

# Report on

## Lab 3: Symmetric encryption & hashing

### Initial Setup

Since I have a windows PC, I will use HxD hex editor software for editing the files. I will use Windows Notebook and Photo Viewer tools to create text files and view images.

### Key and IV(for all tasks)

KEY = 00112233445566778899aabbccddeeff (32 hex = 16 bytes)

IV = 0102030405060708090a0b0c0d0e0f10 (32 hex = 16 bytes)

### Task 1

#### CBC Encryption & Decryption:

**Encryption command:** `openssl enc -aes-128-cbc -e -in plain.txt -out cipher_cbc.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10`

**Decryption command:** `openssl enc -aes-128-cbc -d -in cipher_cbc.bin -out dec_cbc.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10`

Open cipher\_cbc.bin /home/mustakin

plain.txt	cipher_cbc.bin	decrypted_cbc.txt	cipher_ofb.bin
00000000 9C CB D1 C4 D9 C4 9B 4C FA 94 79 D4 B8 C0 2F 38	00000000 24 28 46 1F 1E 2D 1E B5 A1 30 88 B6 90 F2 FF 40		
00000010 6A CA 09 E3 18 5F CA FD 5A 2A E8 02 2E 9F E9 4E			
00000020 88 DC 81 54 32 A2 9B C8 2C 73 26 83 82 F5 B8 EA			
00000030 0E DD 33 63 A2 05 7E E3 8D EF 09 1C 91 EC 52 94			
00000040			

Signed 8 bit: -100 Signed 32 bit: -992883812 Hexadecimal: 9C  
Unsigned 8 bit: 156 Unsigned 32 bit: 3302083484 Octal: 234  
Signed 16 bit: -13412 Signed 64 bit: 552022220786 Binary: 10011100  
Unsigned 16 bit: 52124 Unsigned 64 bit: 552022220786 Stream Length: 8  
Float 32 bit: -1.678363e+0 Float 64 bit: 1.115574e+61  
☒ Show little endian decoding ☐ Show unsigned and float as hexadecimal  
Offset: 0x0

Open decrypted\_cbc.txt /home/mustakin

plain.txt	cipher_cfb.bin	decrypted_cbc	cipher_ofb.bin	cipher_cbc.bin	decrypted_c
00000000 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6D					
00000010 65 73 73 61 67 65 20 66 6F 72 20 65 6E 63 72 79					
00000020 70 74 69 6F 6E 2E 0A 49 74 20 68 61 73 20 6D 75					
00000030 6C 74 69 70 6C 65 20 6C 69 6E 65 73 2E 0A 4C 69					
00000040 6E 65 20 74 68 72 65 65 2E 0A					

Signed 8 bit: 84 Signed 32 bit: 1936287828 Hexadecimal: 54  
Unsigned 8 bit: 84 Unsigned 32 bit: 1936287828 Octal: 124  
Signed 16 bit: 26708 Signed 64 bit: 233832821963 Binary: 01010100  
Unsigned 16 bit: 26708 Unsigned 64 bit: 233832821963 Stream Length: 8  
Float 32 bit: 1.849245e+31 Float 64 bit: 2.316340e+152  
☒ Show little endian decoding ☐ Show unsigned and float as hexadecimal  
Offset: 0x0

#### CFB Encryption & Decryption:

**Encryption command:** `openssl enc -aes-128-cfb -e -in plain.txt -out cipher_cfb.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708`

**Decryption command:** `openssl enc -aes-128-cfb -d -in cipher_cfb.bin -out decrypted_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

The screenshot shows a hex editor with two tabs: `cipher_cfb.bin` and `decrypted_cfb.txt`. The `decrypted_cfb.txt` tab is active, showing the decrypted data. The hex view displays the raw bytes, and the ASCII view shows the resulting text: `...V.V....x.d|. ....8.B....Ns. g53.(. #p..x..[.X e.B...{...t]6M ...)0..K..`. Below the hex view, various numerical representations are shown for the first few bytes: Signed 8 bit: 103, Signed 32 bit: 103, Hexadecimal: 67, Unsigned 8 bit: 103, Unsigned 32 bit: 103, Octal: 147, Signed 16 bit: 103, Signed 64 bit: 103, Binary: 01100111, Unsigned 16 bit: 103, Unsigned 64 bit: 103, Stream Length: 8, Float 32 bit: 1.443337e-43, Float 64 bit: 5.088876e-322. The `Show little endian decoding` checkbox is checked. The offset is 0x49.

## OFB Encryption & Decryption:

**Encryption command:** `openssl enc -aes-128-ofb -e -in plain.txt -out cipher_ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

**Decryption command:** `openssl enc -aes-128-ofb -d -in cipher_ofb.bin -out decrypted_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

Open

cipher\_ofb.bin  
/home/mustakin

DownloadSearchMenuClose

plain.txt	cipher_ofb.bin	decrypted_ofb.txt
00000000 00000010 00000020 00000030 00000040	D3 EE E6 56 E1 56 C9 F1 CE BE CB 78 06 64 7C 00 54 27 EA 66 DC 3E 62 95 98 2A 26 99 3D 0A 3A E6 A5 19 55 0E 0B 4B 55 1C 61 86 03 E4 4E 1D C4 DD 62 51 DB EA 09 F3 C5 4A 52 1F 04 69 F1 74 59 3D 13 17 FD A4 B0 15 20 9D 48 A4	[. . V . V . . . . x . d   . T ' . f . > b . . * & . = . : . .. U . . K U . a . . . N . . b Q . . . . J R . . i . t Y = . . . . . . . H .

Signed 8 bit: -45

Signed 32 bit: 1457974995

Hexadecimal: D3

Unsigned 8 bit: 211

Unsigned 32 bit: 1457974995

Octal: 323

Signed 16 bit: -4397

Signed 64 bit: -10241919144

Binary: 11010011

Unsigned 16 bit: 61139

Unsigned 64 bit: 174225521592

Stream Length: 8 - +

Float 32 bit: 1.269567e+14

Float 64 bit: -1.320026e+2

☒ Show little endian decoding

☐ Show unsigned and float as hexadecimal

Offset: 0x0

Copy

Open

decrypted\_ofb.txt  
/home/mustakin

DownloadSearchMenuClose

plain.txt	cipher_ofb.bin	decrypted_ofb.txt
00000000 00000010 00000020 00000030 00000040	54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6D 65 73 73 61 67 65 20 66 6F 72 20 65 6E 63 72 79 70 74 69 6F 6E 2E 0A 49 74 20 68 61 73 20 6D 75 6C 74 69 70 6C 65 20 6C 69 6E 65 73 2E 0A 4C 69 6E 65 20 74 68 72 65 65 2E 0A	This is a test m essage for encry ption..It has mu ltiple lines..Li ne three..

Signed 8 bit: 84

Signed 32 bit: 1936287828

Hexadecimal: 54

Unsigned 8 bit: 84

Unsigned 32 bit: 1936287828

Octal: 124

Signed 16 bit: 26708

Signed 64 bit: 233832821963

Binary: 01010100

Unsigned 16 bit: 26708

Unsigned 64 bit: 233832821963

Stream Length: 8 - +

Float 32 bit: 1.849245e+31

Float 64 bit: 2.316340e-152

☒ Show little endian decoding

☐ Show unsigned and float as hexadecimal

Offset: 0x0

Copy

I created a simple txt file named plain.txt and encrypted it using the encryption commands. Then I decrypted the encrypted files using the decryption commands. I have added the encrypted and decrypted files in the Task1 folder.

**Encrypted files name:** cipher\_cbc.bin, cipher\_ofb.bin, cipher\_cfb.bin

**Decrypted files name:** dec\_cbc.txt, dec\_ofb.txt, dec\_cfb.txt

## Task 2

First I created a BMP picture from a JPG format picture and named it pic\_original.

### **Encryption:**

#### **ECB:**

```
openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.enc -K  
00112233445566778899aabbccddeeff
```

#### **CBC:**

```
openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc.enc -K  
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

After encrypting I used HxD to replace the header(the first 54 bytes) of the encrypted picture with that

of the original picture. Then I saved the files as bmp files and then opened them in the photos app.

#### **Observation:**

I cannot see the previous images on CBC but for ECB I can see some patterns leaking. Because it doesn't mix block outputs. But CBC hides patterns because each block encryption depends on previous ciphertext + IV. That's why the ECB-encrypted image looks structured, but the CBC-encrypted image looks like random noise.

## Task3

First I created a txt file of 69bytes. Then,

**Encryption:****ECB:**

```
openssl enc -aes-128-ecb -e -in plain.txt -out c_ecb.bin -K  
00112233445566778899aabbccddeeff
```

**CBC:**

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

**CFB:**

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

**OFB:**

```
openssl enc -aes-128-ofb -e -in plain.txt -out c_ofb.bin -K  
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

Open

cipher\_ofb.bin

/home/mustakin

↓

Q

≡

×

plain.txt

cipher\_cbc.bin

cipher\_cfb.bin

cipher\_ecb.bin

cipher\_ofb.bin

00000000

D3 EE E6 56 E1 56 C9 F1 CE BE CB 78 06 64 7C 00

...V.V.....x.d|.

00000010

54 27 EA 66 DC 3E 62 95 98 2A 26 99 3D 0A 3A E6

T'.f.>b..\*&.=...:

00000020

A5 19 55 0E 0B 4B 55 1C 61 86 03 E4 4E 1D C4 DD

..U..KU.a...N...

00000030

62 51 DB EA 09 F3 C5 4A 52 1F 04 69 F1 74 59 3D

bQ.....JR..i.tY=

00000040

13 17 FD A4 B0 15 20 9D 48 A4

.....H.

Signed 8 bit:

-45

Signed 32 bit:

1457974995

Hexadecimal:

D3

Unsigned 8 bit:

211

Unsigned 32 bit:

1457974995

Octal:

323

Signed 16 bit:

-4397

Signed 64 bit:

-10241919144

Binary:

11010011

Unsigned 16 bit:

61139

Unsigned 64 bit:

174225521592

Stream Length:

8

-

+

Float 32 bit:

1.269567e+14

Float 64 bit:

-1.320026e+2

☒ Show little endian decoding

☐ Show unsigned and float as hexadecimal

Offset: 0x0

↕

▼

Open

cipher\_ecb.bin  
/home/mustakin

Download Search Menu Close

plain.txt

cipher\_cbc.bin

cipher\_cfb.bin

cipher\_ecb.bi x

cipher\_ofb.bin

00000000  
00000010  
00000020  
00000030  
00000040

B 60 8F DA F8 73 03 B0 C9 B2 7A 0F A0 63 43 C2  
51 85 71 86 37 E7 FE CC 9C 21 50 37 7E 23 F8 FE  
E5 69 0C E6 5E 8F E5 E9 59 D0 AC 91 FB 45 B2 CD  
31 0C 0F B2 CF B4 B8 3E E9 DE FF 06 6D BD B6 E3  
47 90 5D 6D AC 41 21 94 D0 C7 4E 6B C2 E1 33 3D

...s....z..cC.  
Q.q.7....!P7~#..  
.i..^...Y....E..  
1.....>....m...  
G.]m.A!...Nk..3=

Signed 8 bit:

-5

Signed 32 bit:

-628137733

Hexadecimal:

FB

Unsigned 8 bit:

251

Unsigned 32 bit:

3666829563

Octal:

373

Signed 16 bit:

24827

Signed 64 bit:

-57636355854

Binary:

11111011

Unsigned 16 bit:

24827

Unsigned 64 bit:

126831084882

Stream Length:

8

-

+

Float 32 bit:

-2.017878e+11

Float 64 bit:

-2.099994e-77



Show little endian decoding



Show unsigned and float as hexadecimal

Offset: 0x0



Open

cipher\_cfb.bin  
/home/mustakin

Download Search Menu Close

plain.txt

cipher\_cbc.bin

cipher\_cfb.bin x

cipher\_ecb.bin

cipher\_ofb.bin

00000000	D3 EE E6 56 E1 56 C9 F1 CE BE CB 78 06 64 7C 00	...V.V.....x.d . .
00000010	FF 88 D9 C8 F3 38 CF 42 E8 06 F9 7F 8F 4E 24 F8	.....8.B.....N\$. .
00000020	67 53 33 B5 28 D1 23 70 0A AC 78 87 CA 5B DC 58	gS3.(. #p..x..[.X .
00000030	65 DA 42 F7 00 82 AD 7B 14 F8 D5 C9 74 5D 26 4D	e.B....{....t]&M .
00000040	84 04 0B 29 30 CF EF 4B B7 67	...)0..K.g

Signed 8 bit:	-45	Signed 32 bit:	1457974995	Hexadecimal:	D3
Unsigned 8 bit:	211	Unsigned 32 bit:	1457974995	Octal:	323
Signed 16 bit:	-4397	Signed 64 bit:	-10241919144	Binary:	11010011
Unsigned 16 bit:	61139	Unsigned 64 bit:	174225521592	Stream Length:	8 - +
Float 32 bit:	1.269567e+14	Float 64 bit:	-1.320026e+20		

☒ Show little endian decoding ☐ Show unsigned and float as hexadecimal

Offset: 0x0

Copy

Open

cipher\_cbc.bin

/home/mustakin

Q

x

plain.txt

cipher\_cbc.bi x

cipher\_cfb.bin

cipher\_ecb.bin

cipher\_ofb.bin

00000000	9C CB D1 C4 D9 C4 9B 4C FA 94 79 D4 B8 C0 2F 38	.....L..y.../8
00000010	24 28 46 1F 1E 2D 1E B5 A1 30 88 B6 90 F2 FF 40	\$(F...-...0.....@
00000020	6A CA 09 E3 18 5F CA FD 5A 2A E8 02 2E 9F E9 4E	j....._..Z*.....N
00000030	88 DC 81 54 32 A2 9B C8 2C 73 26 83 82 F5 B8 EA	...T2...,s&.....
00000040	0E DD 33 63 A2 05 7E E3 8D EF 09 1C 91 EC 52 94	..3c..~.....R.

Signed 8 bit: -100

Signed 32 bit: -992883812

Hexadecimal: 9C

Unsigned 8 bit: 156

Unsigned 32 bit: 3302083484

Octal: 234

Signed 16 bit: -13412

Signed 64 bit: 55202222078€

Binary: 10011100

Unsigned 16 bit: 52124

Unsigned 64 bit: 55202222078€

Stream Length: 8 - +

Float 32 bit: -1.678363e+0:

Float 64 bit: 1.115574e+61

☒ Show little endian decoding
 ☐ Show unsigned and float as hexadecimal

Offset: 0x0

Unfortunately, a single bit of the **30th byte** in the encrypted file got corrupted( Used HxD).  
Now,

## Decryption:

### ECB:

```
openssl enc -aes-128-ecb -d -in c_ecb_corrupt.bin -out d_ecb_corrupt.txt -K
00112233445566778899aabbccddeeff
```

### CBC:

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

### CFB:

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

### OFB:

```
openssl enc -aes-128-ofb -d -in c_ofb_corrupt.bin -out d_ofb_corrupt.txt -K
00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```

After decryption,

**Question No 1 and 2: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? And why?**

**Ans:**

**For ECB,** Only the corresponding block is corrupted on decryption. Because each block is encrypted independently. Corruption doesn't propagate beyond that block.

**For CBC,** The current block and the next block become corrupted on decryption. Each plaintext block is XOR-ed with the previous ciphertext block. So, one corrupted ciphertext block breaks its own decryption and corrupts the next block.

**For CFB,** the corruption affects a few bytes but doesn't ruin the rest. CFB uses the previous ciphertext as input to the encryption step, so errors propagate for only a few bytes of plaintext, not indefinitely.

**For OFB,** Only one byte (or one block) is corrupted. OFB generates a keystream independent of the plaintext; errors in ciphertext affect only matching bits on decryption, not future bytes.

**Question no 3: What are the implications of these differences?**

**Ans:** Reasons are-

1. **Error tolerance and data transmission**
  - a. Modes like OFB and CFB are better suited for streaming or communication channels where small bit errors might occur.
  - b. ECB and CBC are not good for noisy channels since bit errors can ruin entire blocks.
2. **Security implications**
  - a. ECB leaks structure. It's deterministic and should never be used for sensitive data like images.
  - b. CBC, CFB, and OFB hide patterns much better.
3. **Performance and parallelism**
  - a. ECB can be parallelized easily; others (especially CBC encryption) are sequential because of block dependencies.
  - b. OFB and CFB behave like stream ciphers — useful for continuous data streams.

## Task 4

Created small.txt with "1234567890"

### Commands Used:

**ECB:** `openssl enc -aes-128-ecb -e -in small.txt -out small_ecb.bin -K 00112233445566778889aabbccddeeff`

**CBC:** `openssl enc -aes-128-cbc -e -in small.txt -out small_cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

**CFB:** `openssl enc -aes-128-cfb -e -in small.txt -out small_cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

**OFB:** `openssl enc -aes-128-ofb -e -in small.txt -out small_ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

### Outputs (Ciphertext Sizes):

ECB: 16 bytes (padded)

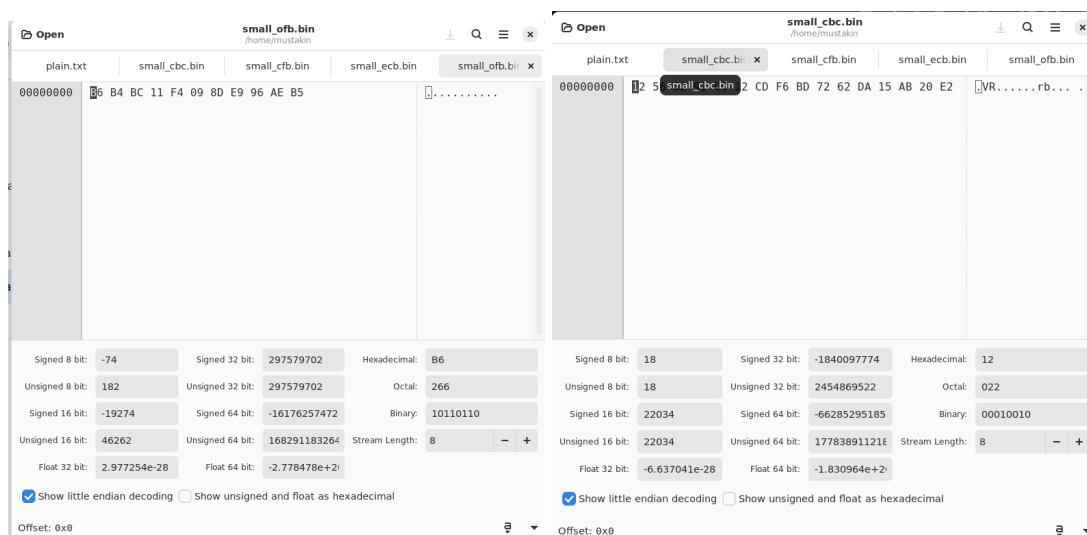
CBC: 16 bytes (padded)

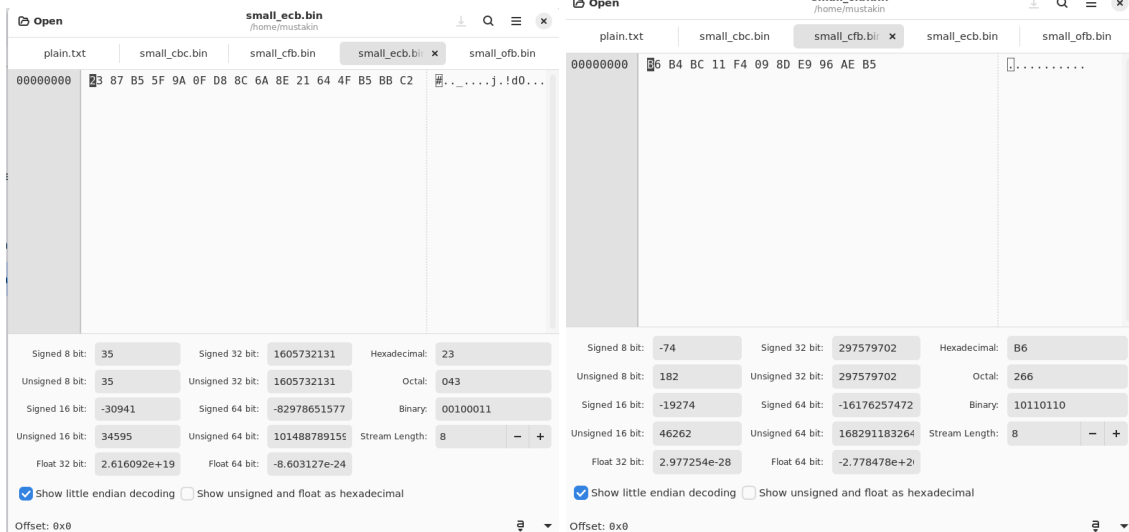
CFB: 10 bytes (no padding)

OFB: 10 bytes (no padding)

### Observations:

- ECB and CBC use block padding (PKCS7 by default in OpenSSL) to reach multiple of 16 bytes.
- CFB and OFB are stream modes; they encrypt byte-by-byte without needing padding.





## Task 5

### Commands Used:

**MD5 H1:** *openssl dgst -md5 plain.txt*

**MD5(plain.txt)=** b0e98dc0588e9b8938036d8960dab393

**SHA256 H1:** *openssl dgst -sha256 plain.txt*

**SHA1(plain.txt)=** 776f07faa3e4c8a85bf1e8a25dee75ad26b39938

**SHA256 H2:** *openssl dgst -sha256 modified.txt*

**SHA256(plain.txt)=**

126503eeaa27867fb59f5c1d560b56c279e0db33f71fa958a38f6f816b52117d

```
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -md5 plain.txt
MD5(plain.txt)= b0e98dc0588e9b8938036d8960dab393
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -sha1 plain.txt
SHA1(plain.txt)= 776f07faa3e4c8a85bf1e8a25dee75ad26b39938
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -sha256 plain.txt
SHA2-256(plain.txt)= 126503eeaa27867fb59f5c1d560b56c279e0db33f71fa9
58a38f6f816b52117d
```

**Observations:**

Different algorithms produce different hash lengths (MD5: 128 bits, SHA1: 160 bits, SHA256: 256 bits) and values.

## Task 6

### Task 6: Keyed Hash and HMAC (3 Marks)

Used plain.txt.

### Commands Used:

**HMAC-MD5:** `openssl dgst -md5 -hmac "abcdefg" plain.txt`

**HMAC-SHA1:** `openssl dgst -sha1 -hmac "abcdefg" plain.txt`

**HMAC-SHA256:** `openssl dgst -sha256 -hmac "abcdefg" plain.txt`

**HMAC-MD5 (short key "a"):** `openssl dgst -md5 -hmac "a" plain.txt`

[illegible]

### Outputs:

**HMAC-MD5(plain.txt)= ff72ad81a4bd7ea26153571bff354604**

HMAC-SHA1(plain.txt)= 1a50fecc73c7ecd5e97661b184f69dca55e952fc

HMAC-SHA2-256(plain.txt)=

f1d6630a2e5662cb8da6f487c110ea65c4a08277f41b9e47ea2749050fb7df05

HMAC-MD5(plain.txt)= 674b7fbe4852ff83f5354c7d5464ff51

HMAC-MD5(long)(plain.txt)= 849afad3bd78b15ca7dbab3f44078893

```
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -md5 -hmac "abcdefg" plain.txt
HMAC-MD5(plain.txt)= ff72ad81a4bd7ea26153571bfff354604
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -sha1 -hmac "abcdefg" plain.txt
HMAC-SHA1(plain.txt)= 1a50fecc73c7ecd5e97661b184f69dca55e952fc
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -sha256 -hmac "abcdefg" plain.t
xt
HMAC-SHA2-256(plain.txt)= f1d6630a2e5662cb8da6f487c110ea65c4a08277f41b9e
47ea2749050fb7df05
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -md5 -hmac "a" plain.txt
HMAC-MD5(plain.txt)= 674b7fbe4852ff83f5354c7d5464ff51
mustakin@DESKTOP-V9TQVMG:~$ openssl dgst -md5 -hmac "aaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa" plain.txt
HMAC-MD5(plain.txt)= 849afad3bd78b15ca7dbab3f44078893
mustakin@DESKTOP-V9TQVMG:~$ █
```

#### Observations:

- No fixed key size required; HMAC accepts any length.
- Why? HMAC pads short keys to block size (e.g., 64 bytes for MD5) and hashes long keys to fit, ensuring security.

## Task7

Commands and generated hash values are shown below:

MD5 H1: *openssl dgst -md5 plain.txt*

SHA256 H1: *openssl dgst -sha256 plain.txt*

MD5 H2: *openssl dgst -md5 modified.txt*

SHA256 H2: *openssl dgst -sha256 modified.txt*

**Outputs:**

**MD5 H1:** a69a785968cd110f4880e16aa2760087

**MD5 H2:** 84d6475271f291fc25a3d253636a32e3

**SHA256 H1:**

a1668d6acc3a96ea04fb63621402daed006193348323c9718d816cce1359423c

**SHA256 H2:** e8b6f8216799253e8eb6a0edf21b53017a4f01a64986f0f67bc54088b93bea7c

**Observations:**

H1 and H2 are completely different (avalanche effect); a single bit change causes ~50% bit flips in hash.

The test file and the program to count how many bits are the same between H1 and H2 is shown in the github code:

[https://github.com/mustakinshvn/INS\\_Lab/blob/main/labManual3/task7.py](https://github.com/mustakinshvn/INS_Lab/blob/main/labManual3/task7.py)