CSE-478 Lab 3: Symmetric Encryption & Hashing

Moloy Banerjee
Reg: 2020831021

November 8, 2025

Overview

This report details the tasks performed in Lab 3 on Symmetric Encryption and Hashing, covering AES encryption modes, comparisons between ECB and CBC, error propagation, padding requirements, message digests, and the avalanche effect. All provided code samples are in **Python 3** or shell commands using `openssl`.
—

1 Task 1: AES Encryption Using Different Modes

1.1 Commands

AES encryption was performed using CBC, ECB, and CFB modes.

```
echo "Secret message for CSE478 Lab 3." > plain_task1.txt
openssl enc -aes-128-cbc -in plain_task1.txt -out cipher_task1_cbc.bin \
 -K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ecb -in plain_task1.txt -out cipher_task1_ecb.bin \
 -K 00112233445566778889aabbccddeeff
openssl enc -aes-128-cfb -in plain_task1.txt -out cipher_task1_cfb.bin \
 -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

1.2 Observation

All three modes (**CBC**, **ECB**, **CFB**) successfully encrypted the file.
—

2 Task 2: ECB vs CBC Mode

This task compared the two encryption modes using image data to visualize pattern retention.

2.1 Steps

1. Created a BMP file: `convert size 100x100 xc:  black pic_original.bmp`
2. Encrypted the image with ECB and CBC modes.
3. Used GHex to copy the first 54 bytes (the BMP header) from the original file to the encrypted files to allow them to be viewed as images.

2.2 Observation

The **ECB mode** showed visible patterns in the encrypted image, while the **CBC mode** appeared as random noise. This demonstrates ECB's vulnerability to pattern analysis because identical plaintext blocks are encrypted into identical ciphertext blocks.

—

3 Task 3: Corrupted Cipher Text

This experiment investigated error propagation by corrupting a single byte in a 64-byte ciphertext.

3.1 Experiment Commands (Partial)

```
python3 -c "print('A'*64)" > plain_task3.txt
# Encrypt with different modes
openssl enc -aes-128-ecb -in plain_task3.txt -out cipher_ecb.bin \
-K 00112233445566778889aabbccddeeff
# Repeat for CBC, CFB, OFB modes
# Corrupt 30th byte using GHex
# Decrypt corrupted files
```

3.2 Results

- **ECB**: Only one block was affected.
- **CBC**: One block was garbled, plus a one-byte error in the next block.
- **CFB/OFB**: Only one byte was corrupted.

3.3 Implication

CFB and **OFB** modes are more suitable for error-prone transmission channels due to their minimal error propagation.

—

4 Task 4: Padding

This task demonstrated the padding requirements for different cipher modes.

4.1 Test Commands

```
echo "23 bytes long text." > plain_task4.txt
openssl enc -aes-128-ecb -in plain_task4.txt -out cipher_ecb.bin \
 -K 00112233445566778889aabbccddeeff -nopad
# Fails needs padding
openssl enc -aes-128-cfb -in plain_task4.txt -out cipher_cfb.bin \
 -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
# Succeeds no padding needed
```

4.2 Result

ECB and **CBC** require padding. **CFB**, **OFB**, and **CTR** do not require padding because they operate as stream ciphers.

—

5 Task 5: Message Digest

This task explored the use of cryptographic hash functions (message digests).

5.1 Commands

```
echo "Hash this data." > hash_input.txt
openssl dgst -md5 hash_input.txt
openssl dgst -sha1 hash_input.txt
openssl dgst -sha256 hash_input.txt
```

5.2 Observation

All algorithms produced **fixed-length outputs** with completely different hash values for the same input, demonstrating the proper **avalanche effect**.

—

6 Task 6: HMAC

This task focused on the Hash-based Message Authentication Code (HMAC).

6.1 Commands

```
echo "HMAC input data." > hmac_input.txt
openssl dgst -md5 -hmac "mykey" hmac_input.txt
openssl dgst -sha256 -hmac "mykey" hmac_input.txt
```

6.2 Answer

HMAC does not require a fixed key size. Keys are automatically hashed or padded to the appropriate block size required by the underlying hash function.

—

7 Task 7: Avalanche Effect

This task explicitly tested the avalanche effect in a hash function by modifying a single bit.

7.1 Steps

```
echo "Original text." > original_task7.txt
openssl dgst -sha256 original_task7.txt > H1_sha256.txt
# Flip one bit using GHex in modified_task7.txt
openssl dgst -sha256 modified_task7.txt > H2_sha256.txt
```

7.2 Observation

The resulting hashes, H_1 and H_2, were completely different, with approximately **50% of their bits changed**. This result clearly demonstrates the strong **avalanche effect** of the SHA-256 algorithm.