Network Security

Cryptography Lab

Muhammad Raahim Khan

21100157

➢ Task 2:

Commands used to encrypt were as follows:

- openssl enc -aes-128-ecb -e -in pic_original.bmp -out ecb.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- openssl enc -aes-128-cbc -e -in pic_original.bmp -out cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Outputs were as follows:

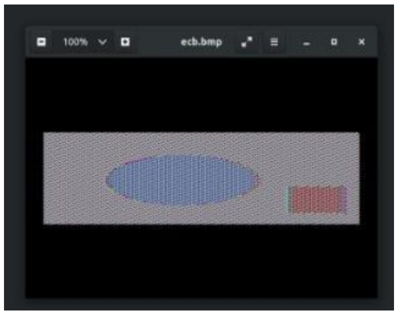


Figure 1: ecb.bmp

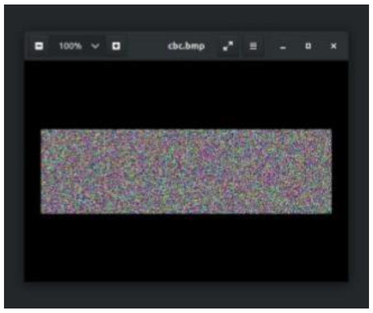


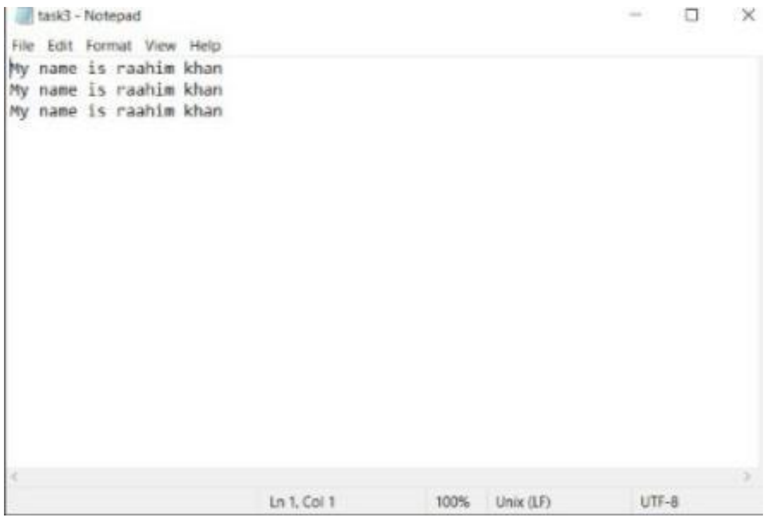
Figure 2: cbc.bmp

CipherType	Observation	Explanation
ECB	Information about original picture from encrypted picture can be derived since shapes are visible.	In ECB mode, blocks of the image that are the same color, and are therefore identical, gets encrypted in the same way. This allows an encryption of a sequence of blocks to reveal a surprising amount of information.

CBC	Encrypted image contains randomness hence useful information about original picture from encrypted picture cannot be derived.	In CBC mode, if identical blocks appear at different places in the input sequence, then they are very likely to have different encryptions in the output sequence. Hence, it becomes difficult to determine patterns in an encryption that is done using CBC mode.

- **<u>Task3:</u>**

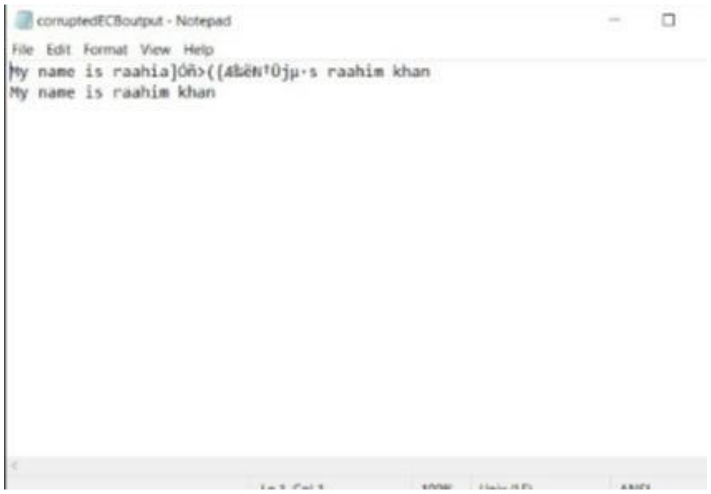
Plaintext file is as follows:



```
My name is raahim khan
My name is raahim khan
My name is raahim khan
```

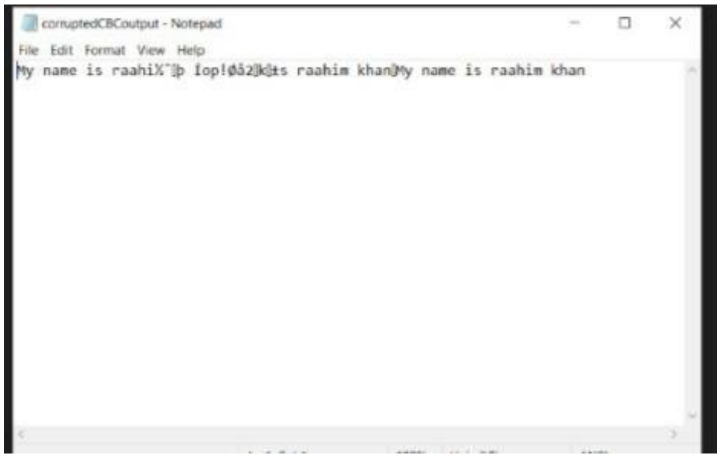
Plaintext file

I encrypted the above plaintext file using ECB, CBC, CFB, and OFB modes. Then I corrupted the single bit of the 30th byte in each of the encrypted file I got. Finally, I decrypted those corrupted (encrypted) files and the outputs were as follows:



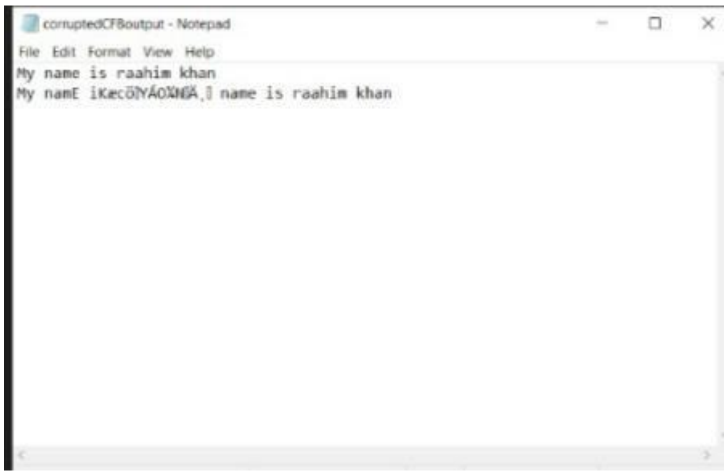
```
My name is raahia]Óñ>(ƒÆ‰ëN†Ûjµ·s raahim khan
My name is raahim khan
```

Decrypted ECB corrupted file



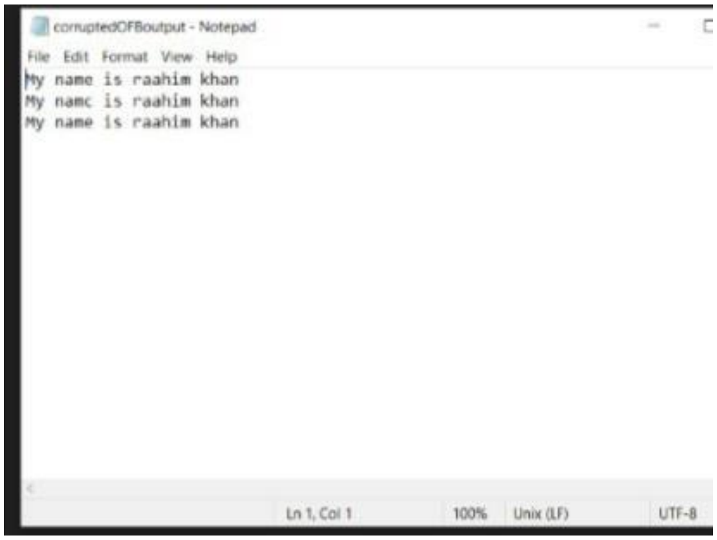
```
My name is raahiX¯þ Íop!Øå2k±s raahim khanMy name is raahim khan
```

Decrypted CBC corrupted file



```
My name is raahim khan
My namE iKæcõYÁO¾NÄ, name is raahim khan
```

Decrypted CFB corrupted file


```
My name is raahim khan
My namc is raahim khan
My name is raahim khan
```

Decrypted OFB corrupted file

CipherType	**Information Recovery**	**Observations after the task**	**Implications**
ECB	Except for the block of ciphertext which contains the corrupted byte, all of the other information should be recoverable.	All the information is fine except for the entire 16-byte cipher block that contained the corrupted byte.	This happened because different message is deciphered when used with the key. Reason is the corrupted bit. This cypher mode is not good for recovering corrupted data as the entire 16-byte block is corrupted.
CBC	Block containing the corrupted byte and the block directly after that block are not recoverable. All of the other blocks can be decrypted	All of the information is fine except for the entire 16-byte cipher block that contained the corrupted byte. Furthermore, the next byte also has minor changes as well.	This happened because the key cannot directly decrypt the block that contained the corrupted byte. Furthermore, the next cipher block requires this block as well during decryption. Hence, that particular bit of the next block is also corrupted. This cypher mode hence is also not good for data recovery from corrupted data.

CFB	Block containing the corrupted byte and the block directly after that block are not recoverable. All of the other blocks can be decrypted	All of the information is fine except for the entire 16-byte cipher block that occurs after the block that contained the corrupted byte. Furthermore, the corrupted bit is still corrupted in decryption.	In CFB mode, current block uses the previous block for decryption. Hence, the block containing the corrupted byte is not altered, but the block after it is corrupted. This cypher mode hence is also not good for data recovery from corrupted data.
OFB	Entire block will not be corrupted. Only the exact string containing the corrupted byte will be corrupted. Apart from this, all of the blocks are recoverable.	All of the information is fine except for the corrupted bit.	OFB mode uses the previous initialization vector. It does this to generate a vector which is then used to perform XOR on the text block. This explains why only the corrupted bit is affected and the rest of the information is fine since the vector is uncorrupted. This cypher mode hence is recommended and good for data recovery from corrupted data out of all the modes discussed in the table.

- **<u>Task 4:</u>**

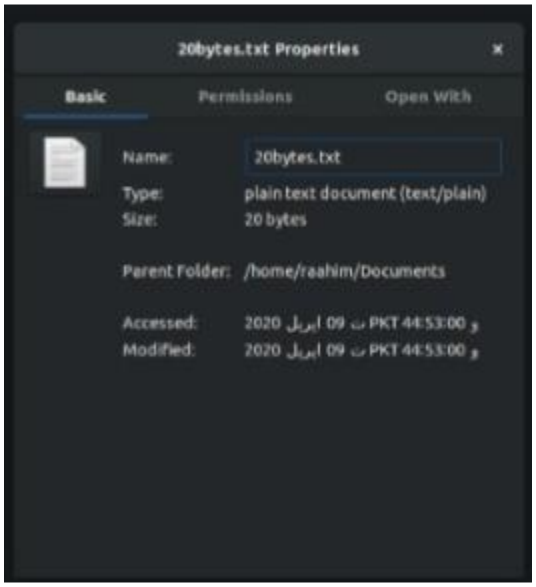
  - **4.1**

    - **<u>Hypothesis:</u>**

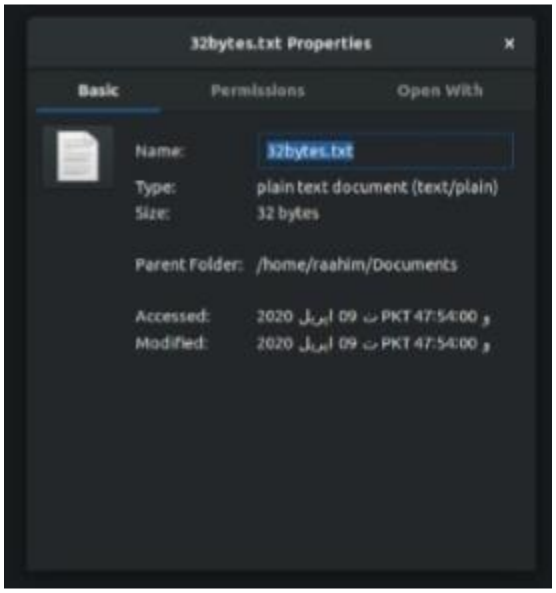
      Block cipher algorithms such as AES require their input to be an exact multiple of block size when ECB and CBC modes are used. Hence, if plaintext that is to be encrypted is not an exact multiple, padding is required. Also, while decrypting, the one who is decrypting needs to know how to remove the padding in an unambiguous manner. OpenSSL uses PKCS5 standard for its padding. I am going to verify this in my experiment.

    - **<u>Method:</u>**

      First of all, I created two files, 20 bytes and 32 bytes long.



Name: 20bytes.txt
Type: plain text document (text/plain)
Size: 20 bytes
Parent Folder: /home/raahim/Documents
Accessed: 2020 اپریل 09 ت PKT 44:53:00 و
Modified: 2020 اپریل 09 ت PKT 44:53:00 و

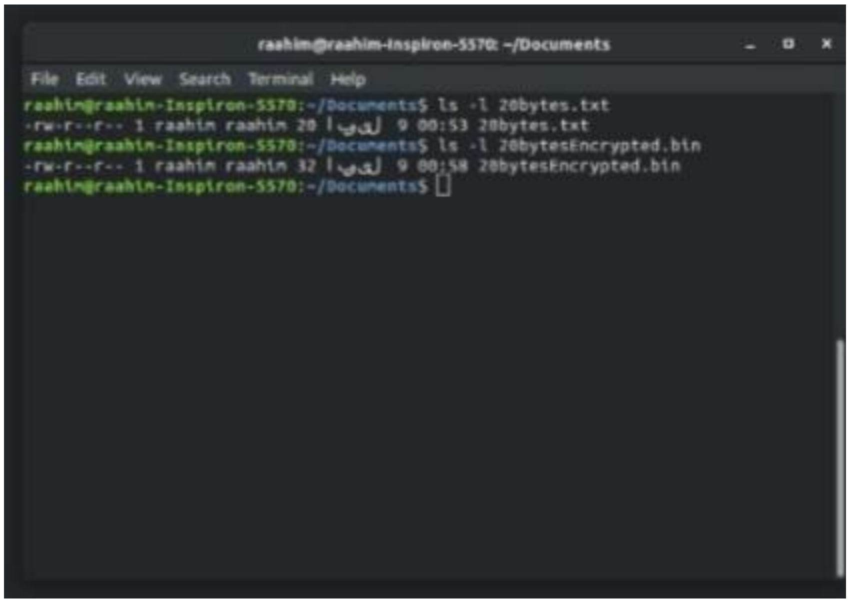


```
32bytes.txt Properties

Basic    Permissions    Open With

Name:           32bytes.txt
Type:           plain text document (text/plain)
Size:           32 bytes

Parent Folder:  /home/raahim/Documents

Accessed:       2020 اپریل 09 ت PKT 47:54:00 و
Modified:       2020 اپریل 09 ت PKT 47:54:00 و
```

I encrypted the plaintext files using aes-128-cbc encryption mode using the following commands: **openssl enc -aes-128-cbc -e -in 20bytes.txt -out 20bytesEncrypted.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708** and **openssl enc -aes-128-cbc -e -in 32bytes.txt -out 32bytesEncrypted.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708**. Finally, I used command **ls -l <filename>** to check the size (in bytes) of the resultant encrypted files. It should be noted that all the block ciphers normally make use of PKCS5 padding and cipher block of 8 bytes. Hence, the block size of 128 bits is equal to 128/8 = 16 bytes.

- **Findings for 20 bytes file:**

  As mentioned above, after encrypting, I made use of command **ls -l <filename>** to check the size (in bytes) of the resultant encrypted file. Here is the output of the command:

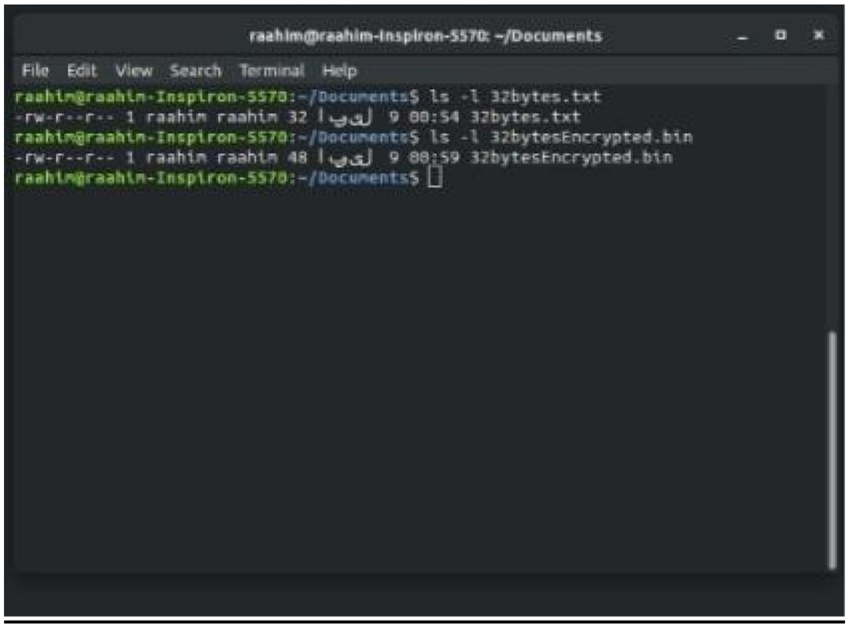


```
raahim@raahim-Inspiron-5570:~/Documents$ ls -l 20bytes.txt
-rw-r--r-- 1 raahim raahim 20 يوليو  9 00:53 20bytes.txt
raahim@raahim-Inspiron-5570:~/Documents$ ls -l 20bytesEncrypted.bin
-rw-r--r-- 1 raahim raahim 32 يوليو  9 00:58 20bytesEncrypted.bin
raahim@raahim-Inspiron-5570:~/Documents$ 
```

It can be seen that size of the encrypted file is 32 bytes. Original file was of size 20 bytes. Hence, encrypted file is 12 bytes more than the original file. Therefore, 12 bytes of padding has been added to the encrypted file. Each block has a size of 16 bytes (since I have used aes-128-cbc encryption and since cipher block is of size 8 bytes hence block size of 128 bits is equal to 128/8 = 16 bytes). Content of original file spans 20 bytes hence first block is of size 16 bytes and second block becomes 4 bytes. Thus, 12 more bytes are needed to fill in 16 bytes for second block. Hence, it is verified OpenSSL uses PKCS5 standard for its padding.

- **Findings for 32 bytes file:**

  As mentioned above, after encrypting, I made use of command **ls -l <filename>** to check the size (in bytes) of the resultant encrypted file. Here is the output of the command:



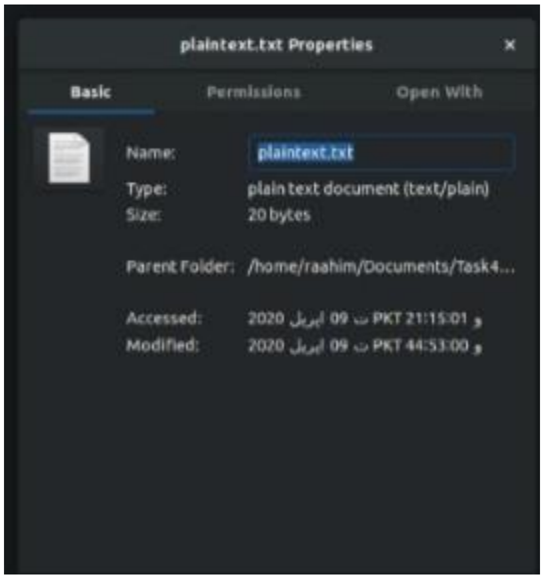
raahim@raahim-Inspiron-5570: ~/Documents
File Edit View Search Terminal Help

```
raahim@raahim-Inspiron-5570:~/Documents$ ls -l 32bytes.txt
-rw-r--r-- 1 raahim raahim 32 اپریل  9 08:54 32bytes.txt
raahim@raahim-Inspiron-5570:~/Documents$ ls -l 32bytesEncrypted.bin
-rw-r--r-- 1 raahim raahim 48 اپریل  9 08:59 32bytesEncrypted.bin
raahim@raahim-Inspiron-5570:~/Documents$ 
```

It can be seen that size of the encrypted file is 48 bytes. Original file was of size 32 bytes. Hence, encrypted file is 16 bytes more than the original file. Each block has a size of 16 bytes (since I have used aes-128-cbc encryption and since cipher block is of size 8 bytes hence block size of 128 bits is equal to 128/8 = 16 bytes). Content of original file spans 32 bytes hence first and second blocks both are 16 bytes. However, 16 more bytes are needed at the end of the encrypted file because it needs padding. So, 16 bytes of padding is added to the encrypted file. Hence, it is verified OpenSSL uses PKCS5 standard for its padding.

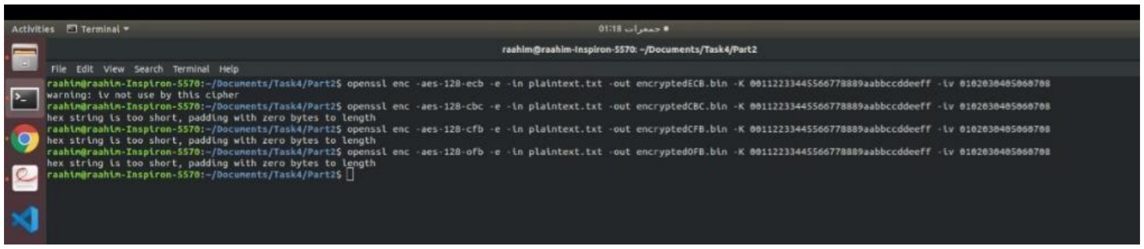
- **4.2**

First of all, I created a plaintext file of size 20 bytes.



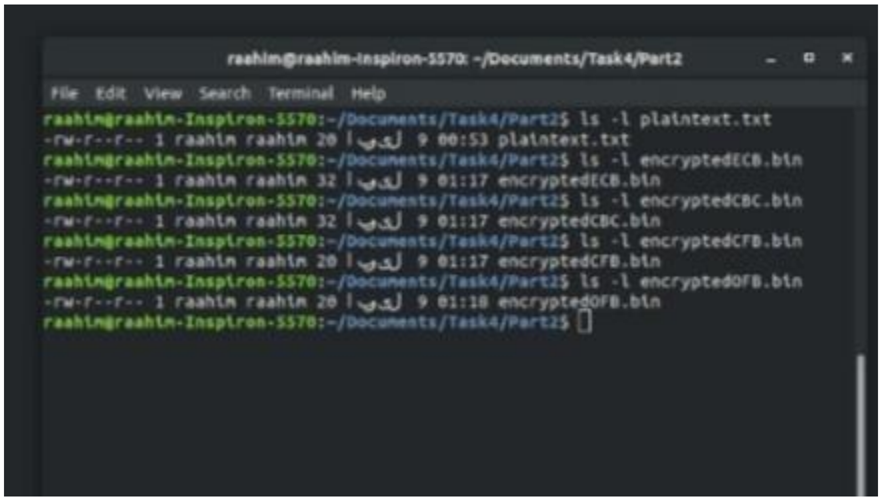
```
plaintext.txt Properties
Name:           plaintext.txt
Type:           plain text document (text/plain)
Size:           20 bytes
Parent Folder:  /home/raahim/Documents/Task4...
Accessed:       2020 ابريل 09 ت PKT 21:15:01 و
Modified:       2020 ابريل 09 ت PKT 44:53:00 و
```

Afterwards, I encrypted this plaintext file using aes-128 encryption and using different modes such as ECB, CBC, CFB, and OFB. Following commands were used:



```
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ openssl enc -aes-128-ecb -e -in plaintext.txt -out encryptedECB.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not use by this cipher
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ openssl enc -aes-128-cbc -e -in plaintext.txt -out encryptedCBC.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ openssl enc -aes-128-cfb -e -in plaintext.txt -out encryptedCFB.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ openssl enc -aes-128-ofb -e -in plaintext.txt -out encryptedOFB.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$
```

Finally, I checked the size of encrypted files to determine whether padding was added or not to each of the encrypted files. Basically, I utilized my experiment from 4.1 to determine this. Following was the resultant output after I ran the command **ls -l <filename>** to each of the encrypted files:



```
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ ls -l plaintext.txt
-rw-r--r-- 1 raahim raahim 20 لائی  9 00:53 plaintext.txt
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ ls -l encryptedECB.bin
-rw-r--r-- 1 raahim raahim 32 لائی  9 01:17 encryptedECB.bin
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ ls -l encryptedCBC.bin
-rw-r--r-- 1 raahim raahim 32 لائی  9 01:17 encryptedCBC.bin
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ ls -l encryptedCFB.bin
-rw-r--r-- 1 raahim raahim 20 لائی  9 01:17 encryptedCFB.bin
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ ls -l encryptedOFB.bin
-rw-r--r-- 1 raahim raahim 20 لائی  9 01:18 encryptedOFB.bin
raahim@raahim-Inspiron-5570:~/Documents/Task4/Part2$ 
```

Findings are summarized in the table below

Ciphermode	**Observations**
ECB	Needs padding
CBC	Needs padding
CFB	Does not need padding because CFB is a stream cipher. It means size of the block is usually fixed.
OFB	Does not need padding because OFB is also a stream cipher just like CFB and hence, size of the block is usually fixed.