Lab Report 3: Symmetric Encryption and Hashing

Setup

The OpenSSL tool was used to perform encryption and hashing operations.
It was verified that OpenSSL was installed in the Ubuntu environment.
Additionally, the GHex hex editor was installed for file-level editing and bit-level inspection.

```
# Update package lists
sudo apt update

# Install OpenSSL
sudo apt install openssl -y

# Verify OpenSSL installation
openssl version

# Install GHex hex editor
sudo apt install ghex -y

# Launch GHex (GUI)
ghex &
```

After setup, both tools were tested to ensure they worked correctly before proceeding with the lab tasks.

Task 1: AES Encryption using Different Modes (2 Marks)

Approach

A simple text file containing a few sentences was used as the input message.
The goal was to encrypt the same file using AES-128 encryption in CBC, CFB, and OFB modes, and then decrypt each one to verify that the original message was successfully recovered.

Procedure

A plain text file was created containing a short message as:

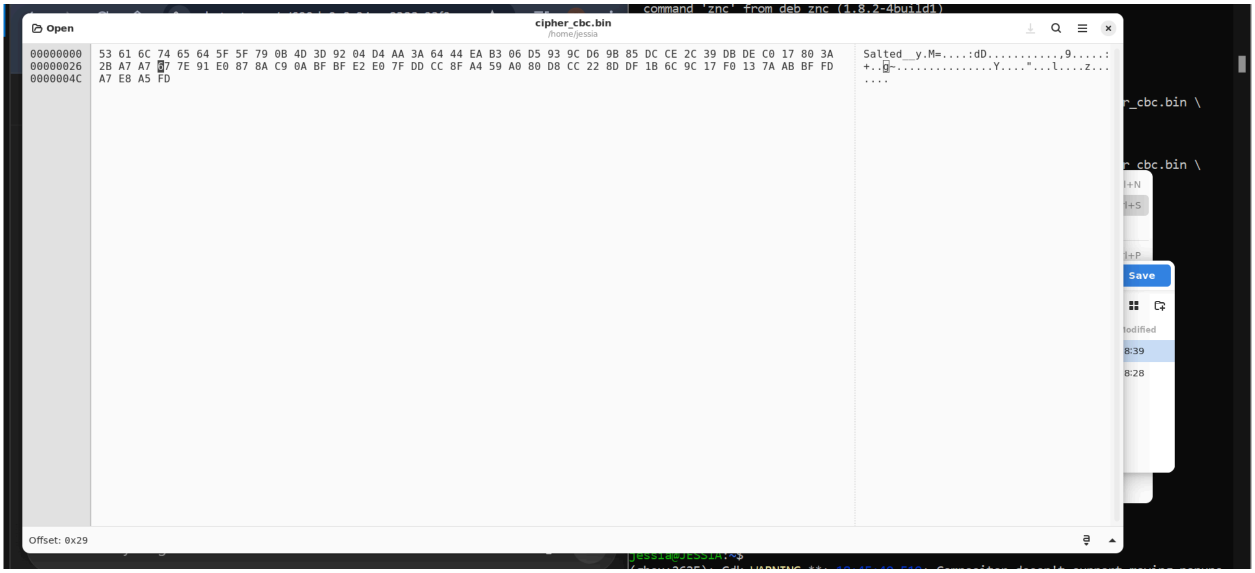
> “This is my symmetric encryption lab test file.”

Using OpenSSL, the file was encrypted three times — once in CBC mode, once in CFB mode, and once in ECB mode.
The resulting encrypted files were saved separately.

Each of these encrypted files was then decrypted again to confirm that the process worked correctly and that the decrypted output matched the original text file.

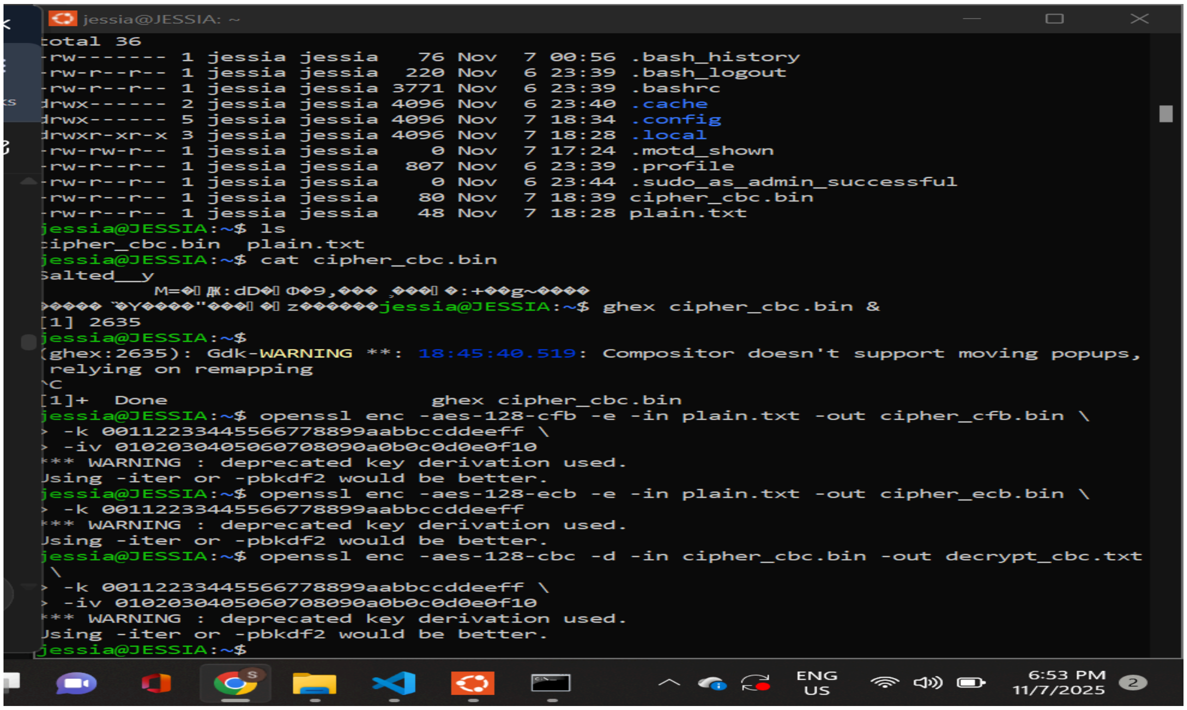
Observation

Decryption produced the original text successfully for all three modes, confirming the correctness of the encryption-decryption process.

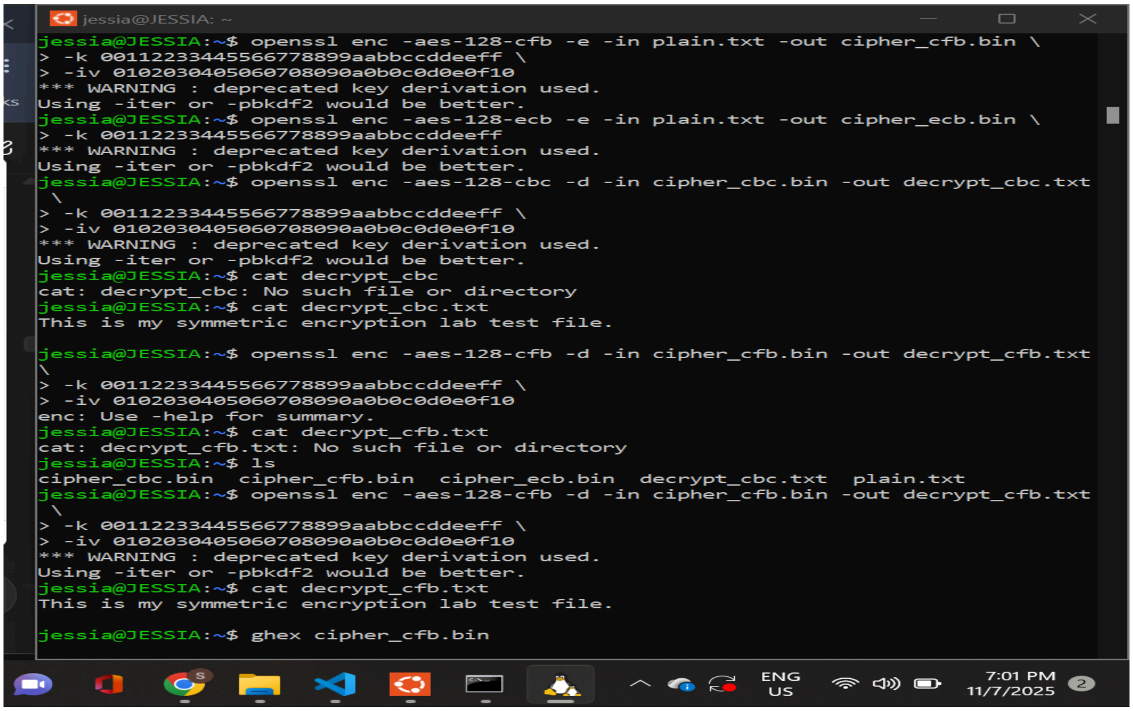


```
command 'znc' from deb znc (1.8.2-4build1)
```

```
00000000  53 61 6C 74 65 64 5F 5F 79 0B 4D 3D 92 04 D4 AA 3A 64 44 EA B3 06 D5 93 9C D6 9B 85 DC CE 2C 39 DB DE C0 17 80 3A   Salted__y.M=....:dD...........,9.....:
00000026  2B A7 A7 67 7E 91 E0 87 8A C9 0A BF BF E2 E0 7F DD CC 8F A4 59 A0 80 D8 CC 22 8D DF 1B 6C 9C 17 F0 13 7A AB BF FD   +..g~..............Y...."...l....z...
0000004C  A7 E8 A5 FD                                                                                                         ....
```



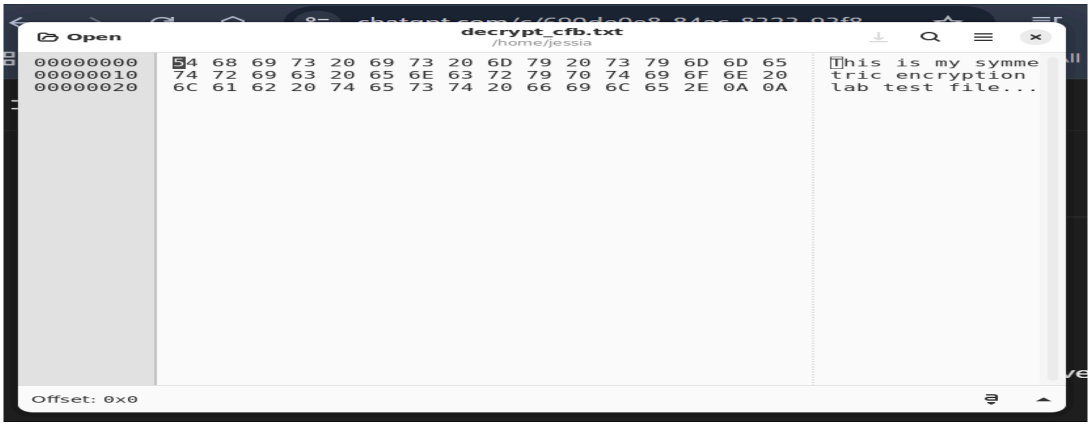
```
total 36
-rw------- 1 jessia jessia   76 Nov  7 00:56 .bash_history
-rw-r--r-- 1 jessia jessia  220 Nov  6 23:39 .bash_logout
-rw-r--r-- 1 jessia jessia 3771 Nov  6 23:39 .bashrc
drwx------ 2 jessia jessia 4096 Nov  6 23:40 .cache
drwx------ 5 jessia jessia 4096 Nov  7 18:34 .config
drwxr-xr-x 3 jessia jessia 4096 Nov  7 18:28 .local
-rw-rw-r-- 1 jessia jessia    0 Nov  7 17:24 .motd_shown
-rw-r--r-- 1 jessia jessia  807 Nov  6 23:39 .profile
-rw-r--r-- 1 jessia jessia    0 Nov  6 23:44 .sudo_as_admin_successful
-rw-r--r-- 1 jessia jessia   80 Nov  7 18:39 cipher_cbc.bin
-rw-r--r-- 1 jessia jessia   48 Nov  7 18:28 plain.txt
jessia@JESSIA:~$ ls
cipher_cbc.bin  plain.txt
jessia@JESSIA:~$ cat cipher_cbc.bin
Salted__y
         M=�▯ ДK:dD�▯ Φ�9,��� ¸���▯ �:+��g~����
����� `�Y����"���▯ �▯ z������jessia@JESSIA:~$ ghex cipher_cbc.bin &
[1] 2635
jessia@JESSIA:~$
(ghex:2635): Gdk-WARNING **: 18:45:40.519: Compositor doesn't support moving popups,
 relying on remapping
^C
[1]+  Done                    ghex cipher_cbc.bin
jessia@JESSIA:~$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher_cfb.bin \
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher_ecb.bin \
> -k 00112233445566778899aabbccddeeff
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-cbc -d -in cipher_cbc.bin -out decrypt_cbc.txt
 \
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$
```



```
jessia@JESSIA:~$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher_cfb.bin \
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher_ecb.bin \
> -k 00112233445566778899aabbccddeeff
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-cbc -d -in cipher_cbc.bin -out decrypt_cbc.txt
 \
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ cat decrypt_cbc
cat: decrypt_cbc: No such file or directory
jessia@JESSIA:~$ cat decrypt_cbc.txt
This is my symmetric encryption lab test file.

jessia@JESSIA:~$ openssl enc -aes-128-cfb -d -in cipher_cfb.bin -out decrypt_cfb.txt
\
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
enc: Use -help for summary.
jessia@JESSIA:~$ cat decrypt_cfb.txt
cat: decrypt_cfb.txt: No such file or directory
jessia@JESSIA:~$ ls
cipher_cbc.bin  cipher_cfb.bin  cipher_ecb.bin  decrypt_cbc.txt  plain.txt
jessia@JESSIA:~$ openssl enc -aes-128-cfb -d -in cipher_cfb.bin -out decrypt_cfb.txt
 \
> -k 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ cat decrypt_cfb.txt
This is my symmetric encryption lab test file.

jessia@JESSIA:~$ ghex cipher_cfb.bin
```



```
00000000  54 68 69 73 20 69 73 20 6D 79 20 73 79 6D 6D 65   This is my symme
00000010  74 72 69 63 20 65 6E 63 72 79 70 74 69 6F 6E 20   tric encryption 
00000020  6C 61 62 20 74 65 73 74 20 66 69 6C 65 2E 0A 0A   lab test file...
```

Task 2 – Comparing ECB and CBC Encryption Modes (3 Marks)

Objective

This task aimed to visually demonstrate the difference between the **ECB** and **CBC** modes of AES encryption when applied to an image file.

Procedure

A bitmap image named pic_original.bmp was used for this experiment.
First, the image was copied into the working directory.
Then, the image was encrypted twice — once using **AES-128 in ECB mode**, and again using **AES-128 in CBC mode**, both with the same key.

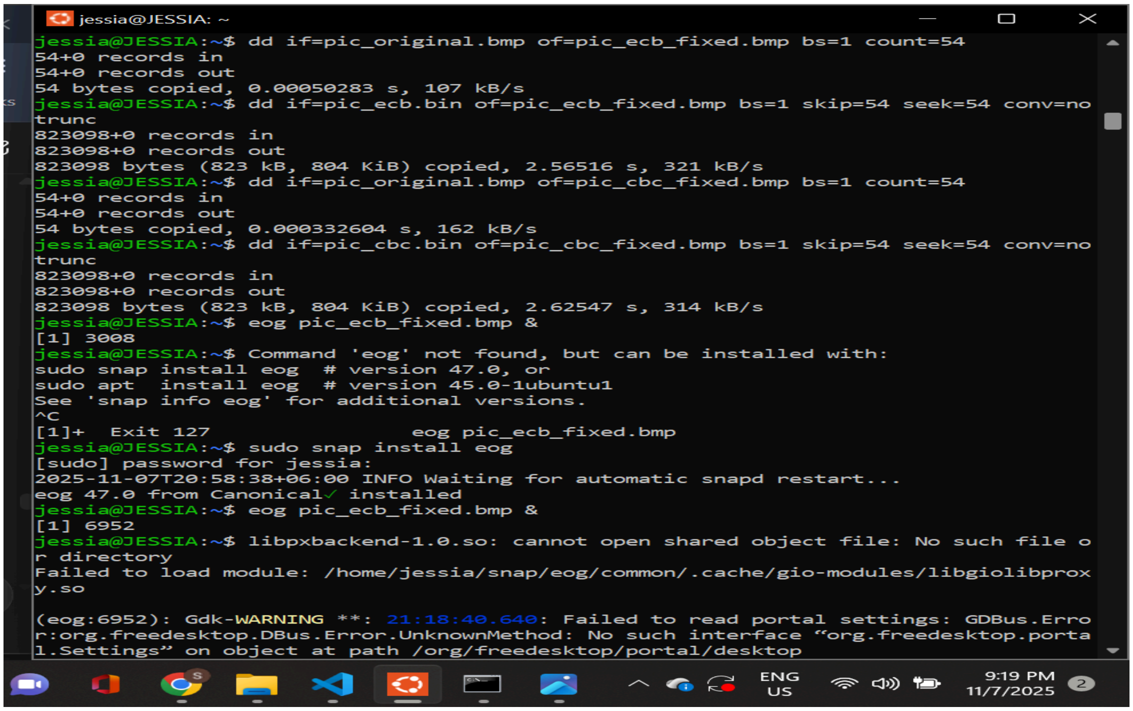
After encryption, the first 54 bytes (the BMP header) from the original image were copied and placed at the beginning of each encrypted file using GHex.
This was necessary to make the encrypted files viewable as images.

Finally, both encrypted images were opened using an image viewer.

Observation

- In **ECB mode**, the general structure and outlines of the original image were still visible, showing block-like patterns.

- In **CBC mode**, the image appeared completely random and unrecognizable, providing much stronger visual security.



```
jessia@JESSIA:~$ dd if=pic_original.bmp of=pic_ecb_fixed.bmp bs=1 count=54
54+0 records in
54+0 records out
54 bytes copied, 0.00050283 s, 107 kB/s
jessia@JESSIA:~$ dd if=pic_ecb.bin of=pic_ecb_fixed.bmp bs=1 skip=54 seek=54 conv=no
trunc
823098+0 records in
823098+0 records out
823098 bytes (823 kB, 804 KiB) copied, 2.56516 s, 321 kB/s
jessia@JESSIA:~$ dd if=pic_original.bmp of=pic_cbc_fixed.bmp bs=1 count=54
54+0 records in
54+0 records out
54 bytes copied, 0.000332604 s, 162 kB/s
jessia@JESSIA:~$ dd if=pic_cbc.bin of=pic_cbc_fixed.bmp bs=1 skip=54 seek=54 conv=no
trunc
823098+0 records in
823098+0 records out
823098 bytes (823 kB, 804 KiB) copied, 2.62547 s, 314 kB/s
jessia@JESSIA:~$ eog pic_ecb_fixed.bmp &
[1] 3008
jessia@JESSIA:~$ Command 'eog' not found, but can be installed with:
sudo snap install eog  # version 47.0, or
sudo apt  install eog  # version 45.0-1ubuntu1
See 'snap info eog' for additional versions.
^C
[1]+  Exit 127                eog pic_ecb_fixed.bmp
jessia@JESSIA:~$ sudo snap install eog
[sudo] password for jessia:
2025-11-07T20:58:38+06:00 INFO Waiting for automatic snapd restart...
eog 47.0 from Canonical✓ installed
jessia@JESSIA:~$ eog pic_ecb_fixed.bmp &
[1] 6952
jessia@JESSIA:~$ libpxbackend-1.0.so: cannot open shared object file: No such file o
r directory
Failed to load module: /home/jessia/snap/eog/common/.cache/gio-modules/libgiolibprox
y.so

(eog:6952): Gdk-WARNING **: 21:18:40.640: Failed to read portal settings: GDBus.Erro
r:org.freedesktop.DBus.Error.UnknownMethod: No such interface “org.freedesktop.porta
l.Settings” on object at path /org/freedesktop/portal/desktop
```

Task 3: Effect of Ciphertext Corruption (3 Marks)

Objective

To analyze how a single-bit corruption in the ciphertext affects the decrypted output in different AES modes.

Procedure

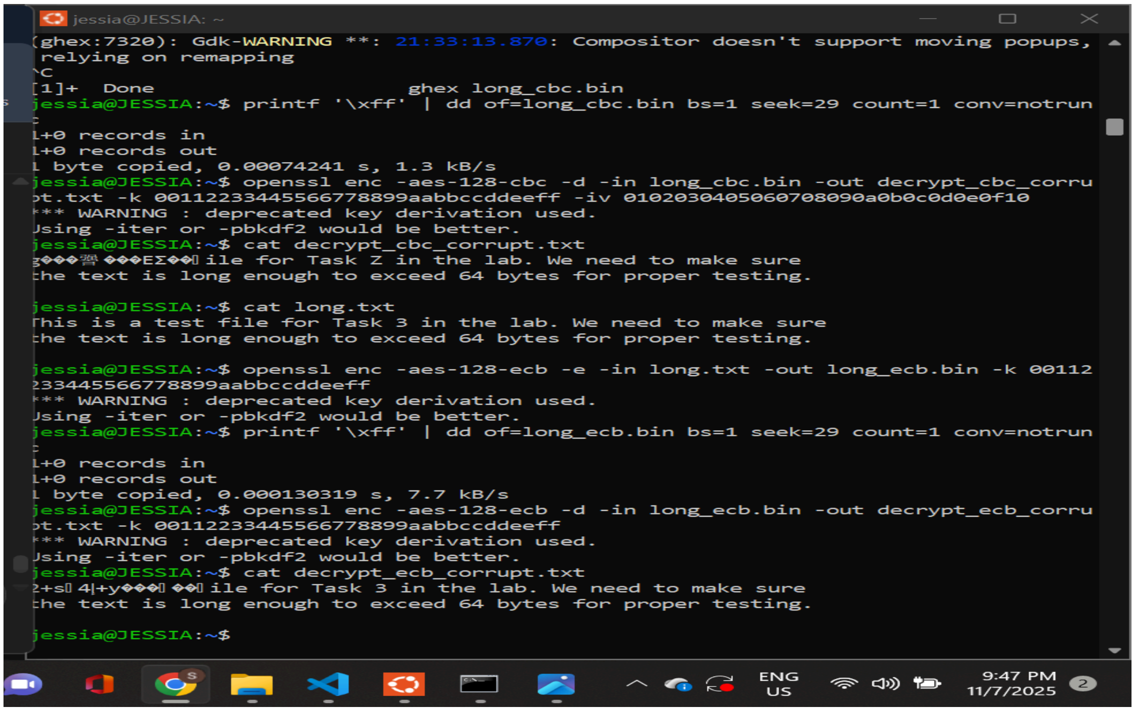
A text file longer than 64 bytes was created and then encrypted using AES-128 in CBC mode.
Next, the encrypted binary file was opened in GHex, and a single bit within the ciphertext was manually flipped to simulate corruption.
The corrupted ciphertext was then decrypted using the same key and initialization vector.

Observation

The corruption had different effects depending on the encryption mode used:

Mode	Effect of Corruption
ECB	Only the block containing the corrupted bit was affected.
CBC	Both the corrupted block and the following block were affected.
CFB	The error spread over a small region of text.
OFB	Only the single corrupted bit was affected in the plaintext.

This experiment demonstrated how error propagation varies among the encryption modes.

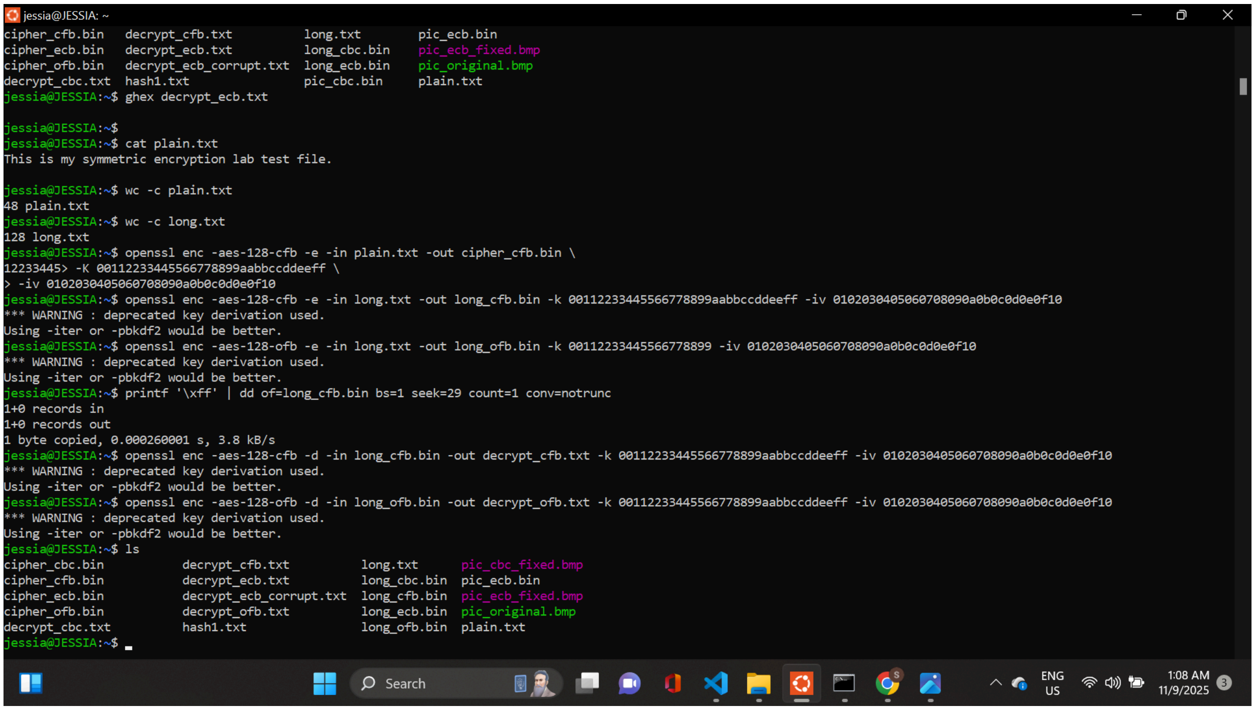


```
(ghex:7320): Gdk-WARNING **: 21:33:13.870: Compositor doesn't support moving popups,
 relying on remapping
^C
[1]+  Done                    ghex long_cbc.bin
jessia@JESSIA:~$ printf '\xff' | dd of=long_cbc.bin bs=1 seek=29 count=1 conv=notrun
c
1+0 records in
1+0 records out
1 byte copied, 0.00074241 s, 1.3 kB/s
jessia@JESSIA:~$ openssl enc -aes-128-cbc -d -in long_cbc.bin -out decrypt_cbc_corru
pt.txt -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ cat decrypt_cbc_corrupt.txt
g���嚳 ���EΣ��ile for Task Z in the lab. We need to make sure
the text is long enough to exceed 64 bytes for proper testing.

jessia@JESSIA:~$ cat long.txt
This is a test file for Task 3 in the lab. We need to make sure
the text is long enough to exceed 64 bytes for proper testing.

jessia@JESSIA:~$ openssl enc -aes-128-ecb -e -in long.txt -out long_ecb.bin -k 00112
233445566778899aabbccddeeff
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ printf '\xff' | dd of=long_ecb.bin bs=1 seek=29 count=1 conv=notrun
c
1+0 records in
1+0 records out
1 byte copied, 0.000130319 s, 7.7 kB/s
jessia@JESSIA:~$ openssl enc -aes-128-ecb -d -in long_ecb.bin -out decrypt_ecb_corru
pt.txt -k 00112233445566778899aabbccddeeff
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ cat decrypt_ecb_corrupt.txt
2+s 4|+y��� ��ile for Task 3 in the lab. We need to make sure
the text is long enough to exceed 64 bytes for proper testing.

jessia@JESSIA:~$
```



```
jessia@JESSIA: ~
cipher_cfb.bin   decrypt_cfb.txt          long.txt        pic_ecb.bin
cipher_ecb.bin   decrypt_ecb.txt          long_cbc.bin    pic_ecb_fixed.bmp
cipher_ofb.bin   decrypt_ecb_corrupt.txt  long_ecb.bin    pic_original.bmp
decrypt_cbc.txt  hash1.txt                pic_cbc.bin     plain.txt
jessia@JESSIA:~$ ghex decrypt_ecb.txt

jessia@JESSIA:~$
jessia@JESSIA:~$ cat plain.txt
This is my symmetric encryption lab test file.

jessia@JESSIA:~$ wc -c plain.txt
48 plain.txt
jessia@JESSIA:~$ wc -c long.txt
128 long.txt
jessia@JESSIA:~$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher_cfb.bin \
12233445> -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10
jessia@JESSIA:~$ openssl enc -aes-128-cfb -e -in long.txt -out long_cfb.bin -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-ofb -e -in long.txt -out long_ofb.bin -k 00112233445566778899 -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ printf '\xff' | dd of=long_cfb.bin bs=1 seek=29 count=1 conv=notrunc
1+0 records in
1+0 records out
1 byte copied, 0.000260001 s, 3.8 kB/s
jessia@JESSIA:~$ openssl enc -aes-128-cfb -d -in long_cfb.bin -out decrypt_cfb.txt -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ openssl enc -aes-128-ofb -d -in long_ofb.bin -out decrypt_ofb.txt -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
jessia@JESSIA:~$ ls
cipher_cbc.bin          decrypt_cfb.txt          long.txt      pic_cbc_fixed.bmp
cipher_cfb.bin          decrypt_ecb.txt          long_cbc.bin  pic_ecb.bin
cipher_ecb.bin          decrypt_ecb_corrupt.txt  long_cfb.bin  pic_ecb_fixed.bmp
cipher_ofb.bin          decrypt_ofb.txt          long_ecb.bin  pic_original.bmp
decrypt_cbc.txt         hash1.txt                long_ofb.bin  plain.txt
jessia@JESSIA:~$ _
```

Task 4 – Padding Analysis (3 Marks)

Approach

The goal was to determine which AES modes require padding when the plaintext length is not a multiple of the block size.

Procedure

A short text file was encrypted using AES in different modes (ECB, CBC, CFB, and OFB).
When encrypting with ECB and CBC modes, OpenSSL automatically added padding to complete the final block.
For CFB and OFB modes, no padding was needed because these modes operate as stream ciphers.

Observation

Mode	Padding Required	Reason
ECB	Yes	Works on fixed-size blocks
CBC	Yes	Works on fixed-size blocks

CFB	No	Stream-based mode
OFB	No	Stream-based mode

Task 5 – Message Digest Generation (3 Marks)

Approach

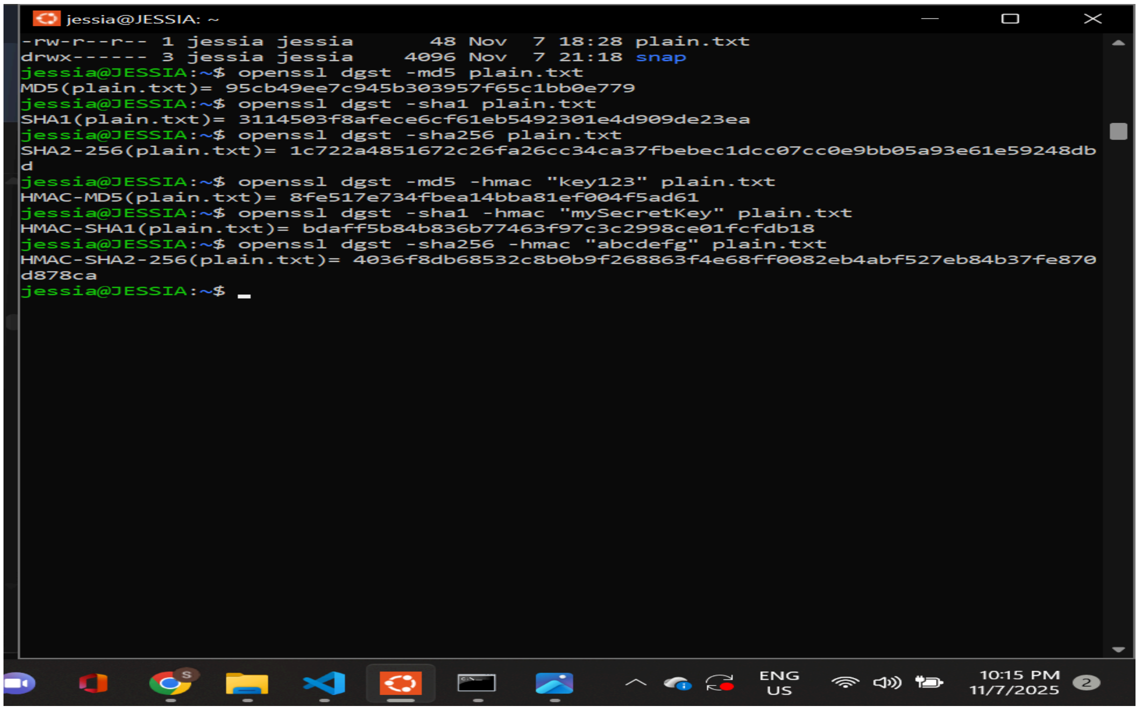
This task demonstrated how message digests (hashes) are generated using OpenSSL with different hashing algorithms.

Procedure

A text file was selected and its hash was generated using three algorithms: MD5, SHA-1, and SHA-256.
Each produced a fixed-length output (digest) unique to the file’s contents.

Observation

Algorithm	Digest Size	Notes
MD5	128 bits	Older algorithm, not secure for modern use
SHA-1	160 bits	Somewhat outdated
SHA-256	256 bits	Secure and widely used today



```
-rw-r--r-- 1 jessia jessia      48 Nov  7 18:28 plain.txt
drwx------ 3 jessia jessia    4096 Nov  7 21:18 snap
jessia@JESSIA:~$ openssl dgst -md5 plain.txt
MD5(plain.txt)= 95cb49ee7c945b303957f65c1bb0e779
jessia@JESSIA:~$ openssl dgst -sha1 plain.txt
SHA1(plain.txt)= 3114503f8afece6cf61eb5492301e4d909de23ea
jessia@JESSIA:~$ openssl dgst -sha256 plain.txt
SHA2-256(plain.txt)= 1c722a4851672c26fa26cc34ca37fbebec1dcc07cc0e9bb05a93e61e59248db
d
jessia@JESSIA:~$ openssl dgst -md5 -hmac "key123" plain.txt
HMAC-MD5(plain.txt)= 8fe517e734fbea14bba81ef004f5ad61
jessia@JESSIA:~$ openssl dgst -sha1 -hmac "mySecretKey" plain.txt
HMAC-SHA1(plain.txt)= bdaff5b84b836b77463f97c3c2998ce01fcfdb18
jessia@JESSIA:~$ openssl dgst -sha256 -hmac "abcdefg" plain.txt
HMAC-SHA2-256(plain.txt)= 4036f8db68532c8b0b9f268863f4e68ff0082eb4abf527eb84b37fe870
d878ca
jessia@JESSIA:~$ _
```

Task 6 – Keyed Hash and HMAC (3 Marks)

Approach

This part involved generating keyed message digests (HMACs) to ensure both data integrity and authenticity.

Procedure

The same text file was used to generate HMAC values using different keys and hashing algorithms (MD5, SHA-1, and SHA-256).
Each key produced a completely different HMAC value, even when the same file was used.

Observation

- HMAC depends on both the data and the key, so two identical files with different keys will never produce the same result.

- Changing even one character in the file or key produces a completely different HMAC value.

- HMACs are useful for verifying that a file has not been modified and that it came from a trusted source.

Task 7 – Randomness and Avalanche Effect (Bonus: 2 Marks)

Approach

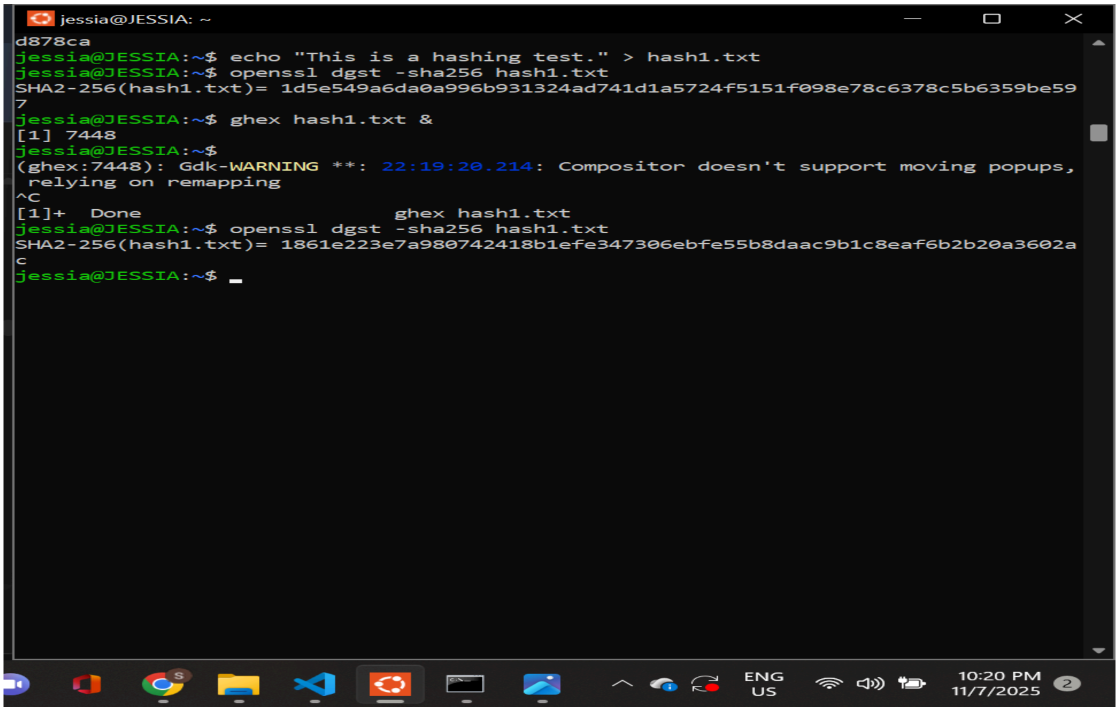
This task demonstrated how a small change in a file leads to a completely different hash output.

Procedure

A text file was hashed using SHA-256.
Then, a single bit of the file was modified using GHex, and the hash was calculated again.
The two hash outputs were compared.

Observation

The two hashes were completely different, even though the file differed by only one bit.
This demonstrated the **avalanche effect**, which is a key property of cryptographic hash functions.



```
d878ca
jessia@JESSIA:~$ echo "This is a hashing test." > hash1.txt
jessia@JESSIA:~$ openssl dgst -sha256 hash1.txt
SHA2-256(hash1.txt)= 1d5e549a6da0a996b931324ad741d1a5724f5151f098e78c6378c5b6359be59
7
jessia@JESSIA:~$ ghex hash1.txt &
[1] 7448
jessia@JESSIA:~$
(ghex:7448): Gdk-WARNING **: 22:19:20.214: Compositor doesn't support moving popups,
 relying on remapping
^C
[1]+  Done                    ghex hash1.txt
jessia@JESSIA:~$ openssl dgst -sha256 hash1.txt
SHA2-256(hash1.txt)= 1861e223e7a980742418b1efe347306ebfe55b8daac9b1c8eaf6b2b20a3602a
c
jessia@JESSIA:~$ _
```