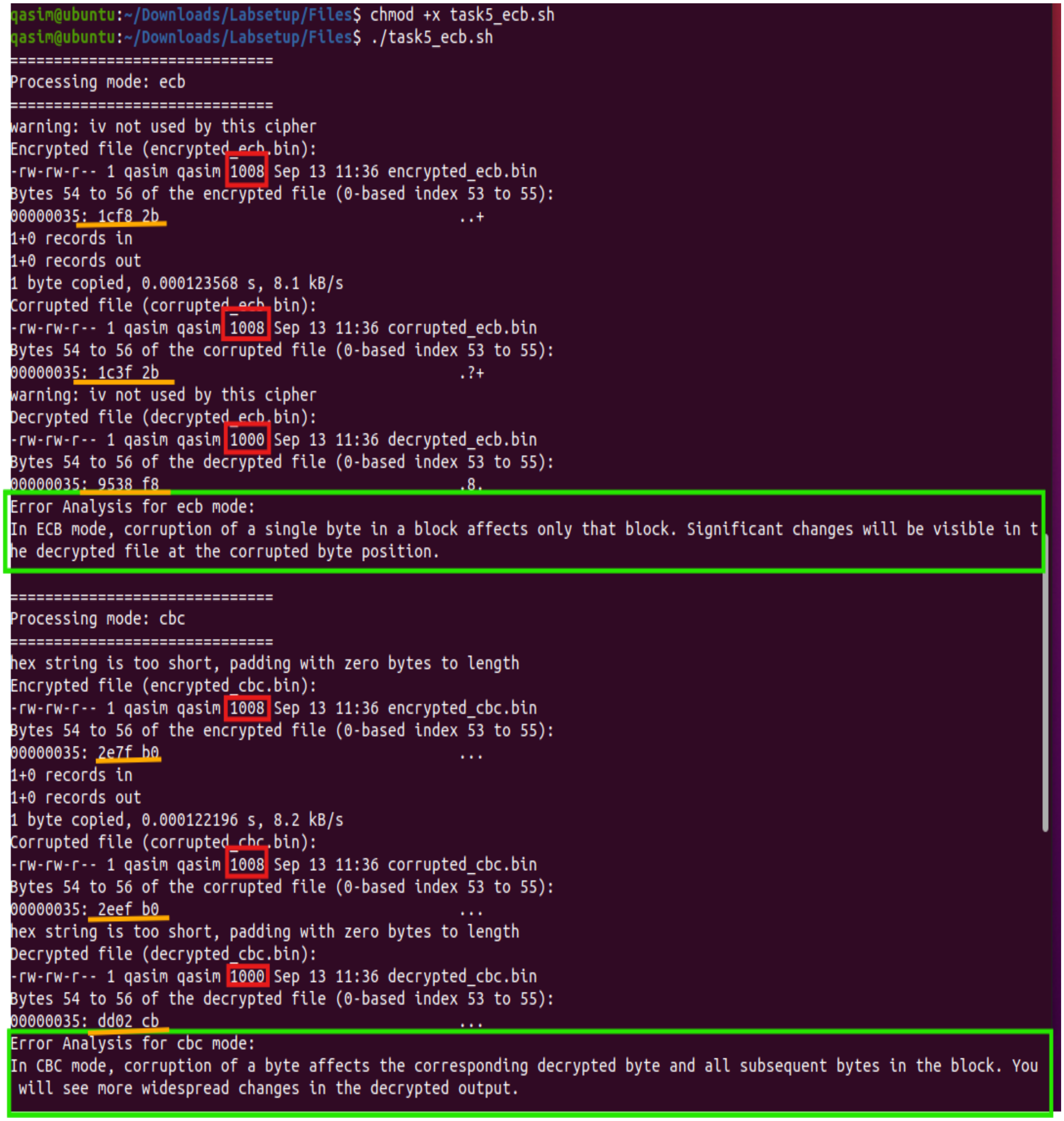
NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

Task 5:

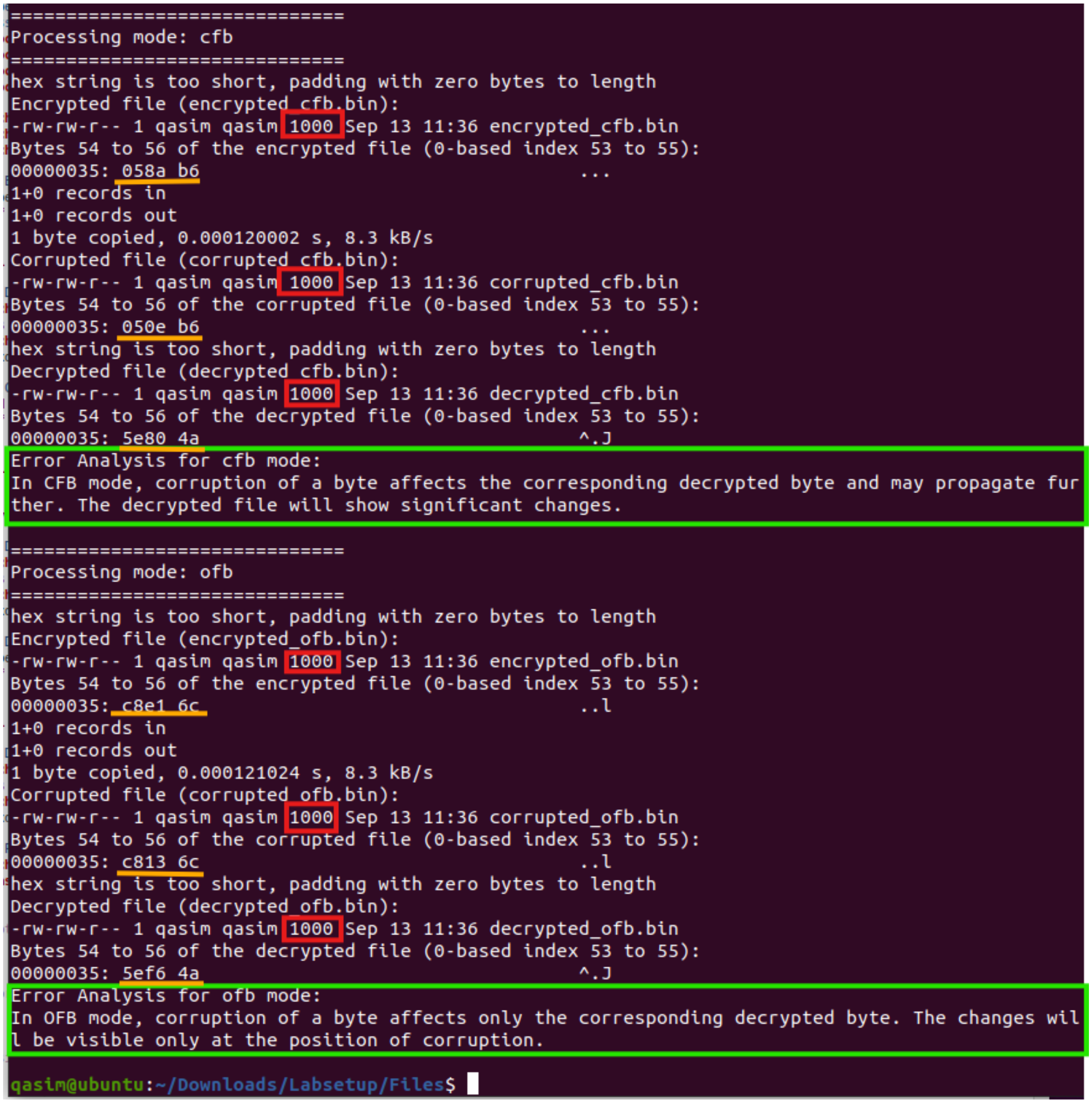
Yellow is the file of our random 1000 byte file we created in terminal

Red is the file size of all encrypted, corrupted, and decrypted files

Green is observation for each mode [EBC,CBC,CFB,OFB]



```
qasim@ubuntu:~/Downloads/Labsetup/Files$ chmod +x task5_ecb.sh
qasim@ubuntu:~/Downloads/Labsetup/Files$ ./task5_ecb.sh
==============================
Processing mode: ecb
==============================
warning: iv not used by this cipher
Encrypted file (encrypted_ecb.bin):
-rw-rw-r-- 1 qasim qasim 1008 Sep 13 11:36 encrypted_ecb.bin
Bytes 54 to 56 of the encrypted file (0-based index 53 to 55):
00000035: 1cf8 2b                                  ..+
1+0 records in
1+0 records out
1 byte copied, 0.000123568 s, 8.1 kB/s
Corrupted file (corrupted_ecb.bin):
-rw-rw-r-- 1 qasim qasim 1008 Sep 13 11:36 corrupted_ecb.bin
Bytes 54 to 56 of the corrupted file (0-based index 53 to 55):
00000035: 1c3f 2b                                  .?+
warning: iv not used by this cipher
Decrypted file (decrypted_ecb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 decrypted_ecb.bin
Bytes 54 to 56 of the decrypted file (0-based index 53 to 55):
00000035: 9538 f8                                  .8.
Error Analysis for ecb mode:
In ECB mode, corruption of a single byte in a block affects only that block. Significant changes will be visible in t
he decrypted file at the corrupted byte position.

==============================
Processing mode: cbc
==============================
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_cbc.bin):
-rw-rw-r-- 1 qasim qasim 1008 Sep 13 11:36 encrypted_cbc.bin
Bytes 54 to 56 of the encrypted file (0-based index 53 to 55):
00000035: 2e7f b0                                  ...
1+0 records in
1+0 records out
1 byte copied, 0.000122196 s, 8.2 kB/s
Corrupted file (corrupted_cbc.bin):
-rw-rw-r-- 1 qasim qasim 1008 Sep 13 11:36 corrupted_cbc.bin
Bytes 54 to 56 of the corrupted file (0-based index 53 to 55):
00000035: 2eef b0                                  ...
hex string is too short, padding with zero bytes to length
Decrypted file (decrypted_cbc.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 decrypted_cbc.bin
Bytes 54 to 56 of the decrypted file (0-based index 53 to 55):
00000035: dd02 cb                                  ...
Error Analysis for cbc mode:
In CBC mode, corruption of a byte affects the corresponding decrypted byte and all subsequent bytes in the block. You
 will see more widespread changes in the decrypted output.
```



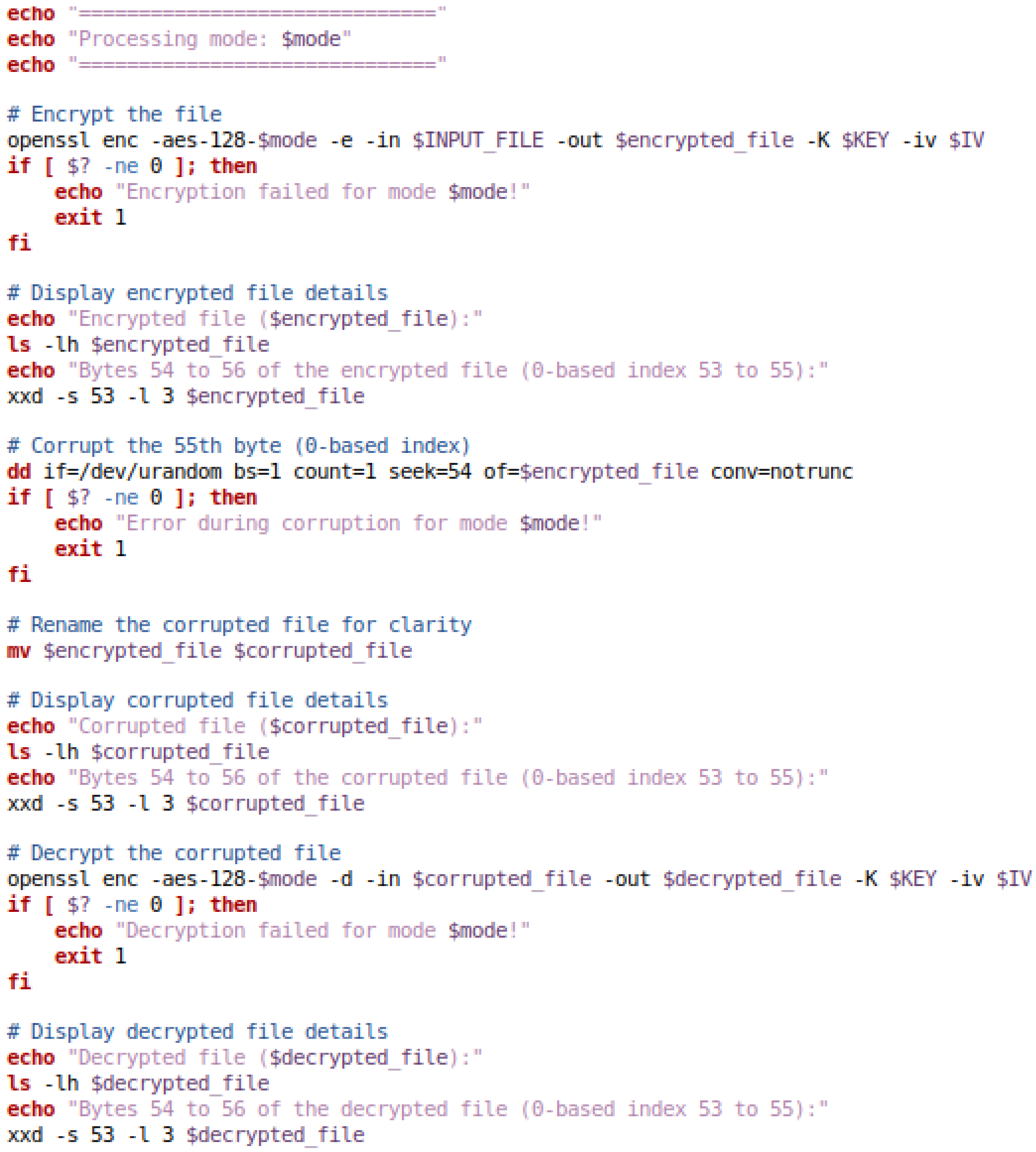
```
==============================
Processing mode: cfb
==============================
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_cfb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 encrypted_cfb.bin
Bytes 54 to 56 of the encrypted file (0-based index 53 to 55):
00000035: 058a b6                                  ...
1+0 records in
1+0 records out
1 byte copied, 0.000120002 s, 8.3 kB/s
Corrupted file (corrupted_cfb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 corrupted_cfb.bin
Bytes 54 to 56 of the corrupted file (0-based index 53 to 55):
00000035: 050e b6                                  ...
hex string is too short, padding with zero bytes to length
Decrypted file (decrypted_cfb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 decrypted_cfb.bin
Bytes 54 to 56 of the decrypted file (0-based index 53 to 55):
00000035: 5e80 4a                                  ^.J
Error Analysis for cfb mode:
In CFB mode, corruption of a byte affects the corresponding decrypted byte and may propagate fur
ther. The decrypted file will show significant changes.

==============================
Processing mode: ofb
==============================
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_ofb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 encrypted_ofb.bin
Bytes 54 to 56 of the encrypted file (0-based index 53 to 55):
00000035: c8e1 6c                                  ..l
1+0 records in
1+0 records out
1 byte copied, 0.000121024 s, 8.3 kB/s
Corrupted file (corrupted_ofb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 corrupted_ofb.bin
Bytes 54 to 56 of the corrupted file (0-based index 53 to 55):
00000035: c813 6c                                  ..l
hex string is too short, padding with zero bytes to length
Decrypted file (decrypted_ofb.bin):
-rw-rw-r-- 1 qasim qasim 1000 Sep 13 11:36 decrypted_ofb.bin
Bytes 54 to 56 of the decrypted file (0-based index 53 to 55):
00000035: 5ef6 4a                                  ^.J
Error Analysis for ofb mode:
In OFB mode, corruption of a byte affects only the corresponding decrypted byte. The changes wil
l be visible only at the position of corruption.

qasim@ubuntu:~/Downloads/Labsetup/Files$
```

Summary of above pics:

- **ECB and OFB Modes:** Corruption affects only the corrupted block.
- **CBC and CFB Modes:** Corruption affects the corrupted block and potentially the next block.

Code used for above analysis is given below:

Mutiple repeated commands merged in sh file



```
echo "=============================="
echo "Processing mode: $mode"
echo "=============================="

# Encrypt the file
openssl enc -aes-128-$mode -e -in $INPUT_FILE -out $encrypted_file -K $KEY -iv $IV
if [ $? -ne 0 ]; then
    echo "Encryption failed for mode $mode!"
    exit 1
fi

# Display encrypted file details
echo "Encrypted file ($encrypted_file):"
ls -lh $encrypted_file
echo "Bytes 54 to 56 of the encrypted file (0-based index 53 to 55):"
xxd -s 53 -l 3 $encrypted_file

# Corrupt the 55th byte (0-based index)
dd if=/dev/urandom bs=1 count=1 seek=54 of=$encrypted_file conv=notrunc
if [ $? -ne 0 ]; then
    echo "Error during corruption for mode $mode!"
    exit 1
fi

# Rename the corrupted file for clarity
mv $encrypted_file $corrupted_file

# Display corrupted file details
echo "Corrupted file ($corrupted_file):"
ls -lh $corrupted_file
echo "Bytes 54 to 56 of the corrupted file (0-based index 53 to 55):"
xxd -s 53 -l 3 $corrupted_file

# Decrypt the corrupted file
openssl enc -aes-128-$mode -d -in $corrupted_file -out $decrypted_file -K $KEY -iv $IV
if [ $? -ne 0 ]; then
    echo "Decryption failed for mode $mode!"
    exit 1
fi

# Display decrypted file details
echo "Decrypted file ($decrypted_file):"
ls -lh $decrypted_file
echo "Bytes 54 to 56 of the decrypted file (0-based index 53 to 55):"
xxd -s 53 -l 3 $decrypted_file
```

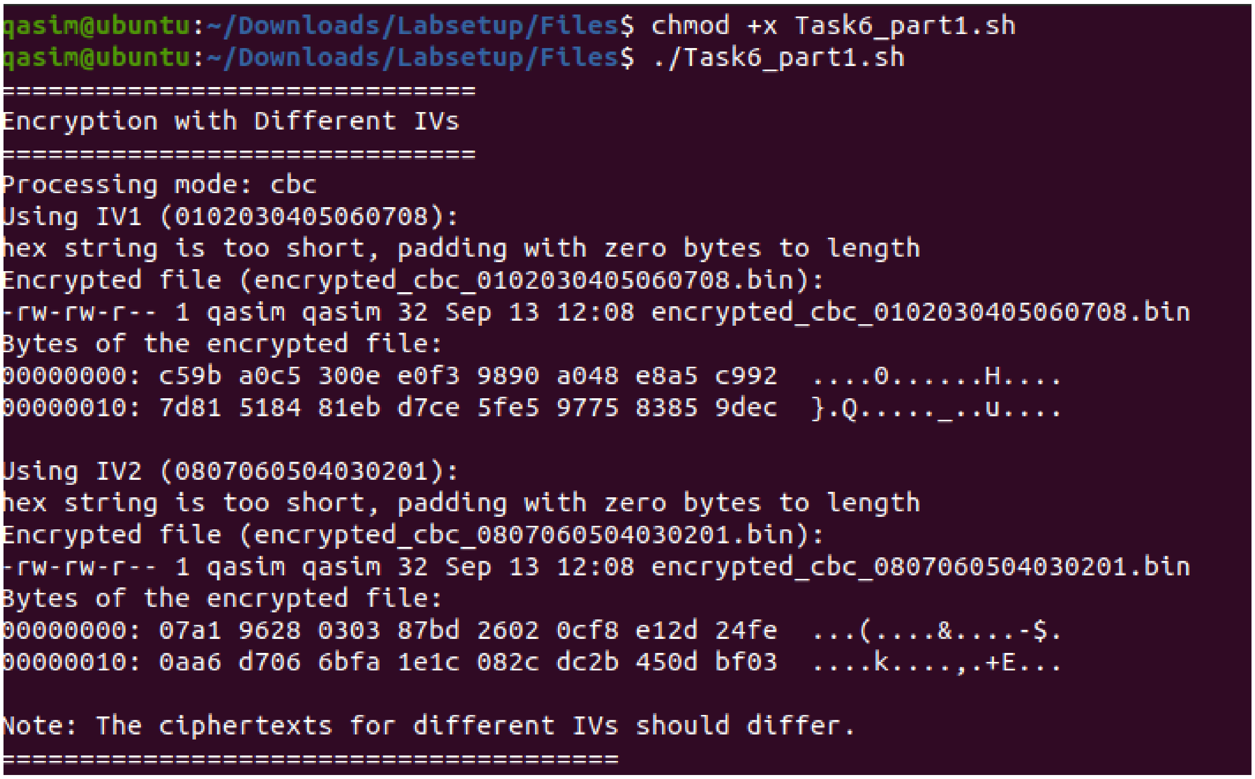
Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

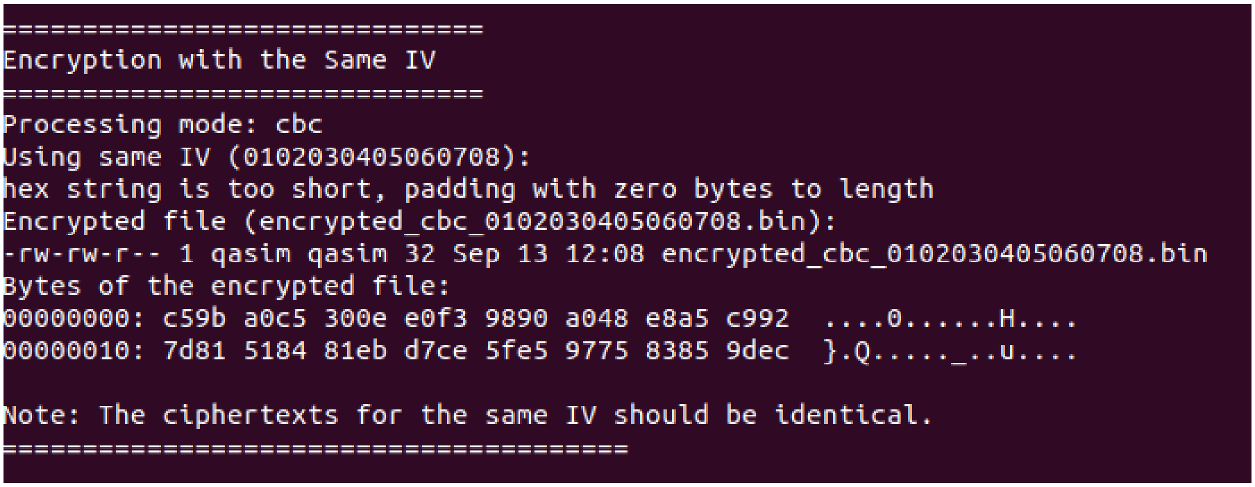
Task 6.1:

Test to see the difference with same IV and two different Ip

- **Unique IVs Ensure Security**: Unique IVs prevent the same plaintext from producing identical ciphertexts, avoiding predictable patterns that could be exploited by attackers.
- **Same/Reused IVs Compromise Security**: Using the same IV with the same key across multiple encryptions results in identical ciphertexts, revealing patterns and potential vulnerabilities.



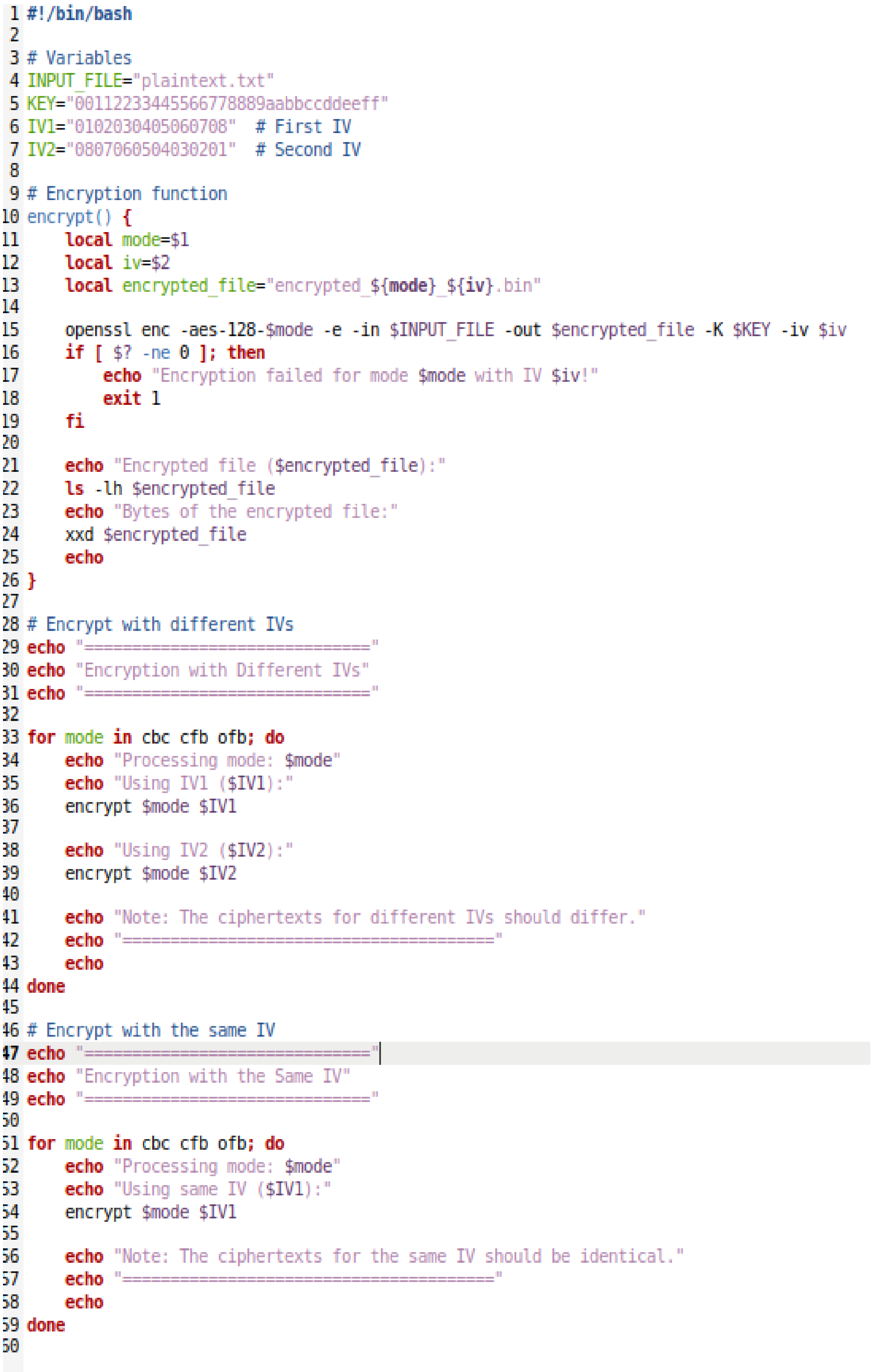
```
qasim@ubuntu:~/Downloads/Labsetup/Files$ chmod +x Task6_part1.sh
qasim@ubuntu:~/Downloads/Labsetup/Files$ ./Task6_part1.sh
==============================
Encryption with Different IVs
==============================
Processing mode: cbc
Using IV1 (0102030405060708):
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_cbc_0102030405060708.bin):
-rw-rw-r-- 1 qasim qasim 32 Sep 13 12:08 encrypted_cbc_0102030405060708.bin
Bytes of the encrypted file:
00000000: c59b a0c5 300e e0f3 9890 a048 e8a5 c992  ....0......H....
00000010: 7d81 5184 81eb d7ce 5fe5 9775 8385 9dec  }.Q....._..u....

Using IV2 (0807060504030201):
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_cbc_0807060504030201.bin):
-rw-rw-r-- 1 qasim qasim 32 Sep 13 12:08 encrypted_cbc_0807060504030201.bin
Bytes of the encrypted file:
00000000: 07a1 9628 0303 87bd 2602 0cf8 e12d 24fe  ...(....&....-$.
00000010: 0aa6 d706 6bfa 1e1c 082c dc2b 450d bf03  ....k....,.+E...

Note: The ciphertexts for different IVs should differ.
=======================================
```



```
==============================
Encryption with the Same IV
==============================
Processing mode: cbc
Using same IV (0102030405060708):
hex string is too short, padding with zero bytes to length
Encrypted file (encrypted_cbc_0102030405060708.bin):
-rw-rw-r-- 1 qasim qasim 32 Sep 13 12:08 encrypted_cbc_0102030405060708.bin
Bytes of the encrypted file:
00000000: c59b a0c5 300e e0f3 9890 a048 e8a5 c992  ....0......H....
00000010: 7d81 5184 81eb d7ce 5fe5 9775 8385 9dec  }.Q....._..u....

Note: The ciphertexts for the same IV should be identical.
=======================================
```

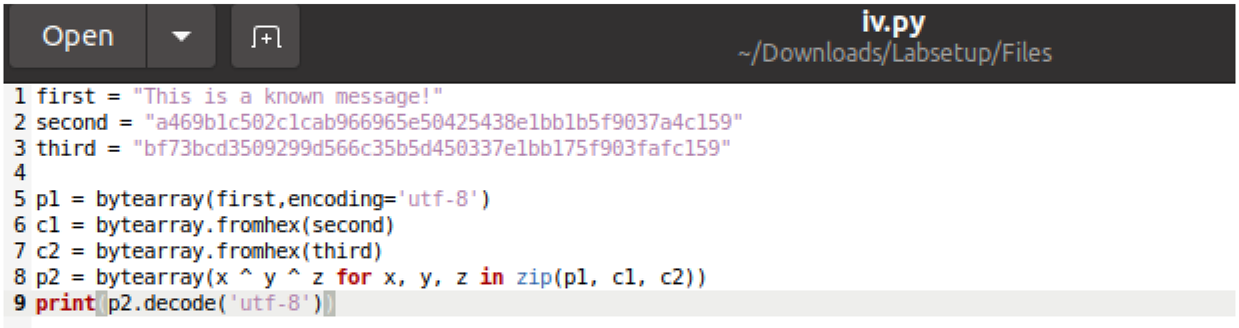
Information Security LAB 1

NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J



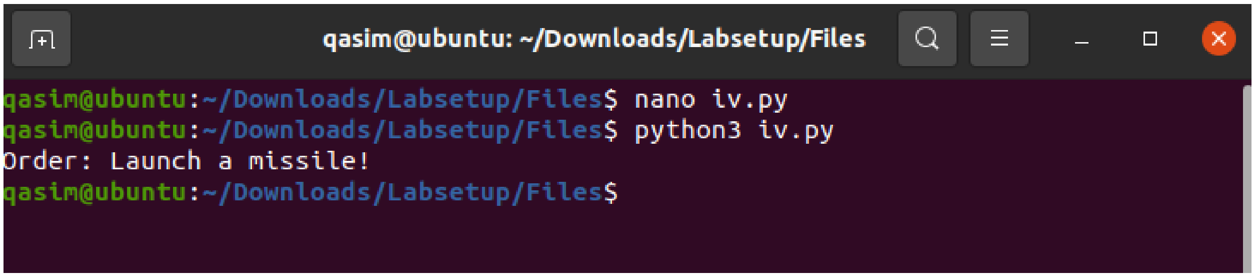
```
#!/bin/bash

# Variables
INPUT_FILE="plaintext.txt"
KEY="00112233445566778899aabbccddeeff"
IV1="0102030405060708"  # First IV
IV2="0807060504030201"  # Second IV

# Encryption function
encrypt() {
    local mode=$1
    local iv=$2
    local encrypted_file="encrypted_${mode}_${iv}.bin"

    openssl enc -aes-128-$mode -e -in $INPUT_FILE -out $encrypted_file -K $KEY -iv $iv
    if [ $? -ne 0 ]; then
        echo "Encryption failed for mode $mode with IV $iv!"
        exit 1
    fi

    echo "Encrypted file ($encrypted_file):"
    ls -lh $encrypted_file
    echo "Bytes of the encrypted file:"
    xxd $encrypted_file
    echo
}

# Encrypt with different IVs
echo "=============================="
echo "Encryption with Different IVs"
echo "=============================="

for mode in cbc cfb ofb; do
    echo "Processing mode: $mode"
    echo "Using IV1 ($IV1):"
    encrypt $mode $IV1

    echo "Using IV2 ($IV2):"
    encrypt $mode $IV2

    echo "Note: The ciphertexts for different IVs should differ."
    echo "========================================"
    echo
done

# Encrypt with the same IV
echo "=============================="
echo "Encryption with the Same IV"
echo "=============================="

for mode in cbc cfb ofb; do
    echo "Processing mode: $mode"
    echo "Using same IV ($IV1):"
    encrypt $mode $IV1

    echo "Note: The ciphertexts for the same IV should be identical."
    echo "========================================"
    echo
done

```

Task 6.2:



```python
first = "This is a known message!"
second = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
third = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

p1 = bytearray(first,encoding='utf-8')
c1 = bytearray.fromhex(second)
c2 = bytearray.fromhex(third)
p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, c1, c2))
print(p2.decode('utf-8'))
```

RESULT:



```
qasim@ubuntu:~/Downloads/Labsetup/Files$ nano iv.py
qasim@ubuntu:~/Downloads/Labsetup/Files$ python3 iv.py
Order: Launch a missile!
qasim@ubuntu:~/Downloads/Labsetup/Files$
```

Analysis of OFB and CFB Modes with Known-Plaintext Attack

OFB Mode (Output Feedback Mode):

- **How It Works:** Encrypts an IV to generate a keystream. This keystream is XORed with plaintext to produce ciphertext.
- **Known-Plaintext Attack:** With known plaintext (P1) and its ciphertext (C1), XOR to find the keystream. Use this keystream to decrypt other ciphertexts (C2) by XORing it with C2.
- **Summary:** Yes, you can decrypt P2 completely using C1, P1, and C2.

CFB Mode (Cipher Feedback Mode):

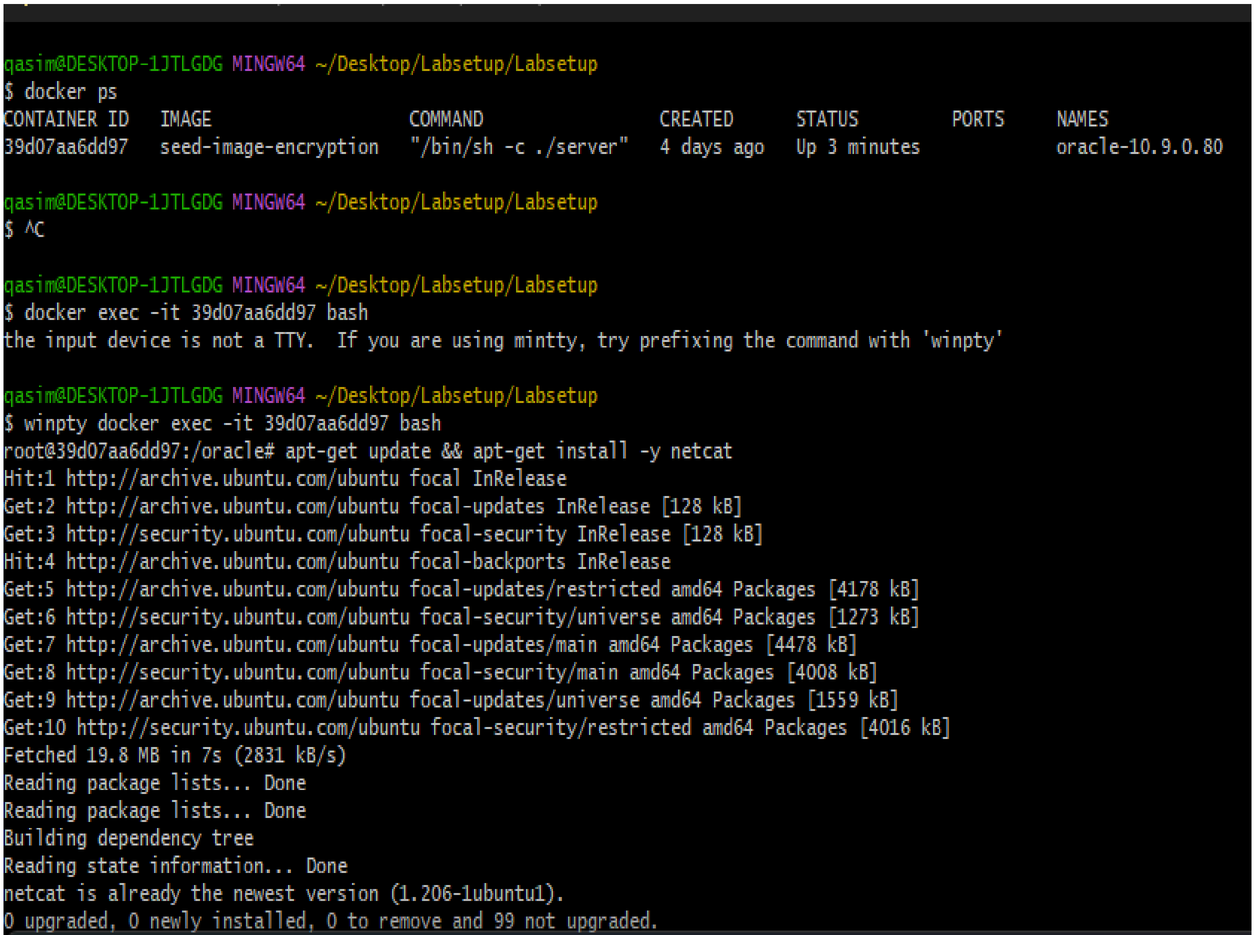
- **How It Works:** Uses IV to initialize the cipher and feedbacks the output for the next block, creating a self-synchronizing stream cipher.
- **Known-Plaintext Attack:** Reusing the same IV does not produce the same keystream due to the feedback mechanism, complicating the derivation of the keystream from known plaintext-ciphertext pairs.
- **Summary:** Decrypting P2 is not straightforward and may reveal only parts of P2.

Information Security LAB 1

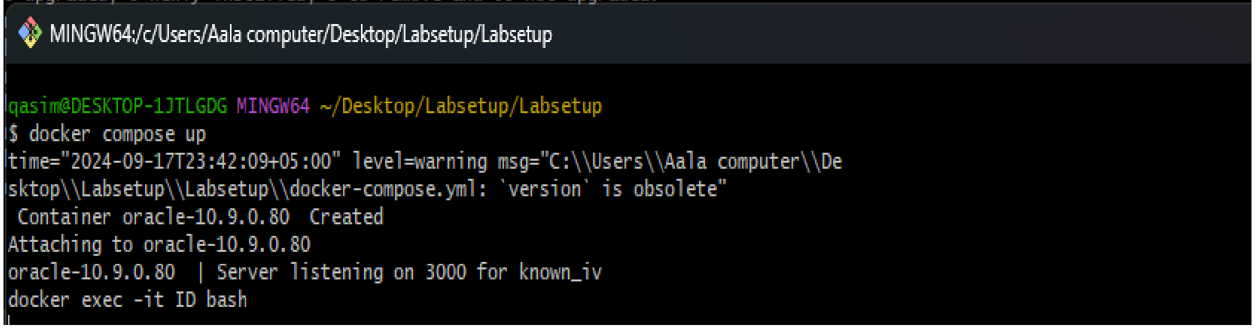
NAME: QASIM HASAN ROLL NO: 21K-3210 SECTION: BCS-7J

Task 6.3: DOCKER TASK

Making the Docker container run correctly.

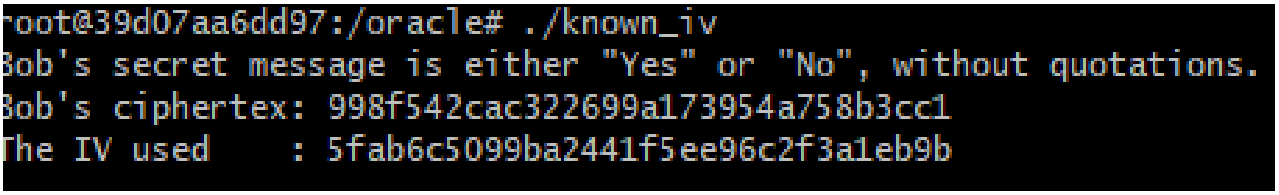


```
qasim@DESKTOP-1JTLGDG MINGW64 ~/Desktop/Labsetup/Labsetup
$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED      STATUS         PORTS     NAMES
39d07aa6dd97   seed-image-encryption   "/bin/sh -c ./server"    4 days ago   Up 3 minutes             oracle-10.9.0.80

qasim@DESKTOP-1JTLGDG MINGW64 ~/Desktop/Labsetup/Labsetup
$ ^C

qasim@DESKTOP-1JTLGDG MINGW64 ~/Desktop/Labsetup/Labsetup
$ docker exec -it 39d07aa6dd97 bash
the input device is not a TTY.  If you are using mintty, try prefixing the command with 'winpty'

qasim@DESKTOP-1JTLGDG MINGW64 ~/Desktop/Labsetup/Labsetup
$ winpty docker exec -it 39d07aa6dd97 bash
root@39d07aa6dd97:/oracle# apt-get update && apt-get install -y netcat
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Hit:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease
Get:5 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [4178 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1273 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [4478 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [4008 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1559 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [4016 kB]
Fetched 19.8 MB in 7s (2831 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
netcat is already the newest version (1.206-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 99 not upgraded.
```

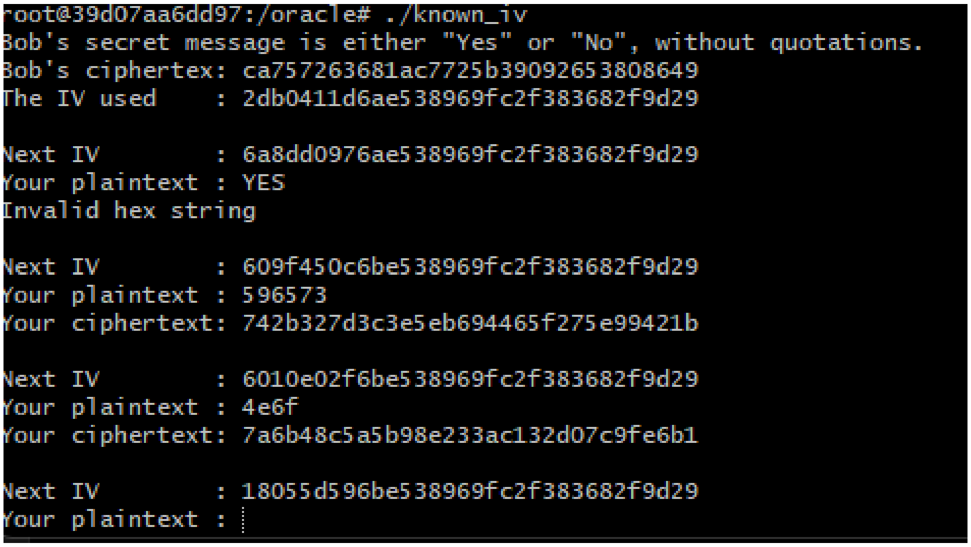


```
qasim@DESKTOP-1JTLGDG MINGW64 ~/Desktop/Labsetup/Labsetup
$ docker compose up
time="2024-09-17T23:42:09+05:00" level=warning msg="C:\\Users\\Aala computer\\De
sktop\\Labsetup\\Labsetup\\docker-compose.yml: `version` is obsolete"
 Container oracle-10.9.0.80  Created
Attaching to oracle-10.9.0.80
oracle-10.9.0.80  | Server listening on 3000 for known_iv
docker exec -it ID bash
```

RESULT:



```
root@39d07aa6dd97:/oracle# ./known_iv
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 998f542cac322699a173954a758b3cc1
The IV used     : 5fab6c5099ba2441f5ee96c2f3a1eb9b
```



```
root@39d07aa6dd97:/oracle# ./known_iv
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: ca757263681ac7725b39092653808649
The IV used     : 2db0411d6ae538969fc2f383682f9d29

Next IV         : 6a8dd0976ae538969fc2f383682f9d29
Your plaintext : YES
Invalid hex string

Next IV         : 609f450c6be538969fc2f383682f9d29
Your plaintext : 596573
Your ciphertext: 742b327d3c3e5eb694465f275e99421b

Next IV         : 6010e02f6be538969fc2f383682f9d29
Your plaintext : 4e6f
Your ciphertext: 7a6b48c5a5b98e233ac132d07c9fe6b1

Next IV         : 18055d596be538969fc2f383682f9d29
Your plaintext :
```

When prompted for **Your plaintext** in the program:

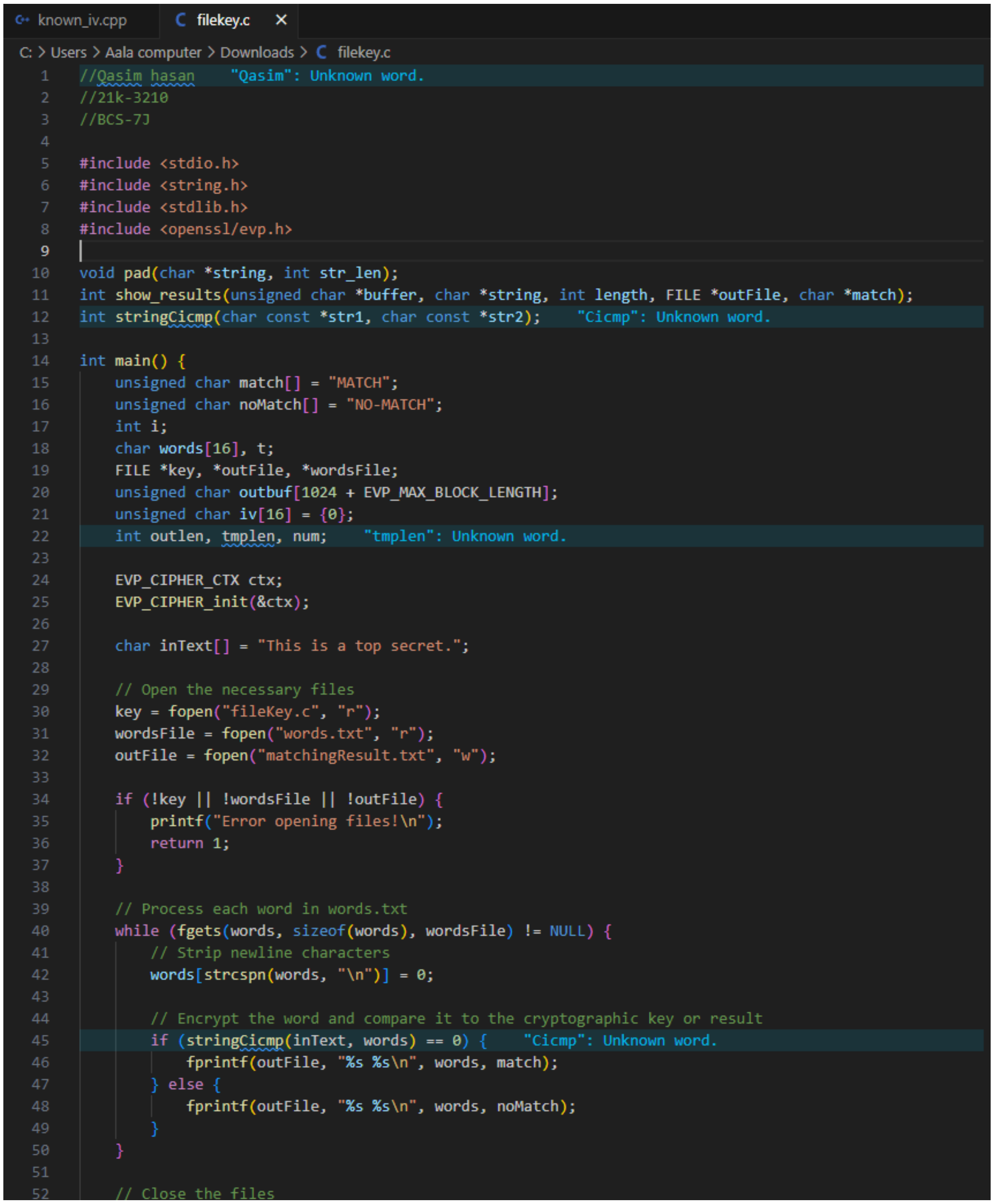
Enter the hex value for "Yes" or "No" based on the guess.

Hexadecimal representation:

1. For **"Yes"**, enter: 596573
2. For **"No"**, enter: 4e6f
3.

This ensures the input is valid and allows the program to process the plaintext correctly without errors. Bob's ciphertext (998f542cac322699a173954a758b3cc1) is **more closely like** the second ciphertext in the first image (7a6b48c5a5b98e233ac132d07c9fe6b1 than the first one. So, based on the similarity, the answer is: **Yes**.

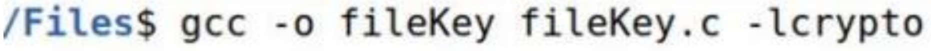
Task 7: Programming using the Crypto Library



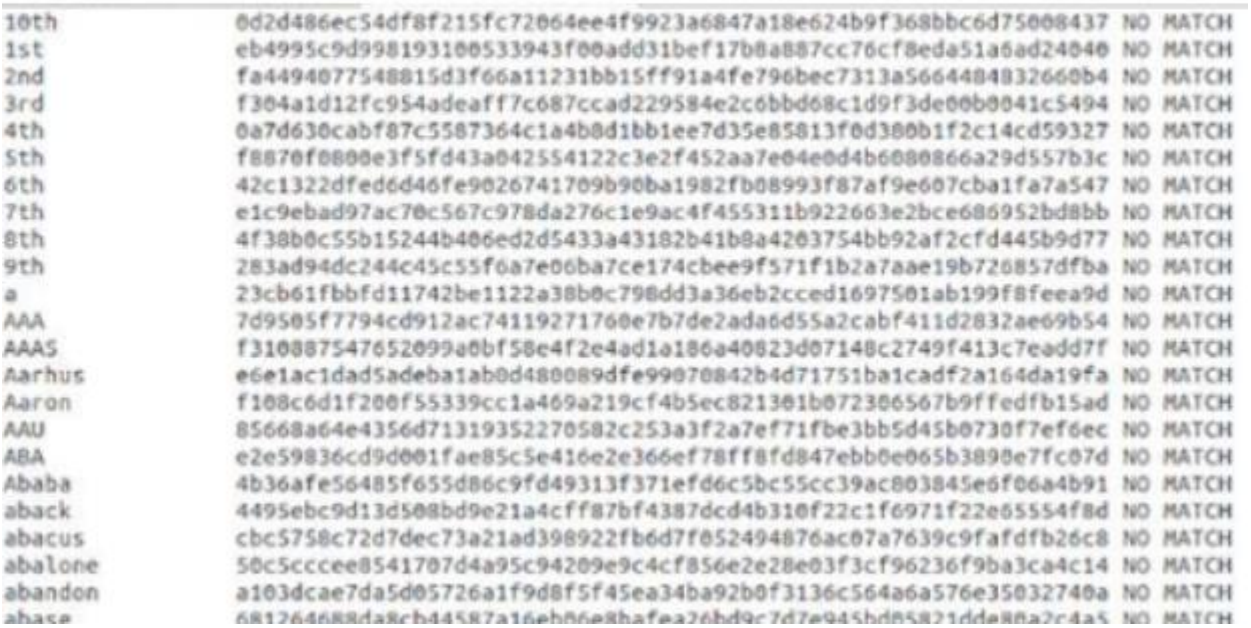
```
//Qasim hasan    "Qasim": Unknown word.
//21k-3210
//BCS-7J

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/evp.h>

void pad(char *string, int str_len);
int show_results(unsigned char *buffer, char *string, int length, FILE *outFile, char *match);
int stringCicmp(char const *str1, char const *str2);    "Cicmp": Unknown word.

int main() {
    unsigned char match[] = "MATCH";
    unsigned char noMatch[] = "NO-MATCH";
    int i;
    char words[16], t;
    FILE *key, *outFile, *wordsFile;
    unsigned char outbuf[1024 + EVP_MAX_BLOCK_LENGTH];
    unsigned char iv[16] = {0};
    int outlen, tmplen, num;    "tmplen": Unknown word.

    EVP_CIPHER_CTX ctx;
    EVP_CIPHER_init(&ctx);

    char inText[] = "This is a top secret.";

    // Open the necessary files
    key = fopen("fileKey.c", "r");
    wordsFile = fopen("words.txt", "r");
    outFile = fopen("matchingResult.txt", "w");

    if (!key || !wordsFile || !outFile) {
        printf("Error opening files!\n");
        return 1;
    }

    // Process each word in words.txt
    while (fgets(words, sizeof(words), wordsFile) != NULL) {
        // Strip newline characters
        words[strcspn(words, "\n")] = 0;

        // Encrypt the word and compare it to the cryptographic key or result
        if (stringCicmp(inText, words) == 0) {    "Cicmp": Unknown word.
            fprintf(outFile, "%s %s\n", words, match);
        } else {
            fprintf(outFile, "%s %s\n", words, noMatch);
        }
    }

    // Close the files
```

To complete a cryptography library task, you need to utilize three files: fileKey.c, matchingResult.txt, and words.txt. First, create the fileKey.c file by using a text editor like gedit. In this file, write the necessary code to compare and match words from the words.txt file, and output the matching results into matchingResult.txt.

RESULT:



```
/Files$ gcc -o fileKey fileKey.c -lcrypto
```



```
10th             0d2d486ec54df8f215fc72064ee4f9923a6847a18e624b9f368bbc6d75008437 NO MATCH
1st              eb4995c9d998193100533943f00add31bef17b8a887cc76cf8eda51a6ad24040 NO MATCH
2nd              fa4494077548815d3f66a11231bb15ff91a4fe796bec7313a5664484832660b4 NO MATCH
3rd              f304a1d12fc954adeaff7c687ccad229584e2c6bbd68c1d9f3de00b0041c5494 NO MATCH
4th              0a7d630cabf87c5587364c1a4b0d1bb1ee7d35e85813f0d380b1f2c14cd59327 NO MATCH
5th              f8870f0800e3f5fd43a042554122c3e2f452aa7e04e0d4b6080866a29d557b3c NO MATCH
6th              42c1322dfed6d46fe9026741709b90ba1982fb08993f87af9e607cba1fa7a547 NO MATCH
7th              e1c9ebad97ac70c567c978da276c1e9ac4f455311b922663e2bce686952bd8bb NO MATCH
8th              4f38b0c55b15244b406ed2d5433a43182b41b8a4203754bb92af2cfd445b9d77 NO MATCH
9th              283ad94dc244c45c55f6a7e06ba7ce174cbee9f571f1b2a7aae19b726857dfba NO MATCH
a                23cb61fbbfd11742be1122a38b0c798dd3a36eb2cced1697501ab199f8feea9d NO MATCH
AAA              7d9505f7794cd912ac74119271760e7b7de2ada6d55a2cabf411d2832ae69b54 NO MATCH
AAAS             f310887547652099a0bf58e4f2e4ad1a186a40823d07148c2749f413c7eadd7f NO MATCH
Aarhus           e6e1ac1dad5adeba1ab0d480089dfe99070842b4d71751ba1cadf2a164da19fa NO MATCH
Aaron            f108c6d1f200f55339cc1a469a219cf4b5ec821301b072306567b9ffedfb15ad NO MATCH
AAU              85668a64e4356d71319352270582c253a3f2a7ef71fbe3bb5d45b0730f7ef6ec NO MATCH
ABA              e2e59836cd9d001fae85c5e416e2e366ef78ff8fd847ebb0e065b3890e7fc07d NO MATCH
Ababa            4b36afe56485f655d86c9fd49313f371efd6c5bc55cc39ac803845e6f06a4b91 NO MATCH
aback            4495ebc9d13d508bd9e21a4cff87bf4387dcd4b310f22c1f6971f22e65554f8d NO MATCH
abacus           cbc5758c72d7dec73a21ad398922fb6d7f052494876ac07a7639c9fafdfb26c8 NO MATCH
abalone          50c5cccee8541707d4a95c94209e9c4cf856e2e28e03f3cf96236f9ba3ca4c14 NO MATCH
abandon          a103dcae7da5d05726a1f9d8f5f45ea34ba92b0f3136c564a6a576e35032740a NO MATCH
abase            681264688da8cb44587a16eb06e8bafea26bd9c7d7e945bd05821dde80a2c4a5 NO MATCH
```