Experiment 3:	Crypto Encryption

Name: Sumeet Haldipur
UID: 2019130018
Class: TE Comps (Batch A)

Note: Students are advised to read through this lab sheet before doing experiment. On-the-spot evaluation may be carried out during or at the end of the experiment. Your performance, teamwork effort, and learning attitude will count towards the marks.

Experiment 3: Crypto Lab – Secret-Key Encryption

1 OBJECTIVE

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2 LAB ENVIRONMENT

Installing OpenSSL. In this lab, we will use openssl commands and libraries. We have already in-stalled openssl binaries in our VM. It should be noted that if you want to use openssl libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have already downloaded the necessary files under the directory openssl-1.0.1. To configure and install openssl libraries, run the following commands.

You should read the INSTALL file first:

```
% ./config
% make
% make test
% sudo make install
```

Installing GHex. In this lab, we need to be able to view and modify files of binary format. We have installed in our VM GHex a hex editor for GNOME. It allows the user to load data from any file, view and edit it in either hex or ascii.

3 LAB TASKS

3.1 Task 1: Encryption using different ciphers and modes

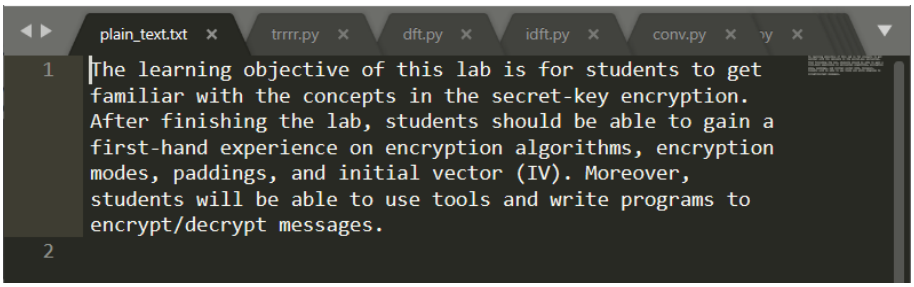
In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man enc.

```
% openssl enc ciphertype -e -in plain.txt -out cipher.bin \ -K
                    00112233445566778889aabbccddeeff \
                    -iv 0102030405060708
```

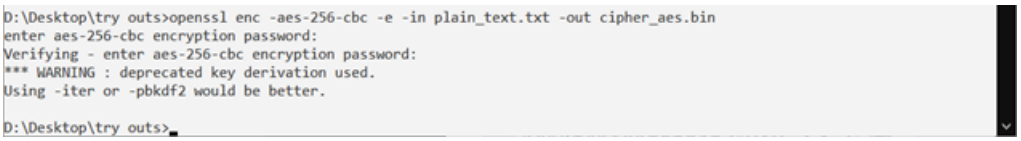
Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -aes-128-cfb, -bf-cbc, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "man enc".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

-in <file>	input file
-out <file>	output file
-e	encrypt
-d	decrypt
-K/-iv	key/iv in hex is the next argument
-[pP]	print the iv/key (then exit if -P)

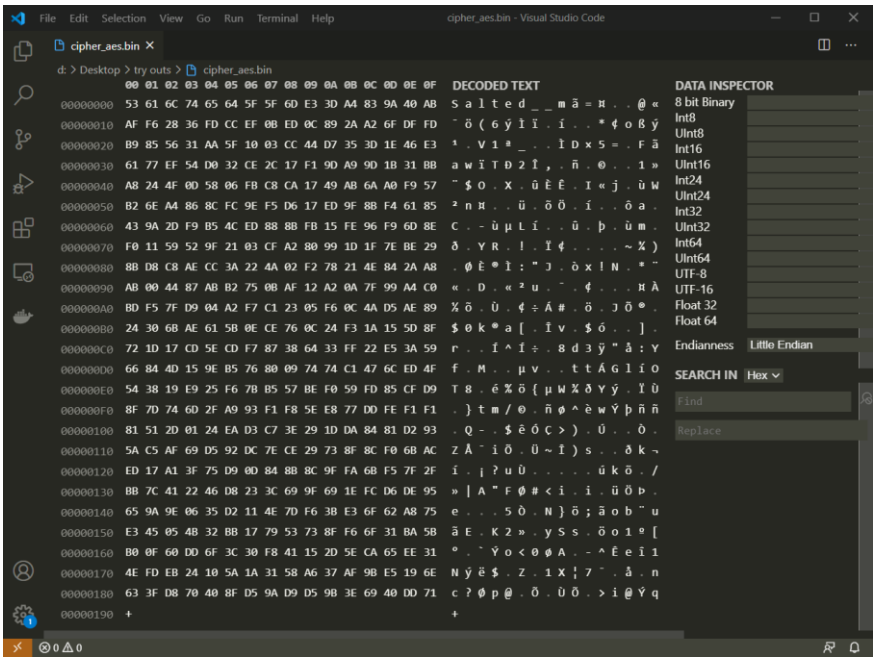
Encryption of plain text using different Ciphers:



1. Encryption Using aes-256-cbc:

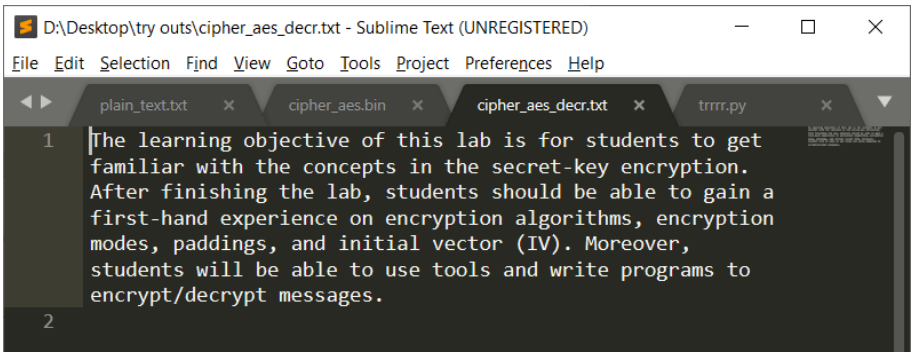


cipher_aes.bin



Decryption of cipher_aes.bin file

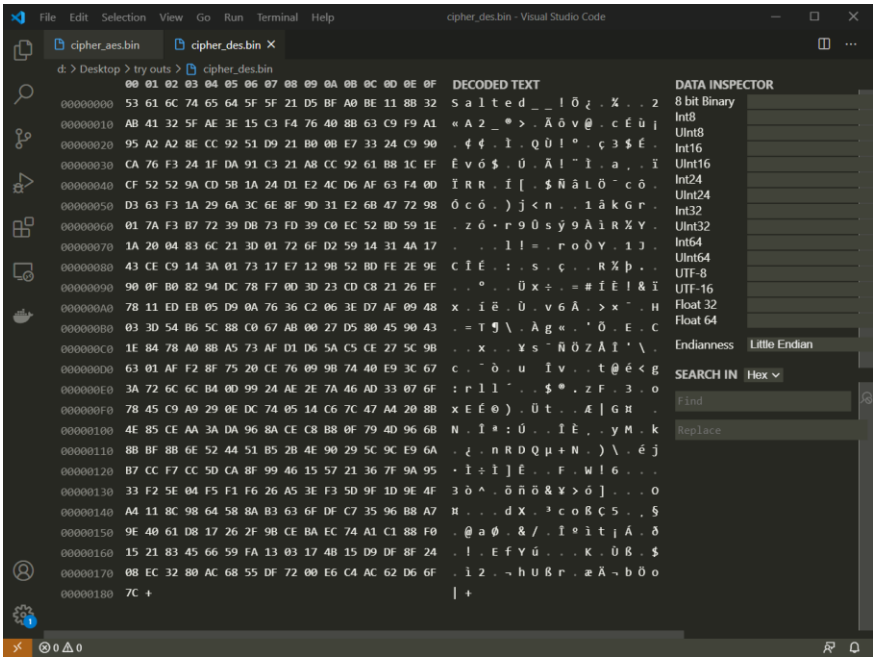
```
D:\Desktop\try outs>openssl enc -aes-256-cbc -d -in cipher_aes.bin -out cipher_aes_decr.txt
enter aes-256-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```



2. Encryption using des-ede3-ofb:

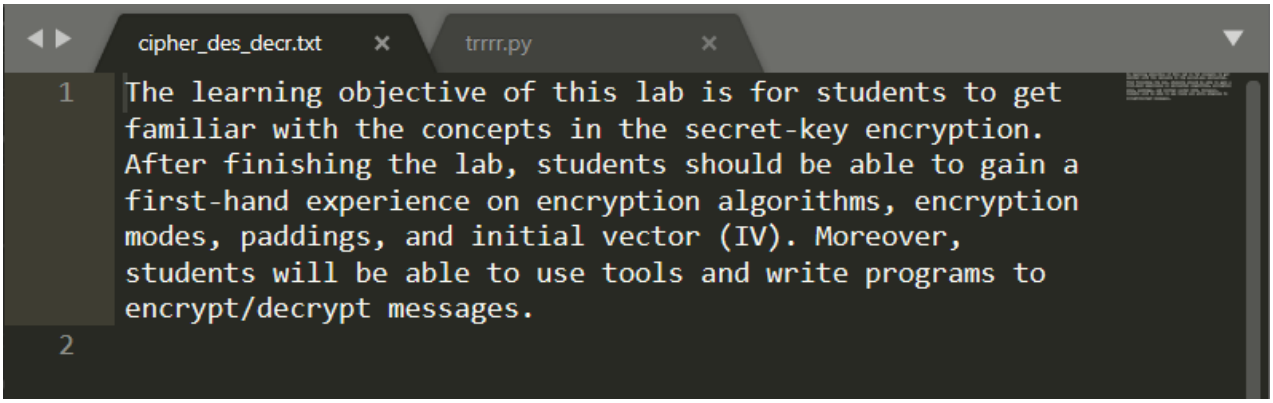
```
D:\Desktop\try outs>openssl enc -des-ede3-ofb -e -in plain_text.txt -out cipher_des.bin
enter des-ede3-ofb encryption password:
Verifying - enter des-ede3-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```

cipher_des.bin



Decryption of cipher_des.bin file

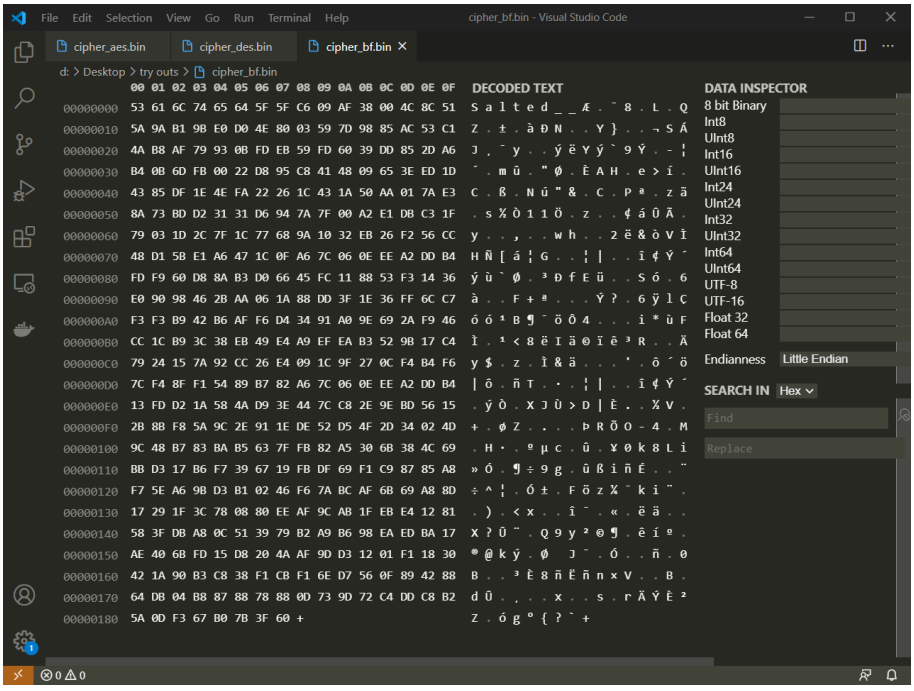
```
D:\Desktop\try outs>openssl enc -des-ede3-ofb -d -in cipher_des.bin -out cipher_des_decr.txt
enter des-ede3-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```



3. **Encryption using bf-ecb:**

```
D:\Desktop\try outs>openssl enc -bf-ecb -e -in plain_text.txt -out cipher_bf.bin
enter bf-ecb encryption password:
Verifying - enter bf-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```

cipher_bf.bin



Decryption of cipher_bf.bin file

```
D:\Desktop\try outs>openssl enc -bf-ecb -d -in cipher_bf.bin -out cipher_bf_decr.txt
enter bf-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```

cipher_bf_decr.txt | trrrr.py

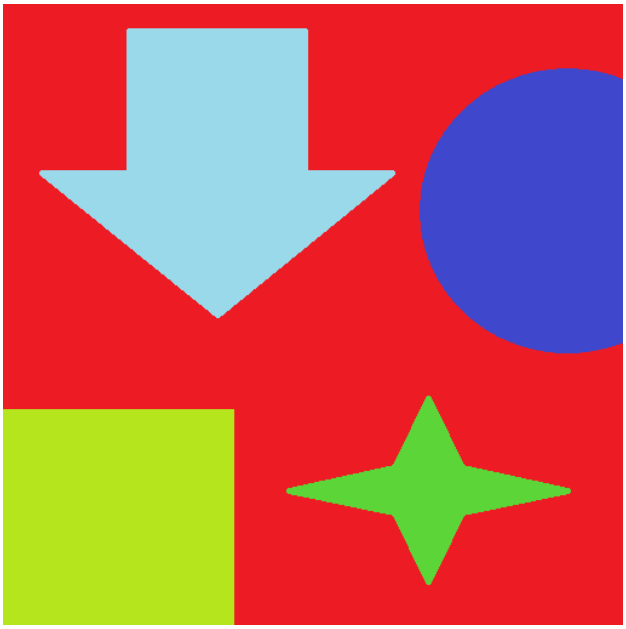
```
1  The learning objective of this lab is for students to get
   familiar with the concepts in the secret-key encryption.
   After finishing the lab, students should be able to gain a
   first-hand experience on encryption algorithms, encryption
   modes, paddings, and initial vector (IV). Moreover,
   students will be able to use tools and write programs to
   encrypt/decrypt messages.
2
```

3.2 Task 2: Encryption Mode – ECB vs. CBC

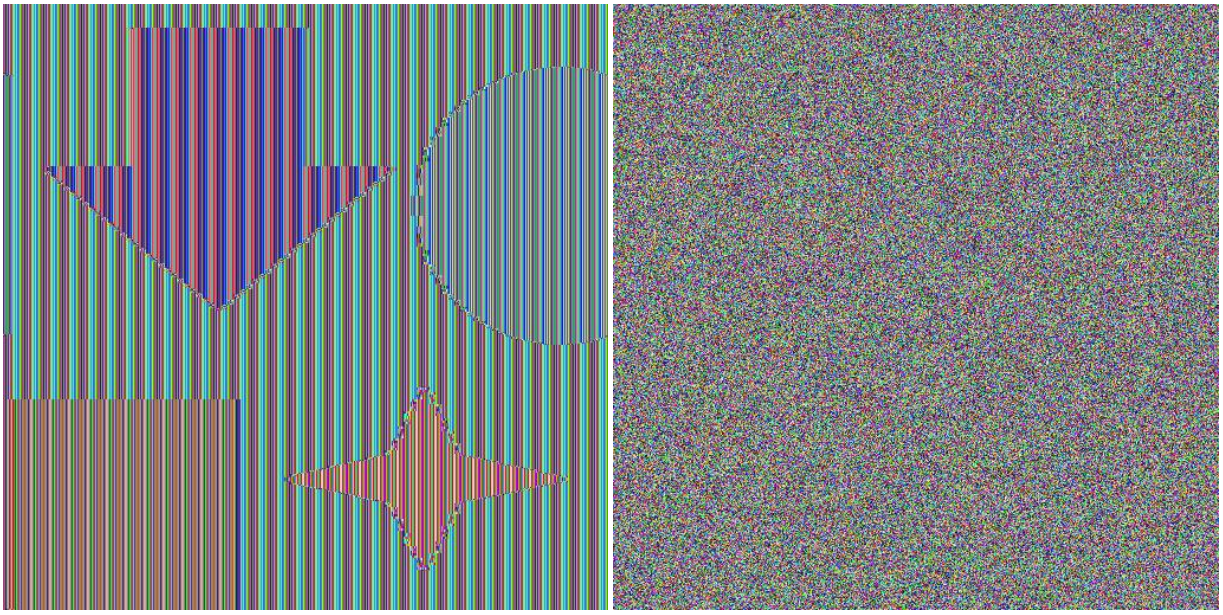
The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, 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:



Picture Encrypted Using ECB: **Picture Encrypted Using CBC:**



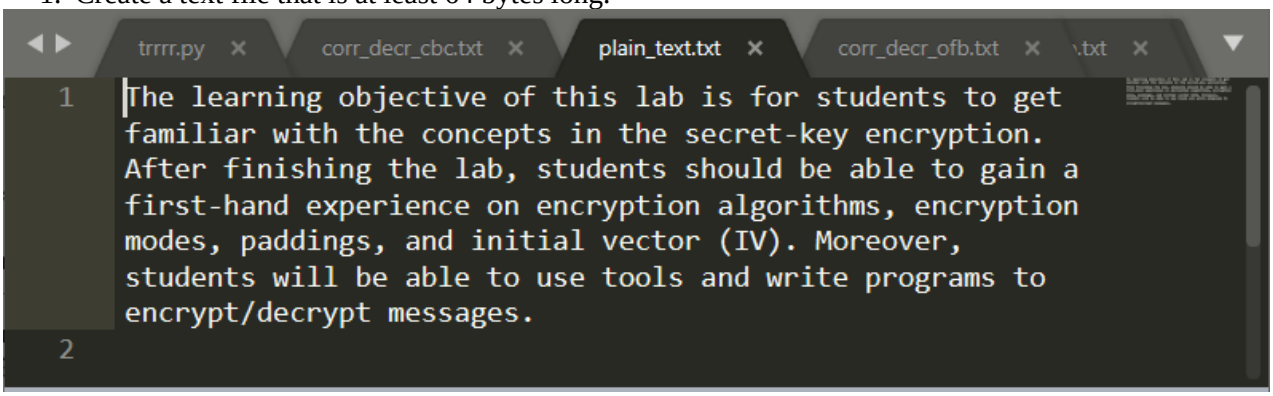
Observations:

- In case of image encrypted using ECB we can see the outline and borders of the original image and figure out a rough shape of the original image by looking at it.
- But in case of the image encrypted using CBC the entire image is distorted and there's no way to find what the original image may look like.
- ECB is the first version of AES and is the most basic block cipher. CBC is more advanced as in this, each ciphertext block is dependent on all plaintext blocks processed up to that point. This adds to the complexity and makes it harder to decrypt.

3.3 Task 3: Encryption Mode – Corrupted Cipher Text

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

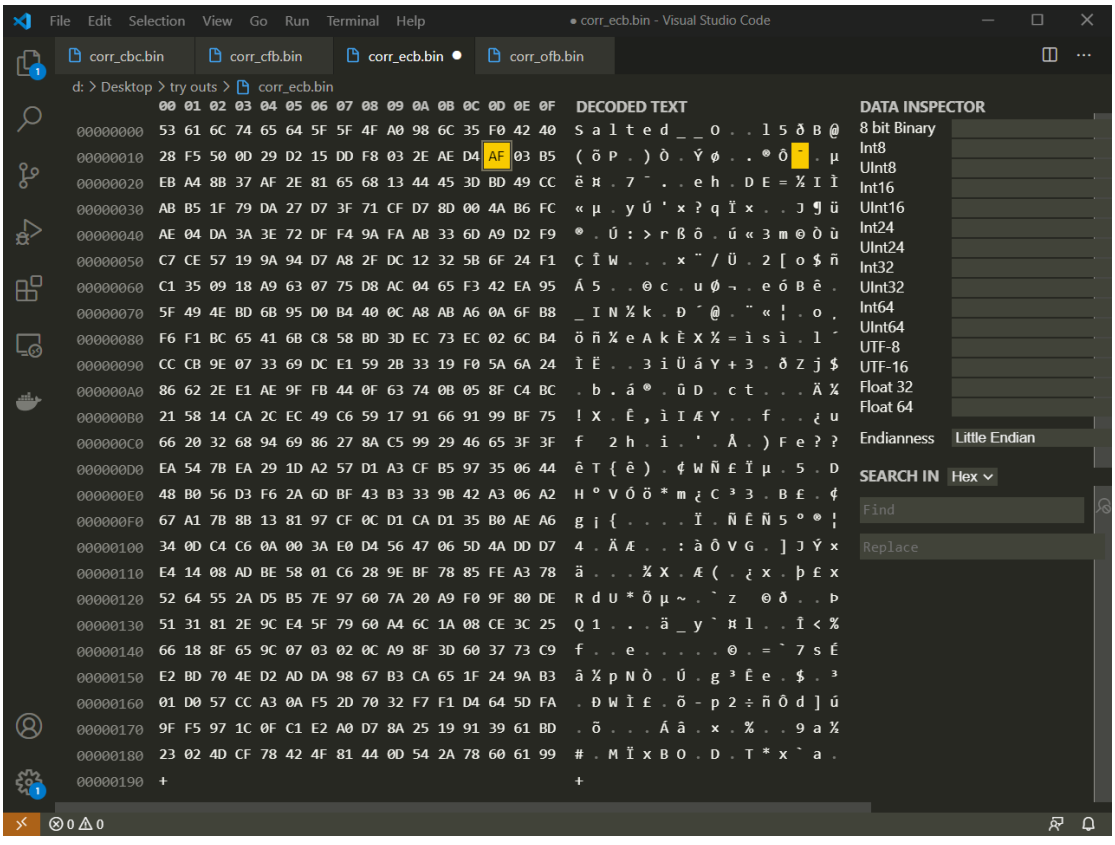
1. Create a text file that is at least 64 bytes long.

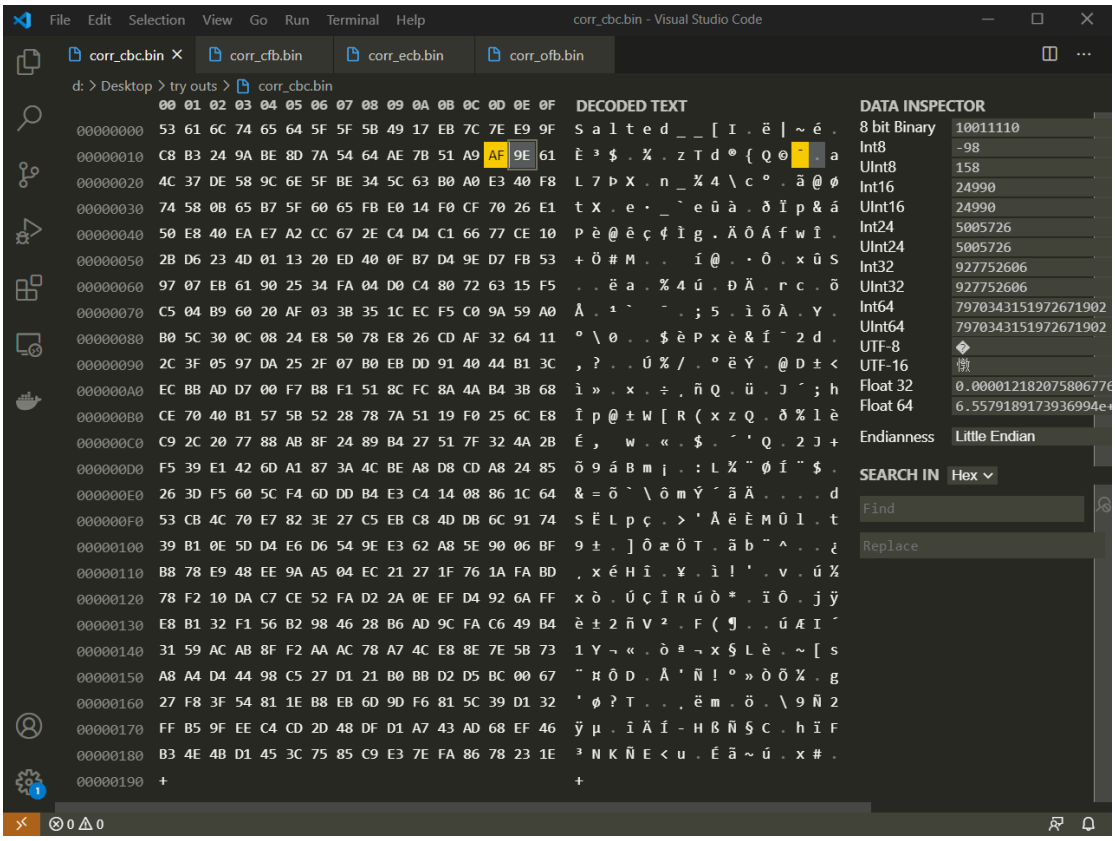


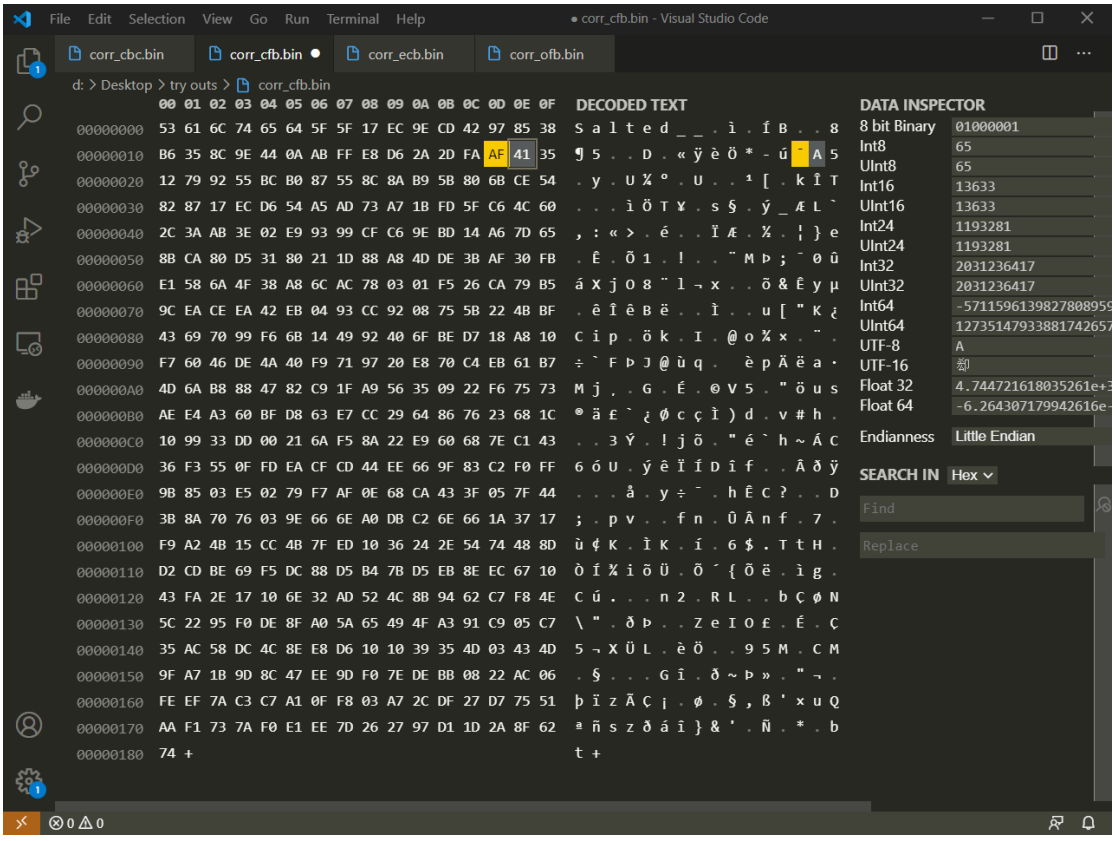
2. Encrypt the file using the AES-128 cipher.

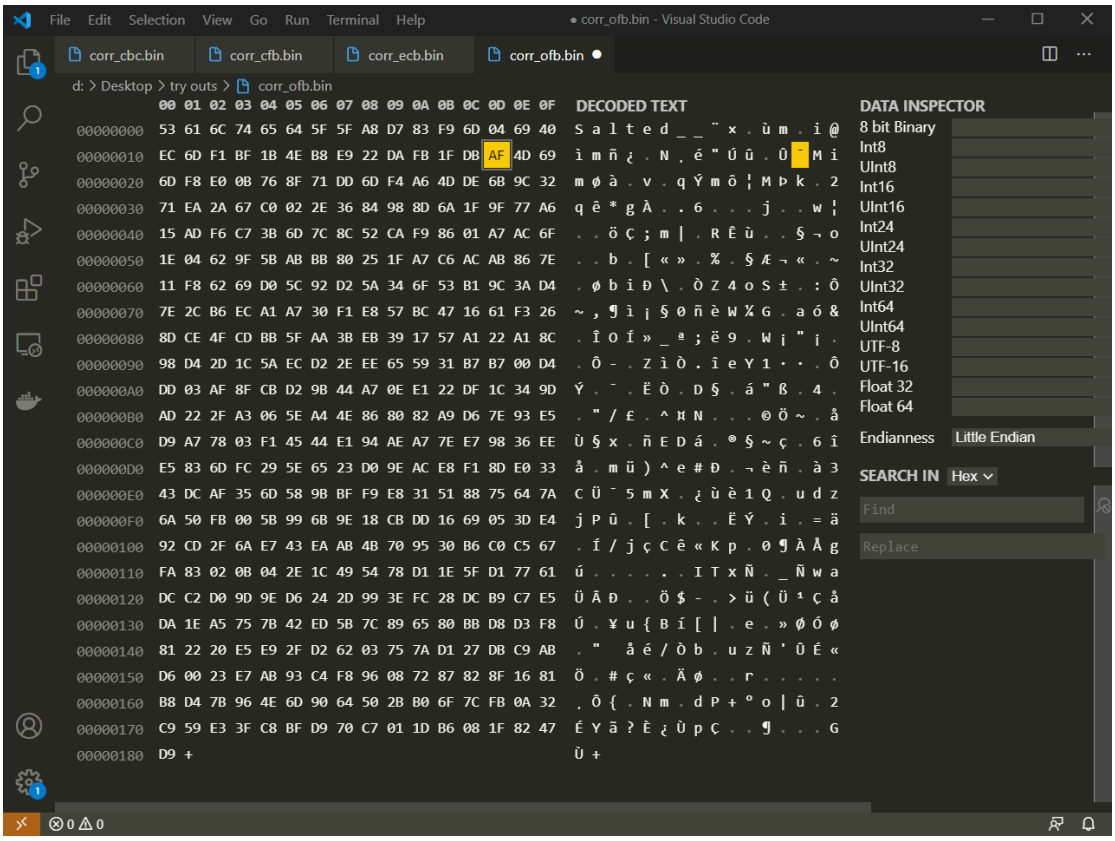
```
D:\Desktop\try outs>openssl enc -aes-128-cbc -e -in plain_text.txt -out corr_cbc.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -aes-128-ofb -e -in plain_text.txt -out corr_ofb.bin
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -aes-128-cfb -e -in plain_text.txt -out corr_cfb.bin
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -aes-128-ecb -e -in plain_text.txt -out corr_ecb.bin
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using VScode.



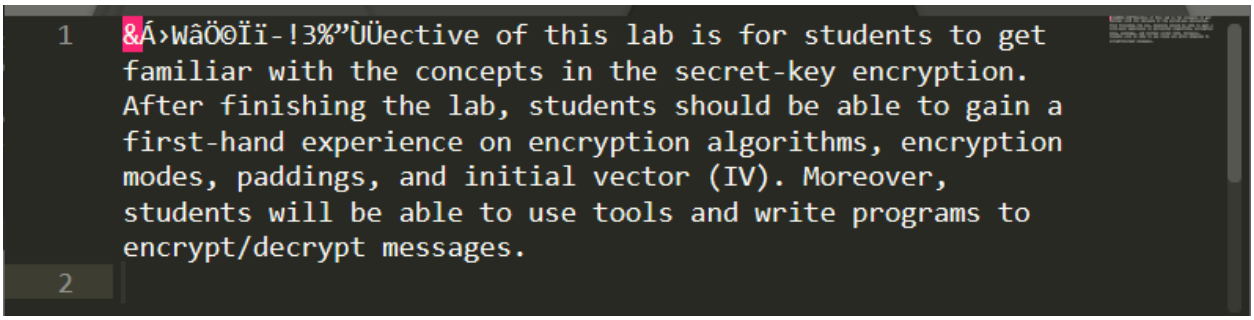

File Edit Selection View Go Run Terminal Help
corr_cbc.bin - Visual Studio Code
corr_cbc.bin
corr_cfb.bin
corr_ecb.bin
corr_ofb.bin
d: > Desktop > try outs > corr_cbc.bin
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
DECODED TEXT
00000000 53 61 6C 74 65 64 5F 5F 5B 49 17 EB 7C 7E E9 9F Salted__[I.ë|~é.
00000010 C8 B3 24 9A BE 8D 7A 54 64 AE 7B 51 A9 AF 9E 61 È³$.¾.zTd®{Q©¯.a
00000020 4C 37 DE 58 9C 6E 5F BE 34 5C 63 B0 A0 E3 40 F8 L7ÞX.n_¾4\c°.ã@ø
00000030 74 58 0B 65 B7 5F 60 65 FB E0 14 F0 CF 70 26 E1 tX.e·_`eûà.ðÏp&á
00000040 50 E8 40 EA E7 A2 CC 67 2E C4 D4 C1 66 77 CE 10 Pè@êç¢Ìg.ÄÔÁfwÎ.
00000050 2B D6 23 4D 01 13 20 ED 40 0F B7 D4 9E D7 FB 53 +Ö#M.. í@.·Ô.×ûS
00000060 97 07 EB 61 90 25 34 FA 04 D0 C4 80 72 63 15 F5 ..ëa.%4ú.ÐÄ.rc.õ
00000070 C5 04 B9 60 20 AF 03 3B 35 1C EC F5 C0 9A 59 A0 Å.¹` ¯.;5.ìõÀ.Y.
00000080 B0 5C 30 0C 08 24 E8 50 78 E8 26 CD AF 32 64 11 °\0..$èPxè&Í¯2d.
00000090 2C 3F 05 97 DA 25 2F 07 B0 EB DD 91 40 44 B1 3C ,?..Ú%/.°ëÝ.@D±<
000000A0 EC BB AD D7 00 F7 B8 F1 51 8C FC 8A 4A B4 3B 68 ì».×.÷¸ñQ.üJ´;h
000000B0 CE 70 40 B1 57 5B 52 28 78 7A 51 19 F0 25 6C E8 Îp@±W[R(xzQ.ð%lè
000000C0 C9 2C 20 77 88 AB 8F 24 89 B4 27 51 7F 32 4A 2B É, w.«.$.´'Q.2J+
000000D0 F5 39 E1 42 6D A1 87 3A 4C BE A8 D8 CD A8 24 85 õ9áBm¡.:L¾¨ØÍ¨$.
000000E0 26 3D F5 60 5C F4 6D DD B4 E3 C4 14 08 86 1C 64 &=õ`\ômÝ´ãÄ....d
000000F0 53 CB 4C 70 E7 82 3E 27 C5 EB C8 4D DB 6C 91 74 SËLpç.>'ÅëÈMÛl.t
00000100 39 B1 0E 5D D4 E6 D6 54 9E E3 62 A8 5E 90 06 BF 9±.]ÔæÖT.ãb¨^..¿
00000110 B8 78 E9 48 EE 9A A5 04 EC 21 27 1F 76 1A FA BD ¸xéHî.¥.ì!'.v.ú½
00000120 78 F2 10 DA C7 CE 52 FA D2 2A 0E EF D4 92 6A FF xò.ÚÇÎRúÒ*.ïÔ.jÿ
00000130 E8 B1 32 F1 56 B2 98 46 28 B6 AD 9C FA C6 49 B4 è±2ñV².F(¶..úÆI´
00000140 31 59 AC AB 8F F2 AA AC 78 A7 4C E8 8E 7E 5B 73 1Y¬«.òª¬x§Lè.~[s
00000150 A8 A4 D4 44 98 C5 27 D1 21 B0 BB D2 D5 BC 00 67 ¨¤ÔD.Å'Ñ!°»ÒÕ¼.g
00000160 27 F8 3F 54 81 1E B8 EB 6D 9D F6 81 5C 39 D1 32 'ø?T..¸ëm.ö.\9Ñ2
00000170 FF B5 9F EE C4 CD 2D 48 DF D1 A7 43 AD 68 EF 46 ÿµ.îÄÍ-HßÑ§C.hïF
00000180 B3 4E 4B D1 45 3C 75 85 C9 E3 7E FA 86 78 23 1E ³NKÑE<u.Éã~ú.x#.
00000190 +
DATA INSPECTOR
8 bit Binary 10011110
Int8 -98
UInt8 158
Int16 24990
UInt16 24990
Int24 5005726
UInt24 5005726
Int32 927752606
UInt32 927752606
Int64 7970343151972671902
UInt64 7970343151972671902
UTF-8 ◆
UTF-16 徻
Float 32 0.000012182075806776
Float 64 6.5579189173936994e+
Endianness Little Endian
SEARCH IN Hex
Find
Replace
0 0


File Edit Selection View Go Run Terminal Help
• corr_cfb.bin - Visual Studio Code
corr_cbc.bin
corr_cfb.bin •
corr_ecb.bin
corr_ofb.bin
d: > Desktop > try outs > corr_cfb.bin
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
DECODED TEXT
00000000 53 61 6C 74 65 64 5F 5F 17 EC 9E CD 42 97 85 38 Salted__.ì.ÍB..8
00000010 B6 35 8C 9E 44 0A AB FF E8 D6 2A 2D FA AF 41 35 ¶5..D.«ÿèÖ*-ú¯A5
00000020 12 79 92 55 BC B0 87 55 8C 8A B9 5B 80 6B CE 54 .y.U¼°.U..¹[.kÎT
00000030 82 87 17 EC D6 54 A5 AD 73 A7 1B FD 5F C6 4C 60 ...ìÖT¥.s§.ý_ÆL`
00000040 2C 3A AB 3E 02 E9 93 99 CF C6 9E BD 14 A6 7D 65 ,:«>.é..ÏÆ.½.¦}e
00000050 8B CA 80 D5 31 80 21 1D 88 A8 4D DE 3B AF 30 FB .Ê.Õ1.!..¨MÞ;¯0û
00000060 E1 58 6A 4F 38 A8 6C AC 78 03 01 F5 26 CA 79 B5 áXjO8¨l¬x..õ&Êyµ
00000070 9C EA CE EA 42 EB 04 93 CC 92 08 75 5B 22 4B BF .êÎêBë..Ì..u["K¿
00000080 43 69 70 99 F6 6B 14 49 92 40 6F BE D7 18 A8 10 Cip.ök.I.@o¾×.¨.
00000090 F7 60 46 DE 4A 40 F9 71 97 20 E8 70 C4 EB 61 B7 ÷`FÞJ@ùq. èpÄëa·
000000A0 4D 6A B8 88 47 82 C9 1F A9 56 35 09 22 F6 75 73 Mj¸.G.É.©V5."öus
000000B0 AE E4 A3 60 BF D8 63 E7 CC 29 64 86 76 23 68 1C ®ä£`¿ØcçÌ)d.v#h.
000000C0 10 99 33 DD 00 21 6A F5 8A 22 E9 60 68 7E C1 43 ..3Ý.!jõ."é`h~ÁC
000000D0 36 F3 55 0F FD EA CF CD 44 EE 66 9F 83 C2 F0 FF 6óU.ýêÏÍDîf..Âðÿ
000000E0 9B 85 03 E5 02 79 F7 AF 0E 68 CA 43 3F 05 7F 44 ...å.y÷¯.hÊC?..D
000000F0 3B 8A 70 76 03 9E 66 6E A0 DB C2 6E 66 1A 37 17 ;.pv..fn.ÛÂnf.7.
00000100 F9 A2 4B 15 CC 4B 7F ED 10 36 24 2E 54 74 48 8D ù¢K.ÌK.í.6$.TtH.
00000110 D2 CD BE 69 F5 DC 88 D5 B4 7B D5 EB 8E EC 67 10 ÒÍ¾iõÜ.Õ´{Õë.ìg.
00000120 43 FA 2E 17 10 6E 32 AD 52 4C 8B 94 62 C7 F8 4E Cú...n2.RL..bÇøN
00000130 5C 22 95 F0 DE 8F A0 5A 65 49 4F A3 91 C9 05 C7 \"..ðÞ..ZeIO£.É.Ç
00000140 35 AC 58 DC 4C 8E E8 D6 10 10 39 35 4D 03 43 4D 5¬XÜL.èÖ..95M.CM
00000150 9F A7 1B 9D 8C 47 EE 9D F0 7E DE BB 08 22 AC 06 .§...GÎ.ð~Þ».".¬.
00000160 FE EF 7A C3 C7 A1 0F F8 03 A7 2C DF 27 D7 75 51 þïzÃÇ¡.ø.§,ß'×uQ
00000170 AA F1 73 7A F0 E1 EE 7D 26 27 97 D1 1D 2A 8F 62 ªñszðáî}&'.Ñ.*.b
00000180 74 +
t +
DATA INSPECTOR
8 bit Binary 01000001
Int8 65
UInt8 65
Int16 13633
UInt16 13633
Int24 1193281
UInt24 1193281
Int32 2031236417
UInt32 2031236417
Int64 -5711596139827808959
UInt64 12735147933881742657
UTF-8 A
UTF-16 㡁
Float 32 4.744721618035261e+3
Float 64 -6.264307179942616e-
Endianness Little Endian
SEARCH IN Hex
Find
Replace
0 0

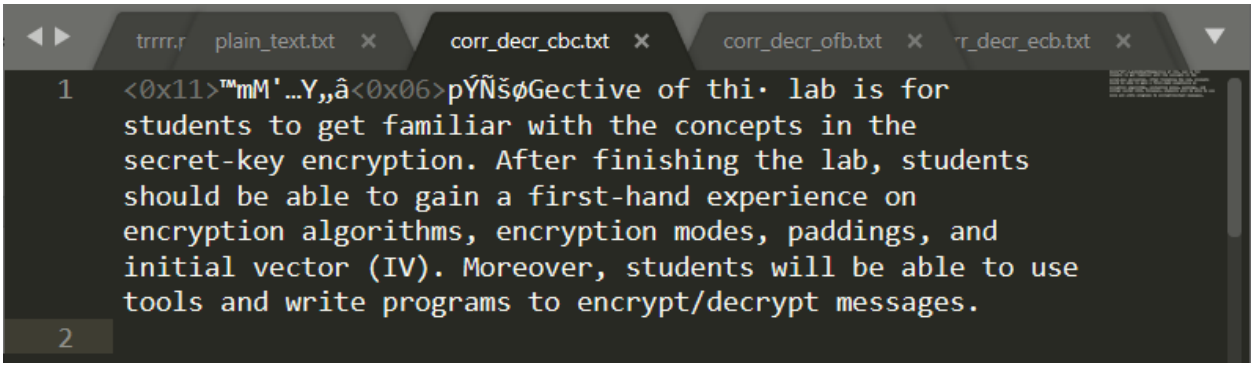


4. Decrypt the corrupted file (encrypted) using the correct key and IV.

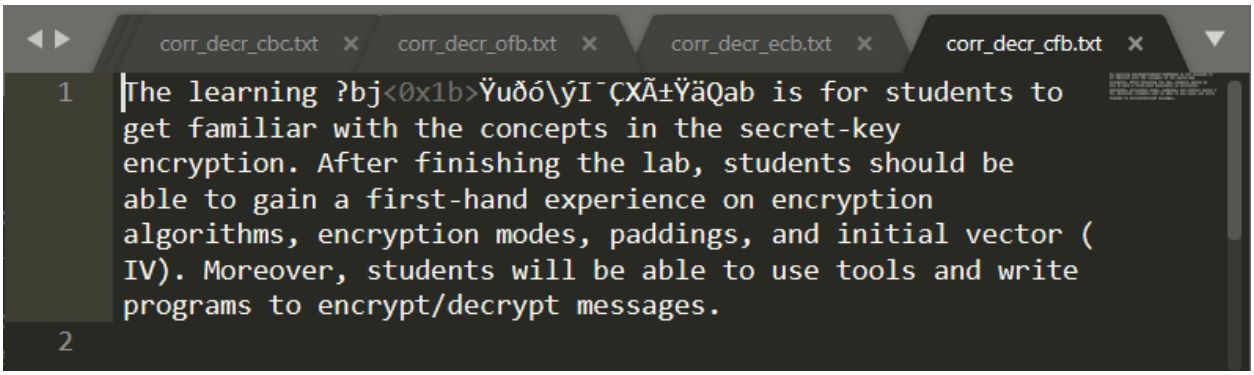
ECB:



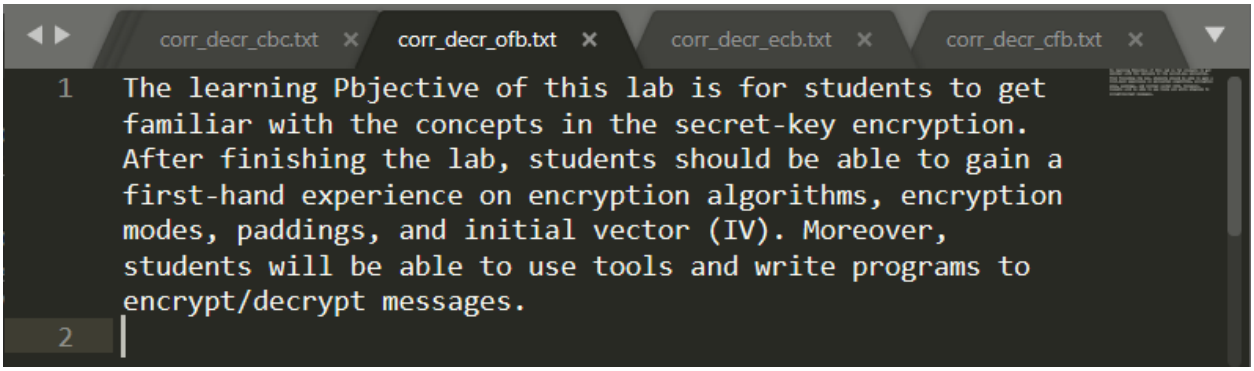
CBC:



CFB:



OFB:



How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?
Please explain why.

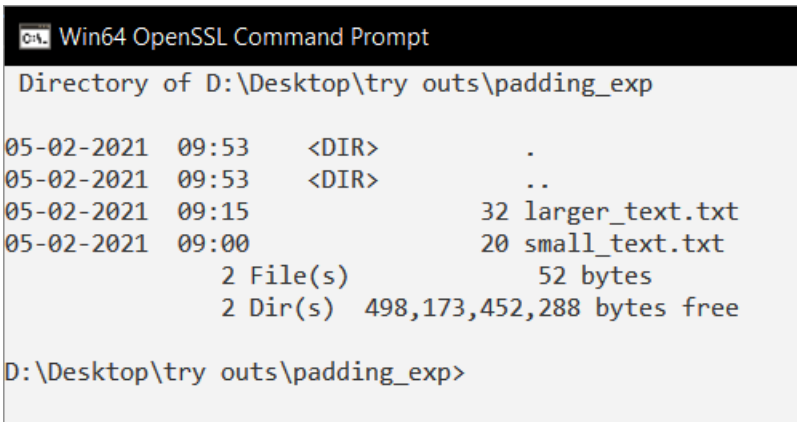
- In ECB mode, only one block is affected i.e. the block having the faulty byte and the rest of the text is unaffected.
- In CBC mode, there was effect in two blocks when one bit of the ciphertext was corrupted.
- In CFB mode, when one ciphertext bit is corrupted, only the next two plaintext blocks are affected.
- In OFB mode, if one bit of a ciphertext or plaintext message is corrupted then only one bit of the plaintext or ciphertext is corrupted.

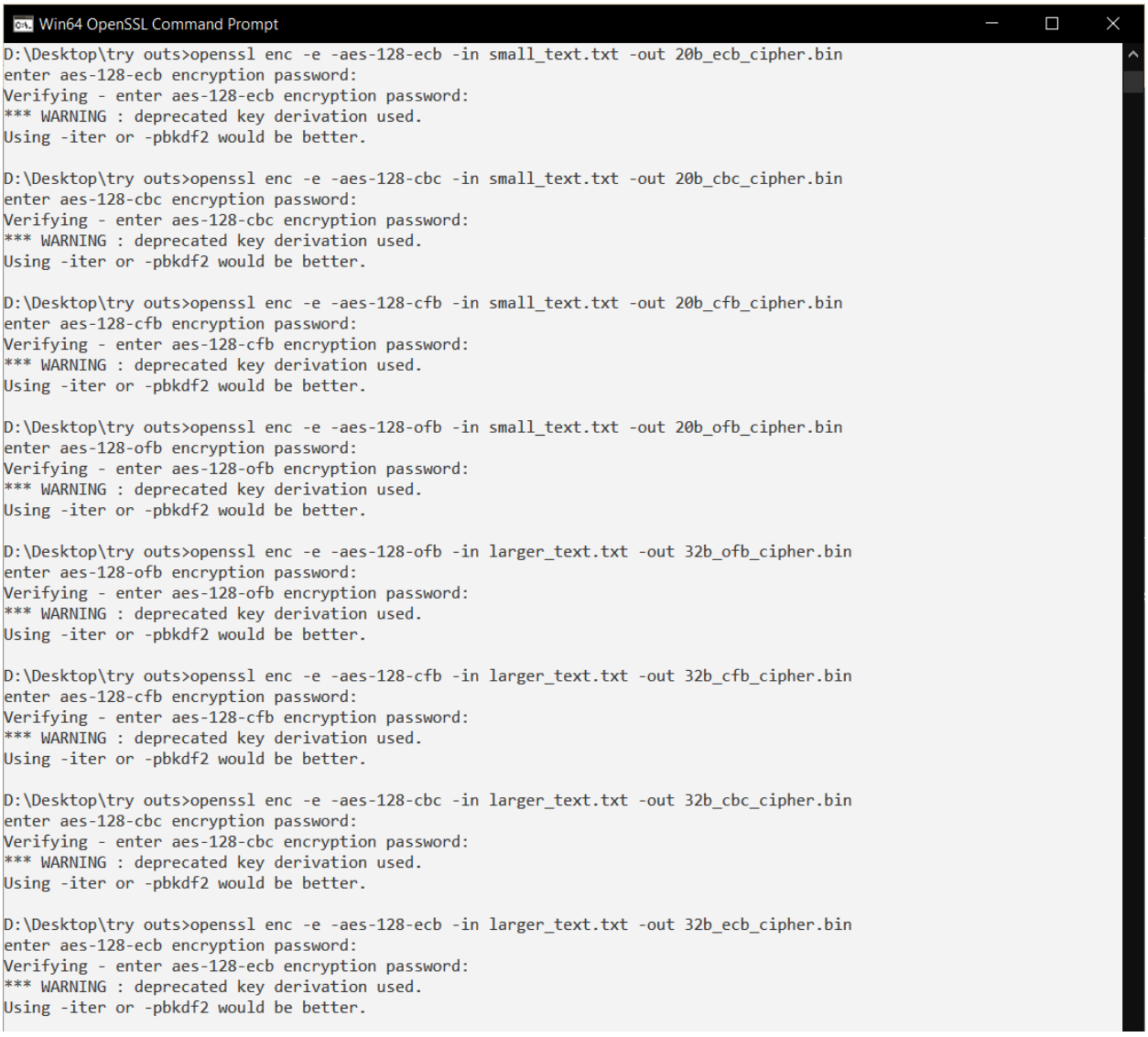
3.4 Task4 : Padding

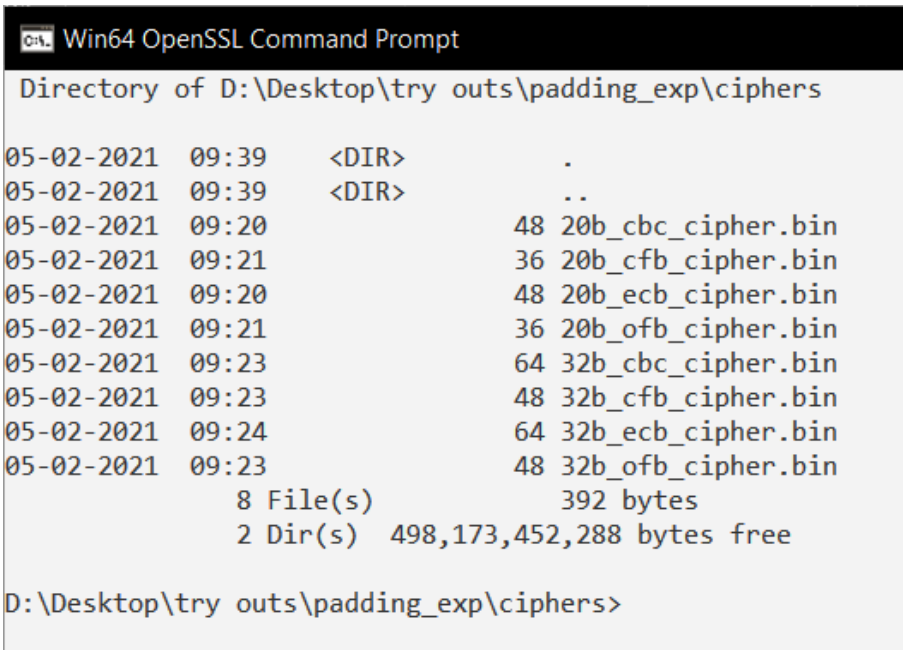
For block ciphers, when the size of the plaintext is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

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.

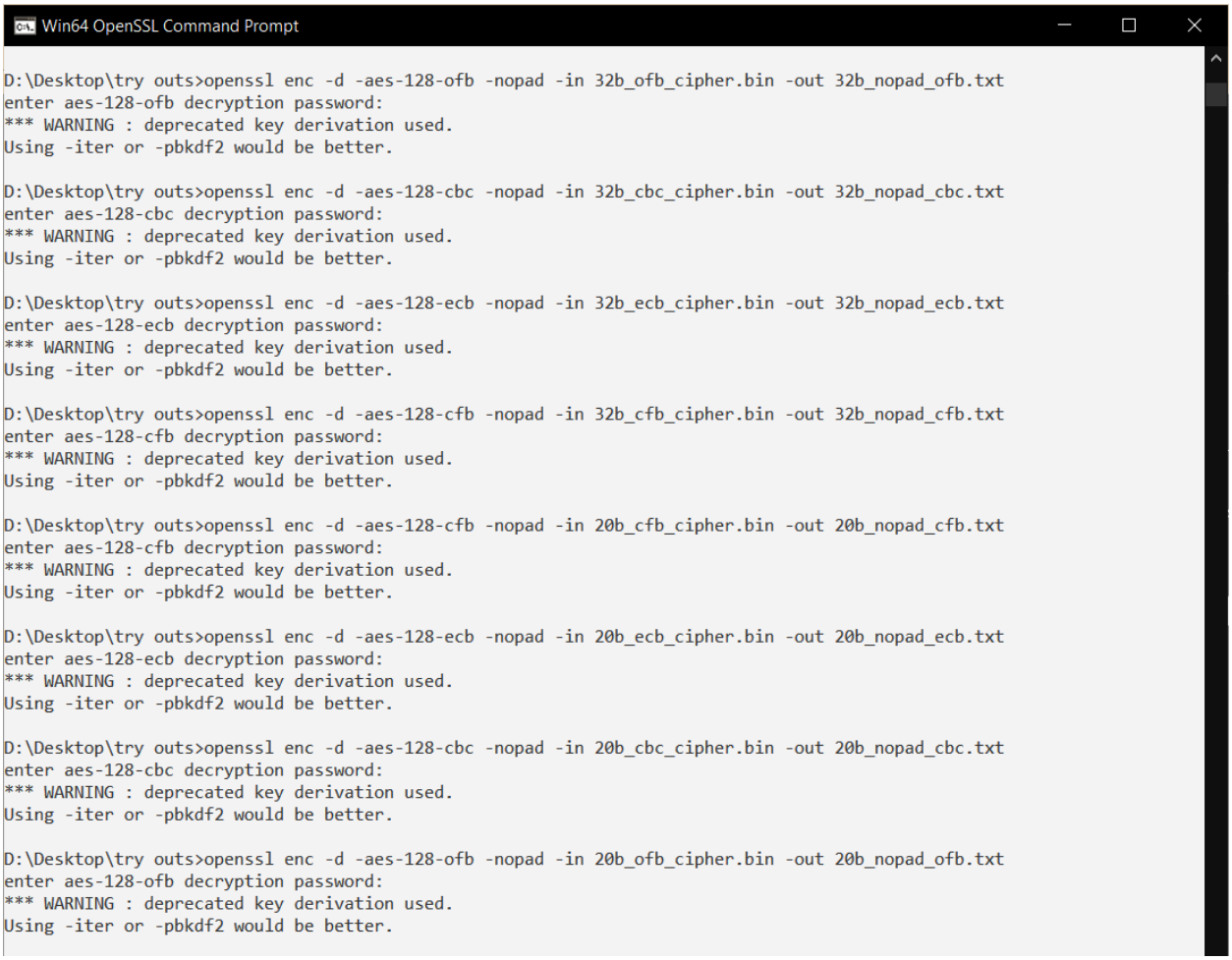
2. Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.


Win64 OpenSSL Command Prompt
Directory of D:\Desktop\try outs\padding_exp
05-02-2021 09:53 <DIR> .
05-02-2021 09:53 <DIR> ..
05-02-2021 09:15 32 larger_text.txt
05-02-2021 09:00 20 small_text.txt
2 File(s) 52 bytes
2 Dir(s) 498,173,452,288 bytes free
D:\Desktop\try outs\padding_exp>

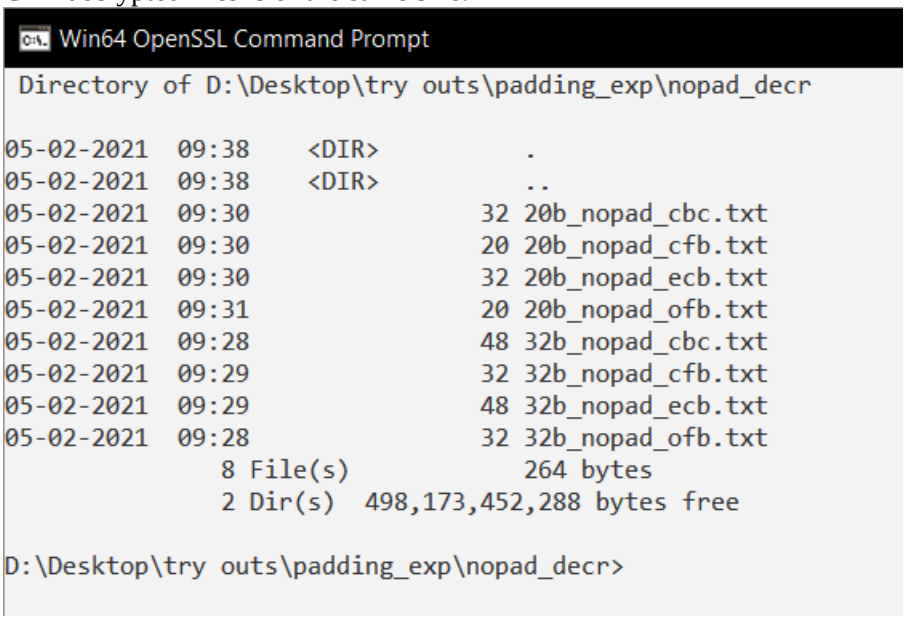

Win64 OpenSSL Command Prompt
D:\Desktop\try outs>openssl enc -e -aes-128-ecb -in small_text.txt -out 20b_ecb_cipher.bin
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-cbc -in small_text.txt -out 20b_cbc_cipher.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-cfb -in small_text.txt -out 20b_cfb_cipher.bin
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-ofb -in small_text.txt -out 20b_ofb_cipher.bin
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-ofb -in larger_text.txt -out 32b_ofb_cipher.bin
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-cfb -in larger_text.txt -out 32b_cfb_cipher.bin
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-cbc -in larger_text.txt -out 32b_cbc_cipher.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>openssl enc -e -aes-128-ecb -in larger_text.txt -out 32b_ecb_cipher.bin
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.



A way to verify that openssl use the PKCS5 padding is to decrypt the encrypt file with option –nopad. This option indicates disable standard block padding. Normally during the encryption by default, it will include the padding, so when decrypt the file if I use the nopad option I can see the padding in the decrypted file.



In the screenshot below it can be seen that the size of CBC and ECB decrypted files using the nopad option is 12 bytes more for the 20 bytes file and 16 bytes more for the 32 bytes file while the size of the OFB and CFB decrypted files is of the same size.



Results:

- Padding is needed for ECB and CBC encryption modes because their inputs contain number of blocks, thus padding could ensure that. Block size depends on the algorithm: AES uses 16 byte blocks.
- There is no need padding for encryption mode CFB and OFB because they are stream ciphers, in which the size of the block is usually fixed (one character).

3.5 Task 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. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides 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:

Plaintext (total 21 characters): This is a top secret. Ciphertext (in hex format): 8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9

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.

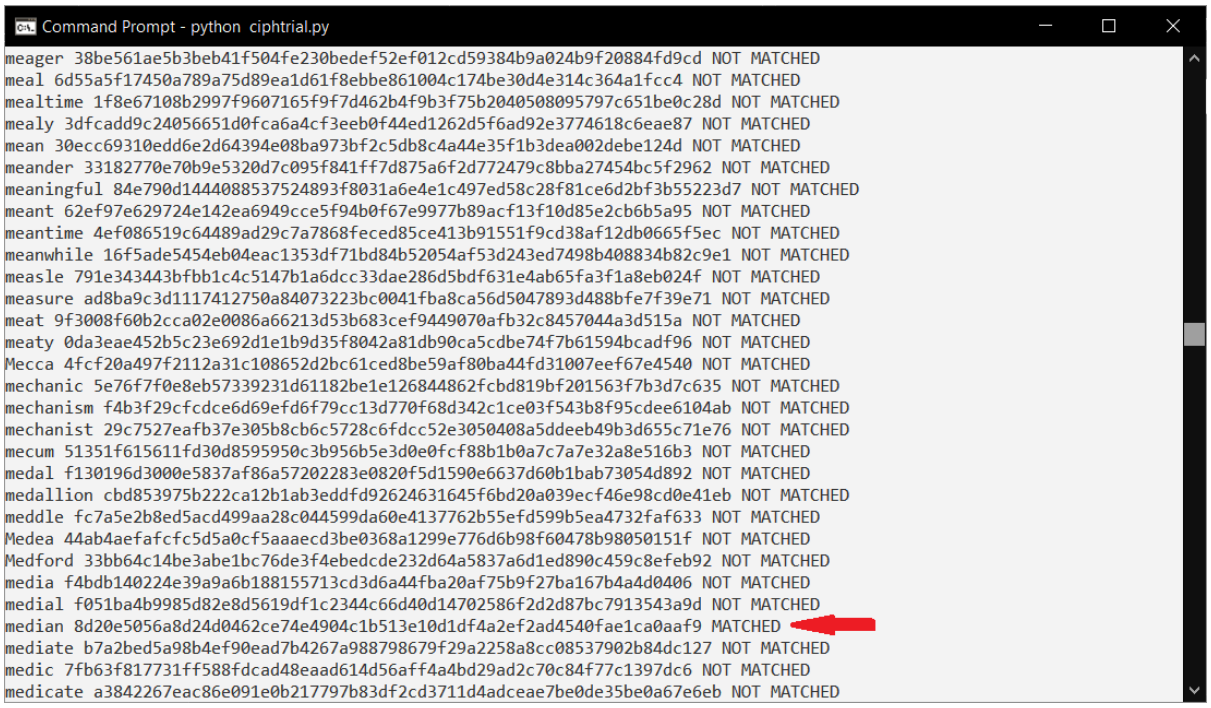
Note 3: To compile your code, you may need to include the header files in openssl, and link to openssl libraries. To do that, you need to tell your compiler where those files are. In your Makefile, you may want to specify the following:

```
INC=/usr/local/ssl/include/
LIB=/usr/local/ssl/lib/

all:
	gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto -ldl
```

Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

plain_text = b"This is a top secret."
cipher_hex = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9
"

res_keys = []

file = open('words.txt', 'r')
lines = file.readlines()
print(len(lines))
words = [str.strip(line) for line in lines]

for word in words:
    if len(word) >= 16:
        continue
    key = word.encode() + b' '*(16-len(word))
    cipher = AES.new(key, AES.MODE_CBC, iv=bytes.fromhex('0'*32))
    ciphertext = cipher.encrypt(pad(plain_text, AES.block_size))

    is_matched = "NOT MATCHED"
    if bytes.hex(ciphertext) == cipher_hex:
        is_matched = "MATCHED"
        res_keys.append(word)

    print(word, bytes.hex(ciphertext), is_matched)

print("\n\nResulting Key:",res_keys)
```

Output:



The result shows that the key that used to encrypt the given plain text is **median**.

Conclusion:

- In this experiment, I learned about various encryption methods such as AES and their different encryption modes such as ECB, CBC, CFB, OFB, etc and I implemented them using Openssl utility.
- ECB mode encrypts the identical plain text blocks to identical encrypted text blocks and hence it is less secure.
- From the experiment, I deduced that ECB and CBC employ padding while encrypting, whereas the other two don't. ECB and CBC are block cyphers, but CFB and OFB are stream cyphers, as evidenced by this.
- In OFB mode, if the single digit of the 30th byte corrupted, then in plain text that only that byte or character is corrupted. Thus, only OFB mode shows the most promising result in task 3 and almost all the texts are recovered.
- I concluded from task 5 that if I have the ciphertext and the key space, I can easily locate the plaintext using the brute force method.

Github Link:

https://github.com/sumeethaldipur/CSS_LAB