

Lab 3: Symmetric Encryption & Hashing

Imroj Hassan Shafi

2020831050

November, 2025

Contents

| | | |
|---|----------------------------------------------|---|
| 1 | Task 1: AES Encryption Using Different Modes | 2 |
| 2 | Task 2: ECB vs CBC Mode | 3 |
| 3 | Task 3: Corrupted Cipher Text | 4 |
| 4 | Task 4: Padding | 5 |
| 5 | Task 5: Message Digest | 6 |
| 6 | Task 6: HMAC | 7 |
| 7 | Task 7: Avalanche Effect | 8 |

1 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 00112233445566778889aabcccddeeff -iv 0102030405060708

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

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

Observation

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

2 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 the 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 reveals visible patterns. CBC mode appears as random noise, demonstrating ECB's vulnerability to pattern analysis.

3 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 0011223344556677889aabbcdddeeff

# Repeat for CBC, CFB, OFB modes

# Corrupt 30th byte using GHex

# Decrypt corrupted files
```

Results

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

Implication

CFB and OFB modes are preferred for error-prone communication channels.

4 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 00112233445566778889aabbcdddeeff -nopad
# Fails - needs padding

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

Result

ECB and CBC require padding. CFB, OFB, and CTR do not require padding, since they operate like stream ciphers.

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

6 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 a fixed key size. Keys are automatically padded or hashed to match the block size of the underlying hash function.

7 Task 7: Avalanche Effect

Steps

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

Observation

Hashes H1 and H2 differ a lot — approximately 50% of bits changed — demonstrating strong avalanche effect.