Name: Param Patil

UID: 2019130049

TE Comps

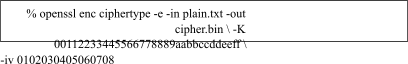
**Aim:**

Get familiar with the concepts in the secret-key encryption. Using different encryption ciphers and modes.

**Requirements:**

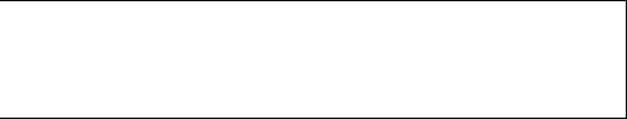
Openssl, Hxd(To edit the files in binary format).

**LAB TASKS:**

**3.1:** In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man

enc.

Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -aes-128-cfb, -bf-cbc, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "man enc". We include some common options for the openssl enc command in the following:



-in <file>

-out <file>

-e

-d

-K/-iv

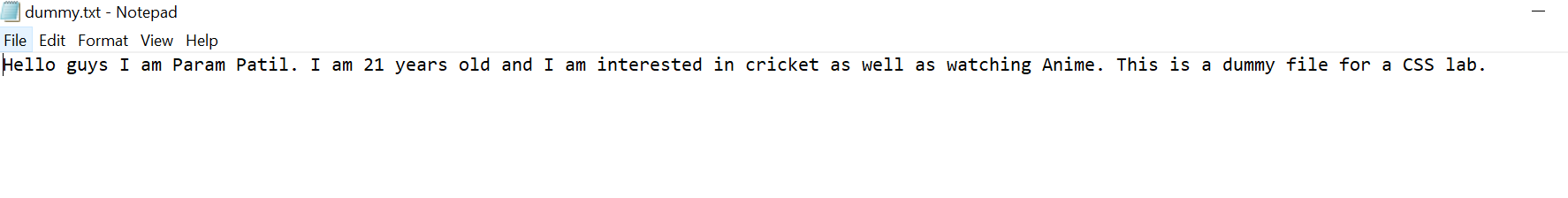
-[pP]

input file output file encrypt decrypt

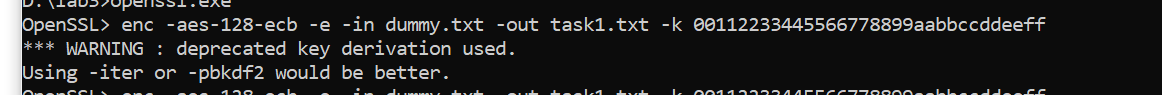
key/iv in hex is the next argument print the iv/key (then exit if -P)

Encryption of Plain text using different ciphers:

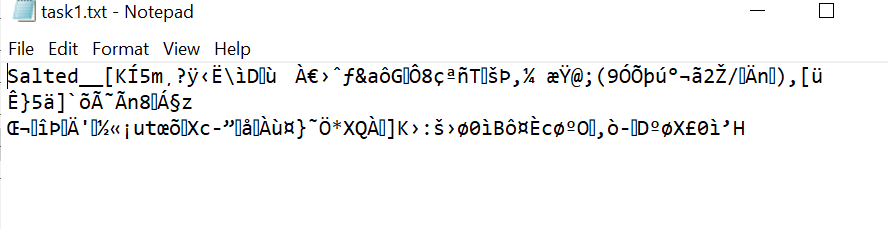
The plaintext is shown below:



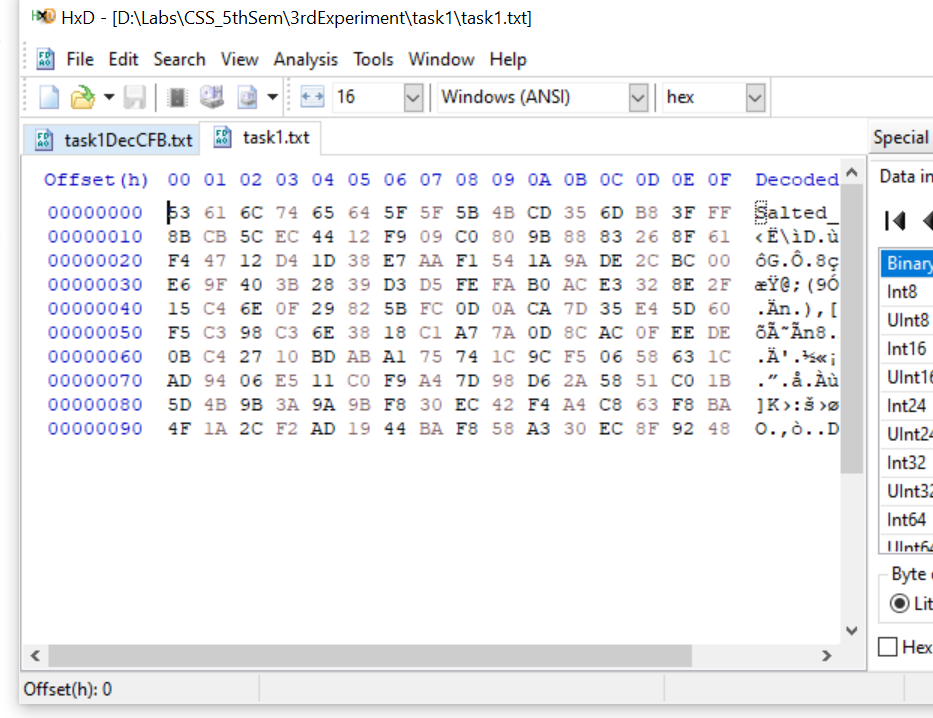
1. Encryption using aes-128-ecb



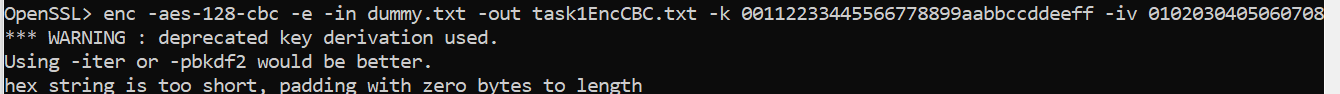
The Encrypted text is :



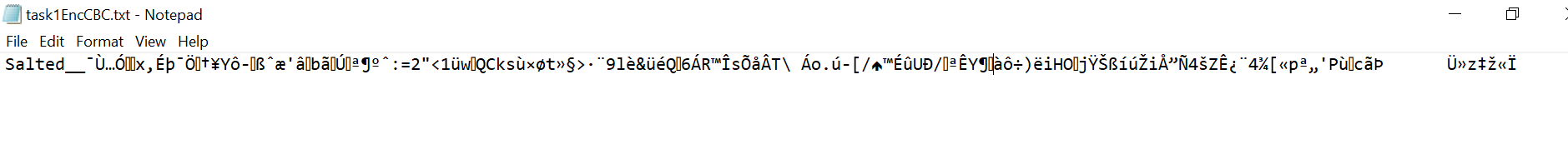
The hexdump for the above cipher text is:



1. Encryption using aes-128-cbc



The Encrypted text is :



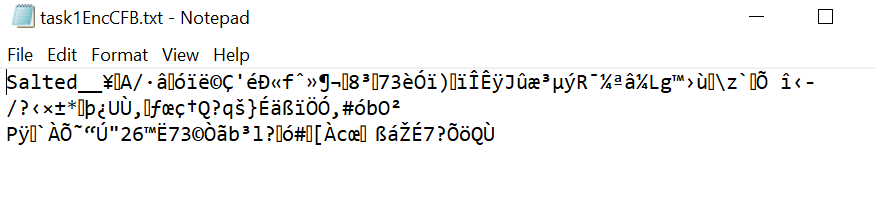
The hexdump for the above cipher text is:



3. Encryption using aes-128-cfb

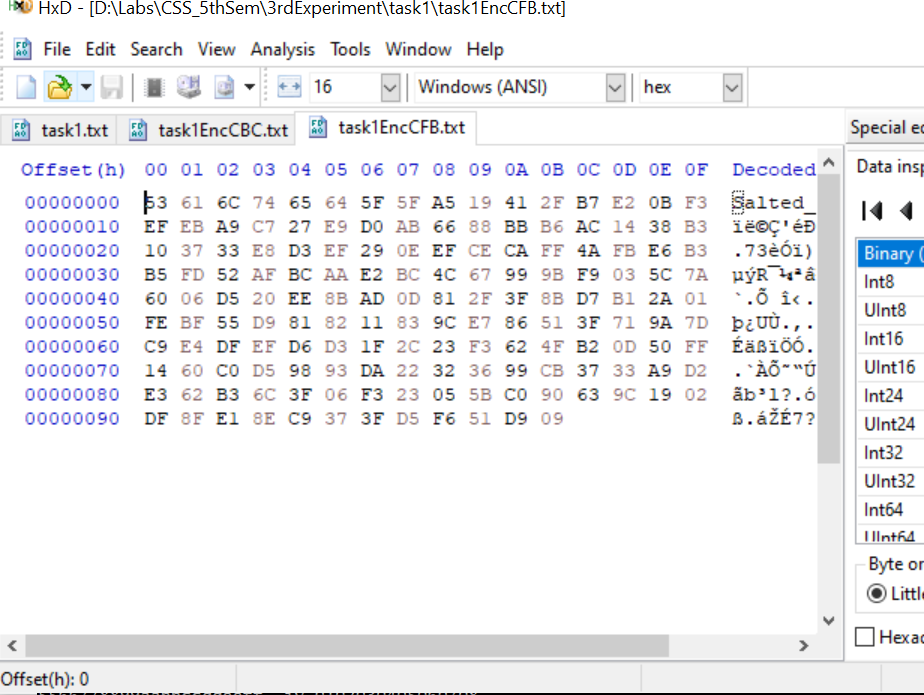
|  |
| --- |
|  |
|  |  |

|  |
| --- |
|  |
|  |  |

The Encrypted text is : 

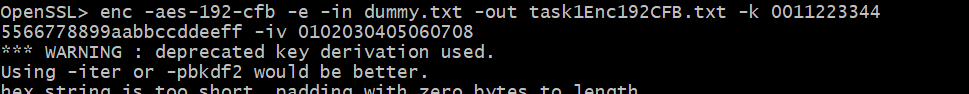
|  |
| --- |
|  |
|  |  |

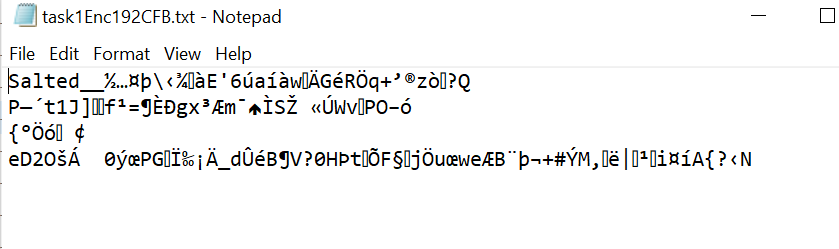
The hexdump for the above cipher text is:

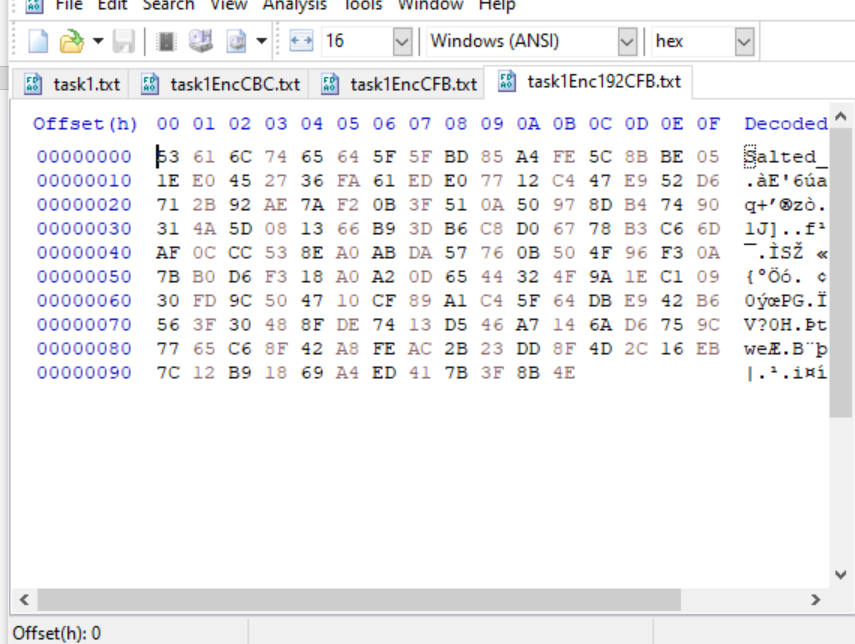


4.Encryption using aes-192-ecb

The cipher text and hexdump of the same is as follows:

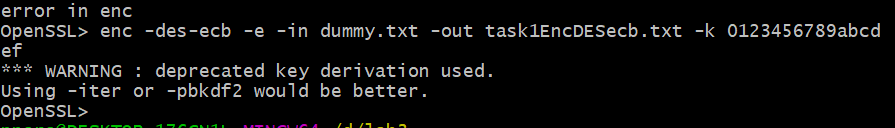


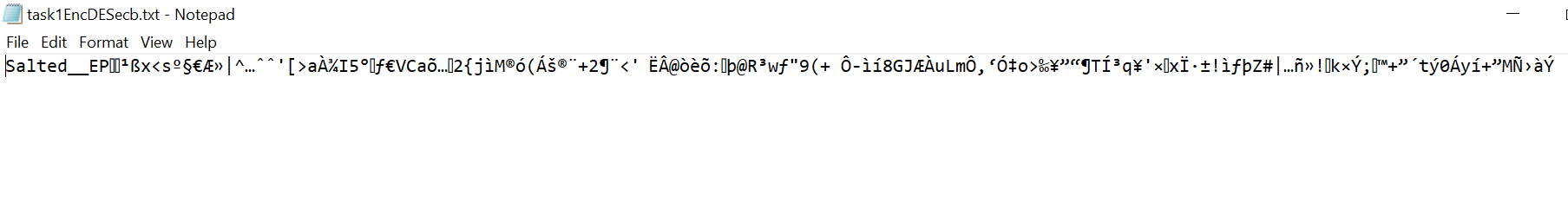


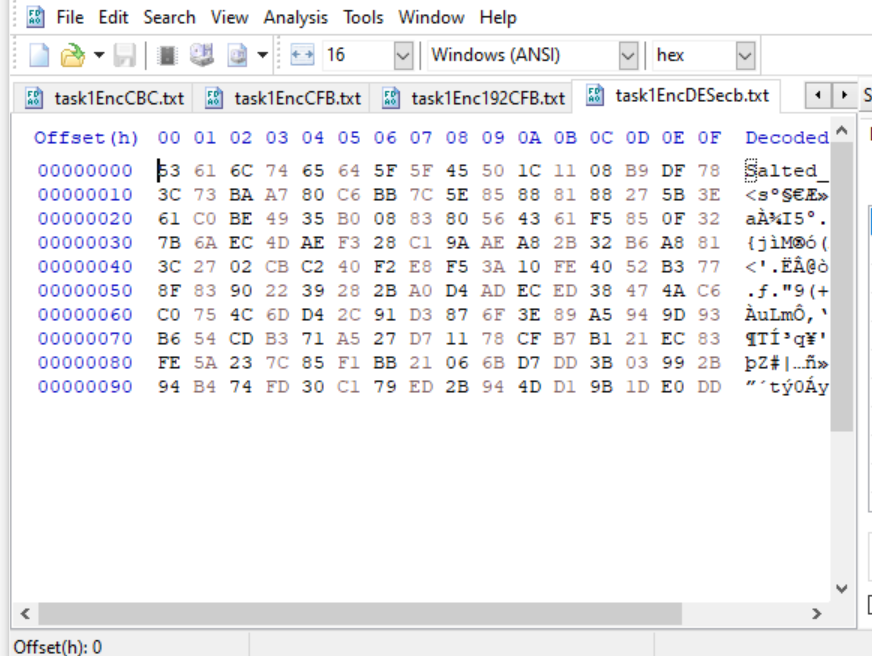


5.Encryption using des-ecb

The cipher text and hexdump of the same is as follows:







3.2: Encryption Mode - ECB vs. CBC

The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

* + 1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. How- ever, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. You can use the ghex tool to directly modify binary files.
    2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

Original Picture



Image Encrypted using ecb

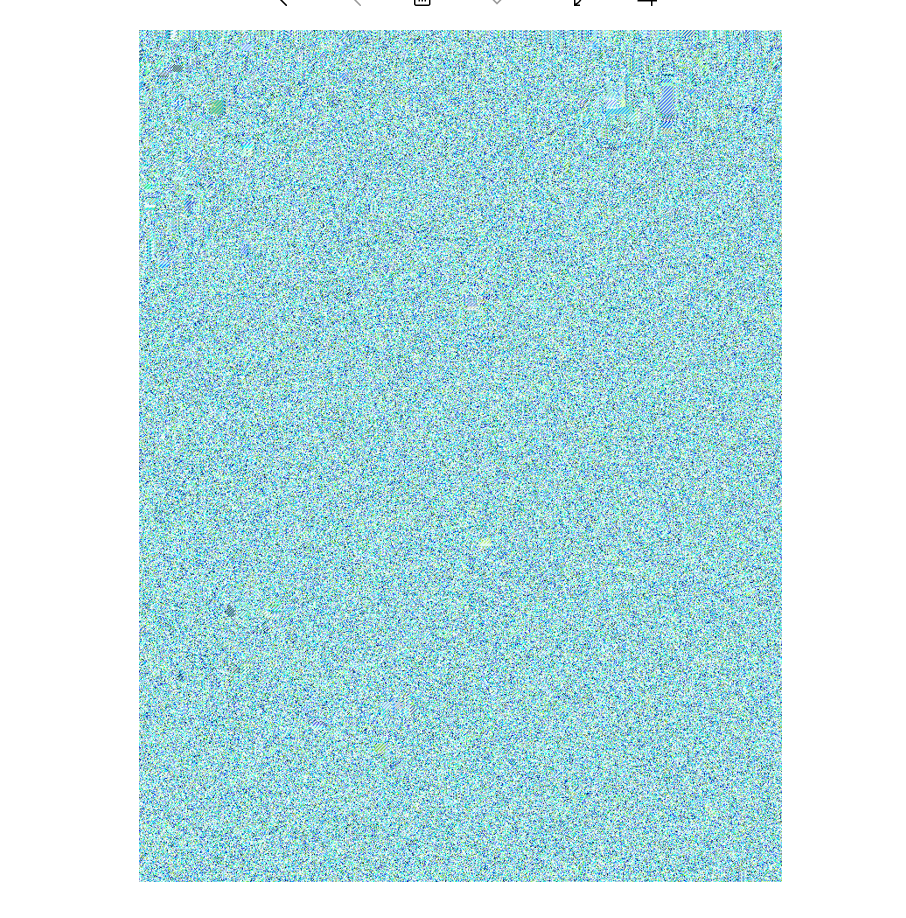
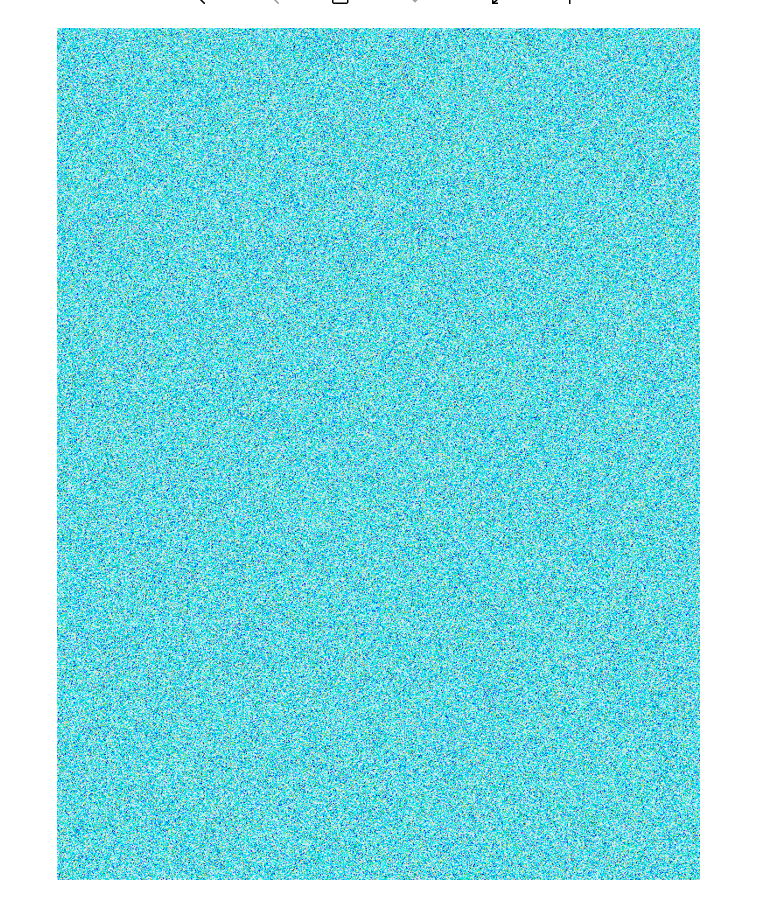


Image encrypted using cbc



Observations:

1. In ECB generally, the encrypted image still has the outlines of some objects which make the shapes in the image easily identifiable and sometimes the shapes are visible even to naked eyes. Even in the above image, although the color of the shapes gets hidden the boundary lines are slightly visible if one looks closely.
2. The CBC uses Initialization Vector, which certainly makes it a better choice to select for image encryption. In CBC encryption the image along with the boundary becomes scattered, thereby making it extremely difficult for a person to understand the picture. With the help of a random IV, even if the attacker has the key it will be difficult to decrypt the image.

**3.3: Encryption Mode – Corrupted Cipher Text**

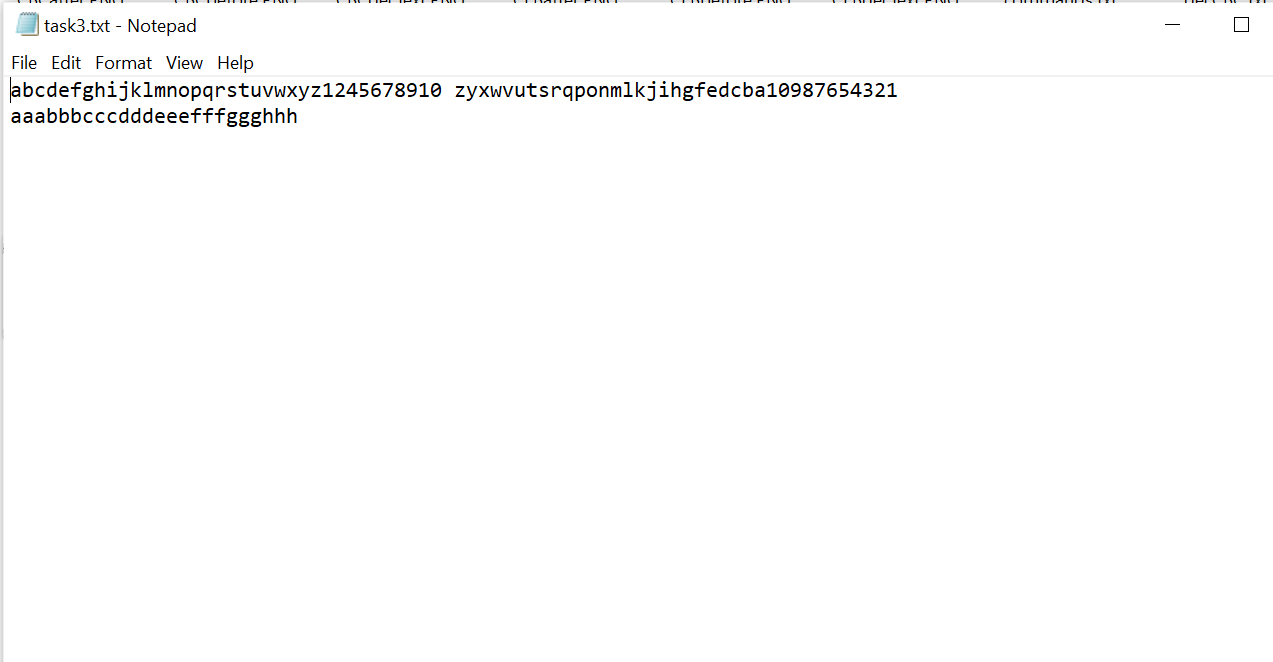
To understand the properties of various encryption modes, we would like to do the following exercise:

* + 1. Create a text file that is at least 64 bytes long.
    2. Encrypt the file using the AES-128 cipher.
    3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using ghex.
    4. Decrypt the corrupted file (encrypted) using the correct key and IV.

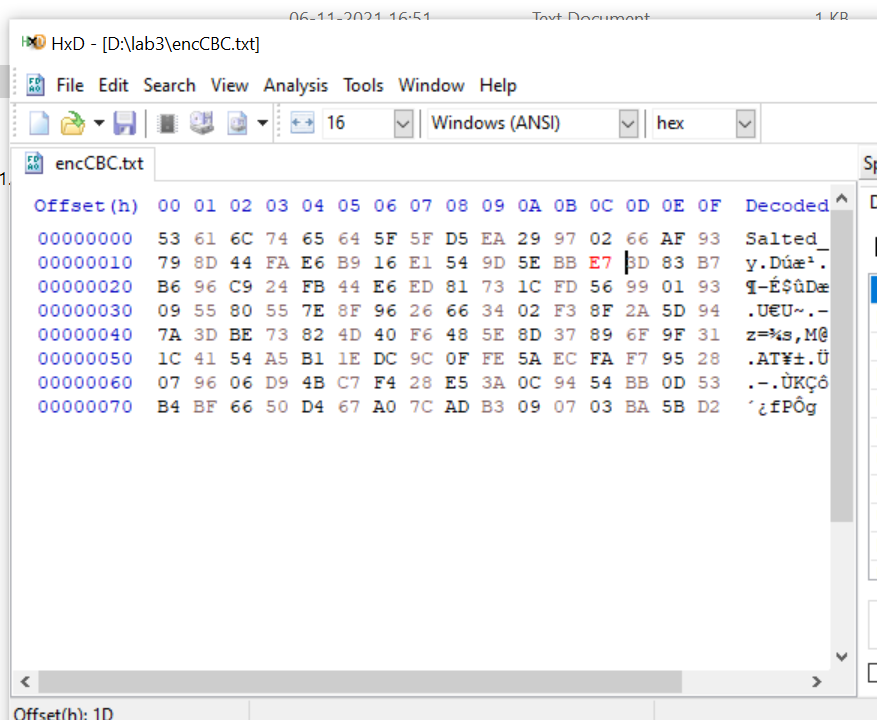
Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task.

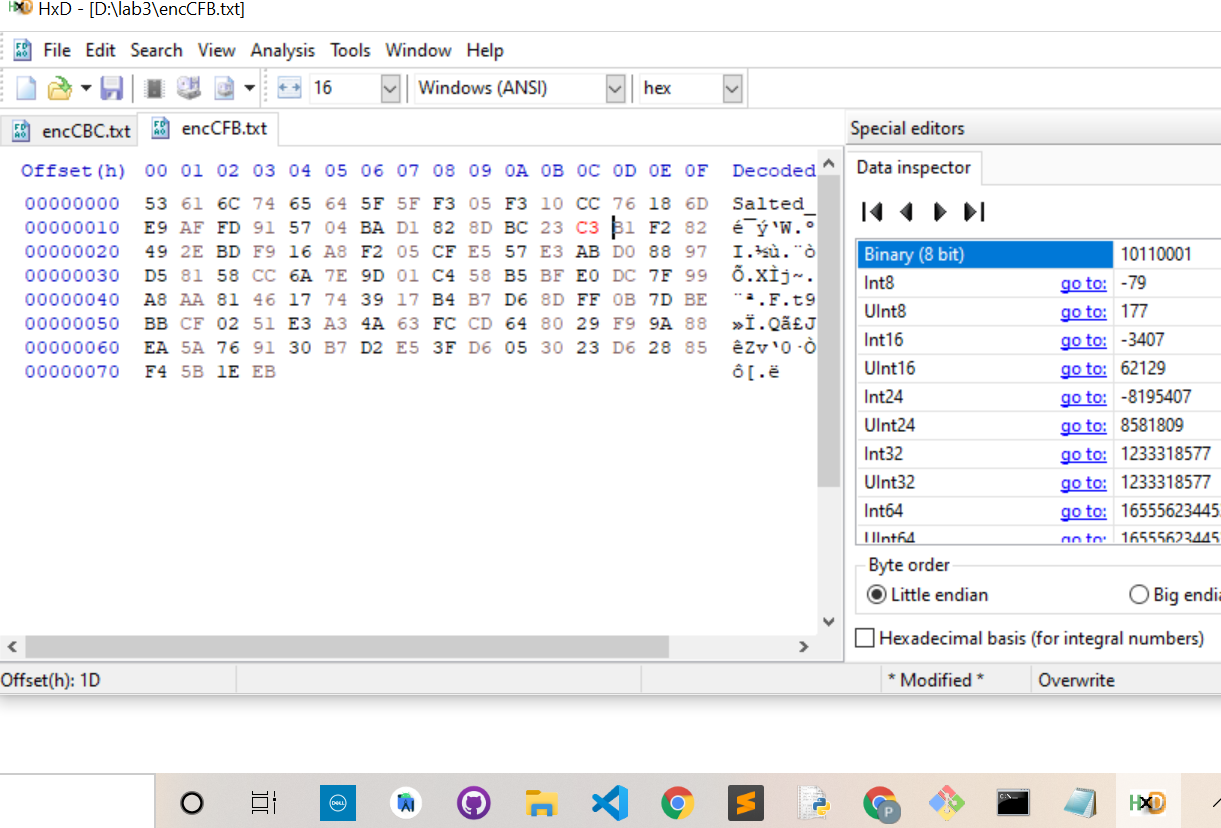
(2) Please explain why. (3) What are the implication of these differences?

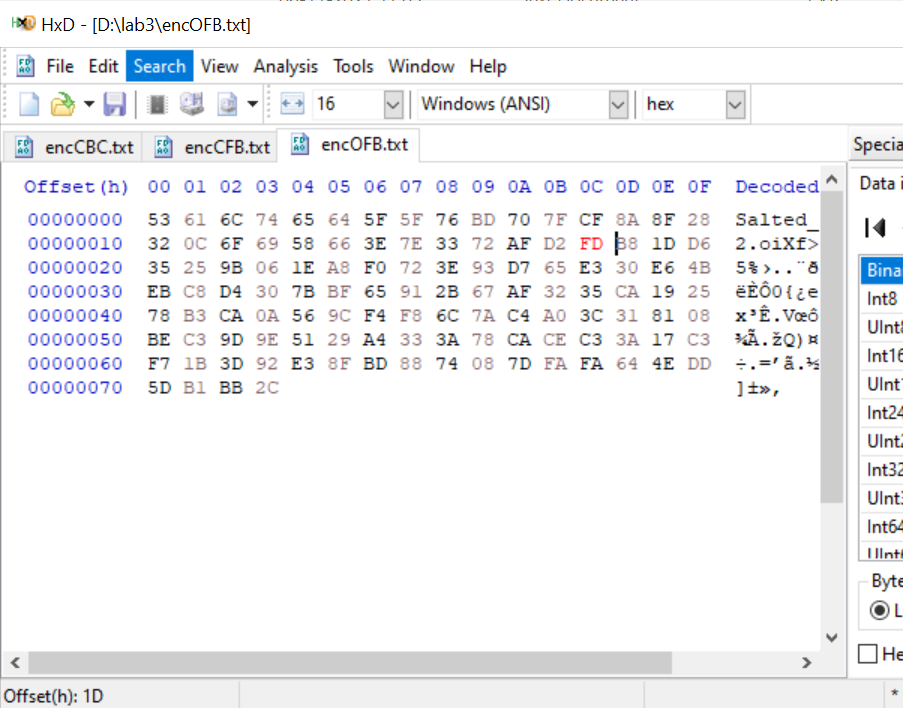
The original plain text is as follows:



The above text is encrypted using aes-128 in 4 different modes: cbc,cfb,ofb and ecb. After corrupting 30th byte of each cbc, cfb, ofb and ecb :

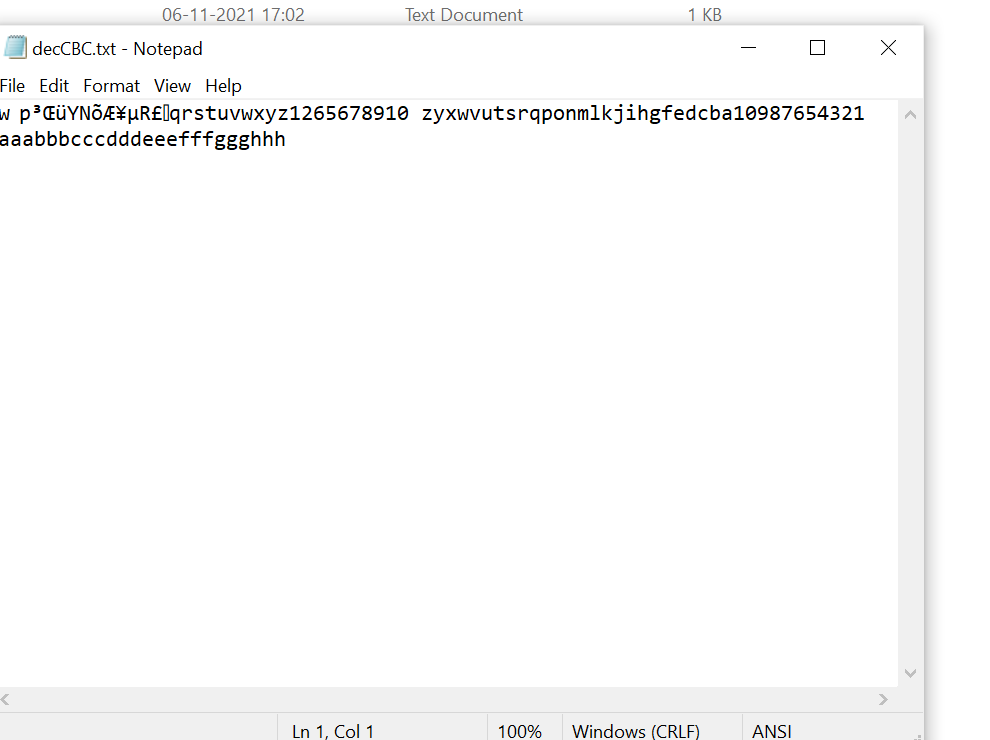




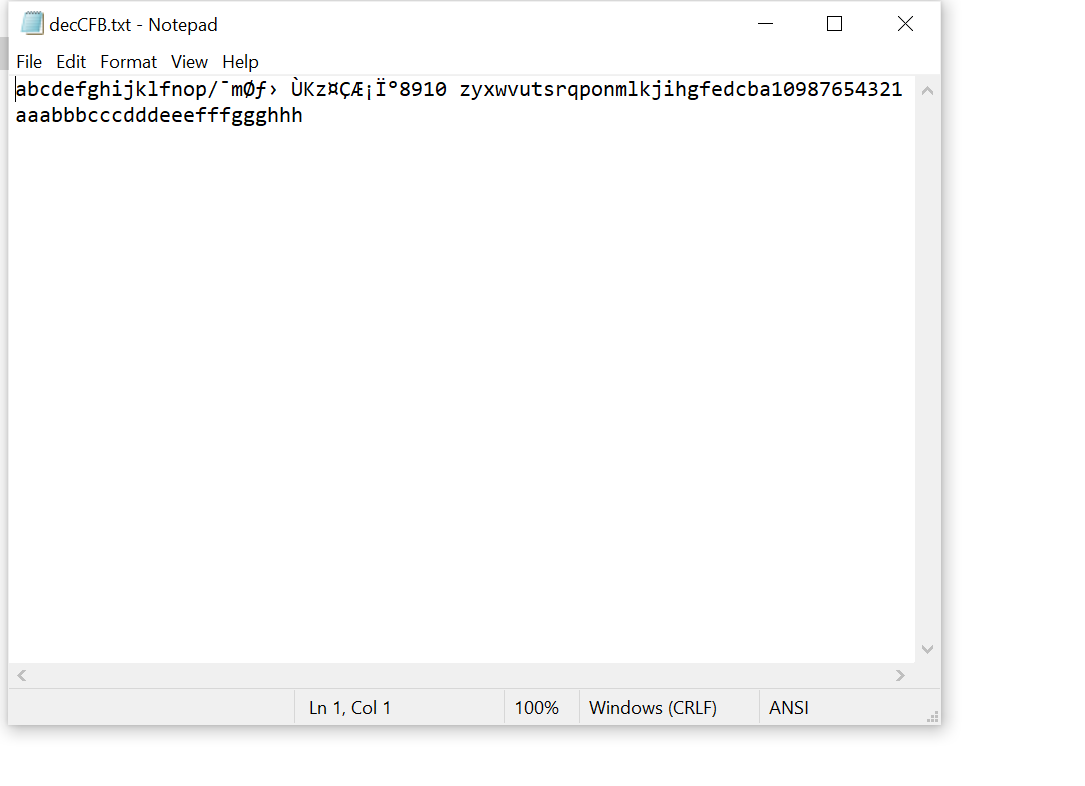




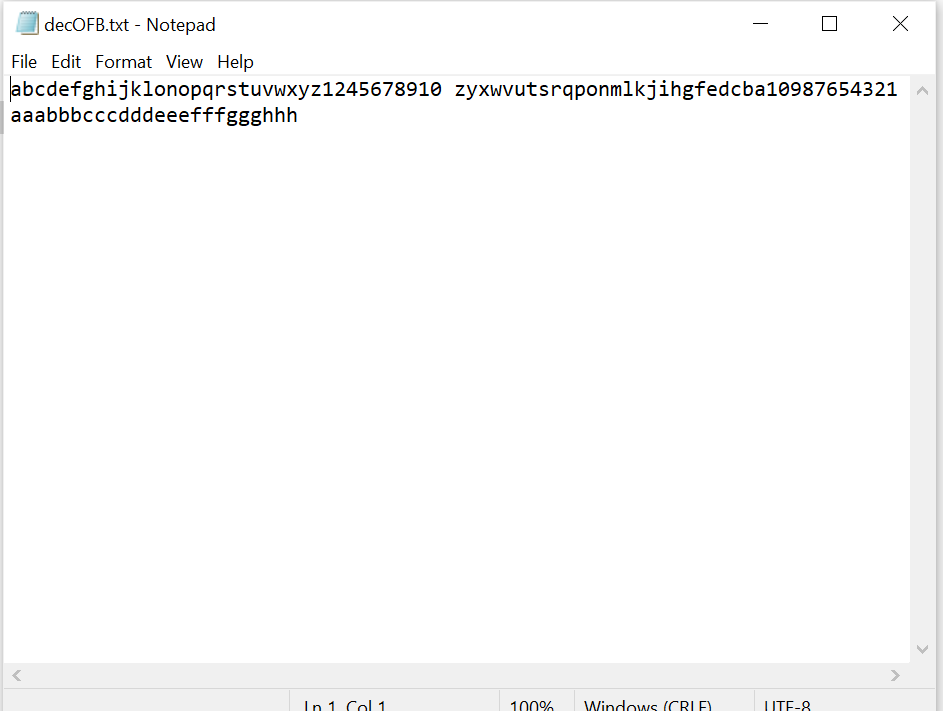
The decrypted text for the same is as follows: CBC



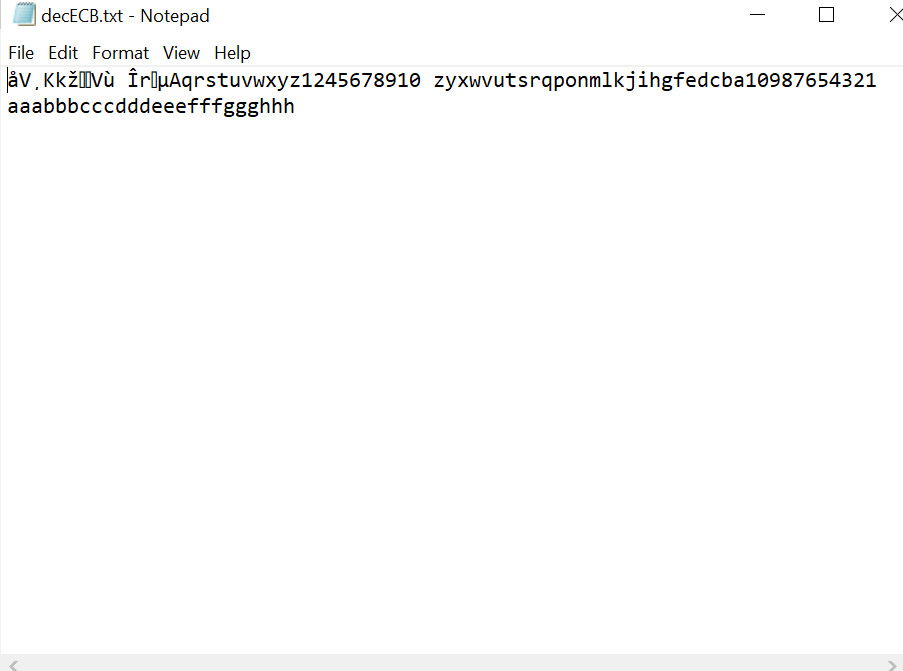
CFB



OFB



ECB



From the results:

ECB mode: If the 30th byte is corrupted then the entire block used while encryption gets corrupted after decryption. This happens mainly because the decryption is done one block at a time.

CBC mode: In CBC mode since two blocks are used at a time for encryption then the two blocks where the 30th byte is present gets corrupted after decryption.

CFB mode: For CFB mode every text block is dependent on the previous text block, as a result, the block that gets corrupted in case of CFB is located a little far from the starting letter of the original text as any hinderance in the previous text block could affect the upcoming text block.

OFB mode: If the 30th byte is corrupted, then only that particular byte of the plaintext gets corrupted, as observed from the decryption, mainly because every text block in OFB is independent from previous.

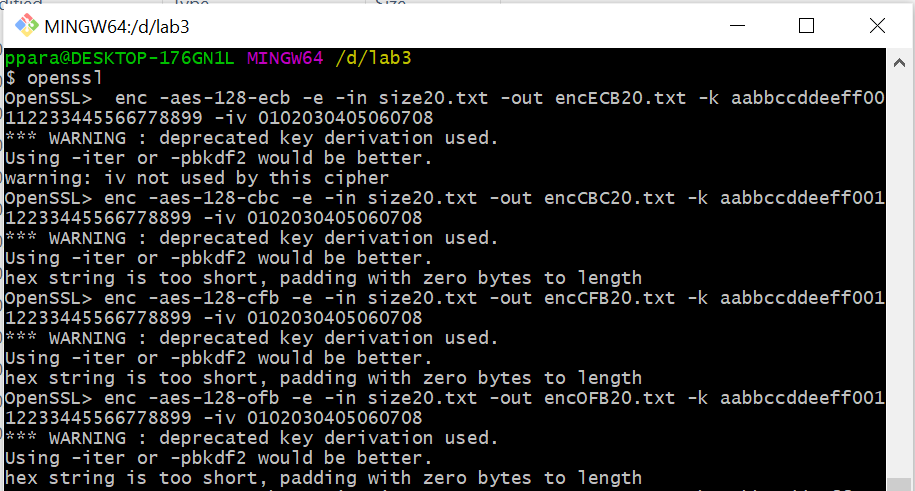
**3.4 : Padding**

For block ciphers, when the size of the plaintex is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

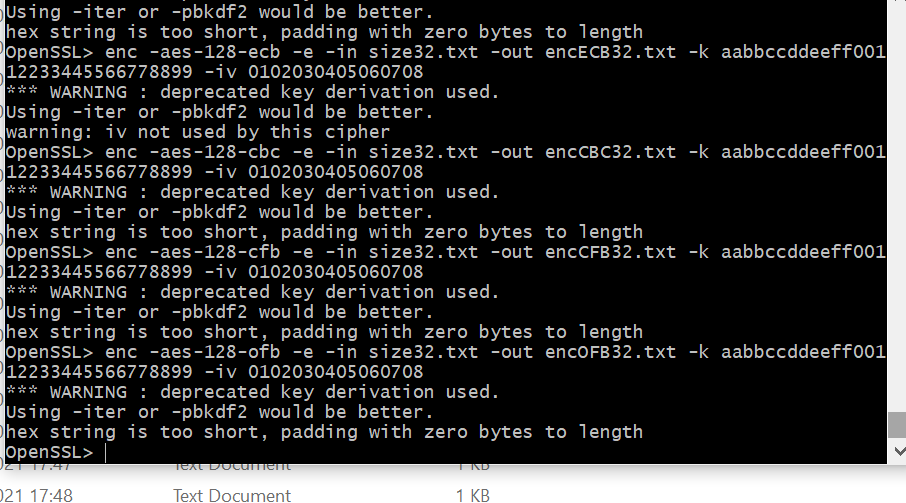
* + 1. The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.
    2. Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

First I encrypted the text using aes-128 in 4 modes namely: ecb, cbc, cfb, ofb, and for both the size of files, i.e 20bytes and 32 bytes (In this I entered a 20 and 32 character plaintext respectively)

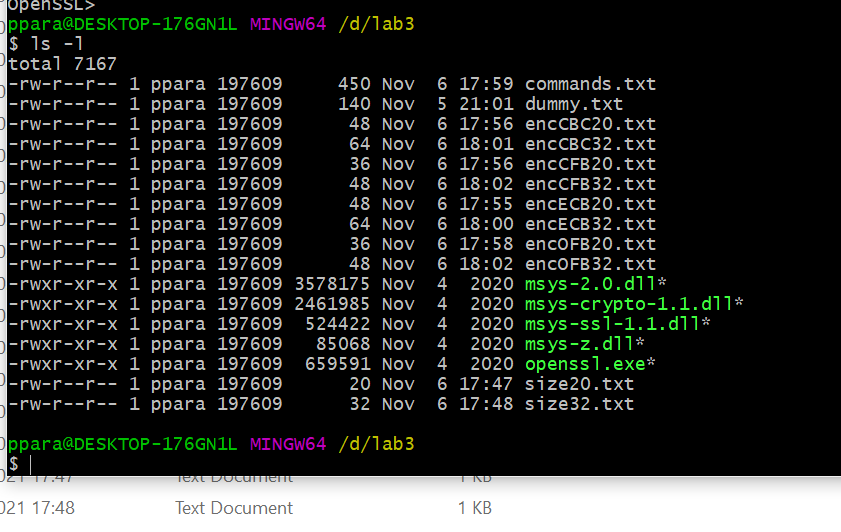
20 bytes size file encryption:



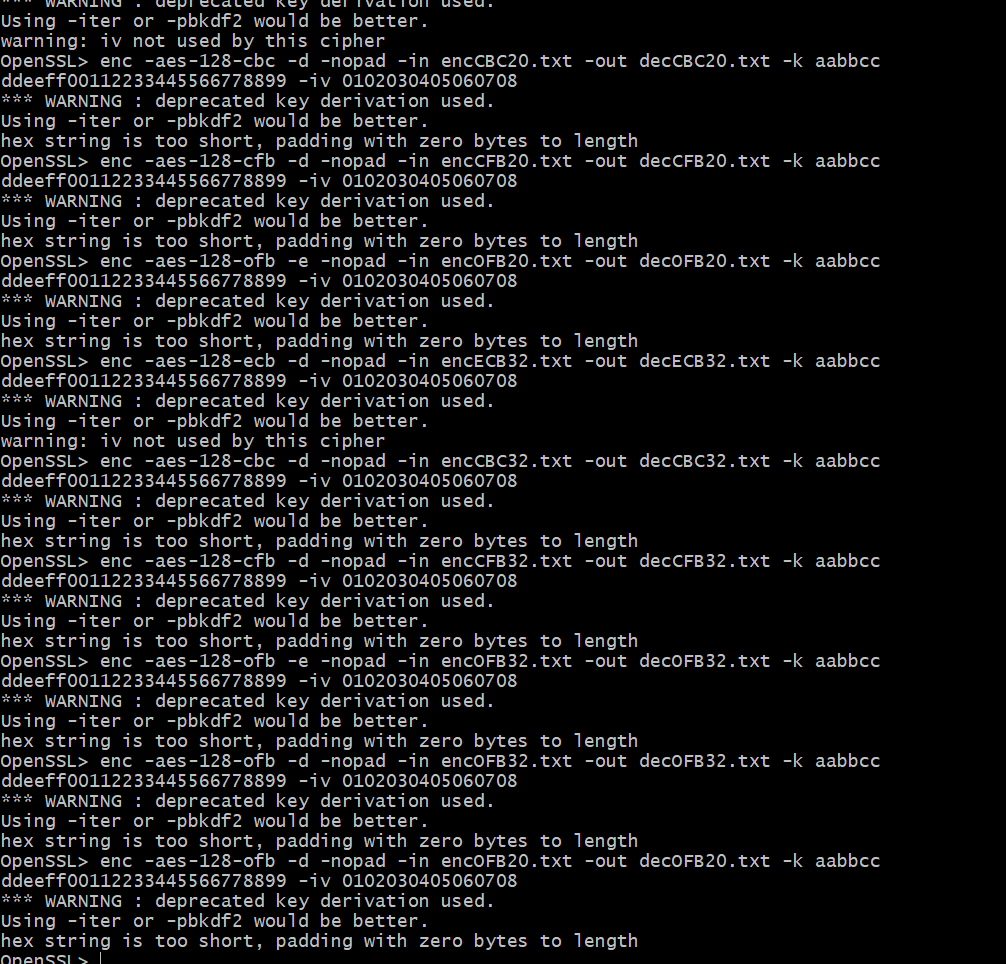
32 bytes size file encryption:



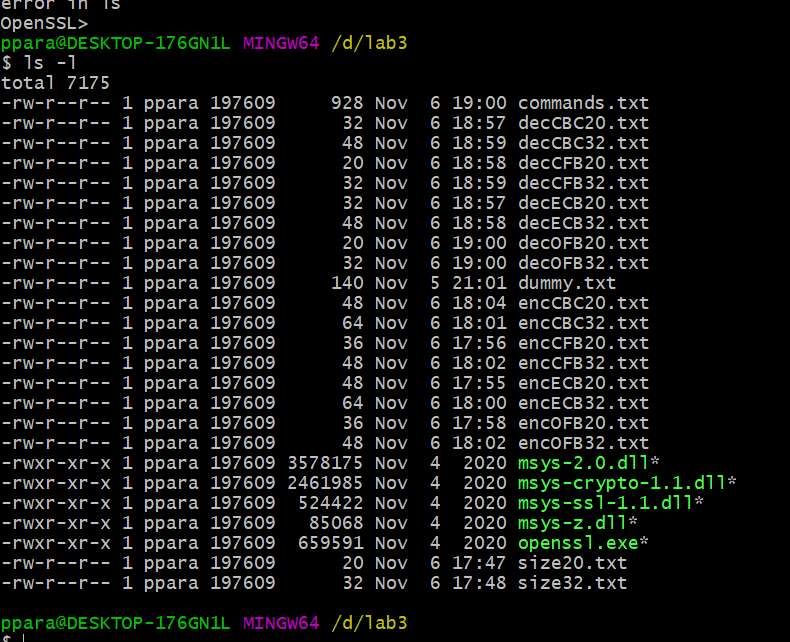
Below you can see the size of all the encrypted files:



To confirm that openssl uses PKCS5 padding, decrypt the encrypted file with option–nopad. This option turns off the standard block padding. Padding is by default used while encryption, by using -nopad the padding in the decrypted file could be seen.



Below is the size of all the decrypted files of both the sizes:



The size of CBC and ECB encrypted files of size 20 file is 12 bytes more and in the case of size 32 the

encrypted file is 16 bytes more thereby working on the formula:

Ciphertext = PlaintText + Block – PlainText%Block

AES uses 16 bytes block.

The decrypted files are also 12 and 16 bytes more for the size of files 20 and 32 respectively.

When compared to OFB and CFB the size of the files remains the same.

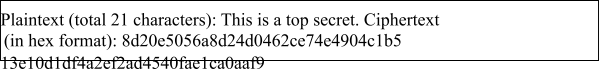
The reason behind this would be that as ECB and CBC are block cipher they need the input to be a multiple of the block and to compensate for the difference padding is needed. Whereas OFB and CFB are ciphers in which the length of the ciphertext is the same as the plaintext. OFB and CFB convert a block cipher into a stream cipher and the generated stream is later XORed.

**3.5 : Programming using the Crypto Library**

So far, we have learned how to use the tools provided by openssl to encrypt and decrypt messages. In this task, we will learn how to use openssl’s crypto library to encrypt/descrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Al- though OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library pro- vides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in http://www.openssl. org/docs/crypto/EVP\_EncryptInit.html. Please get yourself familiar with this program, and then do the

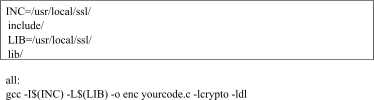
following exercise.

You are given a plaintext and a ciphertext, and you know that aes-128-cbc is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character ‘0’). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. You can download a English word list from the Internet. We have also linked one on the web page of this lab. The plaintext and ciphertext is in the following:

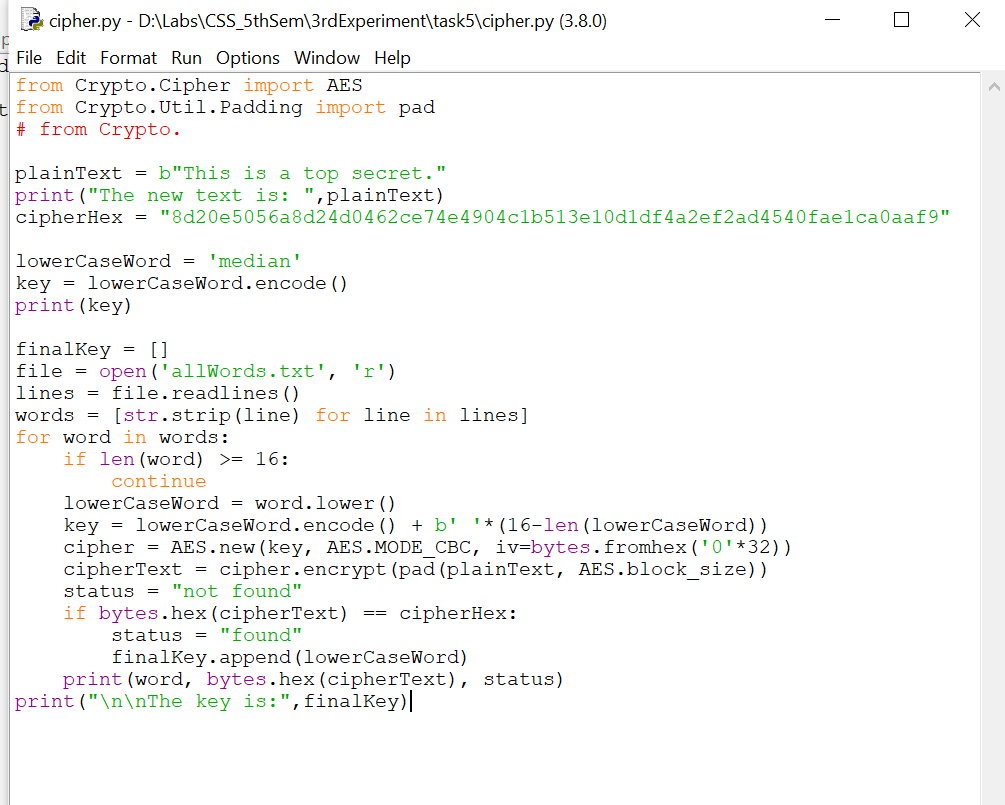
Note 1: If you choose to store the plaintex message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use the ghex tool to remove the special character.

Note 2: In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the openssl commands to do this task.

Note 3: To compile your code, you may need to include the header files in openssl, and link to openssl libraries. To do that, you need to tell your compiler where those files are. In your Makefile, you may want to specify the following:



Code:



Output:



Answer: the encryption key is median.

**Conclusion:**

AES and DES both use same keys for encryption and decryption making them symmetric ciphers but AES data encryption is a more mathematically efficient and elegant cryptographic algorithm, its main strength rests in the option for various key lengths. AES allows you to choose a 128-bit, 192-bit or 256-bit key, making it exponentially stronger than DES where the key length is 56 bits.

When it comes to mode of encryption there are mainly 4 modes which I used in this experiment which are ECB, CBC, CFB and OFB. In case of useability ECB is the weakest as it uses simple substitution, it also prohibits the use of IV (Initialization Vector), which eliminates the option to make the algorithm more robust. By performing the padding task, one could conclude that the ECB and CBC are block ciphers as they need the size of the plaintext to be divided into equal size of blocks and to ensure that for all lengths of plaintext there is a need of padding. In case of OFB and CFB there is no requirement of padding as they are stream ciphers. In case a attacker tries to corrupt a file that has been encrypted by one of the four methods, then OFB is most reliable as it is designed in such a way that only the bytes that are corrupted are affected after decryption, which was not true for the rest of the modes.