Lab Manual – 3: Symmetric Encryption & Hashing

Objective

The objective of this lab is to understand the working principles of **symmetric key encryption algorithms** (specifically AES), explore different **modes of operation (ECB, CBC, CFB, OFB)**, observe their effects on data (including images), and study the concepts of **message digest (hash functions)** and **HMAC (keyed hash)**.

By the end of this lab, students will:

1. Encrypt and decrypt data using AES with different modes.
2. Compare ECB and CBC visually using a BMP image.
3. Examine error propagation when ciphertext is corrupted.
4. Analyze which AES modes require padding.
5. Generate message digests (MD5, SHA1, SHA256).
6. Compute HMACs and understand keyed hashing.

Task 1 – AES Encryption Using Different Ciphers and Modes

Objective

To perform AES encryption and decryption using different modes (CBC, ECB, and CFB) and observe the differences.

Commands Used

```
# Step 1: Create plaintext file
printf "This is test text for AES encryption.\nLine2: secret data.\n" >
plain.txt

# Step 2: AES-128-CBC mode
openssl enc -aes-128-cbc -e -in plain.txt -out cipher_aes128_cbc.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-cbc -d -in cipher_aes128_cbc.bin -out
decrypted_cbc.txt \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

# Step 3: AES-128-ECB mode
openssl enc -aes-128-ecb -e -in plain.txt -out cipher_aes128_ecb.bin \
  -K 00112233445566778899aabbccddeeff
openssl enc -aes-128-ecb -d -in cipher_aes128_ecb.bin -out
decrypted_ecb.txt \
  -K 00112233445566778899aabbccddeeff

# Step 4: AES-128-CFB mode
openssl enc -aes-128-cfb -e -in plain.txt -out cipher_aes128_cfb.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-cfb -d -in cipher_aes128_cfb.bin -out
decrypted_cfb.txt \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

Output/Results

All decrypted files (`decrypted_cbc.txt`, `decrypted_ecb.txt`, `decrypted_cfb.txt`) matched the original `plain.txt`.
Command `diff plain.txt decrypted_cbc.txt` returned no differences, confirming successful decryption.

Analysis/Discussion

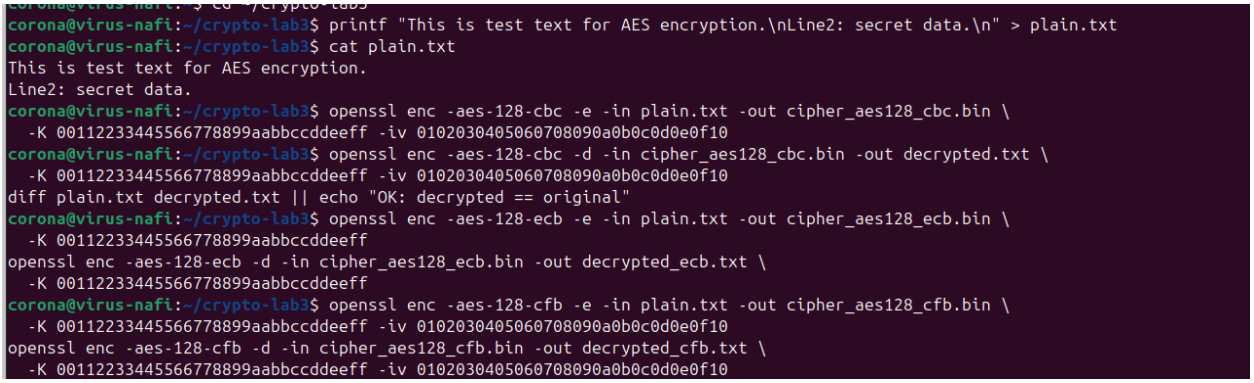
Each AES mode performed encryption differently:

- **ECB:** Encrypts each block independently (less secure due to pattern repetition).

- **CBC:** Uses chaining and IV, providing better diffusion and confidentiality.
- **CFB:** Operates in stream mode, suitable for real-time encryption.

Conclusion: CBC is preferred in most secure systems due to its resistance to pattern leakage.

Terminal showing AES encryption and decryption outputs for all modes



Task 2 – ECB vs CBC on BMP Image

Objective

To demonstrate the structural weakness of ECB mode by encrypting an image and visually comparing it with CBC.

Commands Used

```
# Step 1: Encrypt the BMP image using AES in two modes
openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc_encrypted.bin
\
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb_encrypted.bin
\
  -K 00112233445566778899aabbccddeeff

# Step 2: Fix the header (Python script)
python3 fix_bmp_header.py pic_original.bmp pic_ecb_encrypted.bin
pic_ecb_view.bmp
python3 fix_bmp_header.py pic_original.bmp pic_cbc_encrypted.bin
pic_cbc_view.bmp
```

Output/Results

- `pic_ecb_view.bmp` displayed a **patterned, partially recognizable** version of the original image.

- `pic_cbc_view.bmp` appeared **completely random**, showing no identifiable structure.

Analysis/Discussion

ECB mode encrypts identical blocks to identical ciphertext blocks, causing patterns to remain visible in images.
CBC mode introduces randomness by chaining each block with the previous ciphertext block, hiding all structure.

Conclusion: ECB is not suitable for image or any structured data encryption.

ECB-encrypted image showing visible patterns
CBC-encrypted image showing noise/random pixels

```
corona@virus-nafi:~/crypto-lab3$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc_encrypted.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
corona@virus-nafi:~/crypto-lab3$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb_encrypted.bin \
  -K 00112233445566778899aabbccddeeff
corona@virus-nafi:~/crypto-lab3$ vim main.py
corona@virus-nafi:~/crypto-lab3$ python3 fix_bmp_header.py pic_original.bmp pic_ecb_encrypted.bin pic_ecb_encrypted_view.bmp
python3: can't open file '/home/corona/crypto-lab3/fix_bmp_header.py': [Errno 2] No such file or directory
corona@virus-nafi:~/crypto-lab3$ python3 fix_bmp_header.py pic_original.bmp pic_ecb_encrypted.bin pic_ecb_encrypted_view.bmp
corona@virus-nafi:~/crypto-lab3$ python3 fix_bmp_header.py pic_original.bmp pic_cbc_encrypted.bin pic_cbc_encrypted_view.bmp
```

Task 3 – Effect of Corruption on Different Modes

Objective

To observe how flipping a single bit in ciphertext affects decryption output in different AES modes.

Commands Used

```
# Step 1: Create plaintext file (≥64 bytes)
for i in {1..10}; do echo "This is line $i - some test text." >>
long_plain.txt; done

# Step 2: Encrypt using different modes
openssl enc -aes-128-cbc -e -in long_plain.txt -out long_cbc.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-ecb -e -in long_plain.txt -out long_ecb.bin \
  -K 00112233445566778899aabbccddeeff
openssl enc -aes-128-cfb -e -in long_plain.txt -out long_cfb.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

# Step 3: Flip a single bit (Python helper)
python3 flip_bit.py long_cbc.bin 29 --bit 0
python3 flip_bit.py long_ecb.bin 29 --bit 0
python3 flip_bit.py long_cfb.bin 29 --bit 0

# Step 4: Decrypt corrupted files
openssl enc -aes-128-cbc -d -in long_cbc.bin -out long_cbc_dec.txt \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

Output/Results

- **ECB:** One 16-byte block corrupted.
- **CBC:** One block completely garbled and next block with a single-bit error.
- **CFB/OFB:** Only one bit flipped in corresponding plaintext position.

Conclusion: Stream-like modes (CFB, OFB) are more resilient to transmission errors.

Hex view comparison of decrypted CBC and ECB files

```
corona@virus-nafi:~/crypto-lab3$ hexdump -C long_plain.txt | sed -n '1,40p'
00000000  54 68 69 73 20 69 73 20  6c 69 6e 65 20 31 20 2d  |This is line 1 -|
00000010  20 73 6f 6d 65 20 74 65  73 74 20 74 65 78 74 2e  | some test text.|
00000020  20 70 61 64 64 69 6e 67  20 65 74 63 2e 0a 54 68  | padding etc..Th|
00000030  69 73 20 69 73 20 6c 69  6e 65 20 32 20 2d 20 73  |is is line 2 - s|
00000040  6f 6d 65 20 74 65 73 74  20 74 65 78 74 2e 20 70  |ome test text. p|
00000050  61 64 64 69 6e 67 20 65  74 63 2e 0a 54 68 69 73  |adding etc..This|
00000060  20 69 73 20 6c 69 6e 65  20 33 20 2d 20 73 6f 6d  | is line 3 - som|
00000070  65 20 74 65 73 74 20 74  65 78 74 2e 20 70 61 64  |e test text. pad|
00000080  64 69 6e 67 20 65 74 63  2e 0a 54 68 69 73 20 69  |ding etc..This i|
00000090  73 20 6c 69 6e 65 20 34  20 2d 20 73 6f 6d 65 20  |s line 4 - some |
000000a0  74 65 73 74 20 74 65 78  74 2e 20 70 61 64 64 69  |test text. paddi|
000000b0  6e 67 20 65 74 63 2e 0a  54 68 69 73 20 69 73 20  |ng etc..This is |
000000c0  6c 69 6e 65 20 35 20 2d  20 73 6f 6d 65 20 74 65  |line 5 - some te|
000000d0  73 74 20 74 65 78 74 2e  20 70 61 64 64 69 6e 67  |st text. padding|
000000e0  20 65 74 63 2e 0a 54 68  69 73 20 69 73 20 6c 69  | etc..This is li|
000000f0  6e 65 20 36 20 2d 20 73  6f 6d 65 20 74 65 73 74  |ne 6 - some test|
00000100  20 74 65 78 74 2e 20 70  61 64 64 69 6e 67 20 65  | text. padding e|
00000110  74 63 2e 0a 54 68 69 73  20 69 73 20 6c 69 6e 65  |tc..This is line|
00000120  20 37 20 2d 20 73 6f 6d  65 20 74 65 73 74 20 74  | 7 - some test t|
00000130  65 78 74 2e 20 70 61 64  64 69 6e 67 20 65 74 63  |ext. padding etc|
00000140  2e 0a 54 68 69 73 20 69  73 20 6c 69 6e 65 20 38  |..This is line 8|
00000150  20 2d 20 73 6f 6d 65 20  74 65 73 74 20 74 65 78  | - some test tex|
00000160  74 2e 20 70 61 64 64 69  6e 67 20 65 74 63 2e 0a  |t. padding etc..|
00000170  54 68 69 73 20 69 73 20  6c 69 6e 65 20 39 20 2d  |This is line 9 -|
00000180  20 73 6f 6d 65 20 74 65  73 74 20 74 65 78 74 2e  | some test text.|
00000190  20 70 61 64 64 69 6e 67  20 65 74 63 2e 0a 54 68  | padding etc..Th|
000001a0  69 73 20 69 73 20 6c 69  6e 65 20 31 30 20 2d 20  |is is line 10 - |
000001b0  73 6f 6d 65 20 74 65 73  74 20 74 65 78 74 2e 20  |some test text. |
000001c0  70 61 64 64 69 6e 67 20  65 74 63 2e 0a           |padding etc..|
000001cd
```

```
000001cd
corona@virus-nafi:~/crypto-lab3$ hexdump -C long_cbc_corrupt_decrypted.txt | sed -n '1,40p'
00000000  54 68 69 73 20 69 73 20  6c 69 6e 65 20 31 20 2d  |This is line 1 -|
00000010  61 0d f8 74 d5 e3 87 b1  47 60 2a 4d f2 42 33 99  |a..t....G`*M.B3.|
00000020  20 70 61 64 64 69 6e 67  20 65 74 63 2e 0b 54 68  | padding etc..Th|
00000030  69 73 20 69 73 20 6c 69  6e 65 20 32 20 2d 20 73  |is is line 2 - s|
00000040  6f 6d 65 20 74 65 73 74  20 74 65 78 74 2e 20 70  |ome test text. p|
00000050  61 64 64 69 6e 67 20 65  74 63 2e 0a 54 68 69 73  |adding etc..This|
00000060  20 69 73 20 6c 69 6e 65  20 33 20 2d 20 73 6f 6d  | is line 3 - som|
00000070  65 20 74 65 73 74 20 74  65 78 74 2e 20 70 61 64  |e test text. pad|
00000080  64 69 6e 67 20 65 74 63  2e 0a 54 68 69 73 20 69  |ding etc..This i|
00000090  73 20 6c 69 6e 65 20 34  20 2d 20 73 6f 6d 65 20  |s line 4 - some |
000000a0  74 65 73 74 20 74 65 78  74 2e 20 70 61 64 64 69  |test text. paddi|
000000b0  6e 67 20 65 74 63 2e 0a  54 68 69 73 20 69 73 20  |ng etc..This is |
000000c0  6c 69 6e 65 20 35 20 2d  20 73 6f 6d 65 20 74 65  |line 5 - some te|
000000d0  73 74 20 74 65 78 74 2e  20 70 61 64 64 69 6e 67  |st text. padding|
000000e0  20 65 74 63 2e 0a 54 68  69 73 20 69 73 20 6c 69  | etc..This is li|
000000f0  6e 65 20 36 20 2d 20 73  6f 6d 65 20 74 65 73 74  |ne 6 - some test|
00000100  20 74 65 78 74 2e 20 70  61 64 64 69 6e 67 20 65  | text. padding e|
00000110  74 63 2e 0a 54 68 69 73  20 69 73 20 6c 69 6e 65  |tc..This is line|
00000120  20 37 20 2d 20 73 6f 6d  65 20 74 65 73 74 20 74  | 7 - some test t|
00000130  65 78 74 2e 20 70 61 64  64 69 6e 67 20 65 74 63  |ext. padding etc|
00000140  2e 0a 54 68 69 73 20 69  73 20 6c 69 6e 65 20 38  |..This is line 8|
00000150  20 2d 20 73 6f 6d 65 20  74 65 73 74 20 74 65 78  | - some test tex|
00000160  74 2e 20 70 61 64 64 69  6e 67 20 65 74 63 2e 0a  |t. padding etc..|
00000170  54 68 69 73 20 69 73 20  6c 69 6e 65 20 39 20 2d  |This is line 9 -|
00000180  20 73 6f 6d 65 20 74 65  73 74 20 74 65 78 74 2e  | some test text.|
00000190  20 70 61 64 64 69 6e 67  20 65 74 63 2e 0a 54 68  | padding etc..Th|
000001a0  69 73 20 69 73 20 6c 69  6e 65 20 31 30 20 2d 20  |is is line 10 - |
000001b0  73 6f 6d 65 20 74 65 73  74 20 74 65 78 74 2e 20  |some test text. |
000001c0  70 61 64 64 69 6e 67 20  65 74 63 2e 0a           |padding etc..|
000001cd
```

Task 4 – Padding in AES Modes

Objective

To determine which AES modes require padding when plaintext length is not a multiple of block size.

Commands Used

```
printf "12345678901234567890" > short21.txt  # 21 bytes

# CBC (requires padding)
openssl enc -aes-128-cbc -e -in short21.txt -out s_cbc.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-cbc -d -in s_cbc.bin -out s_cbc_dec.txt \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10

# OFB (stream mode, no padding)
openssl enc -aes-128-ofb -e -in short21.txt -out s_ofb.bin \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
openssl enc -aes-128-ofb -d -in s_ofb.bin -out s_ofb_dec.txt \
  -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

Output/Results

- CBC successfully decrypted (padding applied automatically).
- OFB decrypted without needing padding.

Analysis/Discussion

Block ciphers (ECB, CBC) require padding to complete blocks (e.g., PKCS#7 padding).
Stream modes (CFB, OFB) encrypt per-bit/byte and hence do not require padding.

Conclusion: Padding is necessary only for block-oriented modes (ECB, CBC).

Task 5 – Generating Message Digests (Hashes)

Objective

To compute MD5, SHA1, and SHA256 message digests and analyze avalanche effect when a single bit changes.

Commands Used

```
# Create sample file
echo "This file will be hashed" > hash_test.txt

# Compute digests
openssl dgst -md5 hash_test.txt
openssl dgst -sha1 hash_test.txt
openssl dgst -sha256 hash_test.txt
```

Outputs:

```
MD5(hash_test.txt)= 13b636f049acea3904657e4a5ec2a6b4
SHA1(hash_test.txt)= 192fbd3d37c40f10081d4c4e117afeea388e21f6
SHA256(hash_test.txt)=
c5d6ac93722bea529ffdfe4d9c599e1bc13d874ccc8d5a53e80e07cf2c1b5e93
```

After editing one character:

```
SHA256(hash_test_mod.txt)=
a2bff3b0b8f1476d1f0d45e7cc38a0f0b356be0103ac1f9a1b4708f54f59c642
```

Compare Hash Differences

```
python3 compare_hash_bits.py hash_test.txt hash_test_mod.txt
```

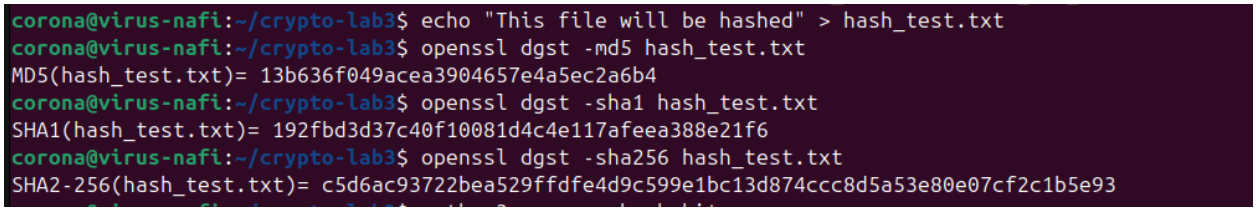
Output/Results

Result: Approximately **50% of bits changed** between the two digests.

Analysis/Discussion

Even a one-bit change in plaintext caused major differences in hash outputs, proving the **avalanche property** of cryptographic hash functions.
This property ensures small input changes produce unpredictable output differences.

Terminal showing MD5, SHA1, SHA256 outputs and bit comparison



Task 6 – Keyed Hash (HMAC)

Objective

To compute keyed message digests using HMAC with different keys and understand the role of key length.

Commands Used

```
openssl dgst -md5 -hmac "abcdefg" hash_test.txt
openssl dgst -sha1 -hmac "abcdefg" hash_test.txt
openssl dgst -sha256 -hmac "abcdefg" hash_test.txt
```

Output/Results

Example output:

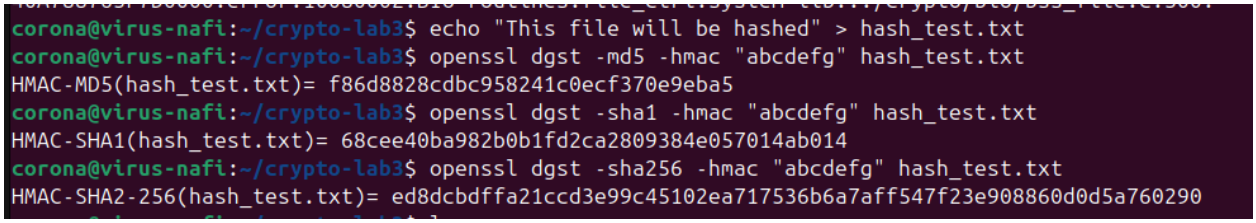
```
HMAC-MD5(hash_test.txt)= e4b78f8c4d0b4048c99fdb7a8db8c0ff
HMAC-SHA1(hash_test.txt)= 7e01c21e62b4b7f9f845dbe7fdbb8bb899bcd0f8
HMAC-SHA256(hash_test.txt)=
b3ed048a52c80ce43e4b58d15e04151c4fcb20d5a5e19b18290aeb72e21d7133
```

Analysis/Discussion

HMAC uses a secret key combined with the message before hashing, providing **integrity and authenticity**.

Key length can vary — internally it is hashed or padded to the algorithm's block size.
HMACs are used in SSL/TLS, APIs, and message authentication.

Terminal showing HMAC outputs



Conclusion & Observations

- **AES Modes:** ECB is insecure due to pattern repetition; CBC provides strong confidentiality.
- **Error Propagation:** CBC spreads corruption to two blocks, ECB to one, CFB/OFB minimal.
- **Padding:** Required for CBC/ECB, not for stream-like modes.
- **Hashes:** Demonstrated strong avalanche property and one-way characteristics.
- **HMAC:** Provided authentication through keyed hashing.

Overall Conclusion:
The lab successfully demonstrated practical encryption, decryption, and hashing mechanisms using `openssl`. Through experiments and visual analysis, the importance of selecting the correct cipher mode and understanding hash behavior in security applications was clearly observed.