

<b>Experiment 3:</b>	<b>Crypto Encryption</b>
----------------------	--------------------------

**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

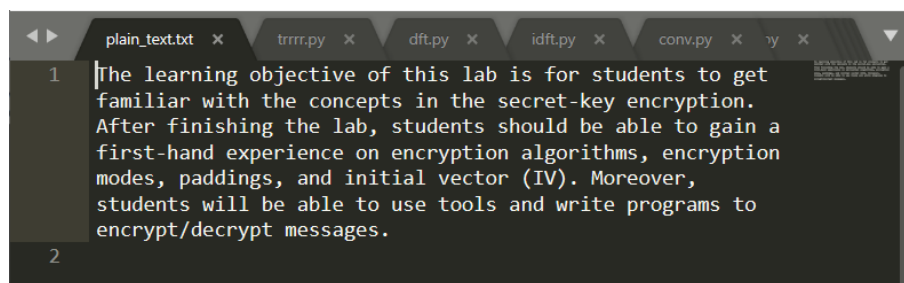
In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man enc.

```
% openssl enc ciphertype -e -in plain.txt -out cipher.bin \
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)

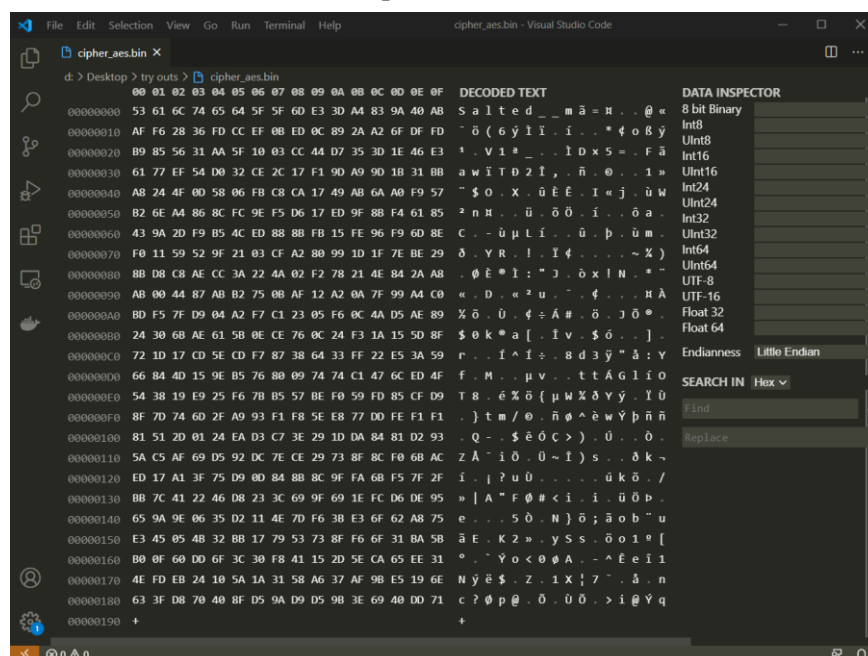
## Encryption of plain text using different Ciphers:



### 1. Encryption Using aes-256-cbc:

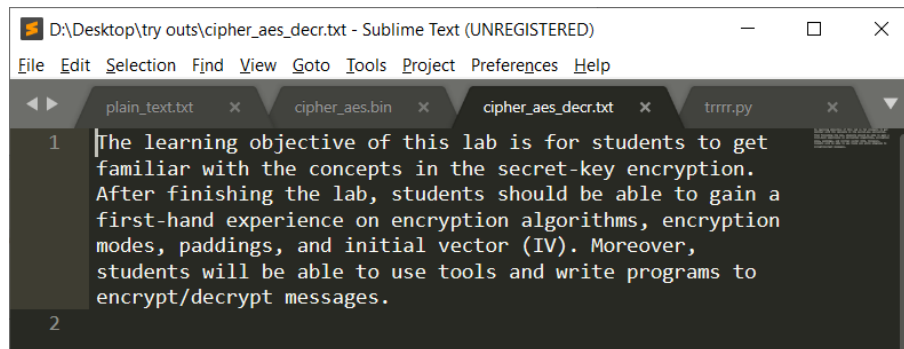
```
D:\Desktop\try outs>openssl enc -aes-256-cbc -e -in plain_text.txt -out cipher_aes.bin
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>
```

#### cipher\_aes.bin



## Decryption of cipher\_aes.bin file

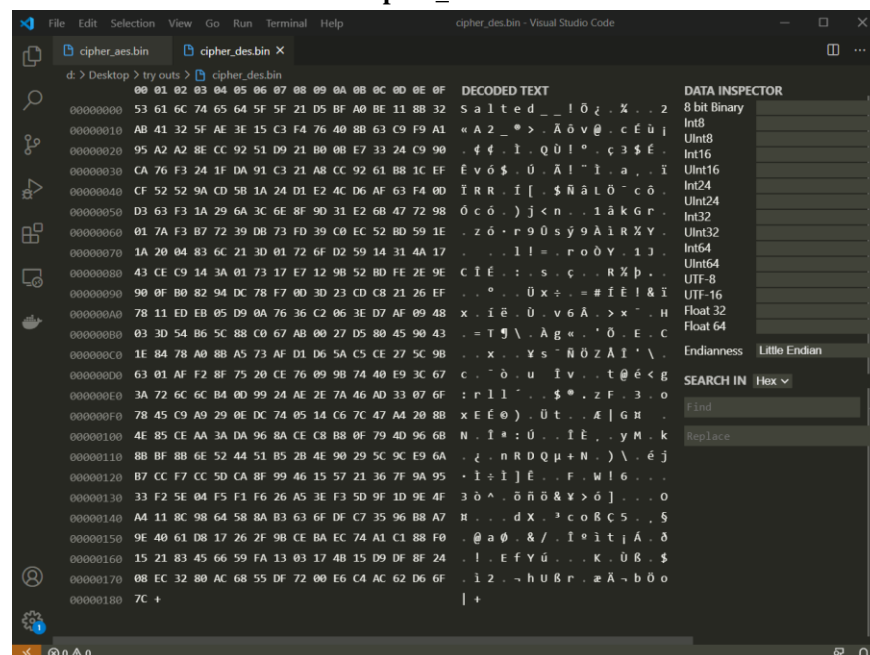
```
D:\Desktop\try outs>openssl enc -aes-256-cbc -d -in cipher_aes.bin -out cipher_aes_decr.txt
enter aes-256-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>
```



## 2. Encryption using des-ede3-ofb:

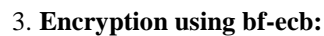
```
D:\Desktop\try outs>openssl enc -des-ede3-ofb -e -in plain_text.txt -out cipher_des.bin
enter des-ede3-ofb encryption password:
Verifying - enter des-ede3-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
D:\Desktop\try outs>
```

### cipher\_des.bin

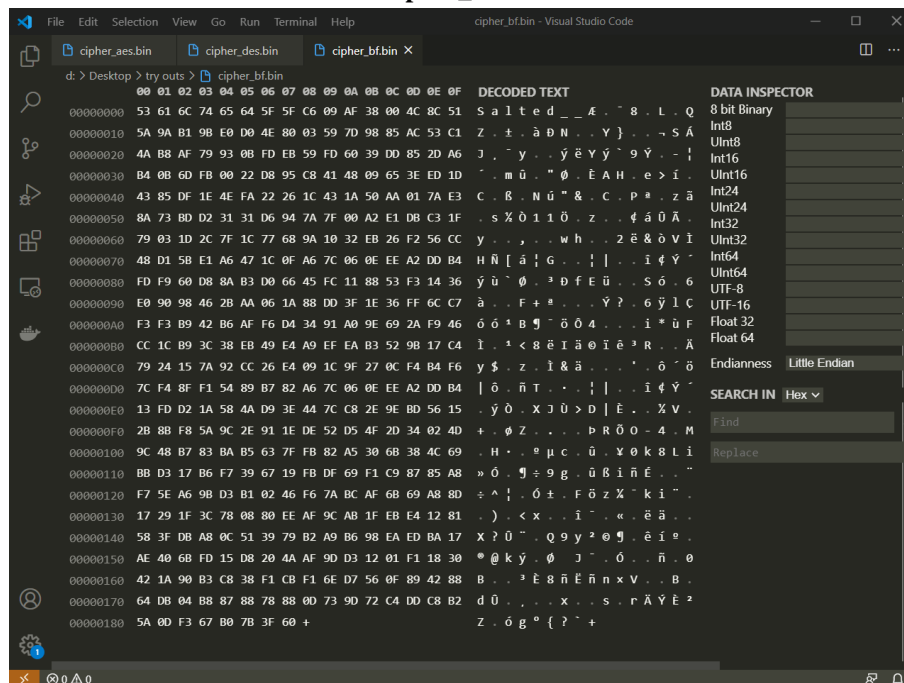


```
D:\Desktop\try outs>openssl enc -des-ede3-ofb -d -in cipher_des.bin -out cipher_des_decr.txt
enter des-ede3-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>
```



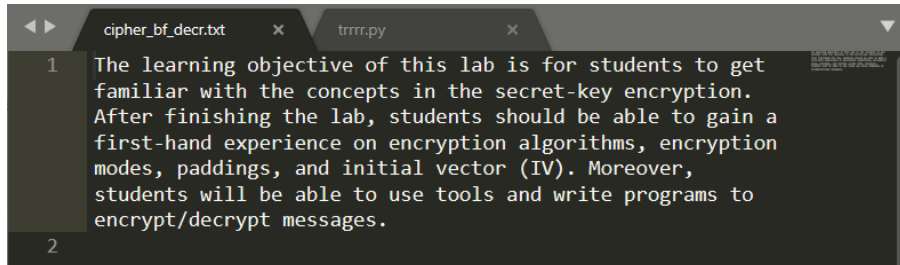
**cipher\_bf.bin**



## 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>
```



## Observations:

I learnt about different encryption techniques and implemented the same in this particular task.

### 3.2 Task 2: Encryption Mode – ECB vs. CBC

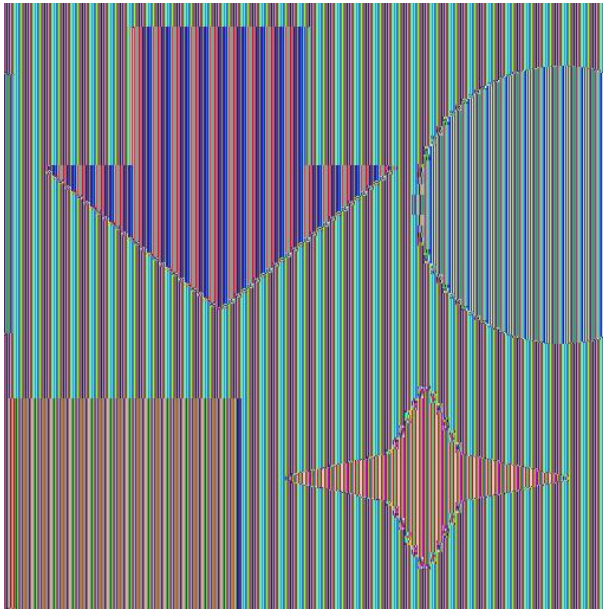
The file `pic original.bmp` contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

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.

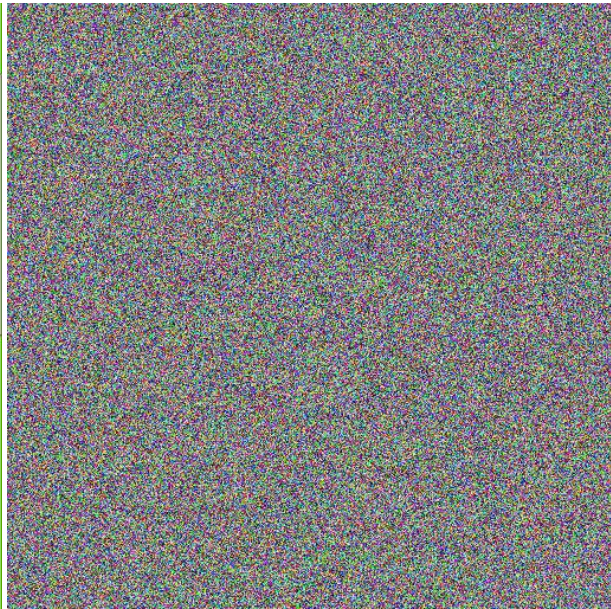
## Original Picture:



**Picture Encrypted Using ECB:**



**Picture Encrypted Using CBC:**



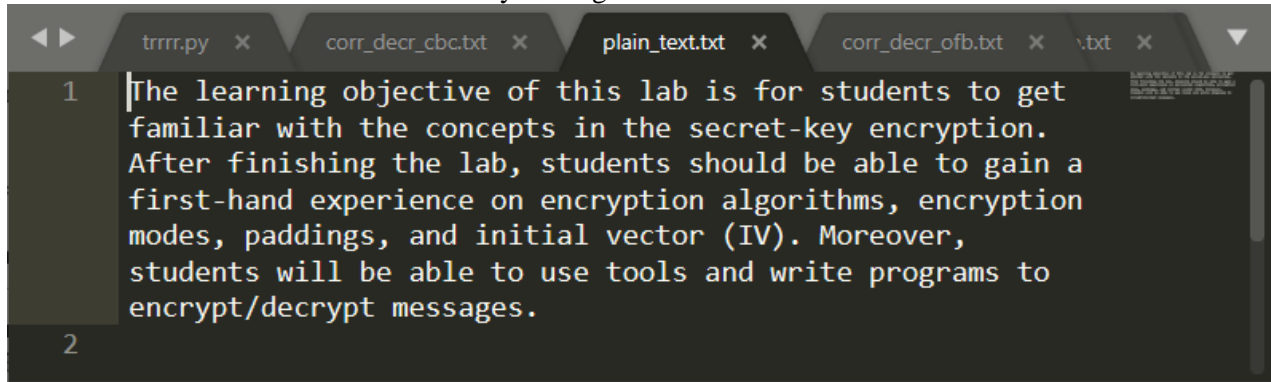
**Observations:**

- 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.



```
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. 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.

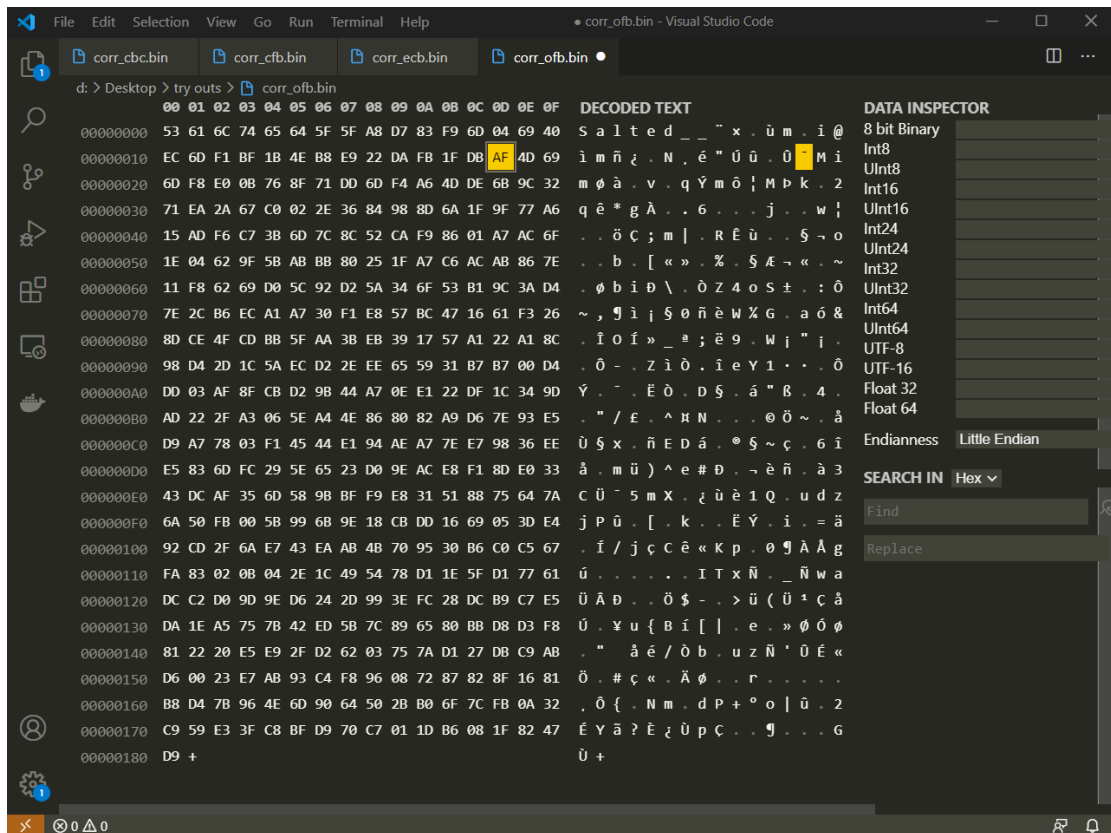
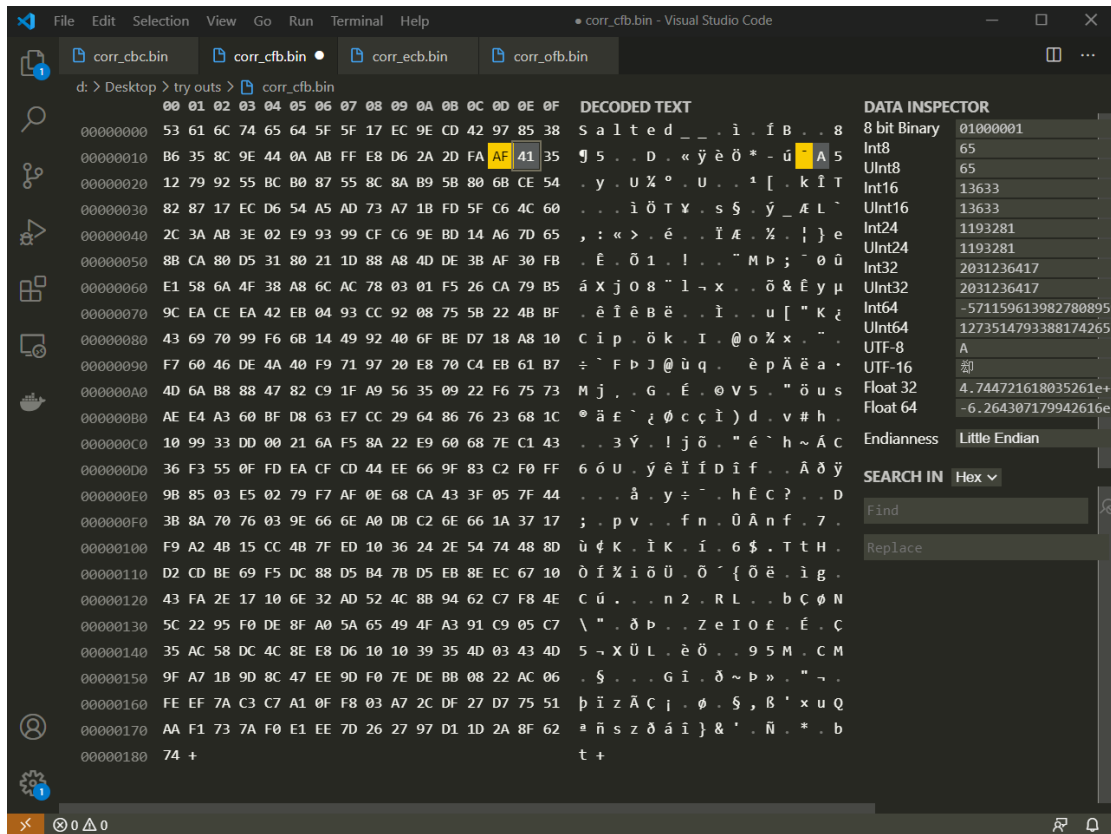


Visual Studio Code interface showing the file explorer with open files: corr\_cbc.bin, corr\_cfb.bin, corr\_ecb.bin, and corr\_ofb.bin. The main editor displays the hex dump and decoded text for corr\_ecb.bin. The hex dump shows a sequence of bytes starting with 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F. The decoded text shows a salted password: "Salted\_\_0...15ðB@ (ôP.)ð.Ýø...øð.μ èπ.7`...e h.D E= %I Ì «μ.y ú'x? q Ì x...J ¶ ü \* . ú : > r ß ò . ú « 3 m ò ù ç î w . . . x " / ü . 2 [ o \$ ñ Ä 5 . . ø c . u ø ~ . e ö B è . \_ I N % k . ð ' @ . " « ! . o . ò ñ % e A k È x % = i s i . l ~ Ì È . . 3 Ì Û á Y + 3 . ð Z j \$ . b . á ° . û ð . c t . . . Ä % ! X . È , Ì I Æ Y . . f . . ð u f 2 h . i . ' . Ä . ) F e ? ? è T { è } . ø W Ñ È Ì μ . 5 . D H ° V Ó ö \* m ð C ³ 3 . B È . ø g i { . . . Ì . Ñ È Ñ 5 ° ø ! 4 . Ä Æ . . : à Ò V G . ] J Ý x ä . . . % X . Æ ( . ð x . þ È x R d u \* ö μ ~ . ` z . ø ð . þ Q 1 . . . ä \_ y ` h l . . Ì < % f . . e . . . . . ø . = ` 7 s È ä % p N Ò . Û . g ³ È e . \$ . ³ . ð W Ì È . ð - p 2 ð ñ Ò d ] ü . ð . . . Ä ä . x . % . . 9 a % # . M Ì x B O . D . T \* x ` a . +

The DATA INSPECTOR panel on the right shows the data type as 8 bit Binary, with various integer and float types listed. The SEARCH IN panel shows the search scope set to Hex.

Visual Studio Code interface showing the file explorer with open files: corr\_cbc.bin, corr\_cfb.bin, corr\_ecb.bin, and corr\_ofb.bin. The main editor displays the hex dump and decoded text for corr\_cbc.bin. The hex dump shows a sequence of bytes starting with 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F. The decoded text shows a salted password: "Salted\_\_ [ I . è | ~ é . è ³ \$ . % . z T d ° { Q ø . a L 7 þ X . n \_ % 4 \ c ° . ä @ ø t X . e . \_ ` e ù ä . ð Ì p & ä P è @ è ç ø Ì g . Ä Ò Ä f w Ì . + ð # M . . Í @ . . ð . x ù S . è a . % 4 ú . ð Ä . r c . ð Ä . ¹ ~ . . ; 5 . ì ò Ä . Y . ° \ ø . . \$ è P x è & f - 2 d . , ? . . ú % / . ° è Ý . @ D ± < Ì » . x . ÷ . ñ Q . ü . J ~ ; h Ì p @ ± W [ R ( x z Q . ð % l è É , w . « . \$ . ' Q . 2 J + ò 9 á B m j . : L % " ø Ì " \$ . & = ð ~ \ ò m Ý ~ ä Ä . . . d S È L p ç . > ' Ä È M Ò l . t 9 ± . ] Ö æ Ò T . ä b " ^ . . ð . x é H Ì . % . ì ! ' . v . ú % x ò . Ü Ç Ì R ú Ò \* . Ì Ò . j ý è ± 2 ñ V ² . F ( ¶ . . ü Ä I ~ 1 Y ~ « . ð æ ~ x \$ L è . ~ [ s " H Ò D . Ä ' Ñ l ° » ð Ö % . g ' ø ? T . . . ' è m . ö . \ 9 Ñ 2 ý μ . Ì Ä Í - H B Ñ \$ C . h Ì F ³ N K Ñ È < u . è ä ~ ú . x # . +

The DATA INSPECTOR panel on the right shows the data type as 8 bit Binary, with various integer and float types listed. The SEARCH IN panel shows the search scope set to Hex.



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

## ECB:

```
1 &Á>WâÖÖİi-!3%”ÜÜective 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
```

## CBC:

```
trrrr plain_text.txt x corr_decr_cbc.txt x corr_decr_ofb.txt x r_decr_ecb.txt x
1 <0x11>""mM'...Y,,â<0x06>pÝÑšøGective of thi· 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
```

## CFB:

```
corr_decr_cbc.txt x corr_decr_ofb.txt x corr_decr_ecb.txt x corr_decr_cfb.txt x
1 |The learning ?bj<0x1b>ÿuðó\ýI`ÇXÃ±ÿäQab 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
```

## OFB:

```
corr_decr_cbc.txt x corr_decr_ofb.txt x corr_decr_ecb.txt x corr_decr_cfb.txt x
1 The learning Pbjective 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
```

**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.
```

```
Win64 OpenSSL Command Prompt
Directory of D:\Desktop\try outs\padding_exp\ciphers

05-02-2021  09:39    <DIR>          .
05-02-2021  09:39    <DIR>          ..
05-02-2021  09:20             48 20b_cbc_cipher.bin
05-02-2021  09:21             36 20b_cfb_cipher.bin
05-02-2021  09:20             48 20b_ecb_cipher.bin
05-02-2021  09:21             36 20b_ofb_cipher.bin
05-02-2021  09:23             64 32b_cbc_cipher.bin
05-02-2021  09:23             48 32b_cfb_cipher.bin
05-02-2021  09:24             64 32b_ecb_cipher.bin
05-02-2021  09:23             48 32b_ofb_cipher.bin
                8 File(s)                392 bytes
                2 Dir(s) 498,173,452,288 bytes free

D:\Desktop\try outs\padding_exp\ciphers>
```

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.

```
Win64 OpenSSL Command Prompt

D:\Desktop\try outs>openssl enc -d -aes-128-ofb -nopad -in 32b_ofb_cipher.bin -out 32b_nopad_ofb.txt
enter aes-128-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-cbc -nopad -in 32b_cbc_cipher.bin -out 32b_nopad_cbc.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-ecb -nopad -in 32b_ecb_cipher.bin -out 32b_nopad_ecb.txt
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-cfb -nopad -in 32b_cfb_cipher.bin -out 32b_nopad_cfb.txt
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-cfb -nopad -in 20b_cfb_cipher.bin -out 20b_nopad_cfb.txt
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-ecb -nopad -in 20b_ecb_cipher.bin -out 20b_nopad_ecb.txt
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-cbc -nopad -in 20b_cbc_cipher.bin -out 20b_nopad_cbc.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

D:\Desktop\try outs>openssl enc -d -aes-128-ofb -nopad -in 20b_ofb_cipher.bin -out 20b_nopad_ofb.txt
enter aes-128-ofb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

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.

```
Win64 OpenSSL Command Prompt

Directory of D:\Desktop\try outs\padding_exp\nopad_decr

05-02-2021  09:38    <DIR>          .
05-02-2021  09:38    <DIR>          ..
05-02-2021  09:30                32 20b_nopad_cbc.txt
05-02-2021  09:30                20 20b_nopad_cfb.txt
05-02-2021  09:30                32 20b_nopad_ecb.txt
05-02-2021  09:31                20 20b_nopad_ofb.txt
05-02-2021  09:28                48 32b_nopad_cbc.txt
05-02-2021  09:29                32 32b_nopad_cfb.txt
05-02-2021  09:29                48 32b_nopad_ecb.txt
05-02-2021  09:28                32 32b_nopad_ofb.txt
                8 File(s)                264 bytes
                2 Dir(s) 498,173,452,288 bytes free

D:\Desktop\try outs\padding_exp\nopad_decr>
```

## 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/decrypt 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](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 plaintext 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:

```
Command Prompt - python ciphtrial.py
meager 38be561ae5b3beb41f504fe230bedef52ef012cd59384b9a024b9f20884fd9cd NOT MATCHED
meal 6d55a5f17450a789a75d89ea1d61f8ebbe861004c174be30d4e314c364a1fcc4 NOT MATCHED
mealtime 1f8e67108b2997f9607165f9f7d462b4f9b3f75b2040508095797c651be0c28d NOT MATCHED
mealy 3dfcadd9c24056651d0fca6a4c3eeb0f44ed1262d5f6ad92e3774618c6eae87 NOT MATCHED
mean 30ecc69310edd6e2d64394e08ba973bf2c5db8c4a44e35f1b3dea002debe124d NOT MATCHED
meander 33182770e70b9e5320d7c095f841ff7d875a6f2d772479c8bba27454bc5f2962 NOT MATCHED
meaningful 84e790d1444088537524893f8031a6e4e1c497ed58c28f81ce6d2bf3b55223d7 NOT MATCHED
meant 62ef97e629724e142ea6949cce5f94b0f67e9977b89acf13f10d85e2cb6b5a95 NOT MATCHED
meantime 4ef086519c64489ad29c7a7868feced85ce413b91551f9cd38af12db0665f5ec NOT MATCHED
meanwhile 16f5ade5454eb04eac1353df71bd84b52054af53d243ed7498b408834b82c9e1 NOT MATCHED
measle 791e343443fbfb1c4c5147b1a6dcc33dae286d5bdf631e4ab65fa3f1a8eb024f NOT MATCHED
measure ad8ba9c3d117412750a84073223bc0041fba8ca56d5047893d488bfe7f39e71 NOT MATCHED
meat 9f3008f60b2cca02e0086a66213d53b683cef9449070afb32c8457044a3d515a NOT MATCHED
meaty 0da3eae452b5c23e692d1e1b9d35f8042a81db90ca5cdbc74f7b61594bcad9f6 NOT MATCHED
Mecca 4fcf20a497f2112a31c108652d2bc61ced8be59af80ba44d31007eef67e4540 NOT MATCHED
mechanic 5e76f7f0e8eb57339231d61182be1e126844862fcbd819bf201563f7b3d7c635 NOT MATCHED
mechanism f4b3f29cfcdce6d69efdd6f79cc13d770f68d342c1ce03f543b8f95cdee6104ab NOT MATCHED
mechanist 29c7527eafbf37e305b8cb6c5728c6fdcc52e3050408a5ddeeb49b3d655c71e76 NOT MATCHED
mecum 51351f615611f30d8595950c3b956b5e3d0e0cf88b1b0a7c7a7e32a8e516b3 NOT MATCHED
medal f130196d3000e5837af86a57202283e0820f5d1590e6637d60b1bab73054d892 NOT MATCHED
medallion cbd853975b222ca12b1ab3edd92624631645f6bd20a039ecf46e98cd0e41eb NOT MATCHED
meddle fca7a5e2b8ed5acd499aa28c044599da60e4137762b55efd599b5ea4732faf633 NOT MATCHED
Medea 44ab4aefafcf5d5a0cf5aaaecd3be0368a1299e776d6b98f60478b98050151f NOT MATCHED
Medford 33bb64c14be3abe1bc76de3f4ebecdce232d64a5837a6d1ed890c459c8efeb92 NOT MATCHED
media f4bdb140224e39a9a6b188155713cd3d6a44fba20af75b9f27ba167b4a4d0406 NOT MATCHED
medial f051ba4b9985d82e8d5619df1c2344c66d40d14702586f2d2d87bc7913543a9d NOT MATCHED
median 8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9 MATCHED
mediate b7a2bed5a98b4ef90ead7b4267a988798679f29a2258a8cc08537902b84dc127 NOT MATCHED
medic 7fb63f817731ff588fdcad48eaad614d56aff4a4bd29ad2c70c84f77c1397dc6 NOT MATCHED
medicate a3842267eac86e091e0b217797b83df2cd3711d4adceae7be0de35be0a67e6eb NOT MATCHED
```

```
Command Prompt
zing f907a0ad5be217e0c71bf99c7d6be0046ee0d6df524d2433159272095c4490 NOT MATCHED
Zion 91923eb0c90eb2b4a7b6610e07848b63c802026b2e10a1bd2ef4d81c36949d8a NOT MATCHED
Zionism de40f1e5200eb205e887357c71bb4bed607d05077e3fb4847892067cbd85f14c NOT MATCHED
zip a3505f2264d430c100d53342f0074497f8060ace858936bfff783e108a8f59488 NOT MATCHED
zippy 6cd6bb040a19e505b3a0b599d5f26d490ffbe37845d0501dfd27b2990ce4933d NOT MATCHED
zircon 3e67f3da7fd47b698bf0bc83d1a640ebd5a18459c0ee4fdbcf4ee415b34dbdb7 NOT MATCHED
Zirconium 4dfefaf712686e075196990c0ebc0140ec0454f716b63e27ce8c520bc4631393c NOT MATCHED
zloty 445a85f023ed28e7792dc64d2a7a541228648b0b4d8e9aa8ebef6824f3e3a0ee NOT MATCHED
zodiac f60249772facc36ecec6b066e4d28318f3ee807962e6792b7ab4aa4f480b95eb1 NOT MATCHED
zodiacal d4a0fc67c0b1e3149cd61b9fd3ef70e57786b2af52cd493592874a922e8687fa NOT MATCHED
Zoe 7f40facd9077eec53ba1c74a12f6cc6d668fad5bc734035b919159b96166fd7a NOT MATCHED
Zomba 891aa32e7b28050c42fc3906640c724a293b4859f9e8e814f52fe1b7aa709a4 NOT MATCHED
zombie bf33265884ec3bf9cfa7ad29104b41ba6f53c05502eb621df6d552d595ec505 NOT MATCHED
zone 6402508dac10daffa266ff136cd450d3a9c749f9997ef9fc3fb178394124f399 NOT MATCHED
zoo 194962aa8e88ee06b3181559dd6ed77df1b37d26d9e8bb9c02d1ef7fe0331a1 NOT MATCHED
zoology ea12d5f693cddb45288980aa414e6b382e8321f99bc6734ac6c5e04d288c6cbb NOT MATCHED
zoom c7160fce104f33d04a4234aefae07855891738043635f6abc4a67716db45c40 NOT MATCHED
Zorn 7be43a4c15ada4d0672a2f638f1088a3c09f78cbb9d819798cc25d206efb540 NOT MATCHED
Zoroaster 2443c9f0280438d8ab2f775257d6c14a0c0d1ea08a49bd4df6f0b5e7a8c31f42 NOT MATCHED
Zoroastrian cc1af4a4dcc779dc9b99c42b7ab0b4de6f78a1606c4ef34506aa59c149c5e887 NOT MATCHED
zounds e1f04902b3a88d87857020411e46332a325a6ba96452103b18ba58a0b348a09 NOT MATCHED
z's add4648882d8499e11f8e0bf3eb28aaca7c5adbe2e4adf58f23d1f5a2a4d93a NOT MATCHED
zucchini d484c4616dfd87c70b537e9ad5a576792c9cb9f576dfd6e5b26c08287ee378cf NOT MATCHED
Zurich adb6b664241480d9e580cbbc0a9f4b3dd405a0830fe1cb413599cd0f58492007 NOT MATCHED
zygote a9c299cc65c3ef64bd2870ba7b5b7ebde968894b23aa476f54a2ac19e0014820 NOT MATCHED

Resulting Key: ['median']
D:\Desktop\try outs>
```

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](https://github.com/sumeethaldipur/CSS_LAB)