Lab 3: Symmetric encryption & hashing

Task 1: AES Encryption Using Different Modes

Plain text:

```
INS Lab 3 - Task 1
This is a sample text for testing purposes.
Tajwar
```

We used the `openssl enc` command with different AES cipher modes.

1. AES-128-CBC

```
openssl enc -aes-128-cbc -e -in plain.txt -out cipher-cbc.bin \
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

2. AES-128-CFB

```
openssl enc -aes-128-cfb -e -in plain.txt -out cipher-cfb.bin \
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

3. AES-128-ECB

```
openssl enc -aes-128-ecb -e -in plain.txt -out cipher-ecb.bin \
-K 00112233445566778899aabbccddeeff
```

For decryption:

Example (for CBC):

```
openssl enc -aes-128-cbc -d -in cipher-cbc.bin -out decrypted-cbc.txt
\
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

The decrypted text matched the original content, confirming correct encryption and decryption.

Step 4: Observations

Mode	Description	Observation
CBC	Cipher Block Chaining	Each block depends on the previous one; identical plaintext blocks encrypt differently.

CFB	Cipher Feedback	Works like a stream cipher; small changes in IV cause large output differences.
ECB	Electronic Codebook	Same plaintext blocks produce identical ciphertext blocks — less secure.

Task 2: Encryption Mode – ECB vs CBC

The file **`weather.bmp`** (a simple image) was used as the input for encryption.

The image was encrypted using AES-128 with ECB and CBC modes.

1. AES-128-ECB

```
openssl enc -aes-128-ecb -e -in weather.bmp -out weather_ecb.bmp \
-K 00112233445566778899aabbccddeeff
```

2. AES-128-CBC

```
openssl enc -aes-128-cbc -e -in weather.bmp -out weather_cbc.bmp \
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

Since BMP images have a 54-byte header, the header from the original image was copied into the encrypted images using **ghex**:

1. Open both `weather.bmp` and `weather_ecb.bmp` in **ghex**.
2. Copy the first **54 bytes** (header) from the original image.
3. Paste/overwrite these bytes in the encrypted file.
4. Save the changes.
5. Repeat for `weather_cbc.bmp`.

When viewing the images using any viewer (`eog weather_ecb.bmp` and `eog weather_cbc.bmp`):

Mode	Observation
ECB	The encrypted image still shows visible patterns of the original (the shapes or outlines can still be recognized).

CBC The image appears completely random; no visual information about the original image can be derived.

Commands

```
# AES-128-ECB
openssl enc -aes-128-ecb -e -in weather.bmp -out weather_ecb.bmp -K
00112233445566778899aabbccddeeff

# AES-128-CBC
openssl enc -aes-128-cbc -e -in weather.bmp -out weather_cbc.bmp -K
00112233445566778899aabbccddeeff -iv 0102030405060708
```

Task 3 : Effect of Corruption on AES Encryption Modes

1. Commands

AES-128 Key & IV

- Key: `00112233445566778899aabbccddeeff`
- IV: `0102030405060708090a0b0c0d0e0f10`

Encryption

```
# ECB
openssl enc -aes-128-ecb -e -in sample.txt -out sample_ecb.bin -K
00112233445566778899aabbccddeeff

# CBC
openssl enc -aes-128-cbc -e -in sample.txt -out sample_cbc.bin -K
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

# CFB
openssl enc -aes-128-cfb -e -in sample.txt -out sample_cfb.bin -K
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

# OFB
openssl enc -aes-128-ofb -e -in sample.txt -out sample_ofb.bin -K
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

Corrupting 30th Byte

```
# ECB example
xxd -p sample_ecb.bin | tr -d '\n' > sample_ecb.hex
perl -pe 's/(..){29}e2/${1}e3/' sample_ecb.hex >
sample_ecb_corrupted.hex
xxd -r -p sample_ecb_corrupted.hex > sample_ecb_corrupted.bin

# (Repeat similarly for CBC, CFB, OFB)
Or direct change to hex file
```

Decryption

```
# ECB
openssl enc -aes-128-ecb -d -in sample_ecb_corrupted.bin -out
sample_ecb_decrypted.txt -K 00112233445566778899aabbccddeeff

# CBC
openssl enc -aes-128-cbc -d -in sample_cbc_corrupted.bin -out
sample_cbc_decrypted.txt -K 00112233445566778899aabbccddeeff -iv
01020304050607080 90a0b0c0d0e0f10

# CFB
openssl enc -aes-128-cfb -d -in sample_cfb_corrupted.bin -out
sample_cfb_decrypted.txt -K 00112233445566778899aabbccddeeff -iv
01020304050607080 90a0b0c0d0e0f10

# OFB
openssl enc -aes-128-ofb -d -in sample_ofb_corrupted.bin -out
sample_ofb_decrypted.txt -K 00112233445566778899aabbccddeeff -iv
01020304050607080 90a0b0c0d0e0f10
```

2. Prediction Before the Task

AES Mode	Predicted Recoverable Information
ECB	Only the corrupted block is garbled; rest of text is readable.
CBC	Corrupted block + next block partially garbled; remaining text readable.

CFB	Only corrupted byte + a few subsequent bytes garbled; rest readable.
OFB	Only the corrupted byte affected; rest of text fully readable.

---

3. Actual Result After Decryption

AES Mode	Actual Recoverable Information
ECB	Correctly predicted: only 16-byte block containing 30th byte corrupted; rest readable.
CBC	Correctly predicted: corrupted block fully garbled, next block has 1-bit error; remaining text readable.
CFB	Correctly predicted: only corrupted byte + minor propagation affected; rest readable.
OFB	Correctly predicted: only the corrupted byte is wrong; all other text intact.

---

4. Explanation

- **ECB:** Each 16-byte block encrypted independently → corruption confined to that block.
- **CBC:** Decryption XORs previous ciphertext → corruption affects current + next block.
- **CFB:** Stream-like mode → limited error propagation to next few bytes.
- **OFB:** Keystream independent of ciphertext → single-bit corruption does not propagate.

5. Implications

Mode	Implication
ECB	Localized corruption; easy to detect/correct. Not recommended for repeated patterns.
CBC	Error spreads to next block → sensitive to corruption, more difficult to recover.
CFB	Minor error propagation; suitable for streaming applications with occasional errors.
OFB	Minimal propagation; best for noisy channels where single-bit errors may occur.

Sample Text

The quick brown fox jumps over the lazy dog. This is a sample text to demonstrate AES encryption in different modes. Enjoy testing!

Analysis of Decryption After Corruption

Decrypted Texts (After 30th Byte Corruption)

AES Mode	Decrypted Text
ECB	The quick brown ``a``x``{``nQrhe lazy dog. This is a sample text to demonstrate AES encryption in different modes. Enjoy testing!
CBC	The quick brown ``&``h``m``he lazy dog. Uhis is a sample text to demonstrate AES encryption in different modes. Enjoy testing!
CFB	The quick brown fox jumps oves t6``'i``?``=``4``3s is a sample text to demonstrate AES encryption in different modes. Enjoy testing!
OFB	The quick brown fox jumps oves the lazy dog. This is a sample text to demonstrate AES encryption in different modes. Enjoy testing!

Observations

1. **ECB:**
   - Only the block containing the 30th byte (brown ...) is garbled (``a``x``{``nQr).
   - Rest of the text is **fully readable**.
2. **CBC:**
   - The corrupted block (brown ...) is completely garbled (``&``h``m``).
   - The next block has minor corruption (Uhis) instead of This.
   - Rest of the text is readable.
3. **CFB:**

    - Only the corrupted byte and a few subsequent bytes are affected (oves t6���'i�?�=�4�3s).
    - Rest of the text is **intact**.
4. **OFB:**
    - Only **one character** (jumps oves instead of jumps over) shows minor corruption.
    - Everything else is **completely correct**.

Task 4: Padding in AES Modes

Sample Plaintext:
This is a 30-byte sample text!

2. Encryption Commands

AES-128 Key & IV

- Key: 00112233445566778899aabbccddeeff
- IV: 0102030405060708090a0b0c0d0e0f10

ECB Mode

```
openssl enc -aes-128-ecb -e -in sample_pad.txt -out sample_ecb_pad.bin
-K 00112233445566778899aabbccddeeff -iv
0102030405060708090a0b0c0d0e0f10
```

CBC Mode

```
openssl enc -aes-128-cbc -e -in sample_pad.txt -out sample_cbc_pad.bin
-K 00112233445566778899aabbccddeeff -iv
0102030405060708090a0b0c0d0e0f10
```

CFB Mode

```
openssl enc -aes-128-cfb -e -in sample_pad.txt -out sample_cfb_pad.bin
-K 00112233445566778899aabbccddeeff -iv
0102030405060708090a0b0c0d0e0f10
```

OFB Mode

```
openssl enc -aes-128-ofb -e -in sample_pad.txt -out sample_ofb_pad.bin
-K 00112233445566778899aabbccddeeff -iv
0102030405060708090a0b0c0d0e0f10
```

Observations

AES Mode	Padding Required?	Explanation
ECB	Yes	ECB encrypts data in **fixed-size blocks independently**. Last block is padded to 16 bytes if plaintext is smaller.
CBC	Yes	CBC also uses **16-byte blocks**. Padding is required for the last block if plaintext length is not multiple of block size.
CFB	No	CFB is a **stream cipher mode** internally; it encrypts plaintext byte-by-byte (or segment-by-segment). Padding is unnecessary.
OFB	No	OFB generates a **keystream** and XORs with plaintext; works on any length. No padding is needed.

Check the **size of the encrypted files**:

```
ls -l sample_ecb_pad.bin sample_cbc_pad.bin sample_cfb_pad.bin
sample_ofb_pad.bin
```

- ECB and CBC files **will be larger** than the plaintext (because of padding).
- CFB and OFB files **will have the same size** as the plaintext.

Task 5: Generating Message Digest

Sample plaintext:
Hello, this is a sample text for hashing.

We are testing multiple one-way hash algorithms using OpenSSL.

The general syntax is:

openssl dgst -dgsttype filename

1. MD5 Command:

openssl dgst -md5 message.txt

2. SHA-1 Command:

openssl dgst -sha1 message.txt

3. SHA-256 Command:

openssl dgst -sha256 message.txt

```
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task5$ openssl dgst -md5 message.txt
MD5(message.txt)= c01e1c2b17be0a16b92c19b5d7e07bc5
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task5$ openssl dgst -sha1 message.txt
SHA1(message.txt)= e3a83d083c455785b5bb685c2bfeb677b127f107
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task5$ openssl dgst -sha256 message.txt
SHA2-256(message.txt)= f7f6efcd149b27d9e75dff810e751e106efc0ba9104971adb3f7646cb47a4143
```

Task 6: Keyed Hash and HMAC

Sample plaintext:
Hello, this is a sample text for hashing using HMAC.
We will generate HMACs with different algorithms and keys.

The general syntax is:

openssl dgst -hmac "your_key" -md5|-sha1|-sha256 filename

1. MD5 Command:

openssl dgst -hmac "key1" -md5 message.txt

2. SHA-1 Command:

openssl dgst -hmac "key123" -sha1 message.txt

3. SHA-256 Command:

openssl dgst -hmac "mysecretkeylong" -sha256 message.txt

```
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task5$ cd ~/INS_LAB/LAB3/Task6
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task6$ openssl dgst -hmac "key1" -md5 message.txt
HMAC-MD5(message.txt)= a75853493add2287112f5fb030f0bfef
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task6$ openssl dgst -hmac "key123" -sha1 message.txt
HMAC-SHA1(message.txt)= d2eae616f628746b29a6eb716db6ef5373eee8db
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task6$ openssl dgst -hmac "mysecretkeylong" -sha256 message.txt
HMAC-SHA2-256(message.txt)= ebe3794380d62def5c11fbf0c1a764d4b89118942d51d01e5d33506027aafd14
```

Task 7: Effect of One-Bit Change on Hash Values

Sample plaintext:
Cryptography ensures secure communication by using mathematical algorithms.
Even a tiny change in the message completely alters its hash value.
This property is called the avalanche effect.

1. MD5 Command:

openssl dgst -md5 message.txt

2. SHA-256 Command:

openssl dgst -sha256 message.txt

```
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task6$ cd ~/INS_LAB/LAB3/Task7
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task7$ openssl dgst -md5 message.txt
MD5(message.txt)= 22ddefd429f01d440596e1c0fbb0ba11
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task7$ openssl dgst -sha256 message.txt
SHA2-256(message.txt)= d33da148eec2467655d5aa7a0573690b2eff1c202ffe8a00dc40450dd7497735
```

Modified plaintext:
Dryptography ensures secure communication by using mathematical algorithms.
Even a tiny change in the message completely alters its hash value.
This property is called the avalanche effect.

1. MD5 Command:

openssl dgst -md5 message_modified.txt

2. SHA-256 Command:

openssl dgst -sha256 message_modified.txt

```
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task7$ openssl dgst -md5 message_modified.txt
MD5(message_modified.txt)= 8eb741c8f8af9280c2c23aec1b13ecac
tajwar46@DESKTOP-5NQP6OI:~/INS_LAB/LAB3/Task7$ openssl dgst -sha256 message_modified.txt
SHA2-256(message_modified.txt)= f3b42231a30feada07b059dfe3bde836fc33661fd15b644264af182574a88709
```