



Information and Network Security

Course Code: SWE 430

Submitted By:

Ridwanur Rashid Siam
Registration No: 2019831060
Session: 2019-20
Department of Software Engineering, IICT, SUST.

Submitted To:

Partha Pratim Paul
Lecturer,
Institute of Information and Communication Technology, SUST.

Lab 3: Symmetric encryption & hashing

Task – 1: AES encryption using different modes

CBC:

```
openssl enc -aes-128-cbc -e -in test.txt -out encryptedCBC.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080910111213141516
```

```
openssl enc -aes-128-cbc -d -in encryptedCBC.txt -out decryptedCBC.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080910111213141516
```

CFB:

```
openssl enc -aria-128-cfb -e -in test.txt -out encryptedCFB.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080910111213141516
```

```
openssl enc -aria-128-cfb -d -in encryptedCFB.txt -out decryptedCFB.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080910111213141516
```

ECB:

```
openssl enc -aes-128-ecb -e -in test.txt -out encryptedECB.txt -K  
00112233445566778889aabbccddeeff
```

```
openssl enc -aes-128-ecb -d -in encryptedECB.txt -out decryptedECB.txt -K  
00112233445566778889aabbccddeeff
```

```
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aes-128-cbc -e -in test.txt]  
-out encryptedCBC.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708091  
0111213141516  
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aes-128-cbc -d -in encrypte]  
dCBC.txt -out decryptedCBC.txt -K 00112233445566778889aabbccddeeff -iv 01020304050  
607080910111213141516  
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aria-128-cfb -e -in test.tx]  
t -out encryptedCFB.txt -K 00112233445566778889aabbccddeeff -iv 010203040506070809  
10111213141516  
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aria-128-cfb -d -in encrypt]  
edCFB.txt -out decryptedCFB.txt -K 00112233445566778889aabbccddeeff -iv 0102030405  
0607080910111213141516  
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aes-128-ecb -e -in test.txt]  
-out encryptedECB.txt -K 00112233445566778889aabbccddeeff  
[(base) ridwan@Ridwans-MacBook-Air task1 % openssl enc -aes-128-ecb -d -in encrypte]  
dECB.txt -out decryptedECB.txt -K 00112233445566778889aabbccddeeff  
(base) ridwan@Ridwans-MacBook-Air task1 %
```

Task – 2: Encryption mode - ECB vs CBC

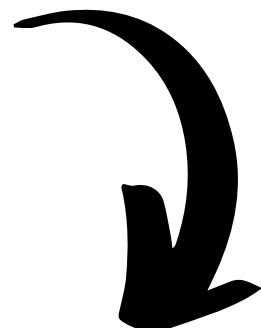
Encryption with ECB Mode

```
openssl enc -aes-128-ecb -e -in lab2.bmp -out encryptedPic.bmp -K  
00112233445566778889aabbccddeeff
```

lab.bmp - GHex

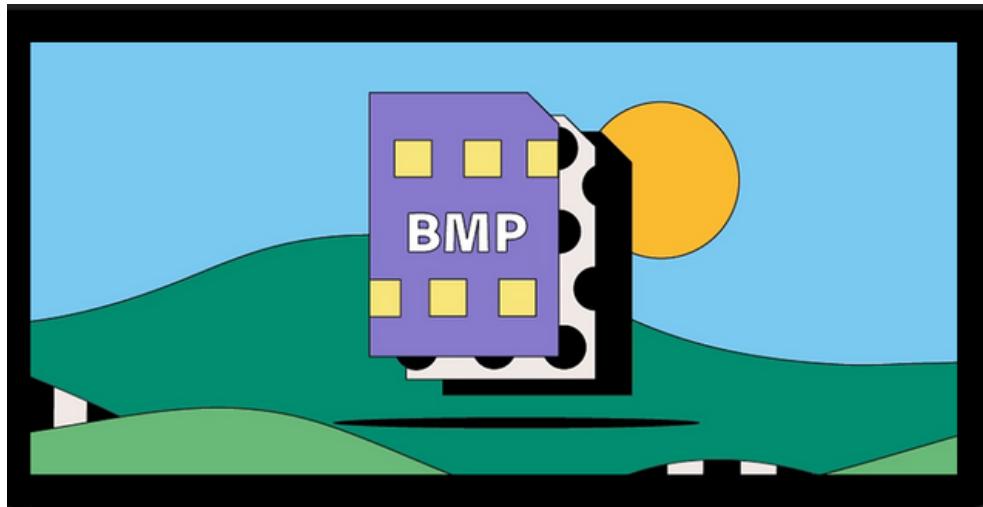
```
File Edit View Windows Help  
00000000 42 4D 72 07 0C 00 00 00 00 00 8A 00 00 00 00 7C 00 00 00 EE 02 00 BMr.....|.  
00000015 00 5E 01 00 00 01 00 18 00 00 00 00 00 E8 06 0C 00 00 00 00 00 ..^.....  
0000002A 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 FF 00 00 FF .....  
0000003F 00 00 00 00 00 FF 42 47 52 73 8F C2 F5 28 51 B8 1E 15 1E 85 .....BGRs...().  
00000054 EB 01 33 33 33 13 66 66 66 26 66 66 66 06 99 99 99 09 3D 0A D7 ..333.fff&ffff...  
00000069 03 28 5C 8F 32 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ..(\.2.....  
0000007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..x.ix  
00000093 78 BA 69 x.ix(ix(ix(ix(ix  
000000A8 78 BA 69 x.ix(ix(ix(ix(ix  
000000BD 78 BA 69 x.ix(ix(ix(ix(ix  
000000D2 78 BA 69 x.ix(ix(ix(ix(ix  
000000E7 78 BA 69 x.ix(ix(ix(ix(ix  
000000FC 78 BA 69 x.ix(ix(ix(ix(ix  
00000111 78 BA 69 x.ix(ix(ix(ix(ix  
00000126 78 BA 69 x.ix(ix(ix(ix(ix  
0000013B 78 BA 69 x.ix(ix(ix(ix(ix  
00000150 78 BA 69 x.ix(ix(ix(ix(ix  
00000165 78 BA 69 x.ix(ix(ix(ix(ix  
Signed 8 bit: 66 Signed 32 bit: 124931394 Hexadecimal: 42  
Unsigned 8 bit: 66 Unsigned 32 bit: 124931394 Octal: 102  
Signed 16 bit: 10778 Signed 64 bit: 51664528046 Binary: 010000  
Selected bytes are replaced
```

First 54 bytes are copied



encryptedImage.bmp - GHex

```
File Edit View Windows Help  
00000000 10 F9 01 7C 8D 24 B3 25 3C B7 F7 28 08 42 CE B3 8E C1 4B 5E B9 ...|.%.<..(.B...K^.  
00000015 AF 46 20 C0 31 5A 75 82 DB DE 0E EC 30 C6 22 04 F7 9C 52 53 57 .F.1Zu.....0."...RSW  
0000002A DF 25 30 8D 83 04 C4 AA 34 E6 EB DD D9 2A 00 F1 5B 02 BD 8C 0D .%0.....4....*...[....  
0000003F B8 A5 EE CA CA FA EF 73 FC 7D 0C 20 EF 21 31 CF E8 F7 A8 59 AA .....s.}. !1....Y.  
00000054 AD 38 6E 44 AF C7 08 53 26 73 C0 F6 9E 65 2F 76 DO 05 1A 87 B9 .8nD...S&s...e/v....  
00000069 2E 0A D2 AC 92 72 EE 30 95 82 69 88 79 8E A5 91 79 37 F5 34 DC .....r..i.y..y7.4.  
0000007E 0A 50 47 F6 B7 6E BA 3F C7 0D FC FA 9E 42 E7 6B F9 3F 68 5E E7 .PG..n?....B.k.?h^.  
00000093 D2 0E 33 D7 A6 6B A1 1A 40 5E BF 38 3B 48 E7 58 26 FC 23 D5 88 ..3..k.@";H.X&#..  
000000A8 37 DA 4D 61 C0 3F 98 81 51 9C 83 42 04 3A 79 5E 37 B6 3A 9C DE 7.Ma.?..Q..B.:y^7...  
000000BD 6B B8 C7 68 5F E7 D2 0F 33 D7 A6 6B A1 1A 40 5E BF 38 3B 48 E7 k..h^..3..k..@^.8;H.  
51 9C 83 42 04 3A 79 X&#.7.Ma.?..Q..B.:y  
33 D7 A6 6B A1 1A 40 ^7...k..h^..3..k..@  
4D 61 C0 3F 98 81 51 ^..8;H.X&#.7.Ma.?..Q..B.:y  
C7 68 5E E7 D2 0E 33 ..B.:y^7...k..h^..3..  
Find String  
08 42 CE B3 8E C1 4B 5E B9 AF 46 20 .B....K^..F  
C0 31 5A 75 82 DB DE 0E EC 30 C6 22 .1Zu.....0.  
04 F7 9C 52 53 57 DF 25 30 8D 83 04 ...RSW.%0.  
C4 AA 34 E6 EB DD D9 0 ..4....  
Replace With  
00 00 7C 00 00 00 EE 02 00 00 5E 01 ..|.....  
00 00 01 00 18 00 00 00 00 00 E8 06 .....  
0C 00 00 00 00 00 00 00 00 00 00 00 .....  
00 00 00 00 00 00 00 00 00 00 00 00 .....  
Find next Replace Replace All Cancel  
Show unsigned and float as hexadecimal
```



Initial Phase



After ECB encryption

Encryption with CBC Mode

```
openssl enc -aes-128-cbc -e -in lab.bmp -out encryptedImageCBC.bmp -K
00112233445566778889aabbccddeeff -iv 01020304050607080102030405060708
```

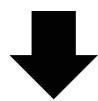
encryptedImage.bmp - GHex

00000000	42 4D 72 07 0C 00 00 00 00 00 00 8A 00 00 00 00 7C 00 00 00 EE	BMr.....
00000013	02 00 00 5E 01 00 00 01 00 18 00 00 00 00 00 E8 06 0C 00	...^.....
00000026	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2A 00*
00000039	F1 5B 02 BD 8C 0D B8 A5 EE CA CA FA EF 73 FC 7D 0C 20 EF	.[.....s..].

encryptedImageCBC.bmp - GHex

00000000	E4 B5 BD 2B 47 BA 11 C7 82 AA 26 C9 9A FA 22 7A E1 5B 5C	...+G.....&....z.[\
00000013	2B 90 9B 2F BF 13 E2 19 41 AB 57 6D 2A DD 6D EC CD 8E 5C	+.../.A.Wm*.m...\\
00000026	55 AF 6F 46 E8 D5 38 95 7D 45 8C 51 CA 83 C8 91 67 BA C5	U.of..8.)E.Q....g..
00000039	26 7F 4B 5C 39 32 F6 C7 6A 52 58 E7 D7 EE 44 D0 B7 42 71	&.K\92..jRX...D..Bq
0000004C	8D E4 24 31 5B 43 55 DA 93 FB 05 73 07 48 7E E4 2D A0 C8	..\$1[CU....s.H-...]
0000005F	06 EA F7 9D F2 67 B9 95 94 C9 56 6B FC 71 23 05 83 5D 3Ag....V.k.q#..]:
00000072	9F CF E0 3E 6C 71 30 0D 20 3F A9 01 F2 FD FE 88 7E 8E 56	...>lq0. ?.....~.V
00000085	B4 F8 82 06 1D EC EE 12 1F 82 31 EB 42 C5 E0 69 3B 9C 8C1.B..i...
00000098	E3 3E 58 06 08 28 13 A4 DD 34 09 05 92 3E 68 89 B7 24 9B	.>X..(...4...>h..\$.
000000AB	8A 08 EA BF F4 21 A8 09 AE B0 36 04 1C 49 6F 7D F1 51 D9!....6..Io}.Q.

Signed 8 bit: -28 Signed 32 bit: 733853156 Hexadecimal: E4
 Unsigned 8 bit: 228 Unsigned 32 bit: 733853156 Octal: 344
 Signed 16 bit: -18972 Signed 64 bit: -4102292970718513692 Binary: 11100100
 Unsigned 16 bit: 46564 Unsigned 64 bit: 14344451102001037024 Stream Length: 8

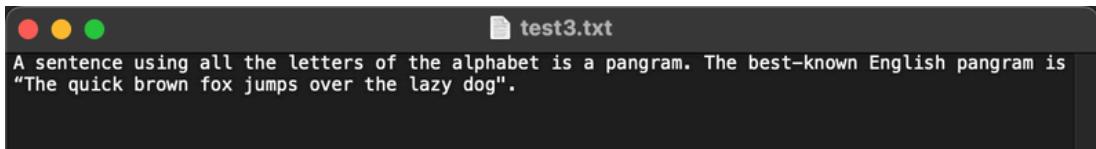


After CBC Mode Conversion

Task – 3: Encryption mode – corrupted cipher text

In the test3 .txt file, we added a text:

A sentence using all the letters of the alphabet is a pangram. The best-known English pangram is “The quick brown fox jumps over the lazy dog”.



Encryption with CBC:

```
openssl enc -aes-128-cbc -e -in test3.txt -out text2Encrypt.txt -K  
00112233445566778889aabcccddeeff -iv 0102030405060708
```

After encrypting the test3.txt file, we opened the encrypted file named text2Encrypt.txt with **ghex**. Then we changed the **30th byte** and got the output.

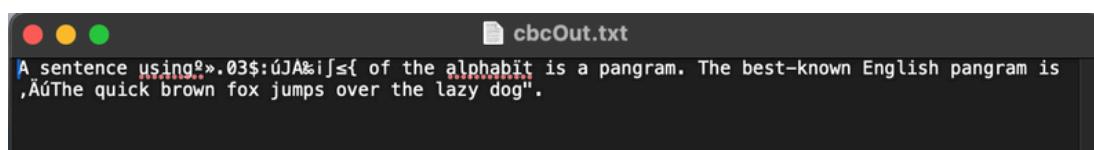


Decryption with CBC:

```
openssl enc -aes-128-cbc -d -in text2Encrypt.txt -out cbcOut.txt -K  
00112233445566778889aabcccddeeff -iv 0102030405060708
```

Output:

A sentence using all the letters of the alphabet is a pangram. The best-known English pangram is “The quick brown fox jumps over the lazy dog”.



Encryption with ECB:

```
openssl enc -aes-128-ecb -e -in test3.txt -out test3ECBEncrypt.txt -K  
00112233445566778889aabccddeeff
```

After encrypting the test3.txt file, we opened the encrypted file named test3ECBEncrypt.txt with **ghex test3ECBEncrypt.txt**. Then we changed the 30th byte from E to 7 and got the output.

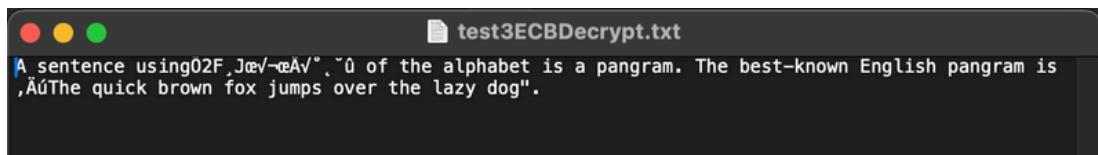


Decryption with ECB:

```
openssl enc -aes-128-ecb -d -in test3ECBEncrypt.txt -out test3ECBDecrypt.txt -K  
00112233445566778889aabccddeeff
```

Output:

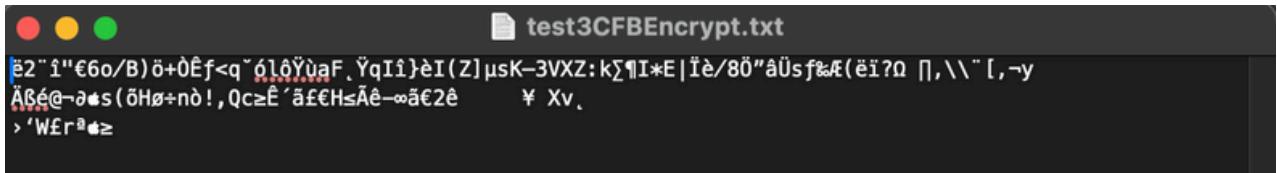
A sentence using Ÿ2F, Jœv/¬œAv°, „û of the alphabet is a pangram. The best-known English pangram is ,ÄúThe quick brown fox jumps over the lazy dog".



Encryption with CFB:

```
openssl enc -aria-128-cfb -e -in test3.txt -out test3CFBEncrypt.txt -K  
00112233445566778889aabccddeeff -iv 0102030405060708
```

After encrypting the test3.txt file, we opened the encrypted file named test3CFBEncrypt.txt with **ghex test3CFBEncrypt.txt**. Then we changed the 30th byte from D to 9 and got the output.

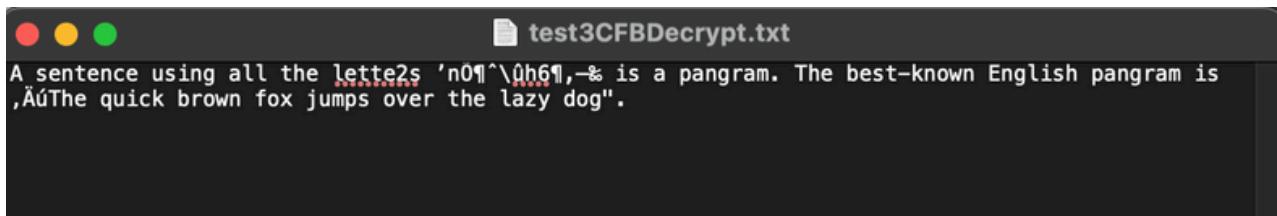


Decryption with CFB:

```
openssl enc -aria-128-cfb -d -in test3CFBEncrypt.txt -out test3CFBDecrypt.txt -K  
00112233445566778889aabccddeeff -iv 0102030405060708
```

Output:

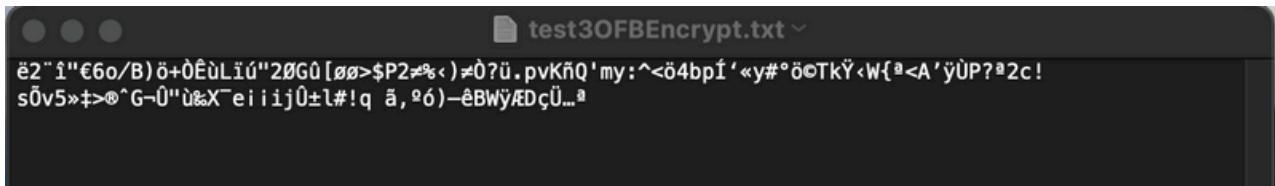
A sentence using all the letters 'nÖ¶|^ûh6¶,-‰ is a pangram. The best-known English pangram is ,ÄúThe quick brown fox jumps over the lazy dog".



Encryption with OFB:

```
openssl enc -aria-128-ofb -e -in test3.txt -out test3OFBEncrypt.txt -K 00112233445566778889aabbcdddeeff -iv 0102030405060708
```

After encrypting the test3.txt file, we opened the encrypted file named test3OFBEncrypt.txt with ghex test3OFBEncrypt.txt. Then we changed the 30th byte from 2 to 9 and got the output.

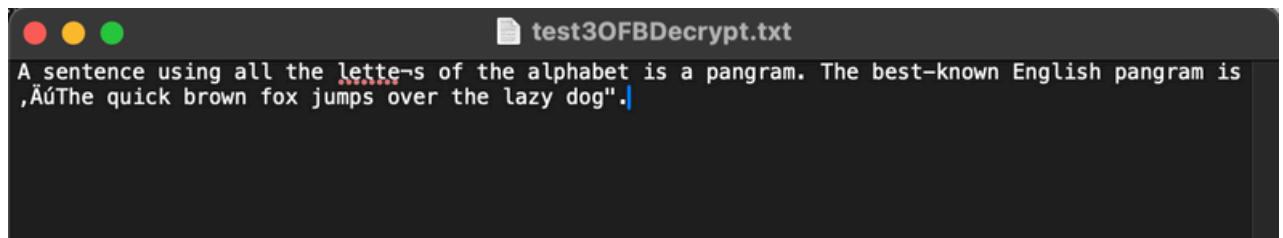


Decryption with OFB:

```
openssl enc -aria-128-ofb -d -in test3OFBEncrypt.txt -out test3OFBDecrypt.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Output:

A sentence using all the letters of the alphabet is a pangram. The best-known English pangram is "The quick brown fox jumps over the lazy dog".



Question Answer:

How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Explain why? What are the differences?

In ECB mode, each block of plaintext is encrypted independently. This means that if one block is corrupted, it will only affect that particular block and not the others. Therefore, in ECB mode, you could potentially recover all the information from the unaffected blocks. In my case, I found 16byte of data to be corrupted consecutively. But sometimes, if a block containing crucial information is corrupted, the data in that block will be lost, potentially leading to partial or incomplete recovery of the file, which happened with my case, where another 2 bytes of data wasn't recovered.

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before encryption. If a block in the middle of the file is corrupted, it will affect the decryption of all subsequent blocks. However, the corruption in one block will only propagate up to the next block; after that, the decryption will become correct again. So, you could potentially recover all the information up to the point of corruption. In my case, I found 32 bytes of data to be corrupted.

In CFB mode, ciphertext is fed back into the encryption algorithm to create the next block of ciphertext. If a block in the middle of the file is corrupted, it will affect the decryption of all subsequent blocks similar to CBC mode. However, like CBC mode, the corruption in one block will only propagate up to the next block. Therefore, you could potentially recover all the information up to the point of corruption. In my case, I found 32 byte of data to be corrupted sequentially.

In OFB mode, the encryption of the previous block is XORed with the plaintext to create the ciphertext. Corruption in one block will only affect the decryption of that block and will not propagate to subsequent blocks. So, similar to ECB mode, you could potentially recover all the information from the unaffected blocks. In my case, I found 16 bytes of data to be corrupted in a serial.

Task – 4: Padding

- **ECB, CBC:** These modes typically require padding because they process data in fixed-size blocks. If the plaintext size isn't a multiple of the block size, incomplete blocks will occur, leading to errors during decryption.

The image shows three terminal windows side-by-side, each displaying a file's content. The top window is titled 'test4.txt' and contains the following text:

```
48 69 20 74 68 65 72 65 2E 20 4D 79 20 6E 61 6D 65 20 69 73 20 53 69 6
6D 2E 20 4E 69 63 65 20 74 6F 20 6D 65 65 74 20 79 6F 75 2E 20 4F 75 7
20 63 6F 75 72 73 65 20 69 6E 73 74 72 75 63 74 6F 72 20 68 61 73 20 6
69 76 65 6E 20 75 73 20 61 20 74 6F 75 67 68 20 74 61 73 6B 20 74 6F 2
63 6F 6D 70 6C 65 74 65 2E 20 57 69 6C 6C 20 73 65 65 20 74 6F 20 79 6
75 20 6C 61 74 65 72 2E 20 42 79 65 2E
Hi there. My name is Si
m. Nice to meet you. Ou
course instructor has !
iven us a tough task to
complete. Will see to yo
u later. Bye.
```

The middle window is titled 'test4ECBEncrypt.txt' and contains the following hex dump:

```
41 70 D6 07 87 08 92 F0 BE 50 21 D7 A4 69 5D FC 39 C3 89 21 62 23 42 5
B7 87 B6 6E 15 5D A9 9D 89 90 B9 B8 29 F6 DA D8 48 1F 74 94 BF 01 82 5
ED 66 93 C4 35 E3 3D CB 50 BF F7 2D CA 44 B0 2A C9 54 39 CC 1B 2A C4 9
F1 BA A3 35 4C AF 4B 19 52 2A A7 BF 57 B4 30 9F FA DE EA 52 C9 62 34 9
60 5C 30 93 E5 7F 80 EC B7 0E 16 EF 72 74 12 7B D5 5C A3 CD BF B5 71 E
5E CF 64 84 7C 69 65 26 C6 65 CE 84 BC 40 47 6A 91 24 26 EC 05 A6 CD A
5 [REDACTED]
```

The bottom window is titled 'test4Encrypt.txt' and contains the following hex dump:

```
9B 72 94 6D 15 55 AB 66 8F 57 33 1E 78 DB 3E 73 E9 2B EA A8 2E 61 94 E
AE C0 89 F0 18 2D A2 84 A5 2C FF AF 9D 6A 54 C9 3A BE 26 38 C7 8A F
B5 00 ED 1C 63 F6 B3 6A E4 C1 15 DF 04 81 C4 24 E7 00 DE AD 6C 58 02 E
EE 4A 7C A4 60 D2 AA 1D 83 71 6E B8 03 8D C8 79 23 80 A7 22 68 72 8B 8
74 6A 0E BE 26 BB 7D B9 7C 93 7D B3 1D EF 9F 5C 70 40 04 1D 58 0F E2 4
34 14 D5 F9 31 3F 5A 67 59 0C 96 B1 45 A2 E5 87 65 31 41 3A B9 1A 17 1
4 [REDACTED]
```

Generating a encrypted file with ECB mode

```
openssl enc -aes-128-ecb -e -in test4.txt -out test4ECBEncrypt.txt -K
00112233445566778889aabbcdddeeff
```

Generating a encrypted file with CBC mode

```
openssl enc -aes-128-cbc -e -in test4.txt -out test4Encrypt.txt -K  
00112233445566778889aabbccddeeff -iv 0102030405060708
```

In the above section, we can see that in the ECB encrypted file, the number of bytes have increased compared to the main test4.txt file. This indicates padding was included in it.

- **CFB, OFB:** These modes operate on a bit-by-bit or byte-by-byte basis using the cipher output as a pseudo-random stream. They do not inherently require padding as they can process any size of plaintext without needing complete blocks.

Generating a encrypted file with CFB mode

```
openssl enc -aria-128-cfb -e -in test4.txt -out test4CFBEncrypt.txt -K  
00112233445566778889aabbccddeeff -iv 0102030405060708
```

Generating a encrypted file with OFB mode

```
openssl enc -aria-128-ofb -e -in test4.txt -out test4OFBEncrypt.txt -K  
00112233445566778889aabbccddeeff -iv 0102030405060708
```

The screenshot displays three hex editor windows side-by-side, each showing a different encryption mode applied to the same input file.

- test4.txt:** The original text file containing the message "Hi there. My name is Si...".
- test4OFBEncrypt.txt:** The output of OFB mode. The byte sequence is identical to the ECB output but contains non-printable characters at the end.
- test4CFBEncrypt.txt:** The output of CFB mode. The byte sequence is identical to the ECB output but contains non-printable characters at the end.

Each window shows the following offset and content:

Offset	test4.txt	test4OFBEncrypt.txt	test4CFBEncrypt.txt
00000000	A8 69 20 74 68 65 72 65 2E 20 4D 79 20 6E 61 6D 65 20 69 73 20 53 69 6	A8 7B FF 1E 92 33 CC 3D 22 9F 2F 25 C9 2C FE EC D8 0D 90 83 22 15 AE 4	A8 7B FF 1E 92 33 CC 3D 22 9F 2F 25 C9 2C FE EC AB 43 17 7B EB 40 F4 2
00000018	6D 2E 20 4E 69 63 65 20 74 6F 20 6D 65 65 74 20 79 6F 75 2E 20 4F 75 7	D3 45 1E 85 A2 38 33 03 66 AD 63 91 38 A0 E0 3F 87 2D 75 30 53 66 11 9	59 E1 5D F2 0B 11 3A 48 D8 ED 0A C8 51 20 7E FC B9 79 4A BF D5 EE 3A 9
00000030	20 63 6F 75 72 73 65 20 69 6E 73 74 72 75 63 74 6F 72 20 68 61 73 20 6	51 2D 71 2A 6A 69 4B 7D 01 93 35 77 6F B1 97 E7 7E 02 23 AB 9E A9 00 4	44 5B A6 34 59 1B 1E D4 44 17 A7 A0 CF BB 55 D5 5B 6C 11 16 15 FF 54 A
00000048	69 76 65 6E 20 75 73 20 61 20 74 6F 75 67 68 20 74 61 73 6B 20 74 6F 2	69 C1 D6 4E 35 EE 0A 0F D3 94 E9 4C 22 FC 2A 22 3B 75 CC 7C 78 9C E6 6	09 3A 38 35 E6 11 CC 4A 1D BB F2 C5 2B BF EE FC 9D 23 7A 17 72 22 A9 A
00000060	63 6F 6D 70 6C 65 74 65 2E 20 57 69 6C 6C 20 73 65 65 20 74 6F 20 79 6	EB 9A 1B AB FA FE 1E 67 C2 55 DA 52 FF 53 5B 64 CB 1D 8F 3D 63 BC B0 2	AA FC 6B CC 1C 62 B6 E1 9E F9 83 62 8B 2A B8 85 26 FC 4A 67 2A 01 21 3
00000078	75 20 6C 61 74 65 72 2E 20 42 79 65 2E	3C 4B 20 60 27 CE FF E4 58 A7 70 C0 56	3B BD 3F 5F 03 1A 5E BD B4 C0 89 C8 69

Each window also displays the file path: /Users/ridwan/Documents/security-labs/task4.

In the above section, we can see that in the ECB encrypted file, the number of bytes haven't changed compared to the main test4.txt file. This indicates padding was not included in it.

Task – 5: Generating message digest

A test5.txt file has been created for encryption.

```
openssl dgst -md5 test5.txt  
MD5(test5.txt)= 56bd5e088857152c415fd6ead4a1781c
```

```
[(base) ridwan@Ridwans-MacBook-Air task5 % openssl dgst -md5 test5.txt  
MD5(test5.txt)= 56bd5e088857152c415fd6ead4a1781c]
```

```
openssl dgst -sha1 test5.txt  
SHA1(test5.txt)= 1f7960bd63c5c0a406e1a7b50f514f164fb55a3f
```

```
[(base) ridwan@Ridwans-MacBook-Air task5 % openssl dgst -sha1 test5.txt  
SHA1(test5.txt)= 1f7960bd63c5c0a406e1a7b50f514f164fb55a3f]
```

```
openssl dgst -sha256 test5.txt  
SHA2-256(test5.txt)=  
eb15cbe2c7b133f3fb424bba67c0469db1ea21424b3dbd8328774789a78ef91f
```

```
[(base) ridwan@Ridwans-MacBook-Air task5 % openssl dgst -sha256 test5.txt  
SHA2-256(test5.txt)= eb15cbe2c7b133f3fb424bba67c0469db1ea21424b3dbd8328774789a78ef  
91f]
```

Task – 6: Keyed hash and HMAC

The commands and their corresponding keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 :

```
openssl dgst -md5 -hmac "siam_valo_chele" test6.txt  
HMAC-MD5(test6.txt)= 8a846e0ad07f2363189284023b8b76b6
```

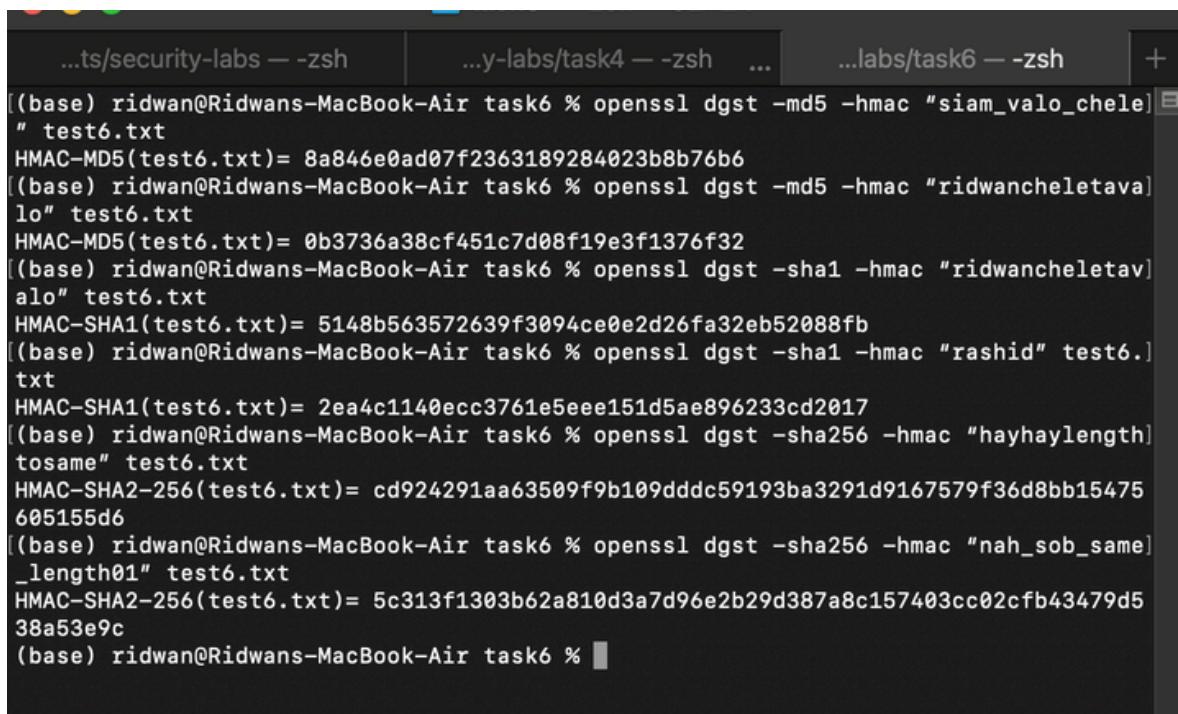
```
openssl dgst -md5 -hmac "ridwancheletaval" test6.txt  
HMAC-MD5(test6.txt)= 0b3736a38cf451c7d08f19e3f1376f32
```

```
openssl dgst -sha1 -hmac "ridwancheletaval" test6.txt  
HMAC-SHA1(test6.txt)= 5148b563572639f3094ce0e2d26fa32eb52088fb
```

```
openssl dgst -sha1 -hmac "rashid" test6.txt  
HMAC-SHA1(test6.txt)= 2ea4c1140ecc3761e5eee151d5ae896233cd2017
```

```
openssl dgst -sha256 -hmac "hayhaylengthtosame" test6.txt  
HMAC-SHA2-256(test6.txt)=  
cd924291aa63509f9b109dddc59193ba3291d9167579f36d8bb15475605155d6
```

```
openssl dgst -sha256 -hmac "nah_sob_same_length01" test6.txt  
HMAC-SHA2-256(test6.txt)=  
5c313f1303b62a810d3a7d96e2b29d387a8c157403cc02cfb43479d538a53e9c
```



```
...ts/security-labs --zsh ...y-labs/task4 --zsh ... ...labs/task6 --zsh +  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -md5 -hmac "siam_valo_chele"]  
" test6.txt  
HMAC-MD5(test6.txt)= 8a846e0ad07f2363189284023b8b76b6  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -md5 -hmac "ridwancheletava]  
lo" test6.txt  
HMAC-MD5(test6.txt)= 0b3736a38cf451c7d08f19e3f1376f32  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -sha1 -hmac "ridwancheletav]  
alo" test6.txt  
HMAC-SHA1(test6.txt)= 5148b563572639f3094ce0e2d26fa32eb52088fb  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -sha1 -hmac "rashid" test6.]  
txt  
HMAC-SHA1(test6.txt)= 2ea4c1140ecc3761e5eee151d5ae896233cd2017  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -sha256 -hmac "hayhaylength]  
tosame" test6.txt  
HMAC-SHA2-256(test6.txt)= cd924291aa63509f9b109dddc59193ba3291d9167579f36d8bb15475  
605155d6  
[(base) ridwan@Ridwans-MacBook-Air task6 % openssl dgst -sha256 -hmac "nah_sob_same]  
_length01" test6.txt  
HMAC-SHA2-256(test6.txt)= 5c313f1303b62a810d3a7d96e2b29d387a8c157403cc02cfb43479d5  
38a53e9c  
(base) ridwan@Ridwans-MacBook-Air task6 %
```

Question: Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

HMAC does not require a key with a fixed size. It can accept keys of any size. However, the security of HMAC depends on the strength of the key. Using a longer key generally increases security, but it's important to choose a key length that is appropriate for the specific cryptographic algorithm being used (e.g., MD5, SHA-1, SHA-256). For example, using a key longer than the block size of the hash function does not necessarily increase security further.

Task – 7: Keyed hash and HMAC

First of all, a new text file named test7.txt has been created. Then several hash values using two of the algorithms are generated.

```
openssl dgst -md5 test7.txt
MD5(test7.txt)= 54f126280be51ff993c11424930c8b52
```

```
[(base) ridwan@Ridwans-MacBook-Air task7 % openssl dgst -md5 test7.txt      ]
MD5(test7.txt)= 54f126280be51ff993c11424930c8b52
[(base) ridwan@Ridwans-MacBook-Air task7 % ghex test7.txt                      ]
```

When we flipped a bit of the input file, the hash value has been changed.

```
41 6E 64 20 69 66 20 61 20 64 6F 75 62 6C 65 2D 64 65 63 6B 65 7
20 64 69 65 20 62 79 20 79 6F 75 72 20 73 69 64 65 20 69 73 20 7
0A 4
54 6
69 7
A1 6E 64 20 69 66 20 61 20 64 6F 75 62 6C 65 2D 64 65
20 64 69 65 20 62 79 20 79 6F 75 72 20 73 69 64 65 20
0A 41 6E 64 20 69 66 20 61 20 74 65 6E 20 74 6F 6E 20
54 6F 20 64 69 65 20 62 79 20 79 6F 75 72 20 73 69 64
69 76 69 6C 65 67 65 20 69 73 20 6D 69 6E 65 2E
```

```
openssl dgst -md5 test7.txt  
MD5(test7.txt)= 681f9aede2f03be7940fc5f9028c0ac6
```

```
[(base) ridwan@Ridwans-MacBook-Air task7 % openssl dgst -md5 test7.txt ]  
MD5(test7.txt)= 681f9aede2f03be7940fc5f9028c0ac6  
[(base) ridwan@Ridwans-MacBook-Air task7 % ghex test7.txt ]
```

Likewise, it gets changed even if we use different algorithms for hash value.

Before flipping bit:

```
openssl dgst -sha256 test7.txt
```

```
SHA2-256(test7.txt)= aeae96038f7b892bfca07225e624bf2b6e9772675477c0aa4108b1032436ee90
```

After flipping bit:

```
openssl dgst -sha256 test7.txt
```

```
SHA2-256(test7.txt)= a64f1d01ea7182224dfaf31a624ab38281711e8c10442d663718904d36c1fdb4
```

```
[(base) ridwan@Ridwans-MacBook-Air task7 % openssl dgst -sha256 test7.txt ]  
SHA2-256(test7.txt)= aeae96038f7b892bfca07225e624bf2b6e9772675477c0aa4108b1  
e90
```

```
[(base) ridwan@Ridwans-MacBook-Air task7 % openssl dgst -sha256 test7.txt ]  
SHA2-256(test7.txt)= a64f1d01ea7182224dfaf31a624ab38281711e8c10442d663718904d36c1f  
db4
```

One-way hash functions are designed to be highly sensitive to changes in the input. Modifying even a single bit in the message significantly alters the internal calculations, resulting in a completely different hash value.

Code for comparing bits difference between two hexadecimal hashed strings are given below:

```
#include <bits/stdc++.h>  
int count_matching_bits(const string& H1, const string& H2) {  
    string bin_H1 = "", bin_H2 = "";  
    for(char c : H1) {  
        switch(c) {  
            case '0': bin_H1 += "0000"; break;  
            case '1': bin_H1 += "0001"; break;  
            case '2': bin_H1 += "0010"; break;  
            case '3': bin_H1 += "0011"; break;  
            case '4': bin_H1 += "0100"; break;  
            case '5': bin_H1 += "0101"; break;  
            case '6': bin_H1 += "0110"; break;
```

```

        case '7': bin_H1 += "0111"; break;
        case '8': bin_H1 += "1000"; break;
        case '9': bin_H1 += "1001"; break;
        case 'a': case 'A': bin_H1 += "1010"; break;
        case 'b': case 'B': bin_H1 += "1011"; break;
        case 'c': case 'C': bin_H1 += "1100"; break;
        case 'd': case 'D': bin_H1 += "1101"; break;
        case 'e': case 'E': bin_H1 += "1110"; break;
        case 'f': case 'F': bin_H1 += "1111"; break;
        default: cout << "Invalid hexadecimal character: " << c << endl;
    }
}

for(char c : H2) {
    switch(c) {
        case '0': bin_H2 += "0000"; break;
        case '1': bin_H2 += "0001"; break;
        case '2': bin_H2 += "0010"; break;
        case '3': bin_H2 += "0011"; break;
        case '4': bin_H2 += "0100"; break;
        case '5': bin_H2 += "0101"; break;
        case '6': bin_H2 += "0110"; break;
        case '7': bin_H2 += "0111"; break;
        case '8': bin_H2 += "1000"; break;
        case '9': bin_H2 += "1001"; break;
        case 'a': case 'A': bin_H2 += "1010"; break;
        case 'b': case 'B': bin_H2 += "1011"; break;
        case 'c': case 'C': bin_H2 += "1100"; break;
        case 'd': case 'D': bin_H2 += "1101"; break;
        case 'e': case 'E': bin_H2 += "1110"; break;
        case 'f': case 'F': bin_H2 += "1111"; break;
        default: cout << "Invalid hexadecimal character: " << c << endl;
    }
}

for(size_t i = 0; i < bin_H1.size() && i < bin_H2.size(); ++i) {
    if(bin_H1[i] == bin_H2[i]) {
        matching_bits++;
    }
}

return matching_bits;
}

int main() {
    string h1, h2;
    cin >> h1 >> h2;

    int matching_bits = count_matching_bits(h1, h2);
    cout << "Number of matching bits: " << matching_bits << endl;

    return 0;
}

```