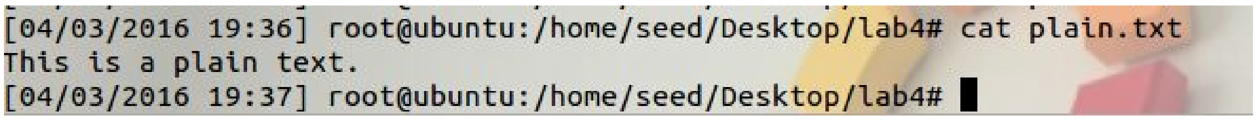
EE609 LAB #4

Hualiang Li

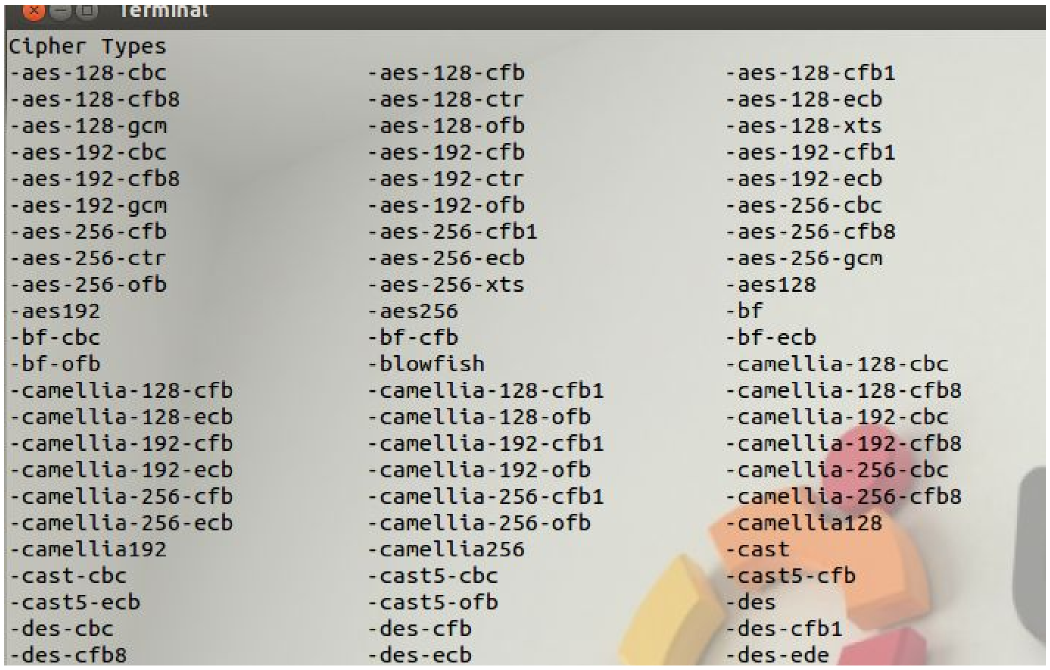
Task 1:

This is my plain text:



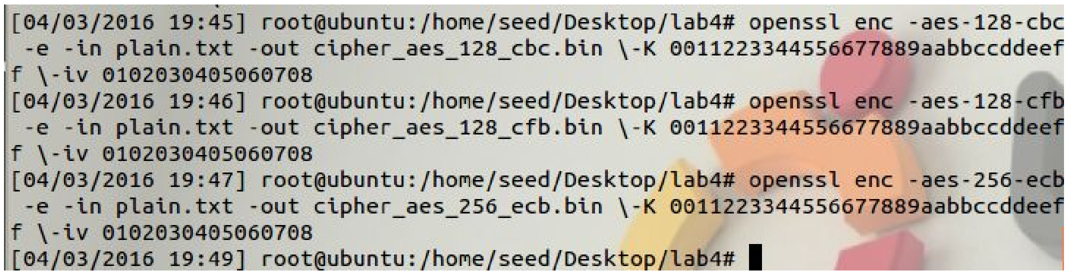
```
[04/03/2016 19:36] root@ubuntu:/home/seed/Desktop/lab4# cat plain.txt
This is a plain text.
[04/03/2016 19:37] root@ubuntu:/home/seed/Desktop/lab4# 
```

I use man enc --help to see different mode of the encryption:



```
Cipher Types
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-gcm               -aes-128-ofb               -aes-128-xts
-aes-192-cbc               -aes-192-cfb               -aes-192-cfb1
-aes-192-cfb8              -aes-192-ctr               -aes-192-ecb
-aes-192-gcm               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-gcm
-aes-256-ofb               -aes-256-xts               -aes128
-aes192                    -aes256                    -bf
-bf-cbc                    -bf-cfb                    -bf-ecb
-bf-ofb                    -blowfish                  -camellia-128-cbc
-camellia-128-cfb          -camellia-128-cfb1         -camellia-128-cfb8
-camellia-128-ecb          -camellia-128-ofb          -camellia-192-cbc
-camellia-192-cfb          -camellia-192-cfb1         -camellia-192-cfb8
-camellia-192-ecb          -camellia-192-ofb          -camellia-256-cbc
-camellia-256-cfb          -camellia-256-cfb1         -camellia-256-cfb8
-camellia-256-ecb          -camellia-256-ofb          -camellia128
-camellia192               -camellia256               -cast
-cast-cbc                  -cast5-cbc                 -cast5-cfb
-cast5-ecb                 -cast5-ofb                 -des
-des-cbc                   -des-cfb                   -des-cfb1
-des-cfb8                  -des-ecb                   -des-ede
```

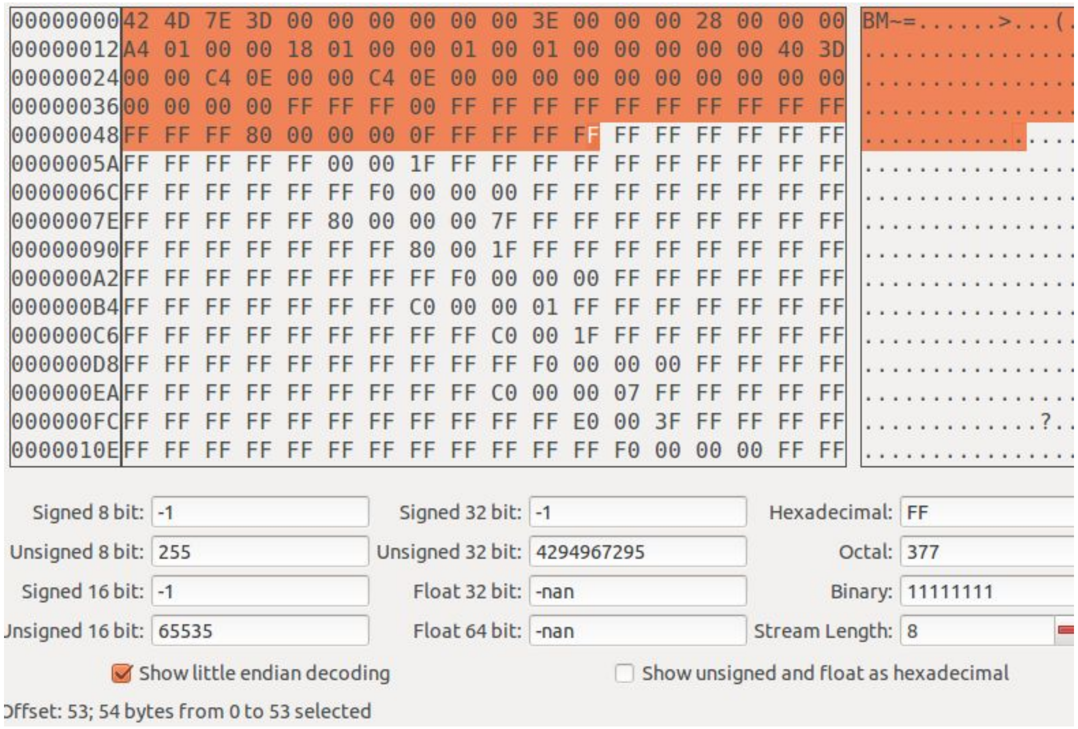
Here is my three different encryption:



```
[04/03/2016 19:45] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cbc
 -e -in plain.txt -out cipher_aes_128_cbc.bin \-K 00112233445566778899aabbccddeef
f \-iv 0102030405060708
[04/03/2016 19:46] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cfb
 -e -in plain.txt -out cipher_aes_128_cfb.bin \-K 00112233445566778899aabbccddeef
f \-iv 0102030405060708
[04/03/2016 19:47] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-256-ecb
 -e -in plain.txt -out cipher_aes_256_ecb.bin \-K 00112233445566778899aabbccddeef
f \-iv 0102030405060708
[04/03/2016 19:49] root@ubuntu:/home/seed/Desktop/lab4# 
```

Task 2:

I use ghex to open an original .bmp image, and see the first 54 bytes as following:



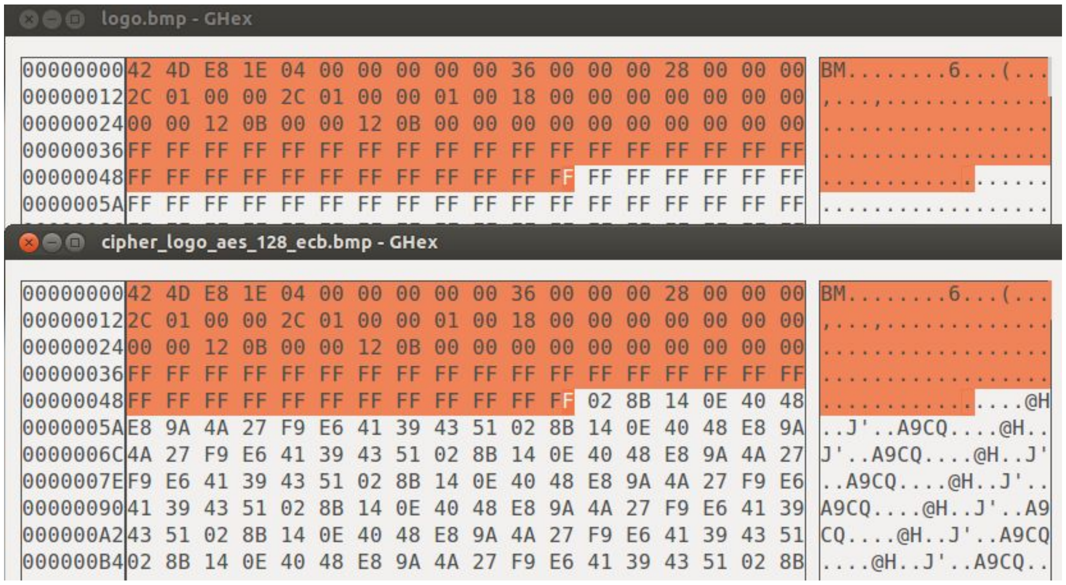
```
00000000 42 4D 7E 3D 00 00 00 00 00 00 3E 00 00 00 28 00 00 00  BM~=......>...(.
00000012 A4 01 00 00 18 01 00 00 01 00 01 00 00 00 00 00 40 3D  ................
00000024 00 00 C4 0E 00 00 C4 0E 00 00 00 00 00 00 00 00 00 00  ................
00000036 00 00 00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF  ................
00000048 FF FF FF 80 00 00 0F FF FF FF FF FF FF FF FF FF FF FF  ................
0000005A FF FF FF FF FF 00 00 1F FF FF FF FF FF FF FF FF FF FF  ................
0000006C FF FF FF FF FF FF F0 00 00 00 FF FF FF FF FF FF FF FF  ................
0000007E FF FF FF FF FF 80 00 00 00 7F FF FF FF FF FF FF FF FF  ................
00000090 FF FF FF FF FF FF FF 80 00 1F FF FF FF FF FF FF FF FF  ................
000000A2 FF FF FF FF FF FF FF FF F0 00 00 00 FF FF FF FF FF FF  ................
000000B4 FF FF FF FF FF FF FF C0 00 00 01 FF FF FF FF FF FF FF  ................
000000C6 FF FF FF FF FF FF FF FF FF C0 00 1F FF FF FF FF FF FF  ................
000000D8 FF FF FF FF FF FF FF FF FF FF F0 00 00 00 FF FF FF FF  ................
000000EA FF FF FF FF FF FF FF FF FF C0 00 00 07 FF FF FF FF FF  ................
000000FC FF FF FF FF FF FF FF FF FF FF FF E0 00 3F FF FF FF FF  .............?..
0000010E FF FF FF FF FF FF FF FF FF FF FF FF F0 00 00 00 FF FF  ................
```

```
   Signed 8 bit: -1             Signed 32 bit: -1             Hexadecimal: FF
 Unsigned 8 bit: 255          Unsigned 32 bit: 4294967295           Octal: 377
  Signed 16 bit: -1              Float 32 bit: -nan                Binary: 11111111
Unsigned 16 bit: 65535           Float 64 bit: -nan         Stream Length: 8

[✓] Show little endian decoding          [ ] Show unsigned and float as hexadecimal

Offset: 53; 54 bytes from 0 to 53 selected
```

After I encrypted the image file, I open that cipher file and selected first 54 bytes, swap it with the original .bmp header:



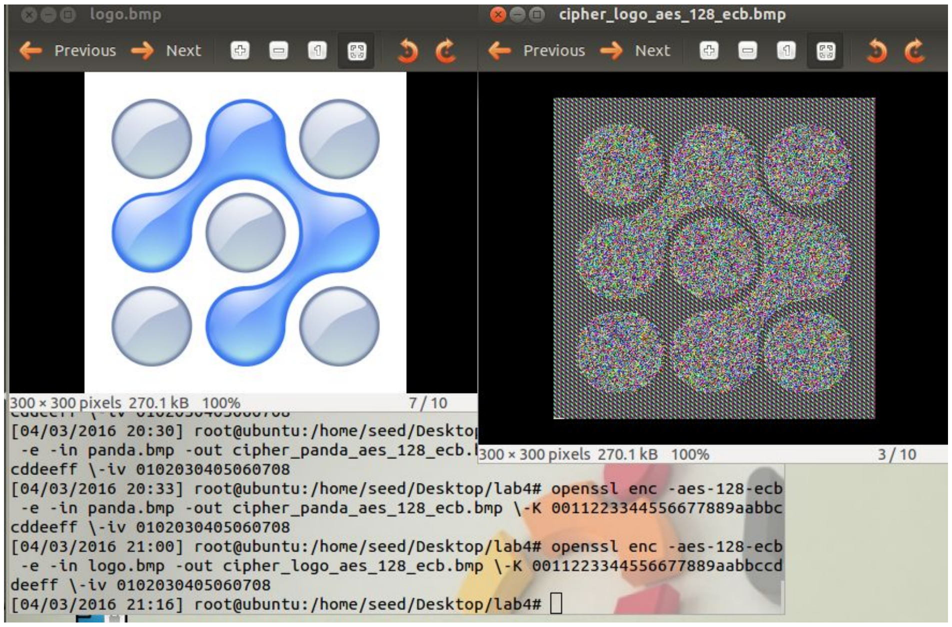
logo.bmp - GHex

```
00000000 42 4D E8 1E 04 00 00 00 00 00 36 00 00 00 28 00 00 00  BM........6...(...
00000012 2C 01 00 00 2C 01 00 00 01 00 18 00 00 00 00 00 00 00  ,...,.............
00000024 00 00 12 0B 00 00 12 0B 00 00 00 00 00 00 00 00 00 00  ..................
00000036 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ..................
00000048 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ..................
0000005A FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ..................
```

cipher_logo_aes_128_ecb.bmp - GHex

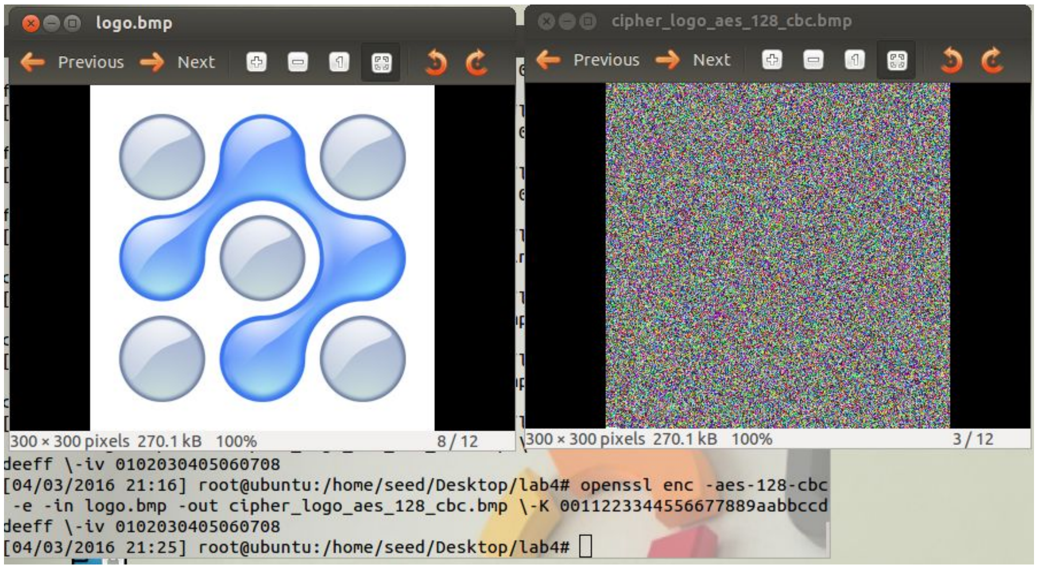
```
00000000 42 4D E8 1E 04 00 00 00 00 00 36 00 00 00 28 00 00 00  BM........6...(...
00000012 2C 01 00 00 2C 01 00 00 01 00 18 00 00 00 00 00 00 00  ,...,.............
00000024 00 00 12 0B 00 00 12 0B 00 00 00 00 00 00 00 00 00 00  ..................
00000036 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ..................
00000048 FF FF FF FF FF FF FF FF FF FF FF FF 02 8B 14 0E 40 48  ................@H
0000005A E8 9A 4A 27 F9 E6 41 39 43 51 02 8B 14 0E 40 48 E8 9A  ..J'..A9CQ....@H..
0000006C 4A 27 F9 E6 41 39 43 51 02 8B 14 0E 40 48 E8 9A 4A 27  J'..A9CQ....@H..J'
0000007E F9 E6 41 39 43 51 02 8B 14 0E 40 48 E8 9A 4A 27 F9 E6  ..A9CQ....@H..J'..
00000090 41 39 43 51 02 8B 14 0E 40 48 E8 9A 4A 27 F9 E6 41 39  A9CQ....@H..J'..A9
000000A2 43 51 02 8B 14 0E 40 48 E8 9A 4A 27 F9 E6 41 39 43 51  CQ....@H..J'..A9CQ
000000B4 02 8B 14 0E 40 48 E8 9A 4A 27 F9 E6 41 39 43 51 02 8B  ....@H..J'..A9CQ..
```

Here is the two modes before and after encryption result:

ECB: 

```
[04/03/2016 20:30] root@ubuntu:/home/seed/Deskto
 -e -in panda.bmp -out cipher_panda_aes_128_ecb.
cddeeff \-iv 0102030405060708
[04/03/2016 20:33] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ecb
 -e -in panda.bmp -out cipher_panda_aes_128_ecb.bmp \-K 00112233445566778899aabbc
cddeeff \-iv 0102030405060708
[04/03/2016 21:00] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ecb
 -e -in logo.bmp -out cipher_logo_aes_128_ecb.bmp \-K 00112233445566778899aabbccd
deeff \-iv 0102030405060708
[04/03/2016 21:16] root@ubuntu:/home/seed/Desktop/lab4# 
```

CBC:



```
deeff \-iv 0102030405060708
[04/03/2016 21:16] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cbc
 -e -in logo.bmp -out cipher_logo_aes_128_cbc.bmp \-K 00112233445566778899aabbccd
deeff \-iv 0102030405060708
[04/03/2016 21:25] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see the ECB mode is not safe when encrypting large files, the data pattern still can be observed after ECB encryption. On the other hand, CBC has a good data hidden feature even the file is large.

Task 3:

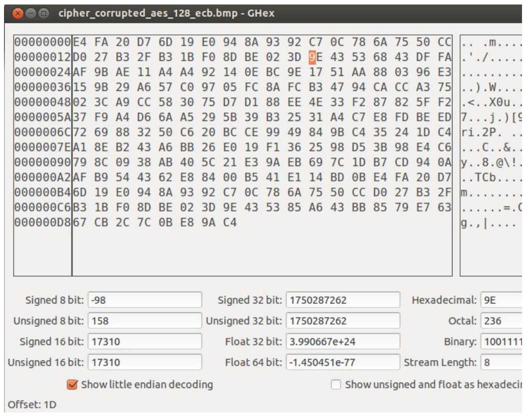
ECB:

My guess for this task is:

ECB mode: most information will be recovered since each block is decrypted independently.

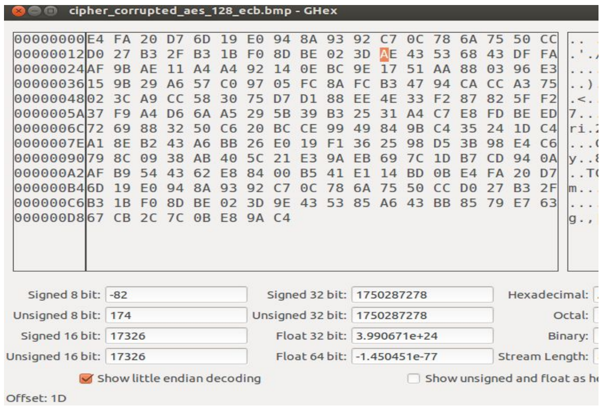
CBC mode: same as ECB.

CFB mode: most part will not be able to recover.
OFB mode: most information will be recovered. The corrupt will only affect the single byte.
The following shows how I flip one bit in the middle of encrypted file:


cipher_corrupted_aes_128_ecb.bmp - GHex

```
00000000E4 FA 20 D7 6D 19 E0 94 8A 93 92 C7 0C 78 6A 75 50 CC  .. .m....
00000012D0 27 B3 2F B3 1B F0 8D BE 02 3D 9E 43 53 68 43 DF FA  .'./.....
00000024AF 9B AE 11 A4 A4 92 14 0E BC 9E 17 51 AA 88 03 96 E3  .........
0000003615 9B 29 A6 57 C0 97 05 FC 8A FC B3 47 94 CA CC A3 75  ..).W....
0000004802 3C A9 CC 58 30 75 D7 D1 88 EE 4E 33 F2 87 82 5F F2  .<..X0u..
0000005A37 F9 A4 D6 6A A5 29 5B 39 B3 25 31 A4 C7 E8 FD BE ED  7...j.)[9
0000006C72 69 88 32 50 C6 20 BC CE 99 49 84 9B C4 35 24 1D C4  ri.2P. ..
0000007EA1 8E B2 43 A6 BB 26 E0 19 F1 36 25 98 D5 3B 98 E4 C6  ...C..&..
0000009079 8C 09 38 AB 40 5C 21 E3 9A EB 69 7C 1D B7 CD 94 0A  y..8.@\!.
000000A2AF B9 54 43 62 E8 84 00 B5 41 E1 14 BD 0B E4 FA 20 D7  ..TCb....
000000B46D 19 E0 94 8A 93 92 C7 0C 78 6A 75 50 CC D0 27 B3 2F  m........
000000C6B3 1B F0 8D BE 02 3D 9E 43 53 85 A6 43 BB 85 79 E7 63  ......=.C
000000D867 CB 2C 7C 0B E8 9A C4                                g.,|....
```

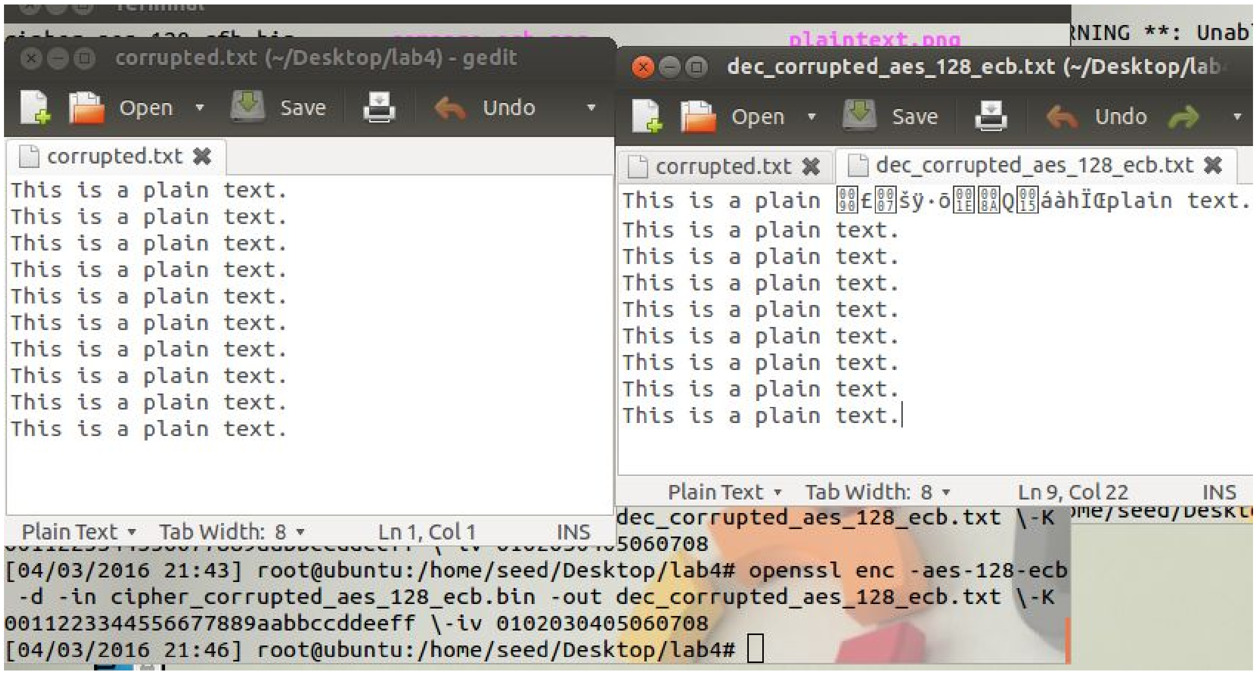
Signed 8 bit: -98 Signed 32 bit: 1750287262 Hexadecimal: 9E
Unsigned 8 bit: 158 Unsigned 32 bit: 1750287262 Octal: 236
Signed 16 bit: 17310 Float 32 bit: 3.990667e+24 Binary: 1001111
Unsigned 16 bit: 17310 Float 64 bit: -1.450451e-77 Stream Length: 8
[✓ Show little endian decoding] Show unsigned and float as hexadeci
Offset: 1D


cipher_corrupted_aes_128_ecb.bmp - GHex

```
00000000E4 FA 20 D7 6D 19 E0 94 8A 93 92 C7 0C 78 6A 75 50 CC  ..
00000012D0 27 B3 2F B3 1B F0 8D BE 02 3D AE 43 53 68 43 DF FA  .'.
00000024AF 9B AE 11 A4 A4 92 14 0E BC 9E 17 51 AA 88 03 96 E3  ...
0000003615 9B 29 A6 57 C0 97 05 FC 8A FC B3 47 94 CA CC A3 75  ..)
0000004802 3C A9 CC 58 30 75 D7 D1 88 EE 4E 33 F2 87 82 5F F2  .<.
0000005A37 F9 A4 D6 6A A5 29 5B 39 B3 25 31 A4 C7 E8 FD BE ED  7..
0000006C72 69 88 32 50 C6 20 BC CE 99 49 84 9B C4 35 24 1D C4  ri.
0000007EA1 8E B2 43 A6 BB 26 E0 19 F1 36 25 98 D5 3B 98 E4 C6  ...
0000009079 8C 09 38 AB 40 5C 21 E3 9A EB 69 7C 1D B7 CD 94 0A  y..
000000A2AF B9 54 43 62 E8 84 00 B5 41 E1 14 BD 0B E4 FA 20 D7  ..T
000000B46D 19 E0 94 8A 93 92 C7 0C 78 6A 75 50 CC D0 27 B3 2F  m..
000000C6B3 1B F0 8D BE 02 3D 9E 43 53 85 A6 43 BB 85 79 E7 63  ...
000000D867 CB 2C 7C 0B E8 9A C4                                g.,
```

Signed 8 bit: -82 Signed 32 bit: 1750287278 Hexadecimal:
Unsigned 8 bit: 174 Unsigned 32 bit: 1750287278 Octal:
Signed 16 bit: 17326 Float 32 bit: 3.990671e+24 Binary:
Unsigned 16 bit: 17326 Float 64 bit: -1.450451e-77 Stream Length:
[✓ Show little endian decoding] Show unsigned and float as h
Offset: 1D

The following is a comparison of the original plaintext and the decrypted text:


corrupted.txt (~/Desktop/lab4) - gedit
Open
Save
Undo
dec_corrupted_aes_128_ecb.txt (~/Desktop/lab
plaintext.png

corrupted.txt

```
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 INS

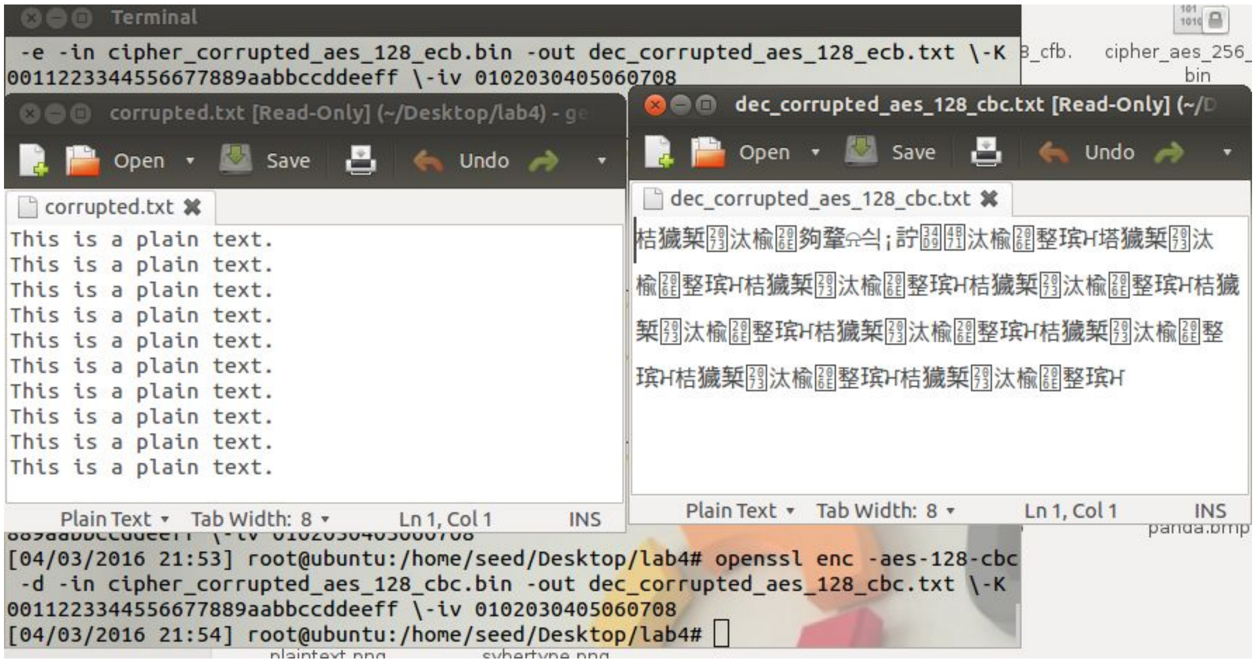
dec_corrupted_aes_128_ecb.txt

```
This is a plain [0090]£[0087]šÿ·ō[001E][008A]Q[0015]áàhÏŒplain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

Plain Text ▾ Tab Width: 8 ▾ Ln 9, Col 22 INS

```
RNING **: Unab
dec_corrupted_aes_128_ecb.txt \-K ome/seed/Deskt
5060708
[04/03/2016 21:43] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ecb
 -d -in cipher_corrupted_aes_128_ecb.bin -out dec_corrupted_aes_128_ecb.txt \-K
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 21:46] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see most of the information can be recovered. This is because we do the decryption one block each time. This will only affect one block long.



```
-e -in cipher_corrupted_aes_128_ecb.bin -out dec_corrupted_aes_128_ecb.txt \-K
00112233445566778899aabbccddeeff \-iv 0102030405060708
```

corrupted.txt [Read-Only] (~/Desktop/lab4) - ge

corrupted.txt

```
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

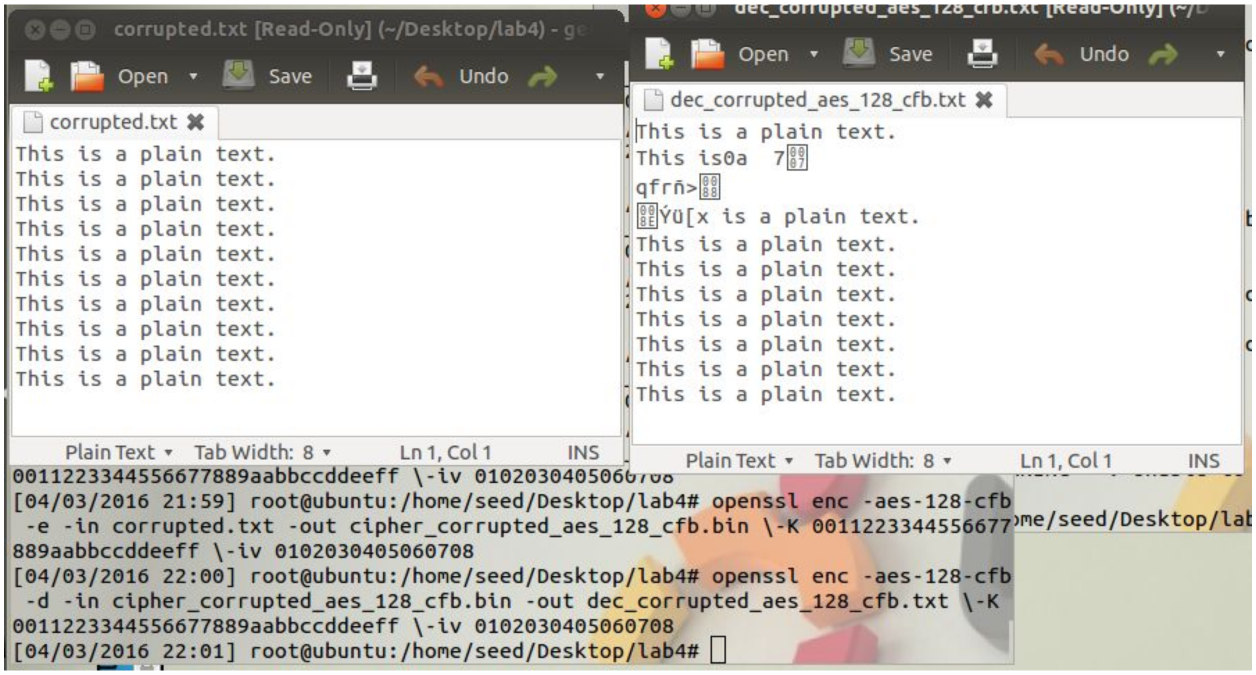
dec_corrupted_aes_128_cbc.txt [Read-Only] (~/D

dec_corrupted_aes_128_cbc.txt

```
桔獵棸□汰榆□夠肇ꪎ싀;訏□□汰榆□整瑸H塔獵棸□汰
榆□整瑸H桔獵棸□汰榆□整瑸H桔獵棸□汰榆□整瑸H桔獵
棸□汰榆□整瑸H桔獵棸□汰榆□整瑸H桔獵棸□汰榆□整
瑸H桔獵棸□汰榆□整瑸H桔獵棸□汰榆□整瑸H
```

```
[04/03/2016 21:53] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cbc
 -d -in cipher_corrupted_aes_128_cbc.bin -out dec_corrupted_aes_128_cbc.txt \-K
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 21:54] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see the whole file is affected. This is because it is chain mode. One single bit corrupted will cause all the decryption invalid.



corrupted.txt [Read-Only] (~/Desktop/lab4) - ge

corrupted.txt

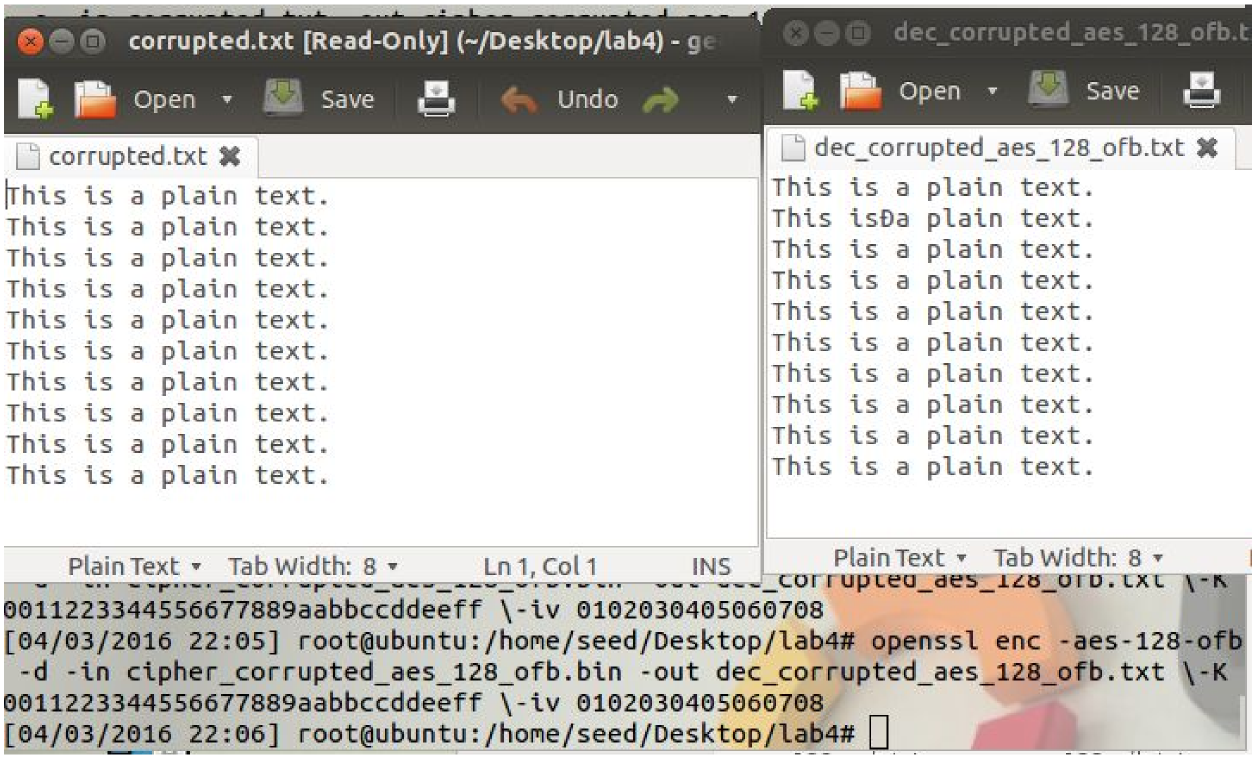
```
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

dec_corrupted_aes_128_cfb.txt

```
This is a plain text.
This is0a   7□
qfrñ>□
□Ýû[x is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

```
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 21:59] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cfb
 -e -in corrupted.txt -out cipher_corrupted_aes_128_cfb.bin \-K 0011223344556677
8899aabbccddeeff \-iv 0102030405060708
[04/03/2016 22:00] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cfb
 -d -in cipher_corrupted_aes_128_cfb.bin -out dec_corrupted_aes_128_cfb.txt \-K
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 22:01] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see there is some problem with the decrypted file. A few number of the blocks are affected. Since it use counter feedback block mode. A few of the blocks will be unable to recover.


corrupted.txt [Read-Only] (~/Desktop/lab4) - ge
Open
Save
Undo
corrupted.txt
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
dec_corrupted_aes_128_ofb.t
Open
Save
dec_corrupted_aes_128_ofb.txt
Plain Text
Tab Width: 8

```
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

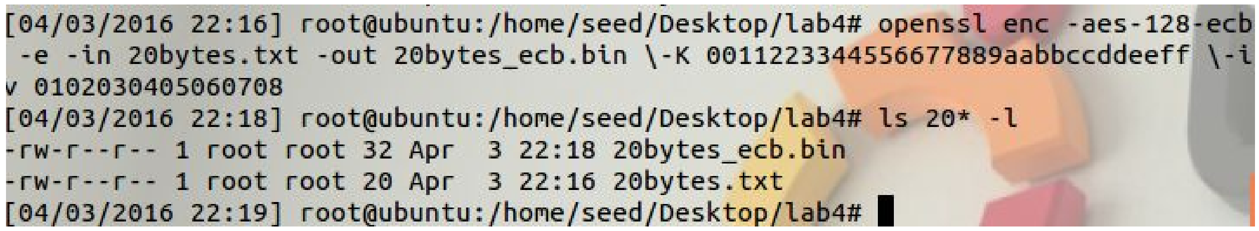
```
This is a plain text.
This isÐa plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
This is a plain text.
```

```
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 22:05] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ofb
 -d -in cipher_corrupted_aes_128_ofb.bin -out dec_corrupted_aes_128_ofb.txt \-K
00112233445566778899aabbccddeeff \-iv 0102030405060708
[04/03/2016 22:06] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see OFB mode is very good at handling corrupting situation. This is because the feedback only in the key-generation step. Thus only the single byte will be affected.

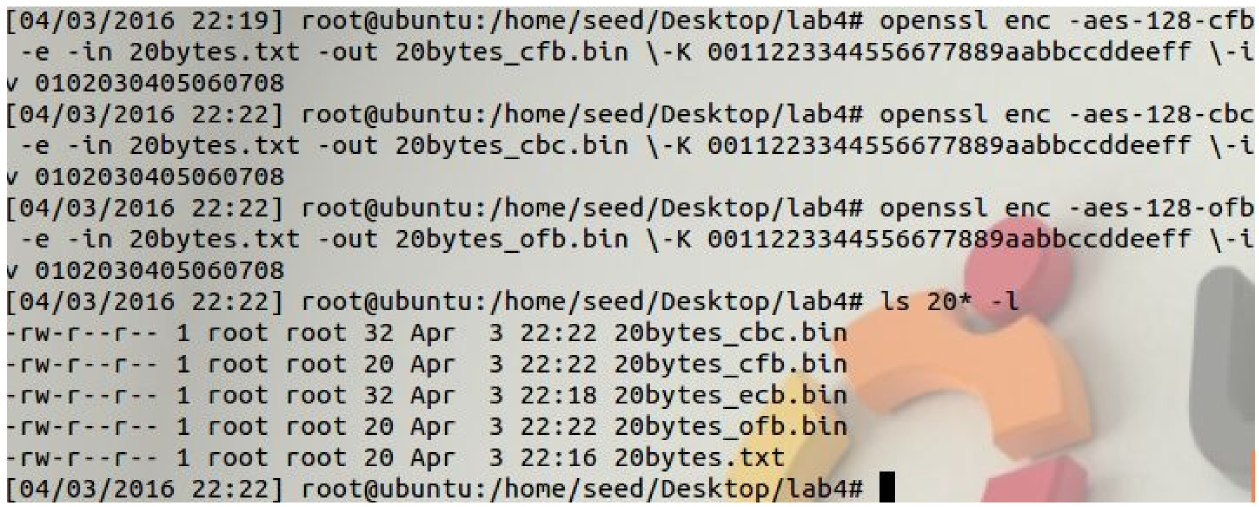
Task 4:

Here is a comparison of the original file and encrypted file. We can see the length of the encrypted file has changed. That means there is padding.



```
[04/03/2016 22:16] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ecb
 -e -in 20bytes.txt -out 20bytes_ecb.bin \-K 00112233445566778899aabbccddeeff \-i
v 0102030405060708
[04/03/2016 22:18] root@ubuntu:/home/seed/Desktop/lab4# ls 20* -l
-rw-r--r-- 1 root root 32 Apr  3 22:18 20bytes_ecb.bin
-rw-r--r-- 1 root root 20 Apr  3 22:16 20bytes.txt
[04/03/2016 22:19] root@ubuntu:/home/seed/Desktop/lab4# 
```

The following shows the other modes of the encryption.



```
[04/03/2016 22:19] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cfb
 -e -in 20bytes.txt -out 20bytes_cfb.bin \-K 00112233445566778899aabbccddeeff \-i
v 0102030405060708
[04/03/2016 22:22] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-cbc
 -e -in 20bytes.txt -out 20bytes_cbc.bin \-K 00112233445566778899aabbccddeeff \-i
v 0102030405060708
[04/03/2016 22:22] root@ubuntu:/home/seed/Desktop/lab4# openssl enc -aes-128-ofb
 -e -in 20bytes.txt -out 20bytes_ofb.bin \-K 00112233445566778899aabbccddeeff \-i
v 0102030405060708
[04/03/2016 22:22] root@ubuntu:/home/seed/Desktop/lab4# ls 20* -l
-rw-r--r-- 1 root root 32 Apr  3 22:22 20bytes_cbc.bin
-rw-r--r-- 1 root root 20 Apr  3 22:22 20bytes_cfb.bin
-rw-r--r-- 1 root root 32 Apr  3 22:18 20bytes_ecb.bin
-rw-r--r-- 1 root root 20 Apr  3 22:22 20bytes_ofb.bin
-rw-r--r-- 1 root root 20 Apr  3 22:16 20bytes.txt
[04/03/2016 22:22] root@ubuntu:/home/seed/Desktop/lab4# 
```

We can see CBC and ECB mode will use padding method. While CFB, OFB do not use padding. This is because when block mode is used, it has to be filled to encrypt. The chain mode does not need.

Task 5:

See my attached code. My code file is called get_key.c which can be used to compile. After run the program, it will print the key to the standard output if any. The dictionary has to be put in the same directory as the code file is.