Lab 4

Crypto Lab - Secret-Key Encryption

CPSC 353-01
Introduction to Computer Security
Professor Kenytt Avery

By Danh Pham

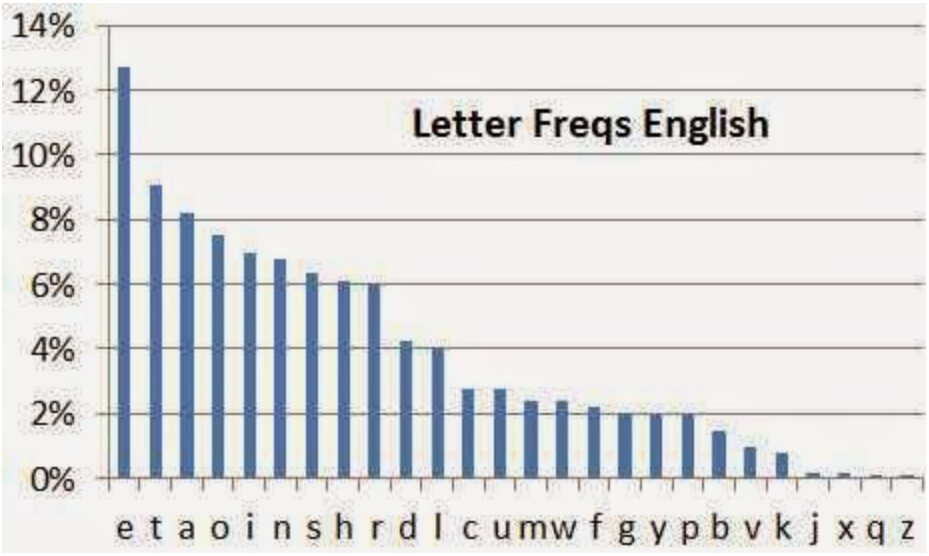
Lab Tasks

Task 1: Frequency Analysis Against Monoalphabetic Substitution Cipher.

First we have to find the frequency of very single letter that appears on the ciphertext.txt.

```
Terminal
[06/25/18]seed@VM:~/.../SOURCE CODE$ python task1.py

 HERE IS THE FREQUENCY TABLES THAT READING FROM CIPHER TEXT

defaultdict(<type 'int'>, {' ': 741, 'a': 116, 'c': 104, 'b': 83, 'e': 76, 'd':
59, 'g': 83, 'f': 49, 'i': 166, 'h': 235, 'k': 5, 'j': 5, 'm': 264, 'l': 90, 'o'
: 4, 'n': 488, 'q': 276, 'p': 156, 's': 19, 'r': 82, 'u': 280, 't': 183, 'w': 1,
 'v': 348, 'y': 373, 'x': 291, 'z': 95})
```

After that we sort the frequency letters by decreasing (expect for \n) after that we found the list of letters:



N y v x -> so on by decreasing frequency

After we mapping the letter with the cipher and checking for every single one by using “string maketrans python”: (note it did it in the long way to check one word, three words, four words)

For example first case will be N -> e because in the ciphertext.txt there is only one N letter in the sentence. Then we can see clearly that ‘v’ in cipher.txt is only by

guessing i assume that ‘v -> a’ . First three letters in ciphertext.txt is ytn so replace with n will be ‘yte’ the english word with e at the end can only be ‘the’ so replace t -> h and y -> t. Only looking for three letters word to finding the rest.

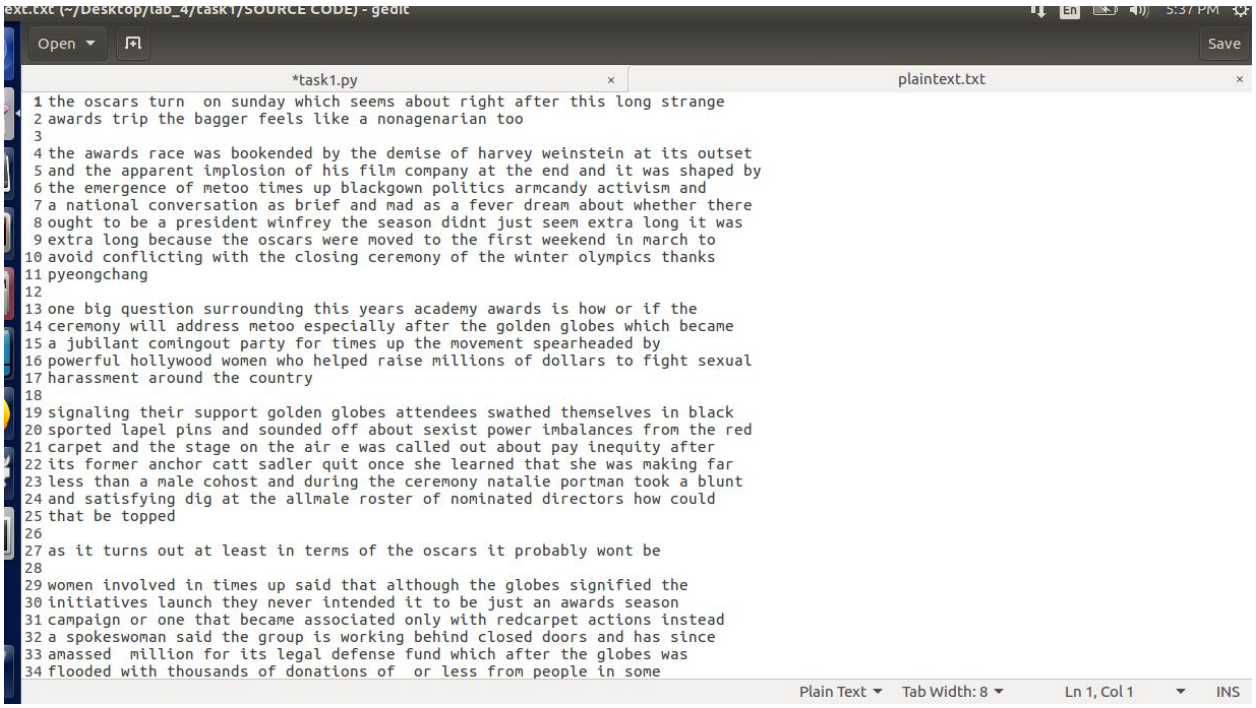
After all that replace i found the solution:

alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
key : c f m y p v b r l q x w i e j d s g k h n a z o t u

Making a python program that can output the plain text with the key and alphabet by using makestran syntax,

```
import collections
import string
from string import maketrans


# mapping function for subtitustion in the cipher text a-> c and b -> f so on

alphabet = "abcdefghijklmnopqrstuvwxyz"

key = "cfmypvbrlqxwiejdsgkhnazotu"

transformer= string.maketrans(alphabet,key)


def frequency(text) :
        freq = collections.defaultdict(int)
        for line in text:

            line = line.rstrip('\n')

            for char in line:

                freq[char] +=1

        return freq


def printplaintext(text):

        plaintext =text.translate(transformer)

        return plaintext

def writeplaintext(text):
```

Result of plaintext.txt

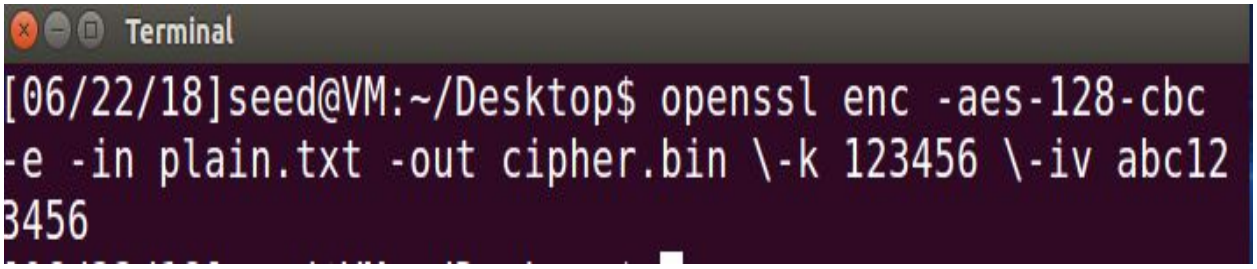


Task 2: Encryption using Different Ciphers and Modes

I am using the same plaintext.txt that i got from first task

Encryption:

-aes-128-cbc



-aes-128-cfb

-bf-cbc

```
[06/22/18]seed@VM:~/Desktop$ openssl enc -bf-cbc -e -i
n plain.txt -out cipher.bin \-k 123456 \-iv abc123456
```

Note: all encrypted files already list on the folder Name task 2.

Task 3: Encryption Mode – ECB vs. CBC

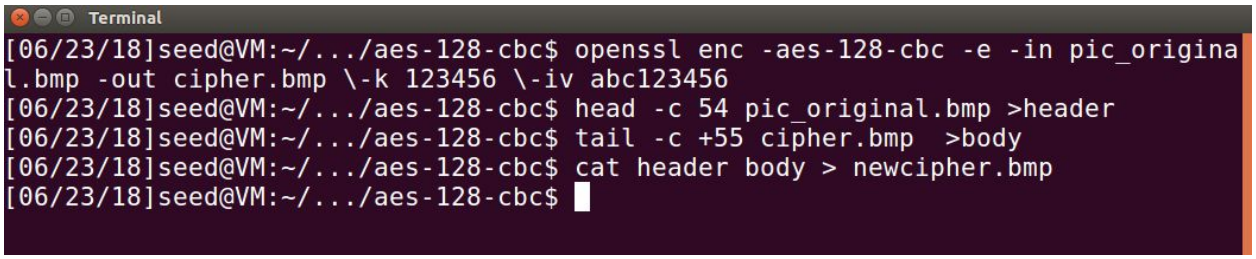
- List all possible cipher mode that can be used to encrypt or decrypt by using command “MAN ENC”

```
       aes-[128|192|256]-cbc  128/192/256 bit AES in
CBC mode
       aes-[128|192|256]      Alias for aes-[128|192|
256]-cbc
       aes-[128|192|256]-cfb  128/192/256 bit AES in
128 bit CFB mode
       aes-[128|192|256]-cfb1 128/192/256 bit AES in
1 bit CFB mode
       aes-[128|192|256]-cfb8 128/192/256 bit AES in
8 bit CFB mode
       aes-[128|192|256]-ecb  128/192/256 bit AES in
ECB mode
       aes-[128|192|256]-ofb  128/192/256 bit AES in
```

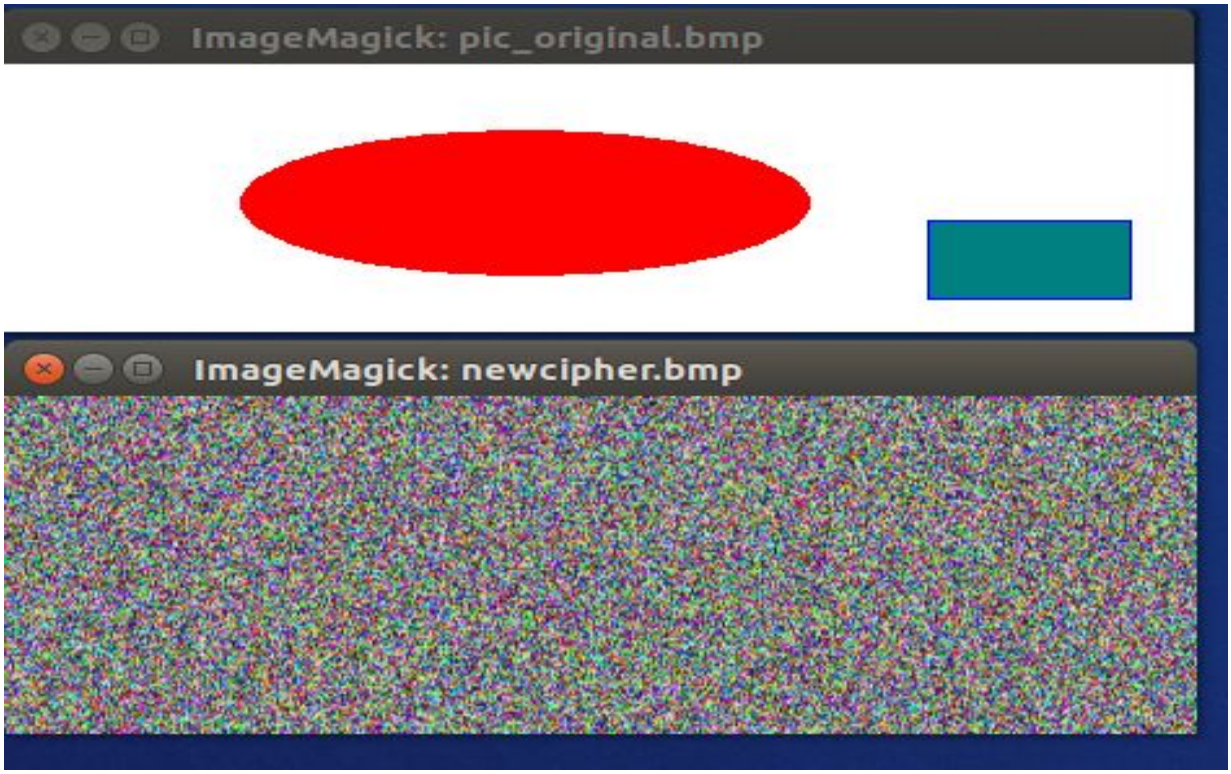
- Encryption the original pic given from the the description:

Mode: aes-128-cbc

Command : by using cat header and body we get header from original pic and combine with the data from encrypted pic so we can view the pic context.



Result : by seeing that we can see the original pic that encrypted to new cipher.bmp the picture turns to be a mess.

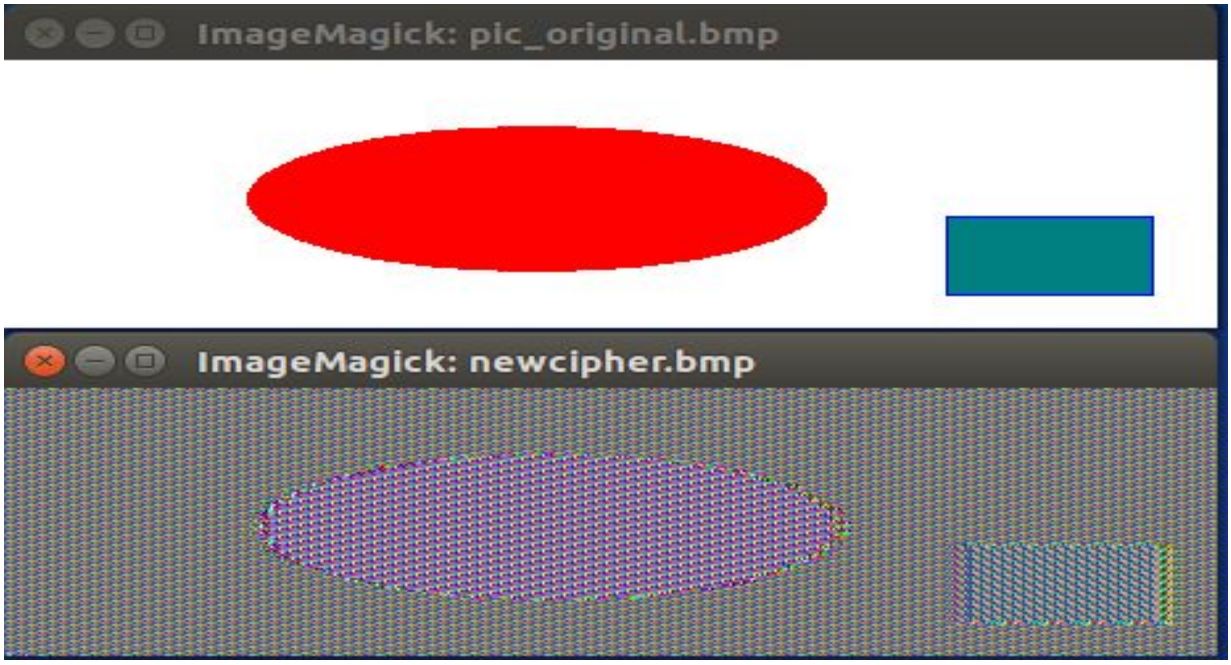


Mode: aes-128-ecb

command :

```
[06/23/18]seed@VM:~/.../encrypted original pic$ head -c 54 pic_original.bmp >hea
der
[06/23/18]seed@VM:~/.../encrypted original pic$ tail -c +55 cipher.bmp  >body
[06/23/18]seed@VM:~/.../encrypted original pic$ cat header body > newcipher.bmp
[06/23/18]seed@VM:~/.../encrypted original pic$
```

Result: as we can see clearly that in ecb mode the pic generated not a mess anymore but it has some change about the color in the new pic.



- Doing the same step with a new bmp pic.

Mode: aes-128-cbc

```
Terminal
[06/23/18]seed@VM:~/.../aes-128-cbc$ openssl enc -aes-128-cbc -e -in index.bmp -
out cipher.bmp \-k 123456789aabbccd \-iv 0000000000000000
[06/23/18]seed@VM:~/.../aes-128-cbc$ head -c 54 index.bmp >header
[06/23/18]seed@VM:~/.../aes-128-cbc$ tail -c +55 cipher.bmp  >body
[06/23/18]seed@VM:~/.../aes-128-cbc$ cat header body > newcipher.bmp
[06/23/18]seed@VM:~/.../aes-128-cbc$
```

result :

Mode: aes-128-ecb

```
Terminal
[06/23/18]seed@VM:~/.../aes-128-ecb$ openssl enc -aes-128-ecb -e -in index.bmp -
out cipher.bmp \-k 123456789aabbccd \-iv 0000000000000000
warning: iv not use by this cipher
[06/23/18]seed@VM:~/.../aes-128-ecb$ head -c 54 index.bmp >header
[06/23/18]seed@VM:~/.../aes-128-ecb$ tail -c +55 cipher.bmp  >body
[06/23/18]seed@VM:~/.../aes-128-ecb$ cat header body > .bmp
[06/23/18]seed@VM:~/.../aes-128-ecb$ cat header body > newcipher.bmp
[06/23/18]seed@VM:~/.../aes-128-ecb$
```

result : when turn into a new bmp pic the second ecb mode still turn into a mess again
I think this pixel of the picture is not good enough for seeing the clearly picture.

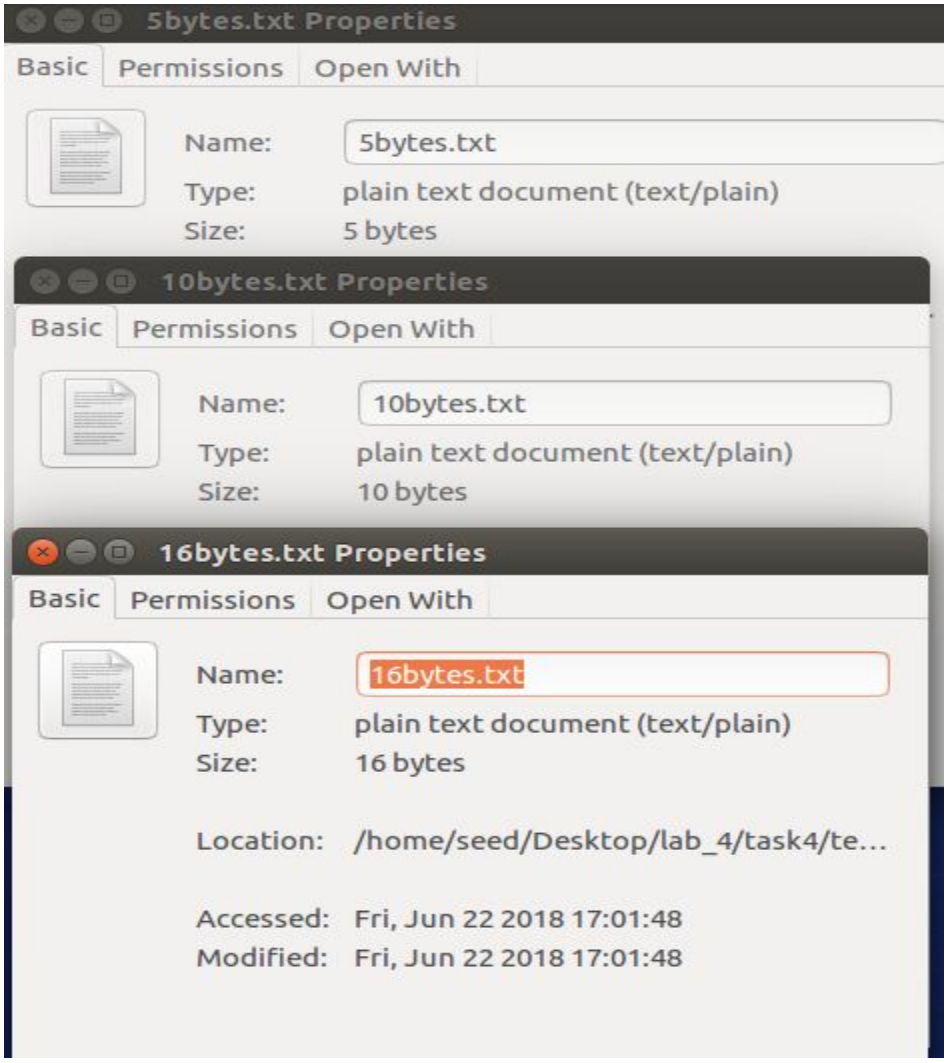
Task 4: Padding

- Creating three texts which difference sizes 5, 10 , 16 bytes.

Command:

```
Terminal
[06/22/18]seed@VM:~/Desktop$ echo -n "12345" > 5bytes.txt
[06/22/18]seed@VM:~/Desktop$ echo -n "1234567891" > 10bytes.txt
[06/22/18]seed@VM:~/Desktop$ echo -n "1234567891012345" > 16bytes.txt
[06/22/18]seed@VM:~/Desktop$
```

Size of texts:



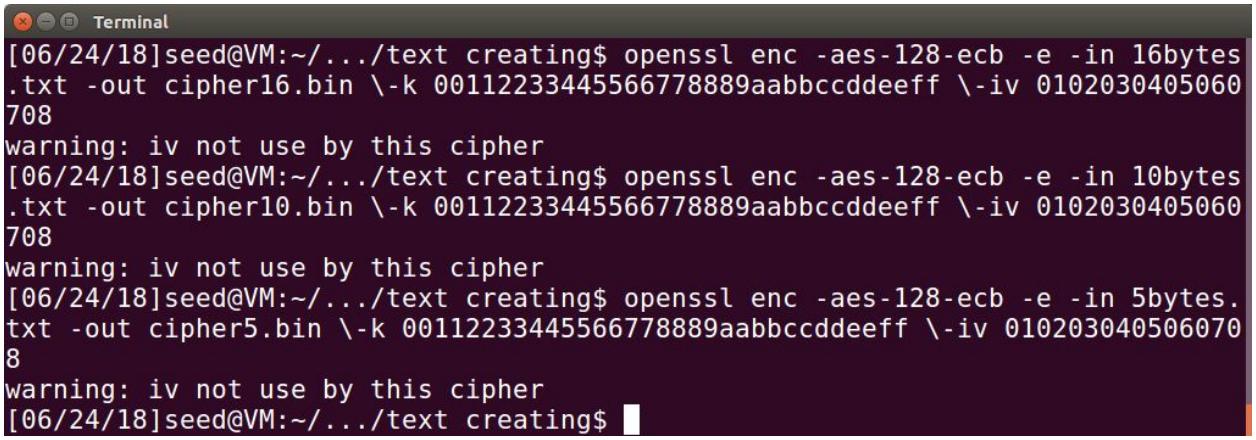
Hexdump commad:

```
Terminal
[06/24/18]seed@VM:~/.../text creating$ hexdump -C 5bytes.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[06/24/18]seed@VM:~/.../text creating$ xxd 5bytes.txt
00000000: 3132 3334 35                             12345
[06/24/18]seed@VM:~/.../text creating$ hexdump -C 10bytes.txt
00000000  31 32 33 34 35 36 37 38  39 31                   |1234567891|
0000000a
[06/24/18]seed@VM:~/.../text creating$ xxd 10bytes.txt
00000000: 3132 3334 3536 3738 3931                 1234567891
[06/24/18]seed@VM:~/.../text creating$ hexdump -C 16bytes.txt
00000000  31 32 33 34 35 36 37 38  39 31 30 31 32 33 34 35  |1234567891012345|
00000010
[06/24/18]seed@VM:~/.../text creating$ xxd 16bytes.txt
00000000: 3132 3334 3536 3738 3931 3031 3233 3435  1234567891012345
[06/24/18]seed@VM:~/.../text creating$
```

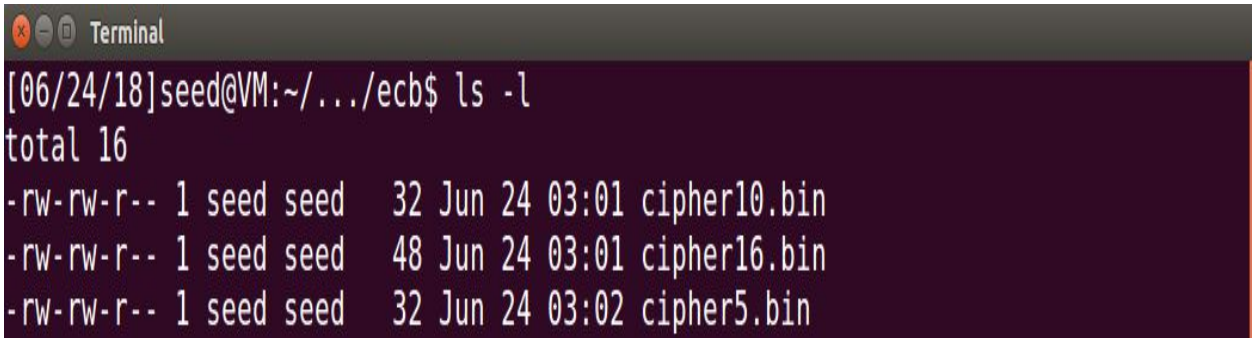
- Use ECB, CBC, CFB, and OFB modes to encrypt those texts:

Mode ECB:

Command:



Size after encrypted:



Hexdump:

```
Terminal
[06/24/18]seed@VM:~/.../ecb$ hexdump -C cipher5.bin
00000000  53 61 6c 74 65 64 5f 5f  2b 8a 5a 53 c5 ba 3b 85  |Salted__+.ZS..;.|
00000010  cb 12 d9 cc 1c e1 cf a5  66 2b c8 2b 12 c3 a9 df  |........f+.+....|
00000020
[06/24/18]seed@VM:~/.../ecb$ xxd cipher5.bin
00000000: 5361 6c74 6564 5f5f 2b8a 5a53 c5ba 3b85  Salted__+.ZS..;.
00000010: cb12 d9cc 1ce1 cfa5 662b c82b 12c3 a9df  ........f+.+....
[06/24/18]seed@VM:~/.../ecb$ hexdump -C cipher10.bin
00000000  53 61 6c 74 65 64 5f 5f  e4 d7 00 fa 03 7f 86 9b  |Salted__........|
00000010  ac 15 43 4d 00 a6 e1 31  9d 32 48 a8 ee a5 09 f2  |..CM...1.2H.....|
00000020
[06/24/18]seed@VM:~/.../ecb$ xxd cipher10.bin
00000000: 5361 6c74 6564 5f5f e4d7 00fa 037f 869b  Salted__........
00000010: ac15 434d 00a6 e131 9d32 48a8 eea5 09f2  ..CM...1.2H.....
[06/24/18]seed@VM:~/.../ecb$ xxd cipher16.bin
00000000: 5361 6c74 6564 5f5f f864 d047 8f5c 34df  Salted__.d.G.\4.
00000010: 805a 1734 4f1e c831 160f 0831 c056 57da  .Z.4O..1...1.VW.
00000020: 04c0 c82c 110e c9b7 8886 729d dcf5 c14e  ...,......r....N
[06/24/18]seed@VM:~/.../ecb$ hexdump -C cipher16.bin
00000000  53 61 6c 74 65 64 5f 5f  f8 64 d0 47 8f 5c 34 df  |Salted__.d.G.\4.|
00000010  80 5a 17 34 4f 1e c8 31  16 0f 08 31 c0 56 57 da  |.Z.4O..1...1.VW.|
00000020  04 c0 c8 2c 11 0e c9 b7  88 86 72 9d dc f5 c1 4e  |...,......r....N|
00000030
[06/24/18]seed@VM:~/.../ecb$
```

Mode CBC:

- *Command:*

```
06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cbc -e -in 5bytes.
xt -out cipher5.bin \-k 00112233445566778889aabbccddeeff \-iv 010203040506070

06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cbc -e -in 10bytes
txt -out cipher10.bin \-k 00112233445566778889aabbccddeeff \-iv 0102030405060
08
06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cbc -e -in 16bytes
txt -out cipher16.bin \-k 00112233445566778889aabbccddeeff \-iv 0102030405060
08
```

- 06/24/18]seed@VM:~/.../text creating$
- *Size :*

```
Terminal
[06/24/18]seed@VM:~/.../cbc$ ls -l
total 16
-rw-rw-r-- 1 seed seed   32 Jun 24 02:56 cipher10.bin
-rw-rw-r-- 1 seed seed   48 Jun 24 02:56 cipher16.bin
-rw-rw-r-- 1 seed seed   32 Jun 24 02:54 cipher5.bin
```

-
- *hexdump :*

```
Terminal
[06/24/18]seed@VM:~/.../cbc$ hexdump -C cipher5.bin
00000000  53 61 6c 74 65 64 5f 5f  7a f9 63 c0 fb 64 34 43  |Salted__z.c..d4C|
00000010  ab a8 c4 3b ff 97 a4 90  87 02 c9 ee 33 18 fe 8d  |...;........3...|
00000020
[06/24/18]seed@VM:~/.../cbc$ xxd cipher5.bin
00000000: 5361 6c74 6564 5f5f 7af9 63c0 fb64 3443  Salted__z.c..d4C
00000010: aba8 c43b ff97 a490 8702 c9ee 3318 fe8d  ...;........3...
[06/24/18]seed@VM:~/.../cbc$ hexdump -C cipher10.bin
00000000  53 61 6c 74 65 64 5f 5f  70 39 04 59 5c 91 7a b0  |Salted__p9.Y\.z.|
00000010  ba 26 a0 bd 1a 26 48 4f  fa 1d ce 5f 71 c3 fb f6  |.&...&HO..._q...|
00000020
[06/24/18]seed@VM:~/.../cbc$ xxd cipher10.bin
00000000: 5361 6c74 6564 5f5f 7039 0459 5c91 7ab0  Salted__p9.Y\.z.
00000010: ba26 a0bd 1a26 484f fa1d ce5f 71c3 fbf6  .&...&HO..._q...
[06/24/18]seed@VM:~/.../cbc$ hexdump -C cipher16.bin
00000000  53 61 6c 74 65 64 5f 5f  43 54 4f 20 e5 90 d8 2f  |Salted__CTO .../|
00000010  a0 c7 1d 76 d5 75 2b 38  ba 62 06 26 88 5b 00 4c  |...v.u+8.b.&.[.L|
00000020  3a d0 1f 22 09 12 39 4d  bf ee d4 85 fa 4d 91 dd  |:.."..9M.....M..|
00000030
[06/24/18]seed@VM:~/.../cbc$ xxd cipher16.bin
00000000: 5361 6c74 6564 5f5f 4354 4f20 e590 d82f  Salted__CTO .../
00000010: a0c7 1d76 d575 2b38 ba62 0626 885b 004c  ...v.u+8.b.&.[.L
00000020: 3ad0 1f22 0912 394d bfee d485 fa4d 91dd  :.."..9M.....M..
[06/24/18]seed@VM:~/.../cbc$
```

Mode CFB:

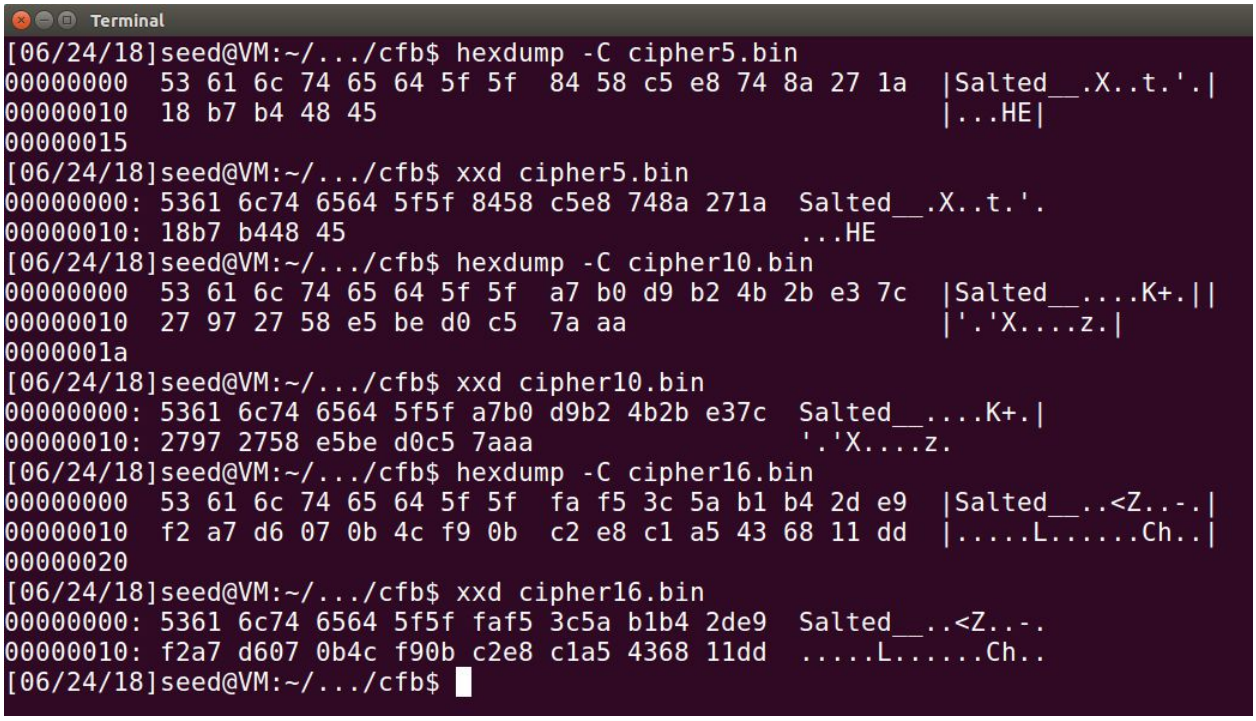
- *Command:*

```
Terminal
[06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cfb -e -in 5bytes.
txt -out cipher5.bin \-k 00112233445566778889aabbccddeeff \-iv 010203040506070
8
[06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cfb -e -in 10bytes
.txt -out cipher10.bin \-k 00112233445566778889aabbccddeeff \-iv 0102030405060
708
[06/24/18]seed@VM:~/.../text creating$ openssl enc -aes-128-cfb -e -in 16bytes
.txt -out cipher16.bin \-k 00112233445566778889aabbccddeeff \-iv 0102030405060
708
```

- *Size :*

```
Terminal
[06/24/18]seed@VM:~/.../cfb$ ls -l
total 16
-rw-rw-r-- 1 seed seed   26 Jun 24 02:59 cipher10.bin
-rw-rw-r-- 1 seed seed   32 Jun 24 03:00 cipher16.bin
-rw-rw-r-- 1 seed seed   21 Jun 24 02:59 cipher5.bin
```

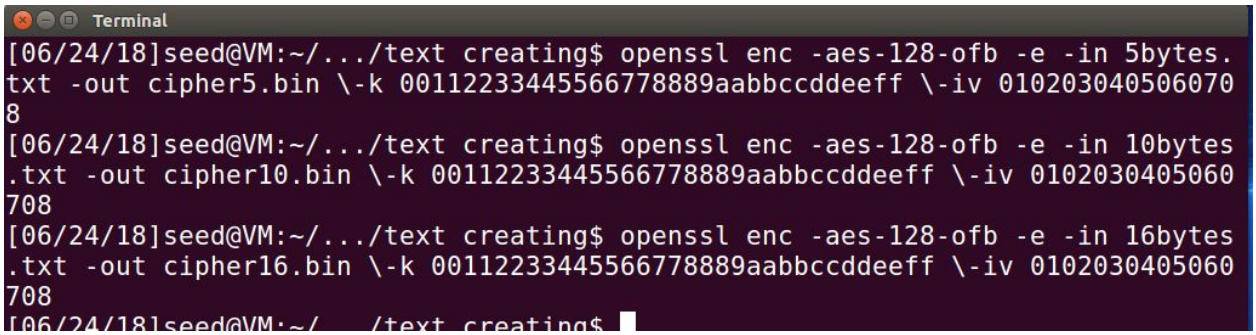
- *hexdump :*



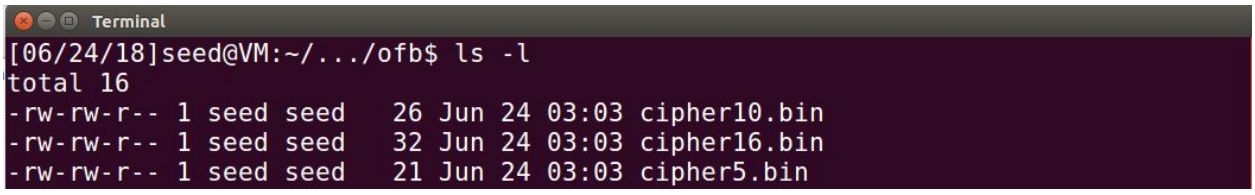
-

Mode OFB:

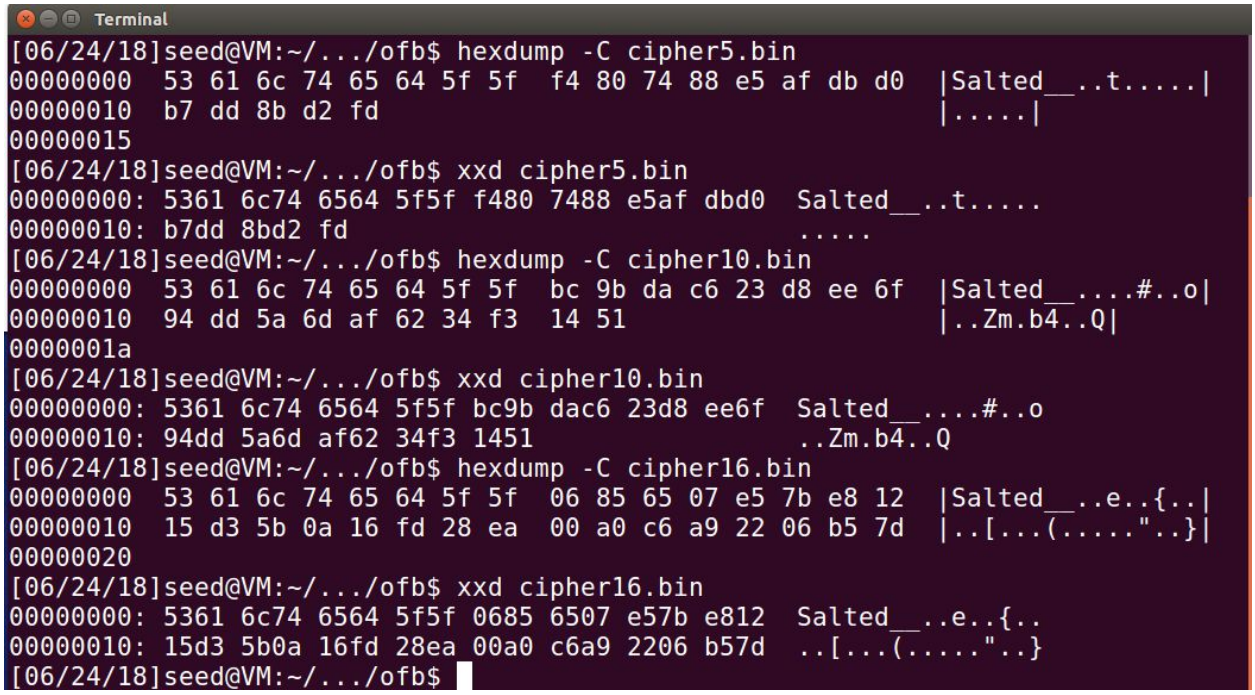
- *Command:*



-
- *Size :*



-
-
-
- *hexdump :*



Conclusion:

- *By default all cipher mode will add 16 bytes iv for encryption but in CBC and ECB will add another padding size for every single one file that encrypted by those mode. Those mode are Blocks cipher so those need to fill every single block with 16 bits*

In CBC mode:

5 bytes -> 32 bytes |32 - 5 = 27 bytes -> 27 - 16 bytes iv = 11 byte padding
10 bytes -> 32 bytes | 6 bytes padding
16 bytes -> 48 bytes | 16 bytes padding

In ECB mode:

5 bytes -> 32 bytes | 11 byte padding
10 bytes -> 32 bytes | 6 bytes padding
16 bytes -> 48 bytes | 16 bytes padding

- In CFB and OFB mode we don't need to add anything padding because those cipher mode are stream cipher:

In CFB mode:

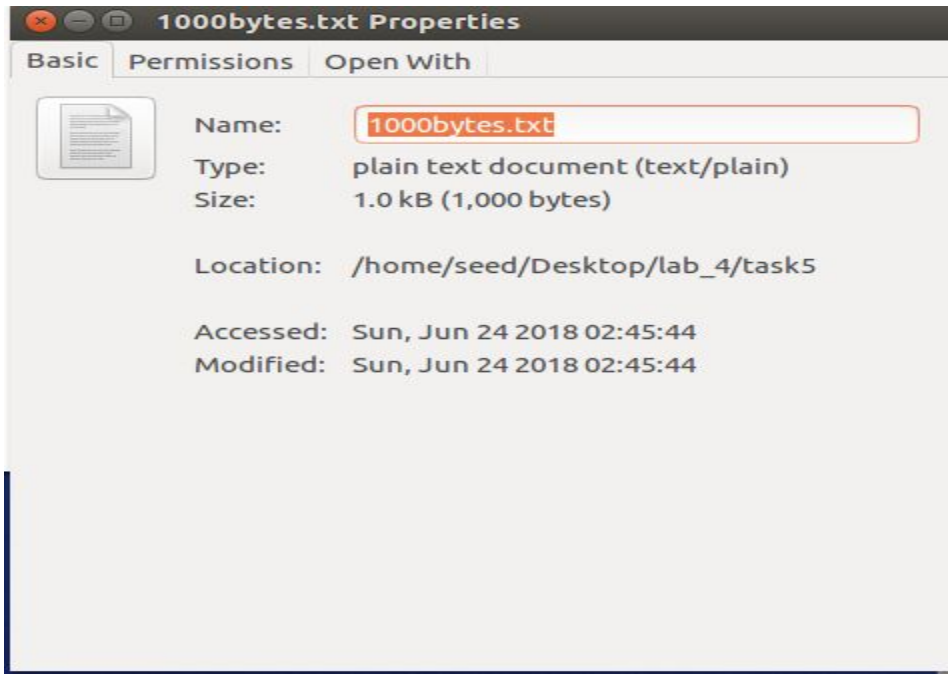
5 bytes -> 21 bytes | 21 - 5 = 16 bytes -> 16 - 16 = 0 bytes padding
10 bytes -> 26 bytes | 0
16 bytes -> 32 bytes | 0

In OFB mode:

5 bytes -> 21 bytes |0
10 bytes -> 26 bytes |0
16 bytes -> 32 bytes |0

Task 5: Error Propagation – Corrupted CipherText

- Creating 1000 bytes plaintext



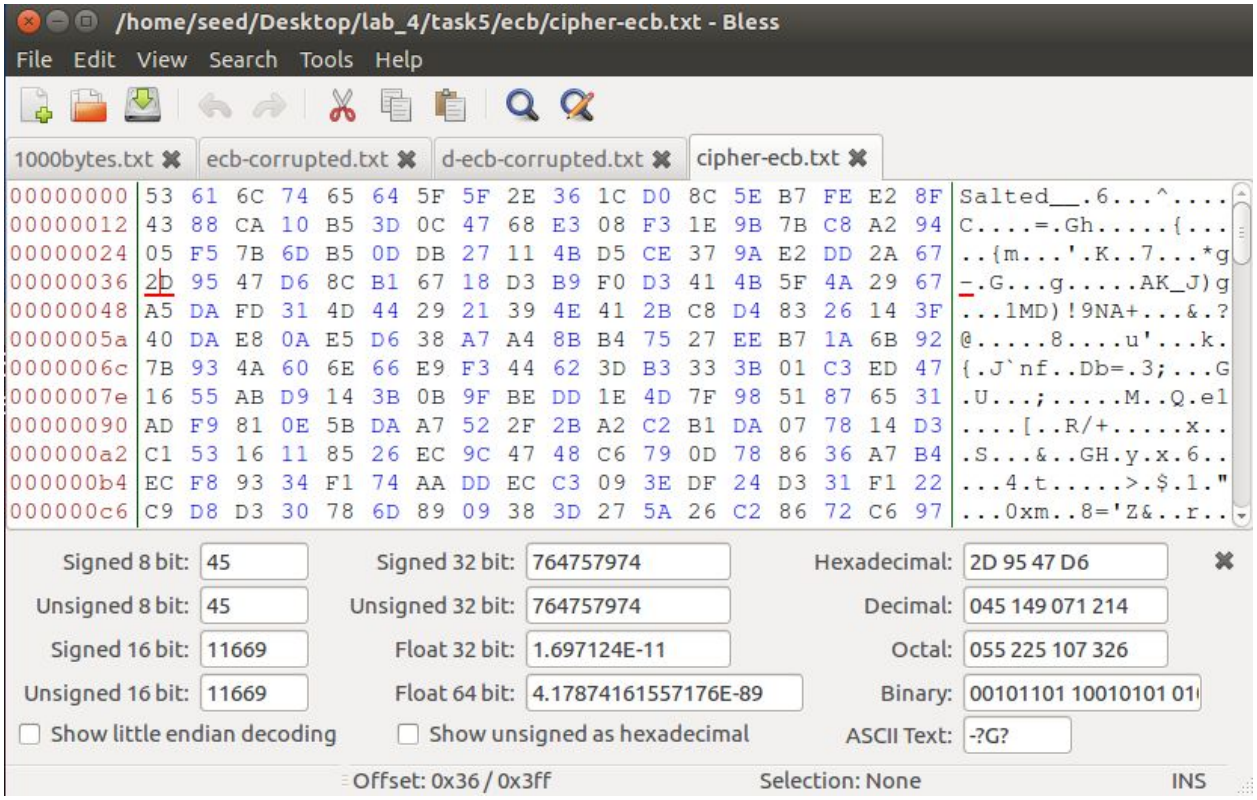
- Encrypted with 4 mode cipher: ECB, CBC, CFB, or OFB, respectively

```
Terminal
[06/24/18]seed@VM:~/.../task5$ openssl enc -aes-128-cbc -e -in 1000bytes.txt -ou
t cipher-cbc.txt \-k 00112233445566778889aabbccddeeff \-iv 0102030405060708
[06/24/18]seed@VM:~/.../task5$ openssl enc -aes-128-cfb -e -in 1000bytes.txt -ou
t cipher-cfb.txt \-k 00112233445566778889aabbccddeeff \-iv 0102030405060708
[06/24/18]seed@VM:~/.../task5$ openssl enc -aes-128-ecb -e -in 1000bytes.txt -ou
t cipher-ecb.txt \-k 00112233445566778889aabbccddeeff \-iv 0102030405060708
warning: iv not use by this cipher
[06/24/18]seed@VM:~/.../task5$ openssl enc -aes-128-ofb -e -in 1000bytes.txt -ou
t cipher-ofb.txt \-k 00112233445566778889aabbccddeeff \-iv 0102030405060708
[06/24/18]seed@VM:~/   /task5$
```

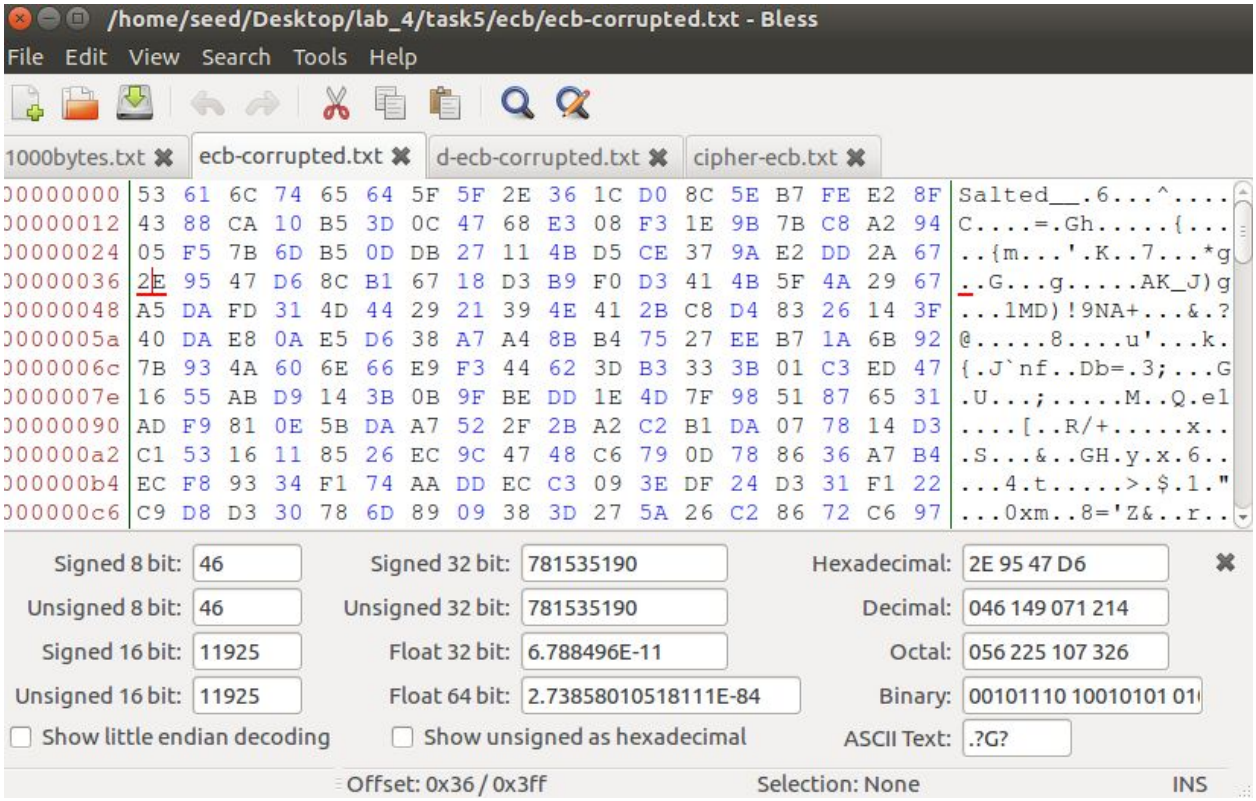
MODE ECB:

HEX command to view cipher block cycles

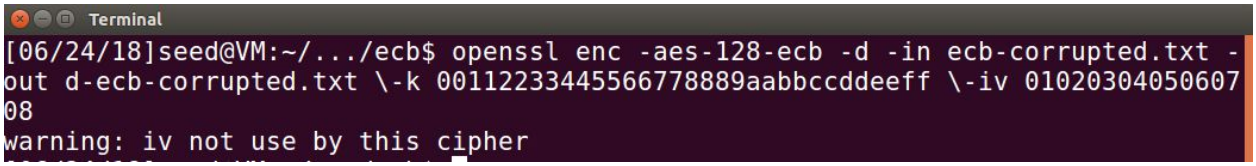
- Original cipher

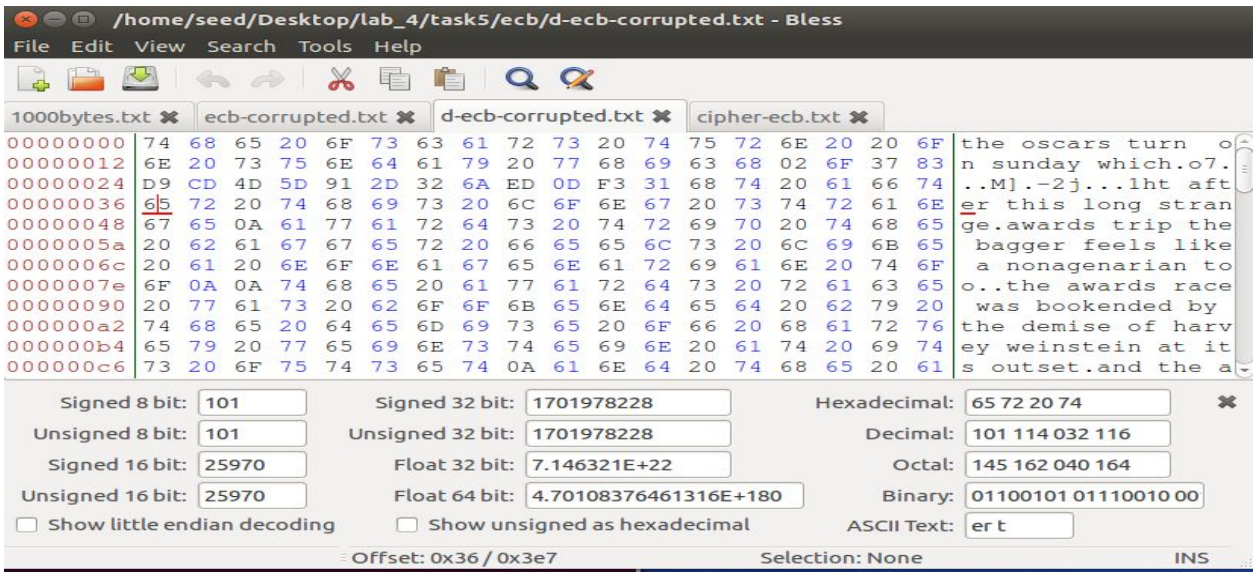


- Corrupted cipher (by changing up one bit at position 55th)

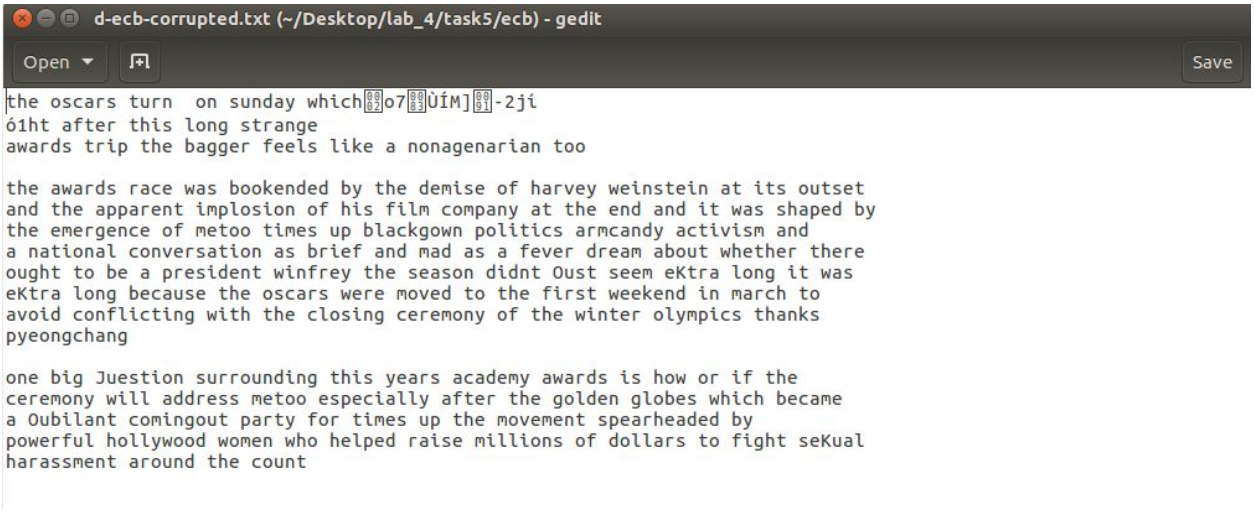


- Decrypted the corrupted ciphertext





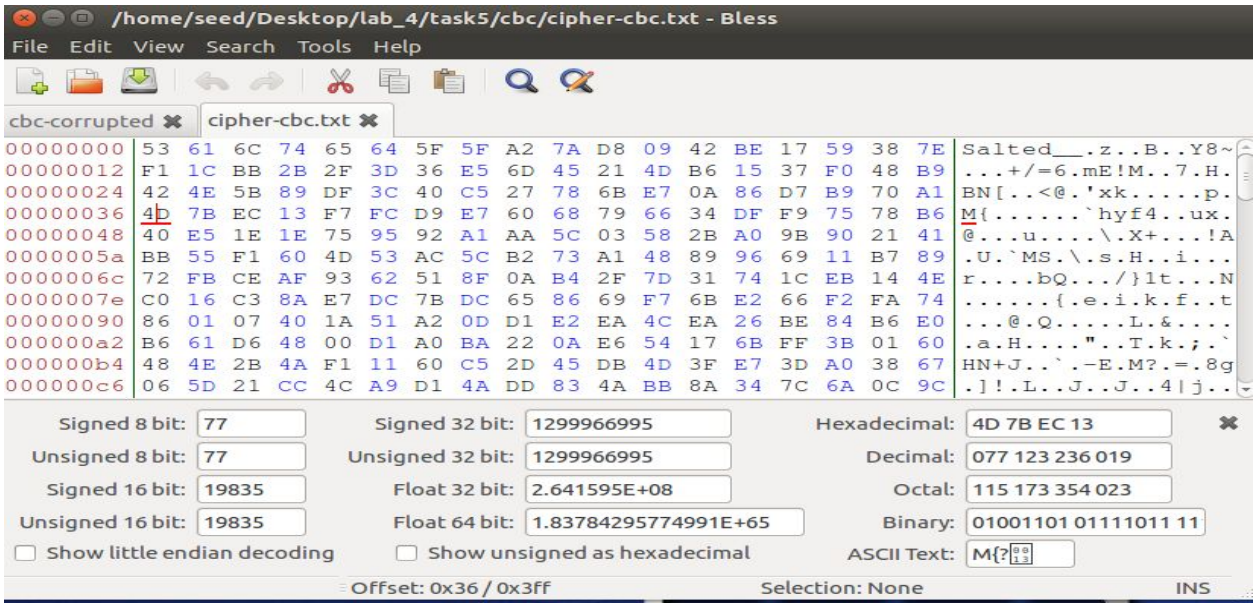
- Result



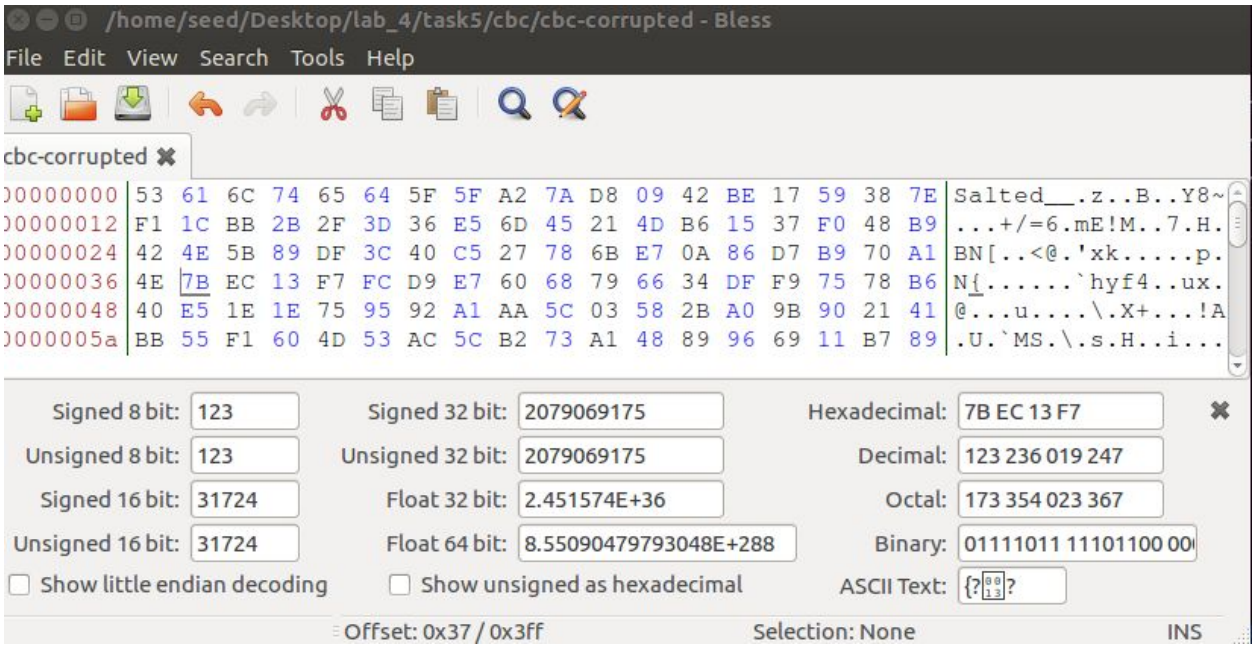
MODE CBC:

HEX command to view cipher block cycles:

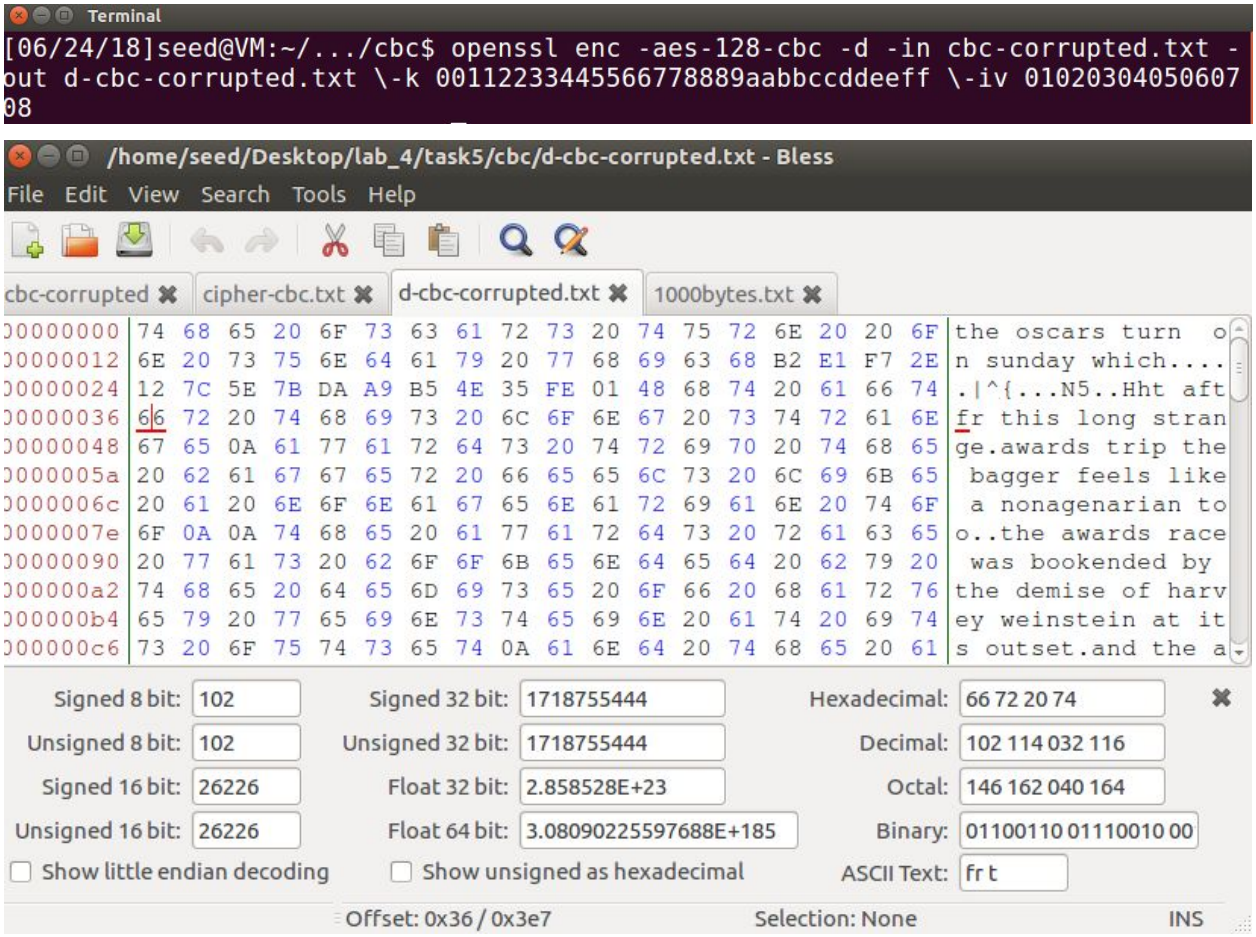
- Original cipher



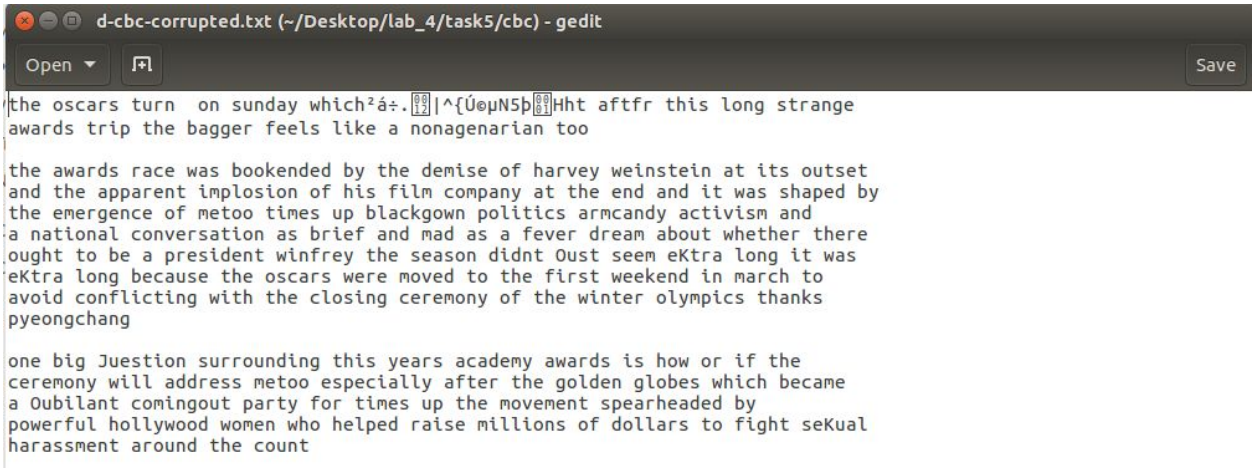
- Corrupted cipher (by changing up one bit at position 55th)



- Decrypted the corrupted ciphertext



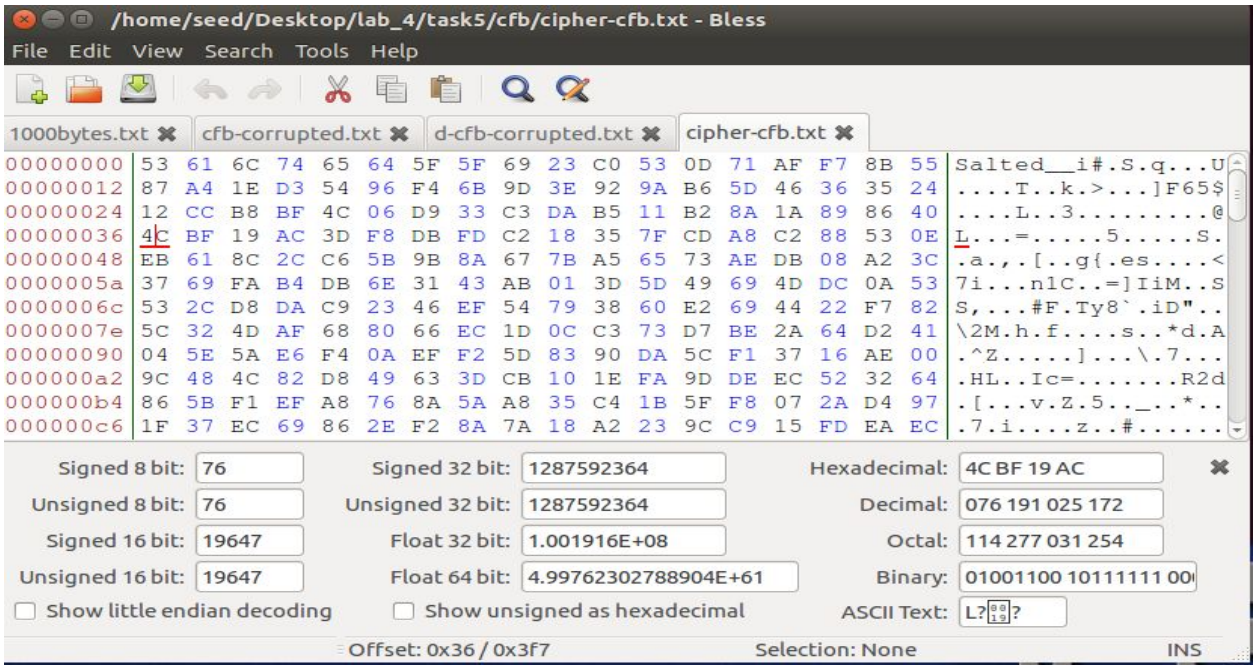
- result



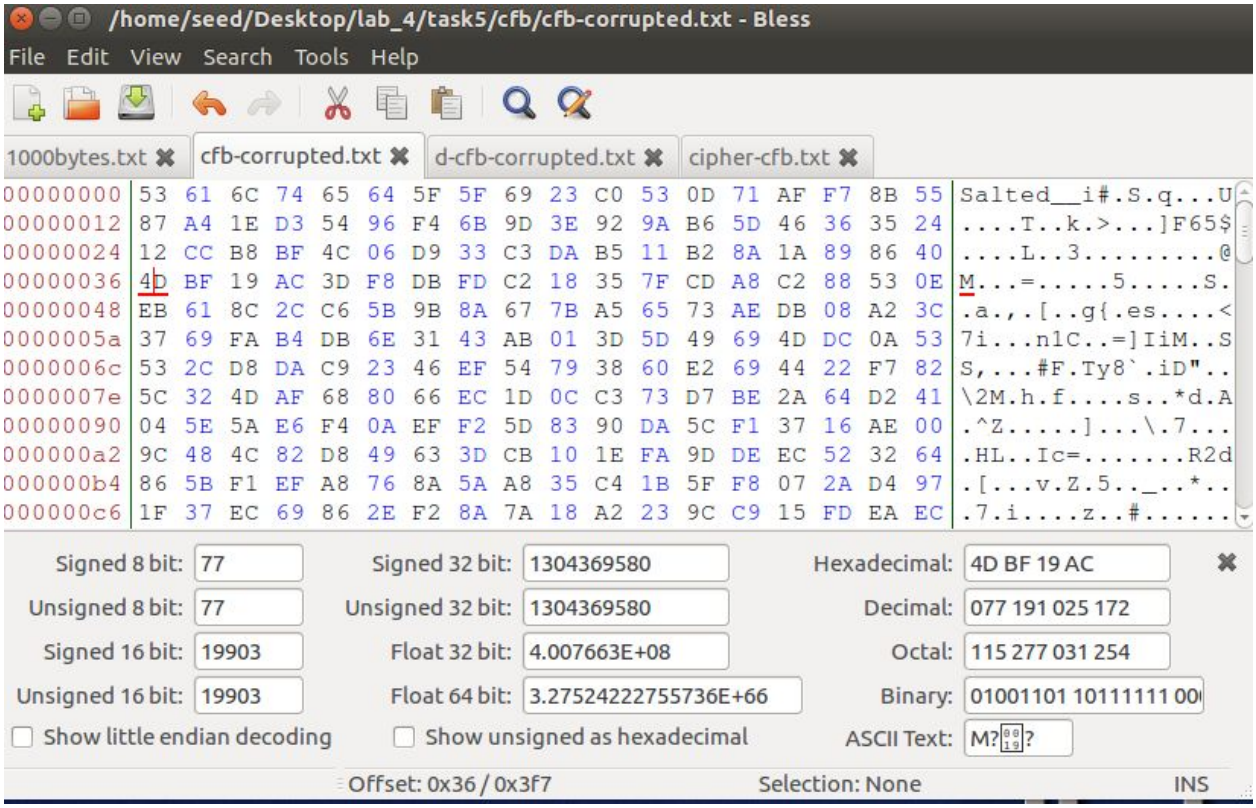
MODE CFB:

HEX command to view cipher block cycles:

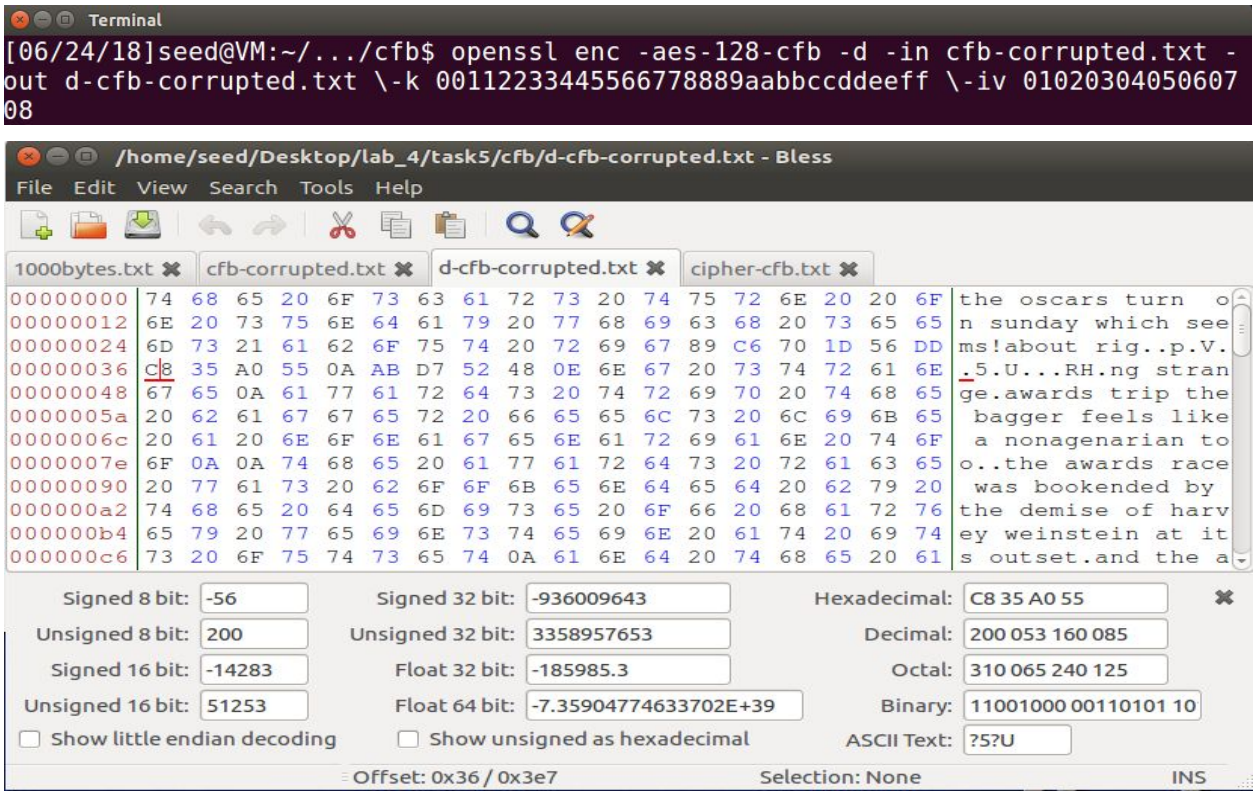
- Original cipher



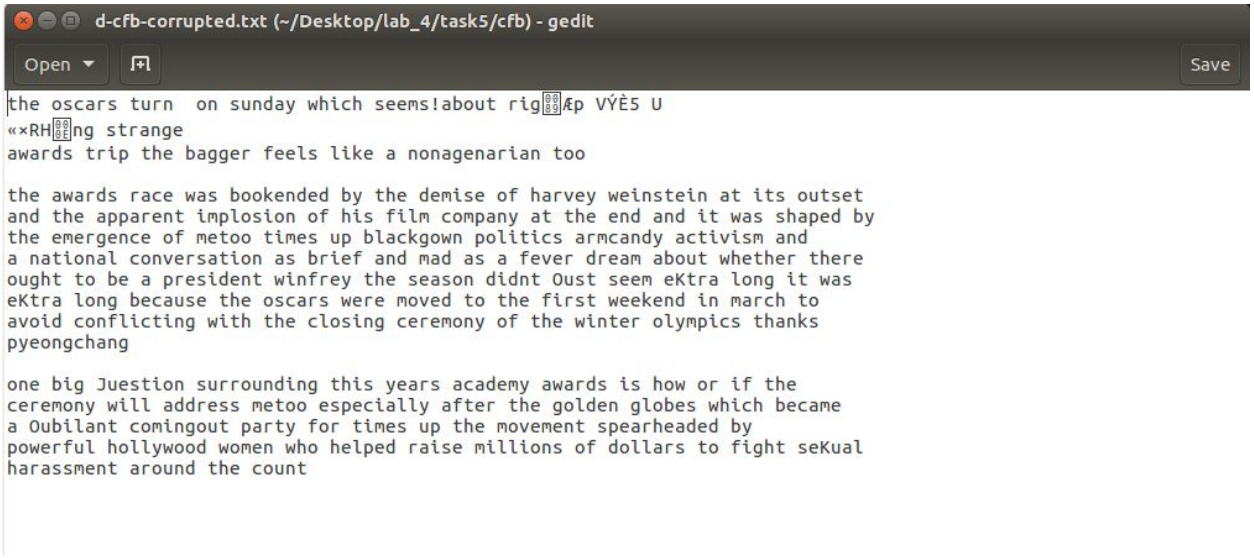
- Corrupted cipher (by changing up one bit at position 55th)



- Decrypted the corrupted ciphertext



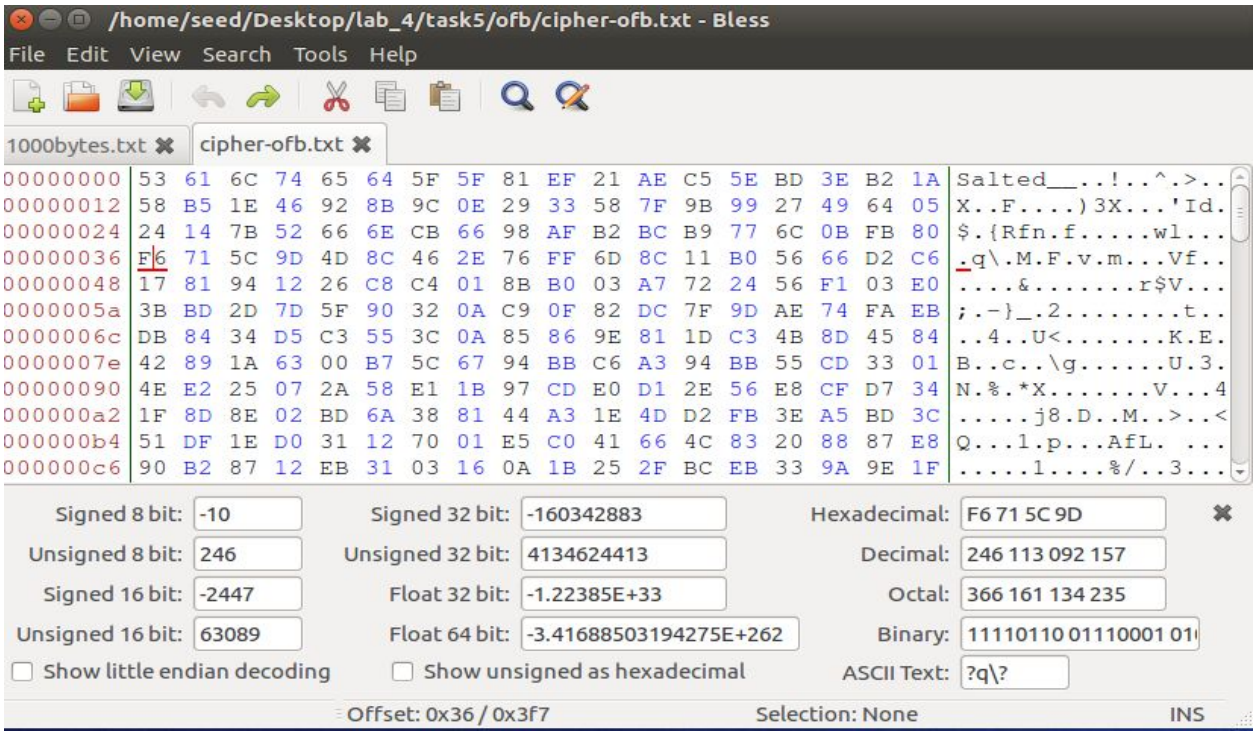
- result



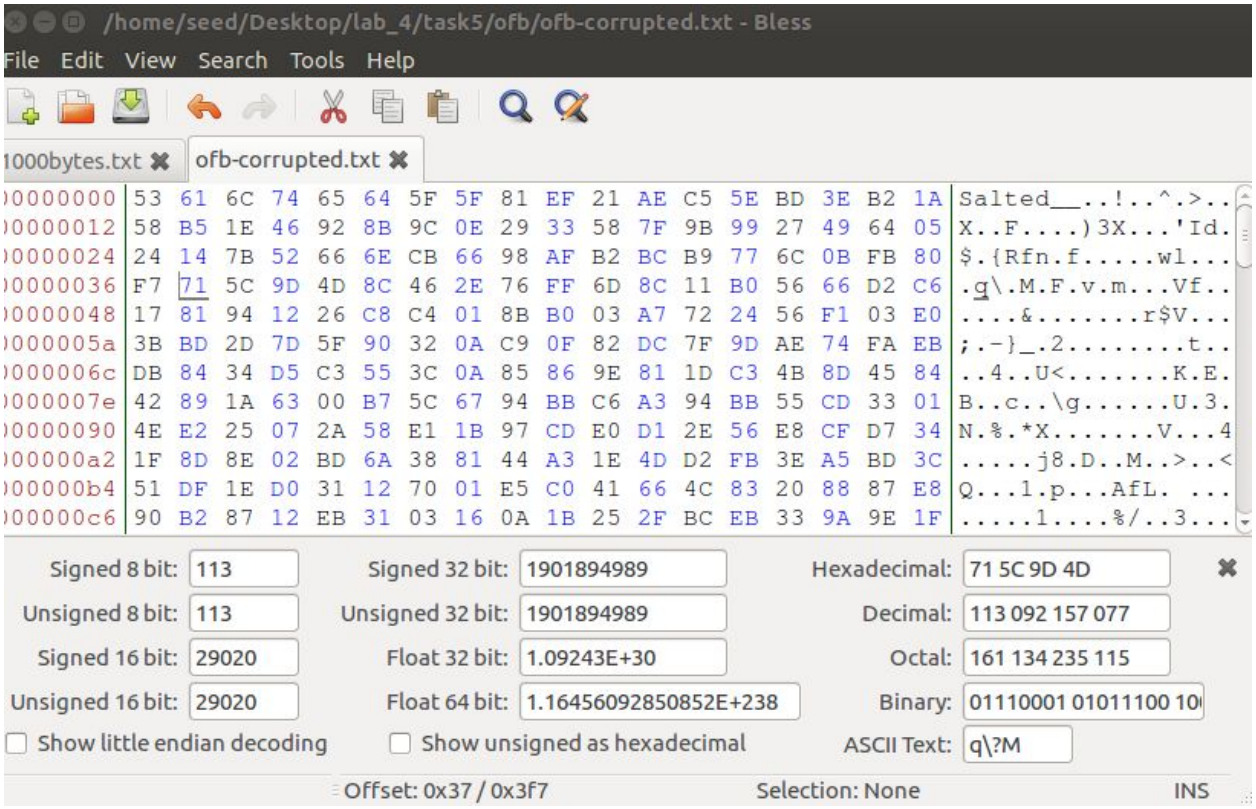
MODE OFB:

HEX command to view cipher block cycles:

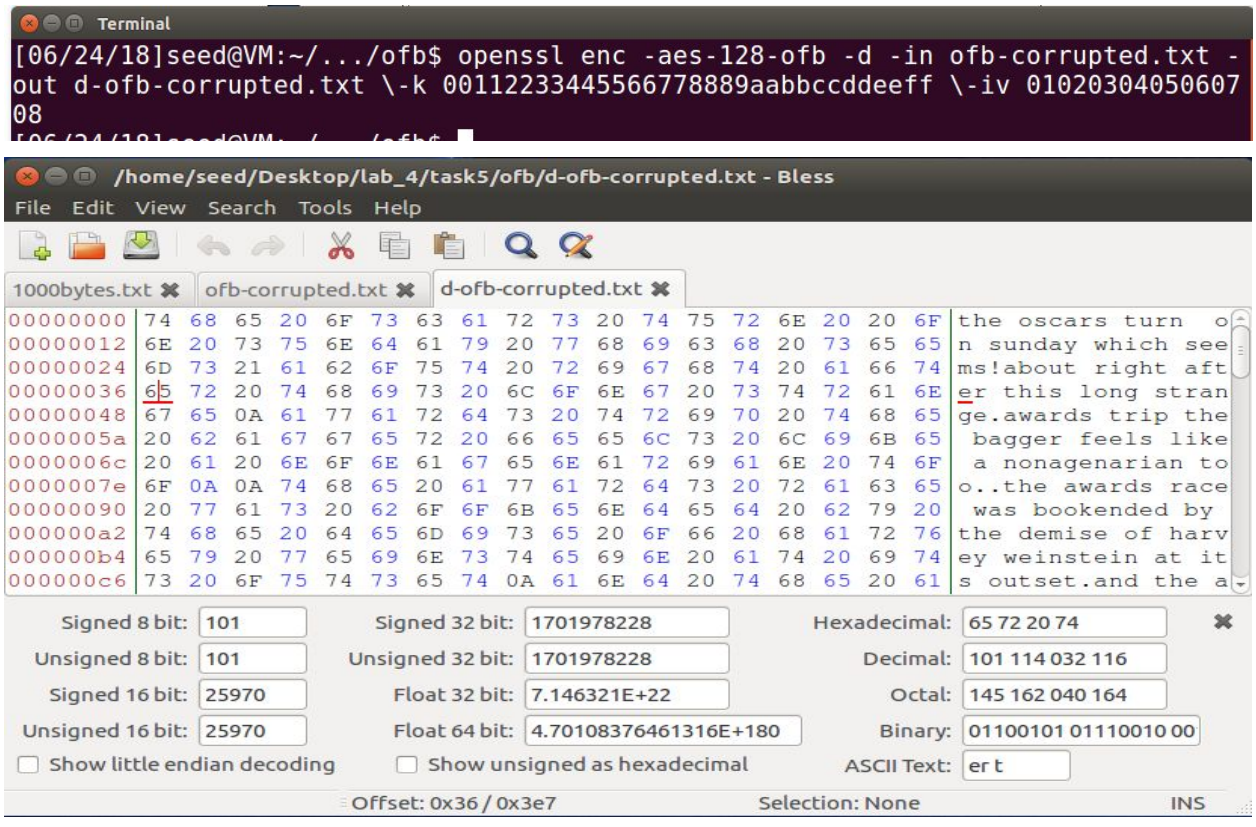
- Original cipher



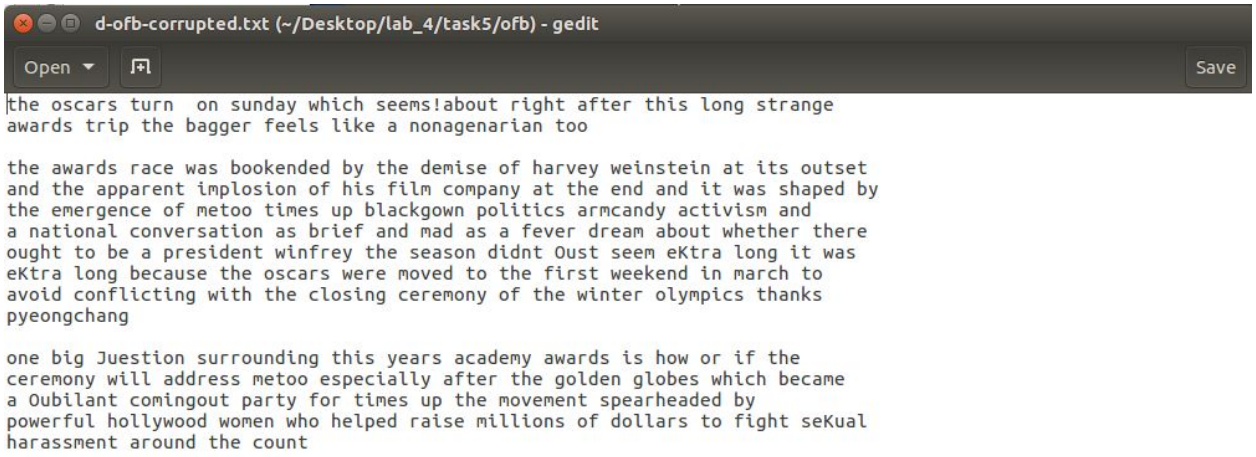
- Corrupted cipher (by changing up one bit at position 55th)



- Decrypted the corrupted ciphertext



- result



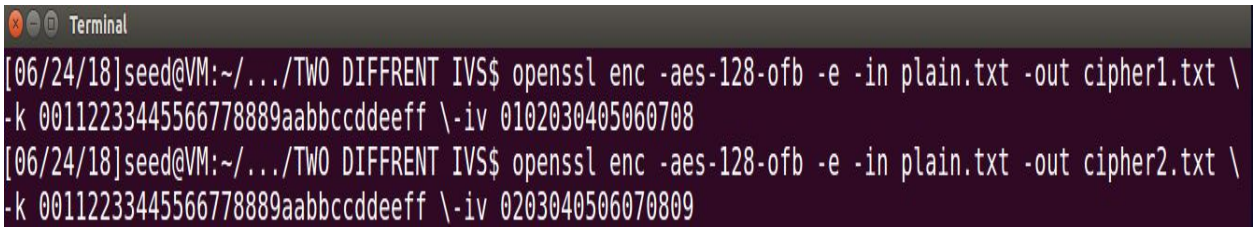
Conclusion: ECB, CBC, CFB, or OFB, mode

As the 4 result we can see clearly that the OFB, ECB, CBC, and CFB mode. We interpreted the 55th bit up to one the decryption will mess up the context of original text in 55th block cycles.

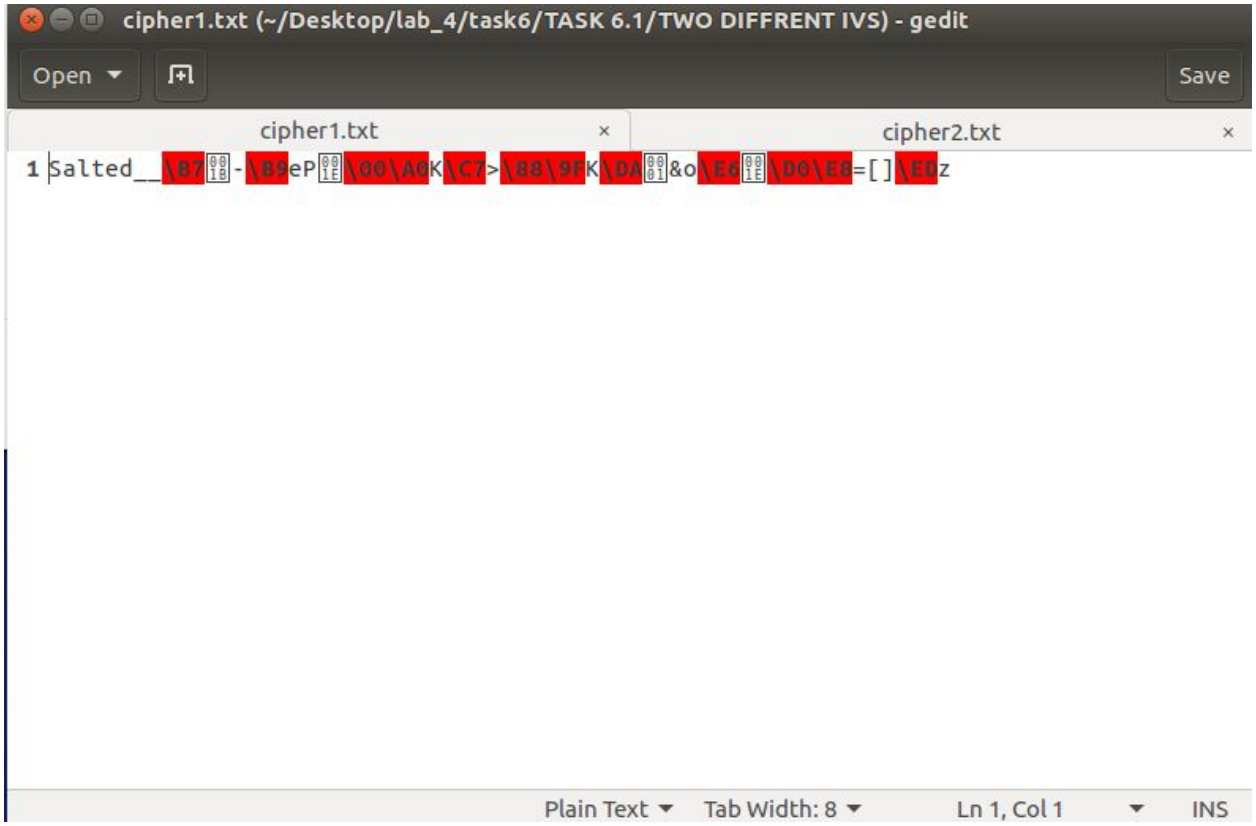
Task 6: Initial Vector (IV)

- Task 6.1. A basic requirement for IV is uniqueness:

a) Two different IVs:



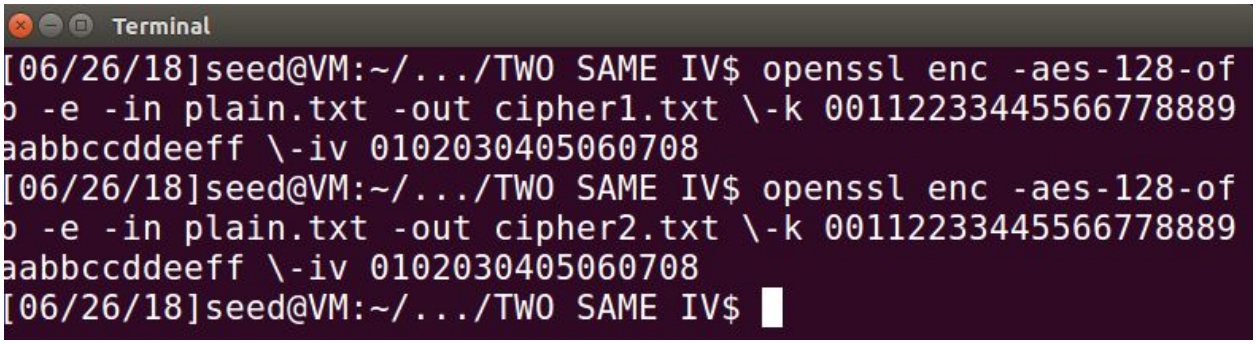
Result from cipher1.txt:



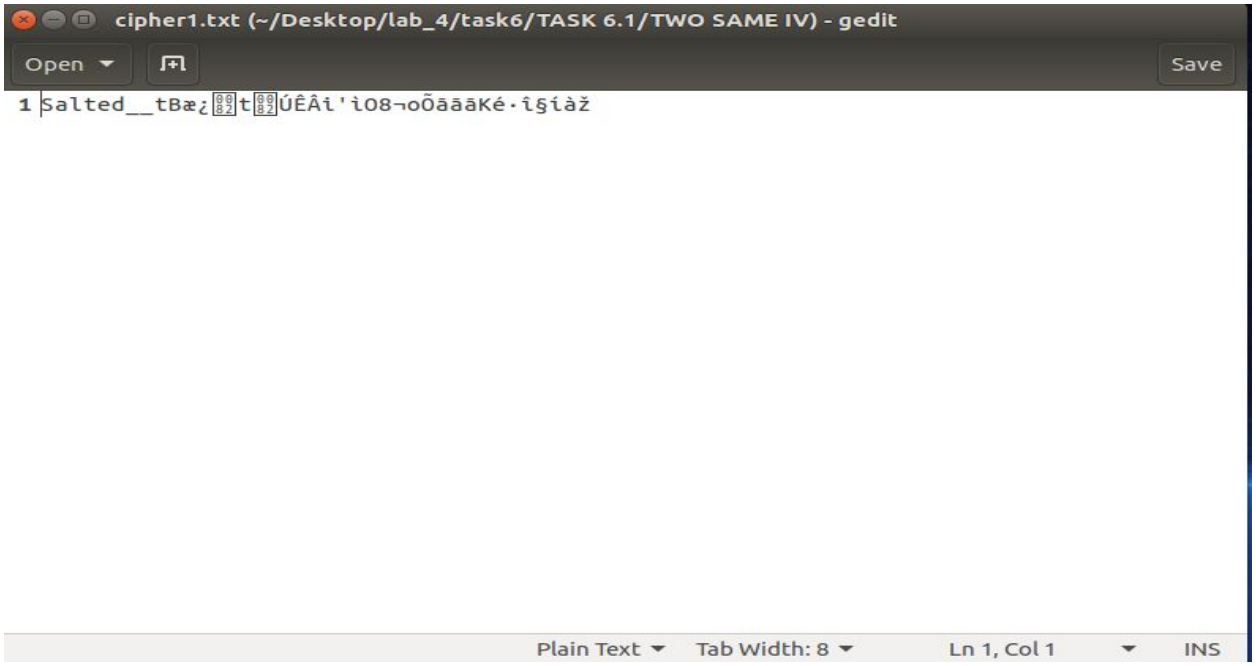
Result from cipher2.txt:

b) same IVs:

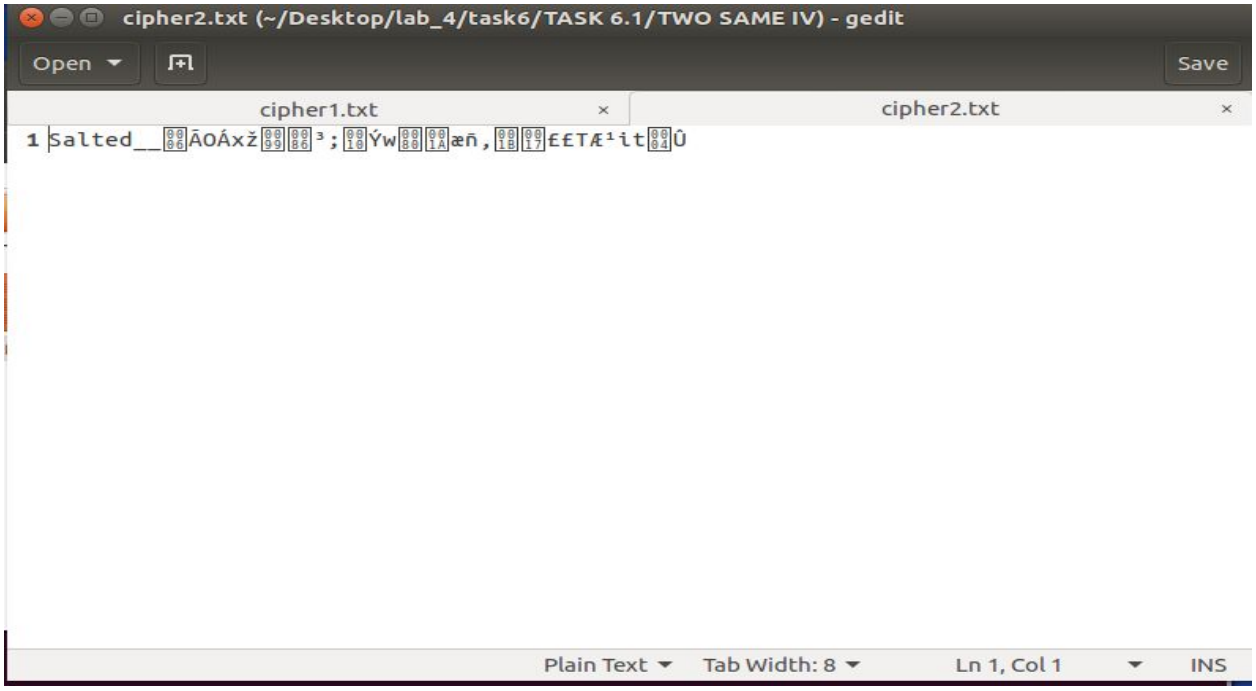
COMMAND:



Result cipher1.txt:



Result cipher2.txt:



Observation: i think the formula that use for OFB mode is **Vi= Ci ⊕ P1** so as long as the IV different the encryption will be changing. On the other hand those on the same IVs the block combine be the same context but different block cipher bits changing.

Task 6.2. One may argue that if the plaintext does not repeat, using the same IV is safe.

Solution: i think he/she decrypt other encrypted messages if the IV is always the same IVs as the formular:

OFB mode => Vi= Ci ⊕ P1

C1= V1 ⊕ P1 then finding the IV => V1 = C1 ⊕ P1
C2 = V1 ⊕ P2 so we have P2 = V1 ⊕ C2

-> P2= C2 ⊕ C1 ⊕ P1 (1)

By following the substitution formula (1):
plain

54 68 69 73 20 69 73 20 61 20 6b 6e 6f 77 6e 20 6d 65 73 73 61 67 65 21

cipher 1

a4 69 b1 c5 02 c1 ca b9 66 96 5e 50 42 54 38 e1 bb 1b 5f 90 37 a4 c1 59

solution p1 xor c1

f0 1 d8 b6 22 a8 b9 99 7 b6 35 3e 2d 23 56 c1 d6 7e 2c e3 56 c3 a4 78

cipher 2

bf 73 bc d3 50 92 99 d5 66 c3 5b 5d 45 03 37 e1 bb 17 5f 90 3f af c1 59

p1 xor c1 xor c2

Message of p2 is: “order: Launch a missile! “

CFB MODE => Ci = Ek(Vi-1) ⊕ Pi

C1 = Ek (Vo) ⊕ P1

V0 = P1 ⊕ C1

As same IV

C2 = EK (V0) ⊕ P2

P2 = C2 ⊕ V0

P2 = P1 ⊕ C1 ⊕ C2

So the message with the same IV will have the same context of OFB

Message of p2 is: “order: Launch a missile! “

TASK 6.3:

By guessing IV assume that Ci = Vi ⊕ Pi

And as the result we can see clearly that the message encryption

```
Encryption method: 128-bit AES with CBC mode.

Key (in hex):    00112233445566778899aabbccddeeff (known only to Bob)
Ciphertext (C1): bef65565572ccee2a9f9553154ed9498 (known to both)
IV used on P1 (known to both)
     (in ascii): 1234567890123456
     (in hex)  : 31323334353637383930313233343536
Next IV (known to both)
     (in ascii): 1234567890123457
     (in hex)  : 31323334353637383930313233343537
```

The iv for p1 and p2 is increasing by one from 36 to 37 so in the case the IV can be predictable.

C1 = IV1 ⊕ P1 ⊕ KEY

2) IV1 = IV(unknown) ⊕ KEY assume that iv unknown first block

Substitution 2 to 1

=> C1 = IV(unknown) ⊕ KEY ⊕ P1 ⊕ KEY

=> C1 = IV(unknown) ⊕ P1

Try to give to Bod the plainttext.txt with content “yes” in the first block then the equation will be

1) C2 = IV2 ⊕ P2

Now we have (USING COMPARISON)

C2 = IV2 ⊕ P2 = IV(unknown) ⊕ P1 = C1

If C2 = C1 then context is “ YES”

If C2 != C1 Then context is “ NO “