CSE-478 Lab 3: Symmetric Encryption & Hashing

Md. Rakibul Kabir

November 8, 2025

Task 1: AES Encryption Using Different Modes

Commands:

```
echo "Secret message for CSE478 Lab 3." > plain_task1.txt

openssl enc -aes-128-cbc -e -in plain_task1.txt -out cipher_task1_cbc.bin \
  -K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-ecb -e -in plain_task1.txt -out cipher_task1_ecb.bin \
  -K 00112233445566778889aabbccddeeff

openssl enc -aes-128-cfb -e -in plain_task1.txt -out cipher_task1_cfb.bin \
  -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Observation:

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

Task 2: ECB vs CBC Mode

Steps:

1. Created BMP file: `convert -size 100x100 xc:black pic_original.bmp`
2. Encrypted with ECB and CBC modes
3. Used GHex to copy first 54 bytes (BMP header) from original to encrypted files
4. Renamed: `pic_ecb_encrypted.bin` → `pic_ecb.bmp`, `pic_cbc_encrypted.bin` → `pic_cbc.bmp`

Observation:

ECB mode shows visible patterns; CBC mode appears as random noise, demonstrating ECB's vulnerability to pattern analysis.

Task 3: Corrupted Cipher Text

Experiment:

```
python3 -c "print('A'*64)" > plain_task3.txt

# Encrypt with different modes
openssl enc -aes-128-ecb -e -in plain_task3.txt -out cipher_ecb.bin \
  -K 00112233445566778889aabbccddeeff
# Repeat for CBC, CFB, OFB modes

# Corrupt 30th byte using GHex
# Decrypt corrupted files
```

Results:

- **ECB:** Only one block affected
- **CBC:** One block garbled + one byte error in next block
- **CFB/OFB:** Only one byte corrupted

Implication:

CFB and OFB modes are more suitable for error-prone transmission channels.

Task 4: Padding

Test:

```
echo "23 bytes long text." > plain_task4.txt

openssl enc -aes-128-ecb -e -in plain_task4.txt -out cipher_ecb.bin \
  -K 00112233445566778889aabbccddeeff -nopad
# Fails - needs padding

openssl enc -aes-128-cfb -e -in plain_task4.txt -out cipher_cfb.bin \
  -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
# Succeeds - no padding needed
```

Result:

ECB and CBC require padding; CFB, OFB, and CTR do not require padding due to their stream cipher nature.

Task 5: Message Digest

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
```

Observation:

All algorithms produce fixed-length outputs with completely different hash values, demonstrating proper avalanche effect.

Task 6: HMAC

Commands:

```
echo "HMAC input data." > hmac_input.txt

openssl dgst -md5 -hmac "mykey" hmac_input.txt
openssl dgst -sha256 -hmac "mykey" hmac_input.txt
```

Answer:

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

Task 7: Avalanche Effect

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
```

Observation:

H_1 and H_2 are completely different with approximately 50% of bits changed, demonstrating strong avalanche effect.