Report on Lab 2 - Symmetric Encryption & Hashing

Setup: I am using Windows for this Lab task. Using HxD replacement for GHex. Using notepad for text files and Paint for images.

Keys:
There are the key and initialize vector -

Key: 00112233445566778889aabbccddeeff (128-bit/16 bytes)
IV: 0102030405060708090a0b0c0d0e0f10 (128-bit/16 bytes)

Task-1 (AES Encryption using Different Modes)

1st we need to create text file in a folder. Named it original.txt. After that open powershell and copy the path of directory where the original is located.

CBC Encryption & Decryption:

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

- openssl enc -aes-128-cbc -d -in cbc.bin -out cbc-oroginal.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

ECB Encryption & Decryption:

- openssl enc -aes-128-ecb -e -in original.txt -out ecb.bin -K 00112233445566778899aabbccddeeff

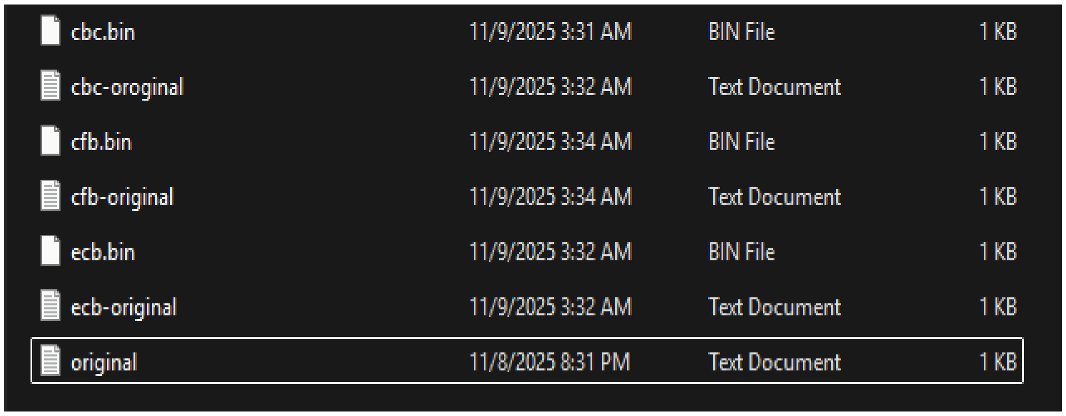
- openssl enc -aes-128-ecb -d -in ecb.bin -out ecb-original.txt -K 00112233445566778899aabbccddeeff

CFB Encryption & Decryption:

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

- openssl enc -aes-128-cfb -d -in cfb.bin -out cfb-original.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

Here original is my main text File.



```
cbc.bin         11/9/2025 3:31 AM   BIN File        1 KB
cbc-oroginal    11/9/2025 3:32 AM   Text Document   1 KB
cfb.bin         11/9/2025 3:34 AM   BIN File        1 KB
cfb-original    11/9/2025 3:34 AM   Text Document   1 KB
ecb.bin         11/9/2025 3:32 AM   BIN File        1 KB
ecb-original    11/9/2025 3:32 AM   Text Document   1 KB
original        11/8/2025 8:31 PM   Text Document   1 KB
```

Task-2 (ECB vs CBC Image)

1stly a BMP image has been generated from a JPG image, which has been named as 'dog-original'.

Encryption:

- **ECB:** openssl enc -aes-128-ecb -e -in dog-original.bmp -out ecb.enc -K 00112233445566778899aabbccddeeff
- **CBC:** openssl enc -aes-128-cbc -e -in dog-original.bmp -out cbc.enc -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

After the encryption, I utilized the tool Hxd to replace the header, comprising the initial 54 bytes, within the encrypted image with of the original picture. Then, the files were saved as bmp pictures, which were then opened using Paint.

Observation:
I can see the previous pictures on CBC while for ECB some pattern leakage can be observed. It is because it does not mix the outputs of the blocks. But CBC suppresses the pattern because it depends on the previous ciphertext + IV. That's why the image encrypted using ECB has a pattern, while the image encrypted using CBC has random noise.

Task-3 (Corrupted Cipher Text)

1stly i created a text file name original.txt

Encryption:

- **ECB:** openssl enc -aes-128-ecb -e -in original.txt -out e-ecb.bin -K 00112233445566778899aabbccddeeff

- **CBC:** openssl enc -aes-128-cbc -e -in original.txt -out e-cbc.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

- **CFB:** openssl enc -aes-128-cfb -e -in original.txt -out e-cfb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

- **OFB:** openssl enc -aes-128-ofb -e -in original.txt -out e-ofb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

A single bit of the **30th byte** in the encrypted file got corrupted.

Decryption:

- **ECB:** openssl enc -aes-128-ecb -d -in e-ecb-corrupt.bin -out d-ecb-corrupt.txt -K 00112233445566778899aabbccddeeff

- **CBC:** openssl enc -aes-128-cbc -d -in e-cbc-corrupt.bin -out d-cbc-corrupt.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

- **CFB:** openssl enc -aes-128-cfb -d -in e-cfb-corrupt.bin -out d-cfb-corrupt.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f1

- **OFB:** openssl enc -aes-128-ofb -d -in e-ofb-corrupt.bin -out d-ofb-corrupt.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

1. In the case of ECB, only the affected block gets corrupted while decrypting as the blocks are encrypted independently. The malicious code does not propagate to other blocks.
2. In CBC, both the current block as well as the next block are corrupted since every plaintext block is XOR with the previous ciphertext block. Thus, a corrupted ciphertext block will affect the decryption of both the corrupted as well as the next block.
3. In the case of CFB, it is observed that the corruption affects a few bytes, but it does not affect the whole message. The reason is that in CFB, the previous ciphertext is fed back to the encryption algorithm, leading to a limited spread of the error.
4. In OFB, it is only the corresponding byte or block that is affected. Since OFB generates a keystream independently of the plaintext, a single error in the ciphertext affects only the corresponding bits, without affecting the remaining bytes.

Task-4 (Padding)

Encryption:

- **ECB:** openssl enc -aes-128-ecb -e -in original.txt -out ecb.bin -K 00112233445566778899aabbccddeeff
- **CBC:** openssl enc -aes-128-cbc -e -in original.txt -out cbc.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
- **CFB:** openssl enc -aes-128-cfb -e -in original.txt -out cfb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
- **OFB:** openssl enc -aes-128-ofb -e -in original.txt -out ofb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

After Encrypting, it's observed that the file sizes of both CFB & OFB are the same as the original file, which is 89 byte. In contrast, the file sizes of both ECB & CBC are larger, which are 96 byte.

Padding is provided for
Block ciphers such as AES, operate on fixed-size blocks - 16 bytes for AES. If the plaintext message is not a multiple of the block length, it means that we have to pad the final block.

Padding is added prior to the encryption process, as well as removed following the decryption process.

Since CFB & OFB operates on streams, it does not require padding. The file size remains the same. But ECB & CBC operates on a block basis, it requires padding, the last 7 byte because 96 - 89 = 7, so it increased the file size.

Task-5 (Message Digest Hashing)

Commands and generated hash values:

openssl dgst -md5 original.txt

MD5 (original.txt)= 121959b186fbd6d79391c0fc35656ca5

openssl dgst -sha1 original.txt

SHA1 (original.txt)= c32934c7a54b3f4fada0f181611907097991d4b2

openssl dgst -sha256 original.txt

SHA256 (original.txt)=

b7412a488e4dc3097ed67d422a07efc58194a14e828a8531de3502c73a207a9c

Observation:

1. Fixed Length of Output - It has a fixed length that is irreproachable of input.
2. Avalanche Effect - Small changes in input cause drastically different hash values.
3. Deterministic - Given the same input, it always generates the same hash.
4. One-way Function - A hash value can never be reversibly transformed back into the input.
5. Security - SHA256 is secure, while MD5 is considered obsolete, so too is SHA1.

Task-6 (Keyed Hash and HMAC)

Commands and generated hash values:

openssl dgst -md5 -hmac "key99" original.txt

HMAC-MD5 (original.txt) = 401eee9b367eaab6bf97d2d7f18aa6c9

openssl dgst -sha1 -hmac "key99" original.txt

HMAC-SHA1 (original.txt) = 1f2236083252542758ec45c332bfe94e2a2d5d4b

openssl dgst -sha256 -hmac "key99" original.txt

HMAC-SHA256 (original.txt) =

885c19152cae4b026e6e45994136a5e8f5d1bcf6d95a019451def384a5b2cf36 openssl dgst -sha256 -hmac "a much longer sample key" original.txt

HMAC-SHA256 (original.txt) =

74109b9f41aecc3af120d9ba4b9b99220a58f935939429f633744cde1f05e548

- HMAC accepts keys of any length
- If the key is shorter than the hash function's block size (64 bytes for MD5/SHA1/SHA256), it is padded with zeros
- If the key is longer than the block sizer it is first hashed to reduce it to the hash output length
- This ensures the key is always processed consistently regardless of its original length
- The HMAC algorithm handles key normalization internally, making it flexible for keys of any size

Task-7 (Hashing effect)

Commands and generated hash values:

openssl dgst -md5 original.txt

MD5 (original.txt) = 121959b186fbd6d79391c0fc35656ca5

openssl dgst -sha256 original.txt

SHA256 (original.txt) =

1b28918dd1b7183d1d250d8f0b58a0f77b4cab66ed0caeee06b078843e407630

After flipping one bit :

openssl dgst -md5 original_flipped.txt

MD5 (original_flipped.txt)= 38ceb7812d807f484b61c715df4e985a

openssl dgst -sha256 original_flipped.txt

SHA256 (original_flipped.txt) =

fc522bd21c82b2ad2f8682c337ff40a84a54956b18673a5706aef783ab5ba764

Observations:

1. **Complete Change:** Flipping a single bit in the input file resulted in completely different hash values for both MD5 and SHA256
2. **No Similarity:** There is no visible similarity between them, confirming the one-way property of hash functions
3. **Deterministic:** The same input always produces the same hash, but any change produces a completely different hash