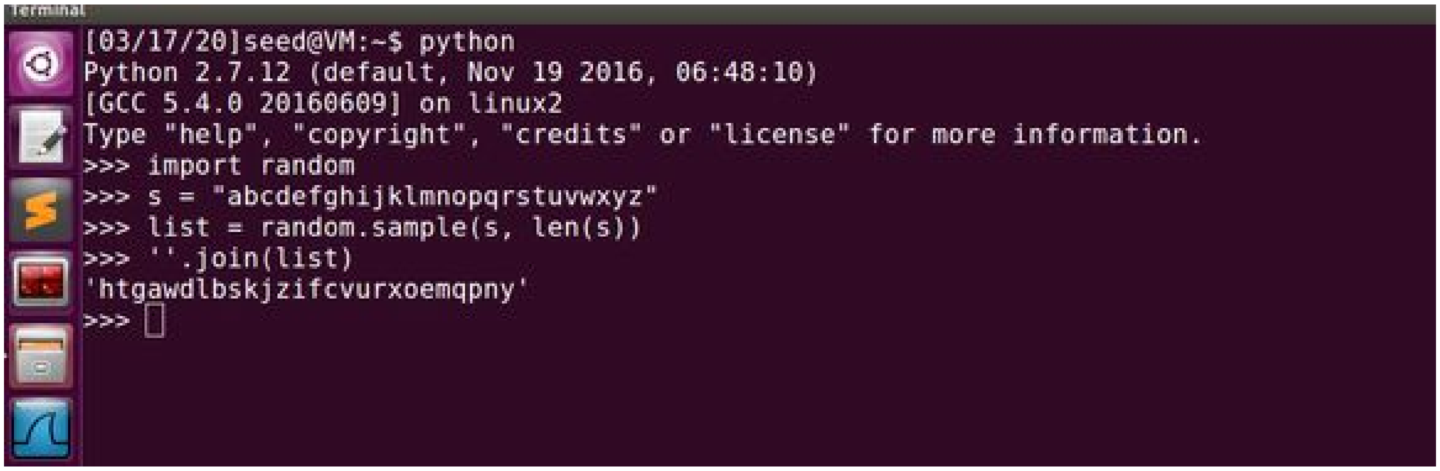
Lab 11 - Secret-Key Encryption Lab ***Darshan K (A20446137)***

Task 1 - Frequency analysis against Monoalphabetic Substitution Cipher:

In this task we describe how we encrypt the original text/article by finding its cipher text using frequency analysis and then decrypt.

Step1: In this step we removed all the punctuations and numbers, converted all Uppercase to Lowercase. The spaces are still preserved as boundaries to the words. In real encryption, spaces will be removed using the monoalphabetic cipher.

Step2: In this step we generate the encryption key by substitution table. We will permute the alphabets from a to z and use this as a key.



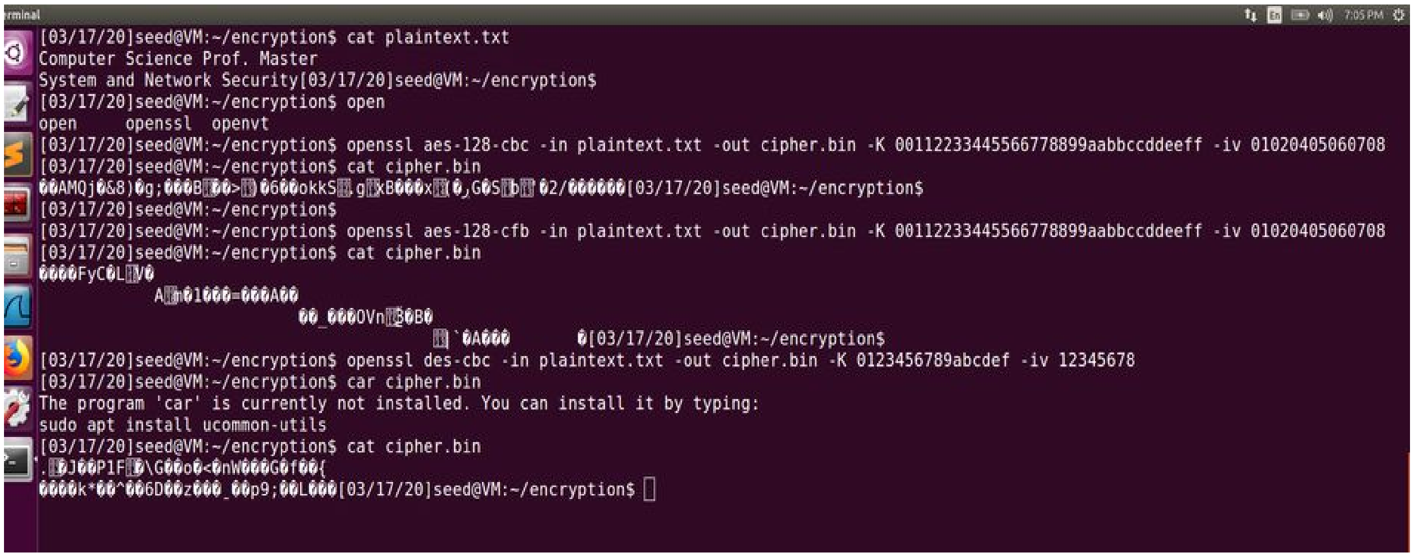
```
[03/17/20]seed@VM:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> ''.join(list)
'htgawdlbskjzifcvurxoemqpny'
>>> 
```

The key we are going to use here is “**htgawdlbskjzifcvurxoemqpny**”

Step3: Here the content of plaintext is “System and Network Security” and once it is encrypted by “**htgawdlbskjzifcvurxoemqpny**” we get the cipher text as “**Snxowi hfa Nwoqcrj Swgerson**”

Task 2 - Encryption using different Ciphers and Modes:

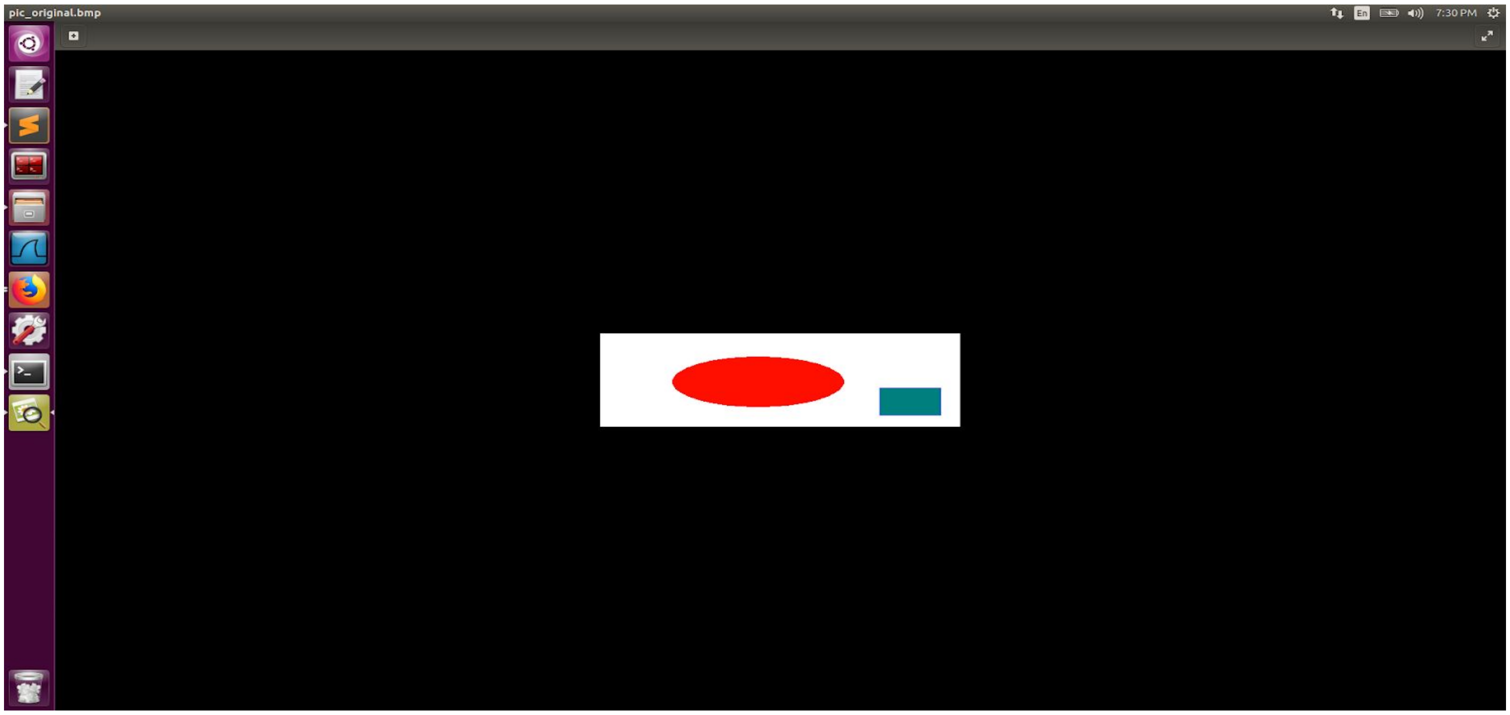
In this task, we make use of various encryption algorithms and nodes. We make use of *openssl enc* command to encrypt/decrypt a file. We make use of *-aes-128-cbc, -bf-cbc, -aes-128-cfb* cipher types. For this task, the plain text we use is **“Computer Science Prof. Master System and Network Security”.** We now encrypt using the 3 cipher types. Below is the screenshot of the various encryption mechanisms applied to the plaintext.

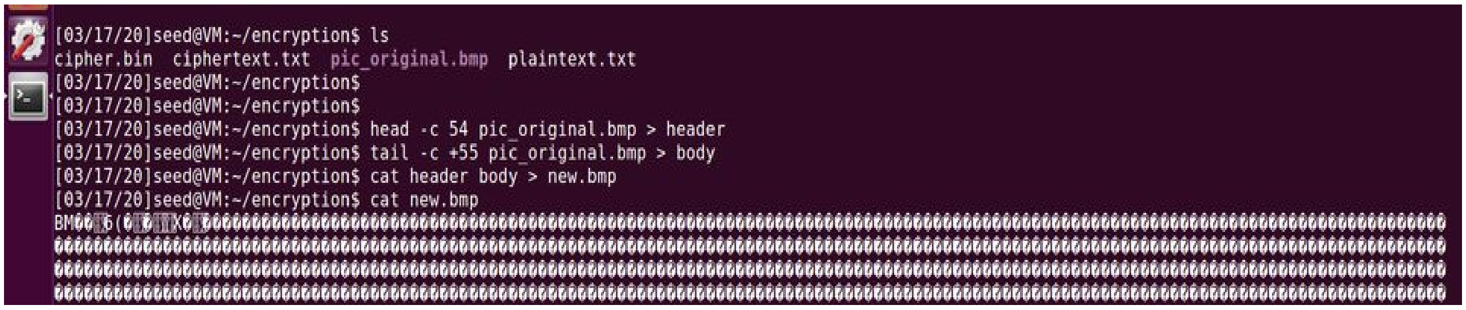


```
[03/17/20]seed@VM:~/encryption$ cat plaintext.txt
Computer Science Prof. Master
System and Network Security[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ open
open      openssl   openvt
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -in plaintext.txt -out cipher.bin -K 00112233445566778899aabbccddeeff -iv 01020405060708
[03/17/20]seed@VM:~/encryption$ cat cipher.bin
��AMQj�&8)�g;���B▯��>▯)�6��okkS▯.g▯xB���x▯(�,G�S▯b▯'�2/������[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cfb -in plaintext.txt -out cipher.bin -K 00112233445566778899aabbccddeeff -iv 01020405060708
[03/17/20]seed@VM:~/encryption$ cat cipher.bin
����FyC�L▯V�
           A▯n�1���=���A��
                             ��_���OVn▯B�B�
                                            ▯`�A���       �[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ openssl des-cbc -in plaintext.txt -out cipher.bin -K 0123456789abcdef -iv 12345678
[03/17/20]seed@VM:~/encryption$ car cipher.bin
The program 'car' is currently not installed. You can install it by typing:
sudo apt install ucommon-utils
[03/17/20]seed@VM:~/encryption$ cat cipher.bin
.▯�J��P1F▯�\G��o�<�nW���G�f��{
����k*��^��6D��z���_��p9;��L���[03/17/20]seed@VM:~/encryption$ ▯
```

Task 3 - Encryption Mode - ECB vs CBC

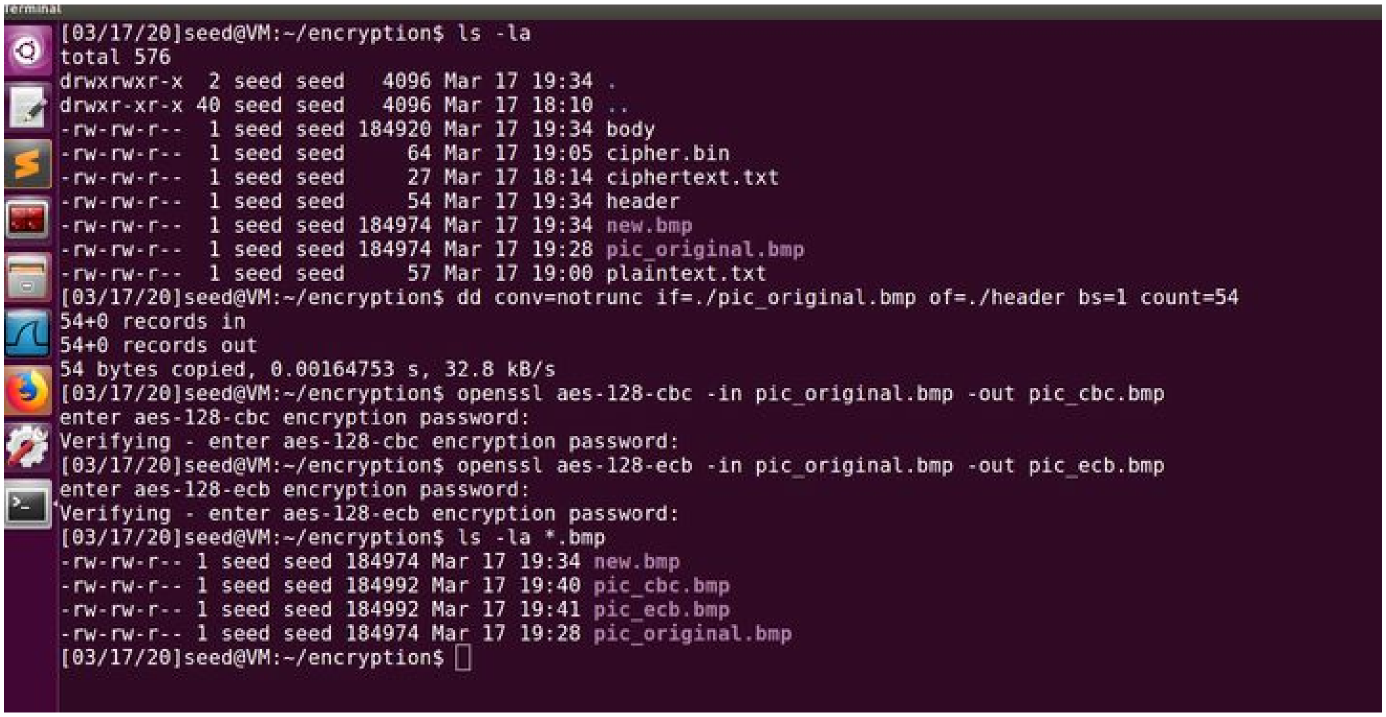
In this task, we will be encrypting the image that we downloaded from the lab site. People without the encryption key wouldn't know the contents of the picture. We encrypt the file using ECB and CBC. This is the image that we are going to encrypt.





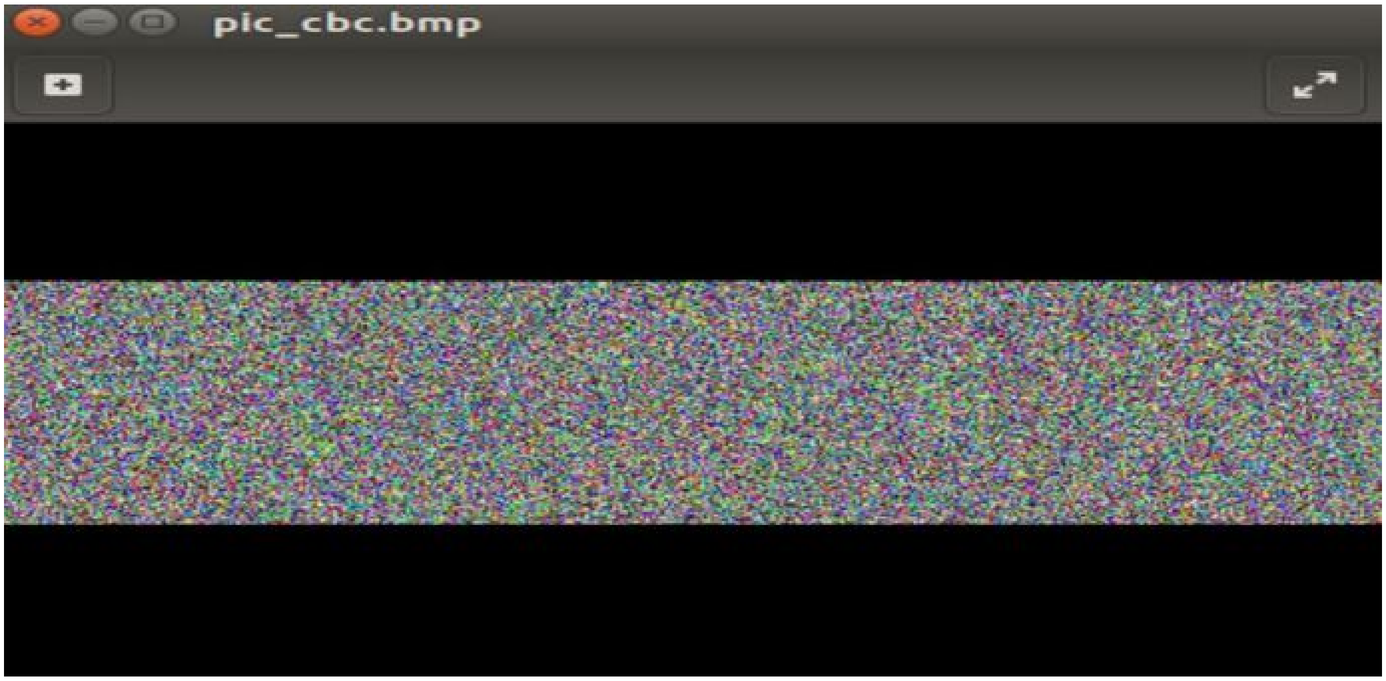
```
[03/17/20]seed@VM:~/encryption$ ls
cipher.bin  ciphertext.txt  pic_original.bmp  plaintext.txt
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ head -c 54 pic_original.bmp > header
[03/17/20]seed@VM:~/encryption$ tail -c +55 pic_original.bmp > body
[03/17/20]seed@VM:~/encryption$ cat header body > new.bmp
[03/17/20]seed@VM:~/encryption$ cat new.bmp
BM��6(��X��������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������
����������������������������������������������������������������������������������������������������
```

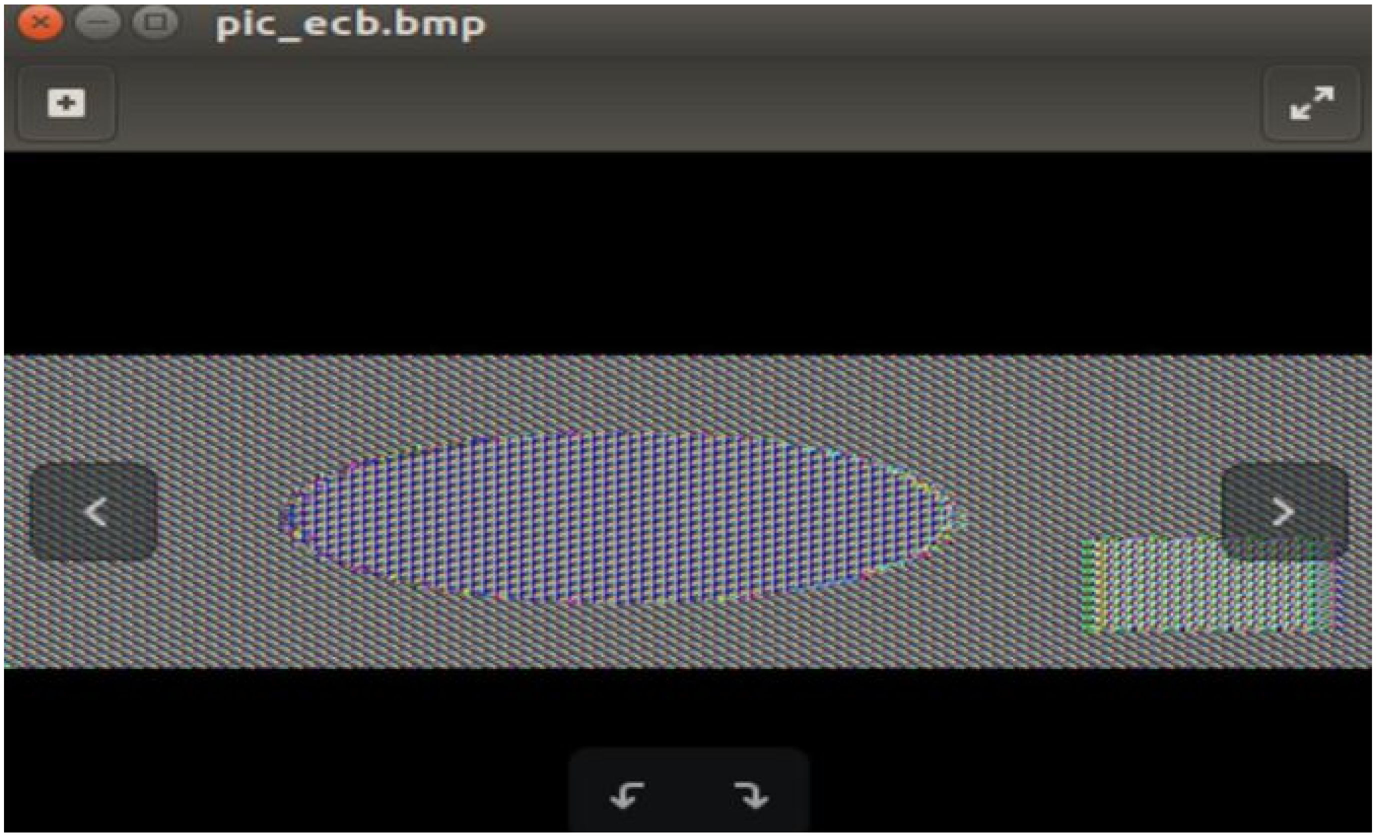
With the .bmp file, the first 54 bytes has the information of the header and we have to replace it with the header of the encrypted picture. We do that using the bless hex editor too. Below is the screenshot of what is executed.



```
[03/17/20]seed@VM:~/encryption$ ls -la
total 576
drwxrwxr-x  2 seed seed   4096 Mar 17 19:34 .
drwxr-xr-x 40 seed seed   4096 Mar 17 18:10 ..
-rw-rw-r--  1 seed seed 184920 Mar 17 19:34 body
-rw-rw-r--  1 seed seed     64 Mar 17 19:05 cipher.bin
-rw-rw-r--  1 seed seed     27 Mar 17 18:14 ciphertext.txt
-rw-rw-r--  1 seed seed     54 Mar 17 19:34 header
-rw-rw-r--  1 seed seed 184974 Mar 17 19:34 new.bmp
-rw-rw-r--  1 seed seed 184974 Mar 17 19:28 pic_original.bmp
-rw-rw-r--  1 seed seed     57 Mar 17 19:00 plaintext.txt
[03/17/20]seed@VM:~/encryption$ dd conv=notrunc if=./pic_original.bmp of=./header bs=1 count=54
54+0 records in
54+0 records out
54 bytes copied, 0.00164753 s, 32.8 kB/s
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -in pic_original.bmp -out pic_cbc.bmp
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-ecb -in pic_original.bmp -out pic_ecb.bmp
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
[03/17/20]seed@VM:~/encryption$ ls -la *.bmp
-rw-rw-r-- 1 seed seed 184974 Mar 17 19:34 new.bmp
-rw-rw-r-- 1 seed seed 184992 Mar 17 19:40 pic_cbc.bmp
-rw-rw-r-- 1 seed seed 184992 Mar 17 19:41 pic_ecb.bmp
-rw-rw-r-- 1 seed seed 184974 Mar 17 19:28 pic_original.bmp
[03/17/20]seed@VM:~/encryption$ 
```

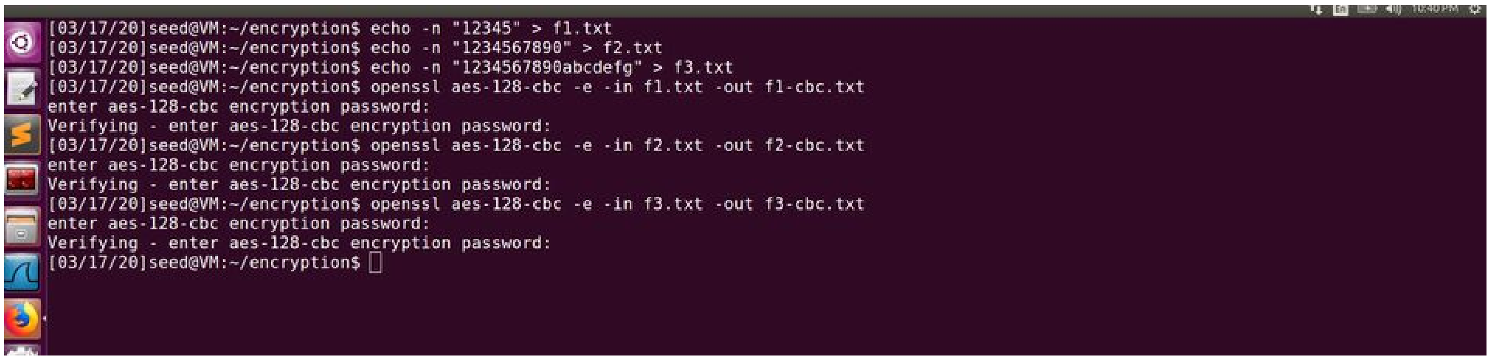
From the output of the last command of the screenshot, we can see the 2 bmp files that we generated that are encrypted. The below 2 screenshots show how the image is encrypted making it highly impossible to notice it's original contents.


pic_cbc.bmp



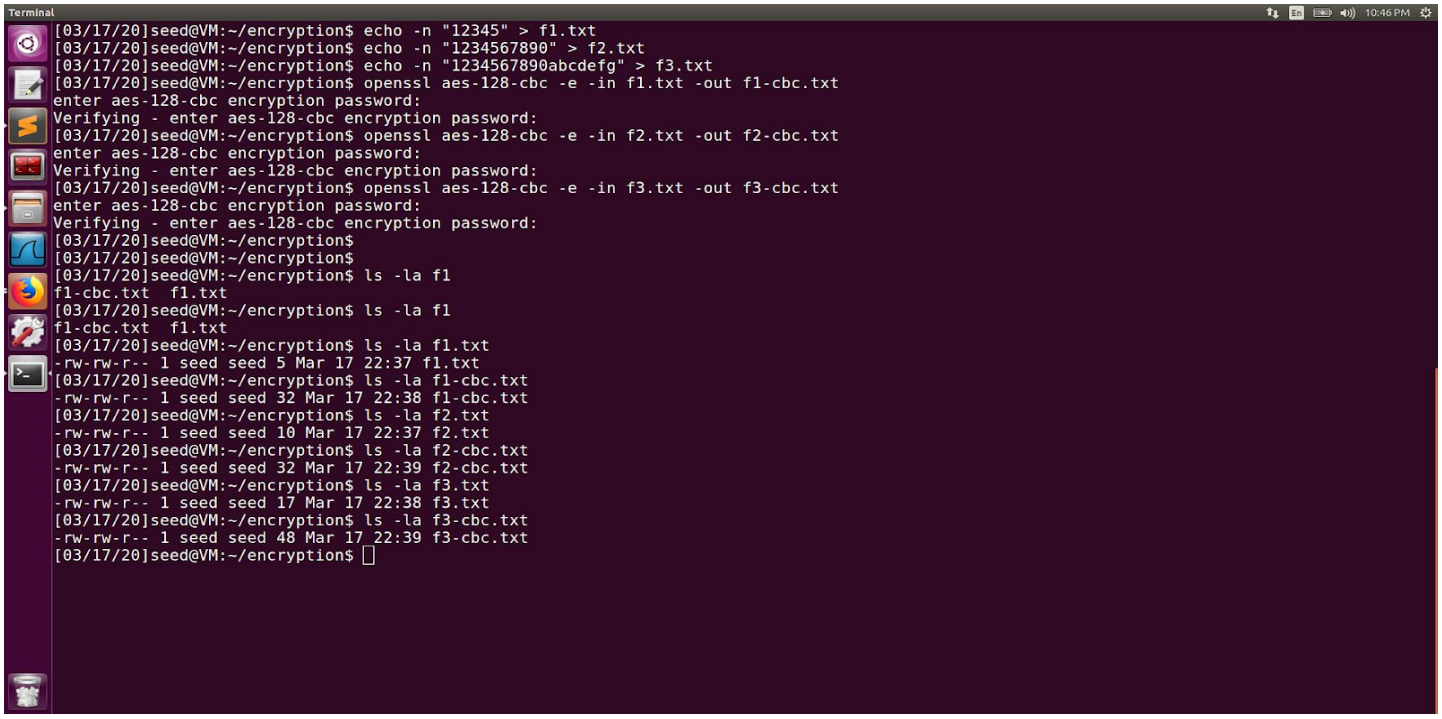
Task 4 - Padding :

For Blockciphers, if the size of the plaintext is not a multiple of a block size, padding may be required. All block ciphers use the standard block padding. Here we created 3 files of sizes 5,10,16 bytes each respectively. We encrypted using aes-128.



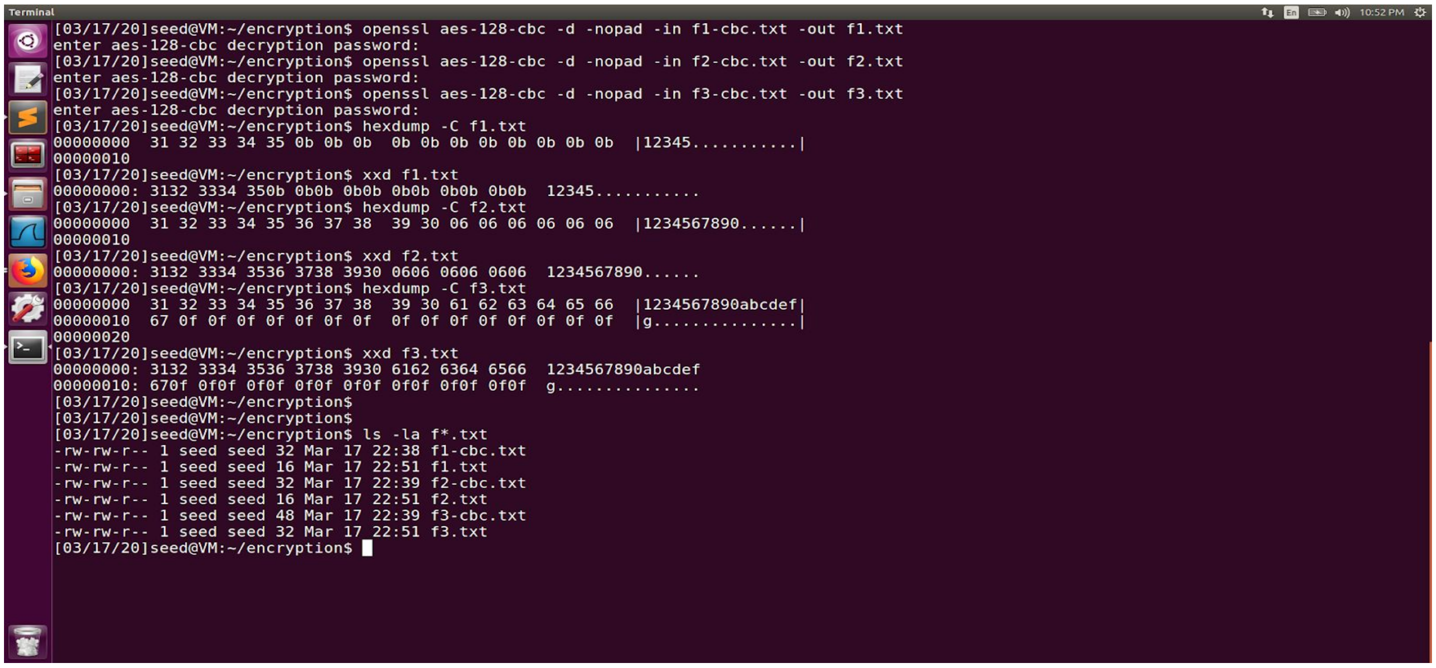
```
[03/17/20]seed@VM:~/encryption$ echo -n "12345" > f1.txt
[03/17/20]seed@VM:~/encryption$ echo -n "1234567890" > f2.txt
[03/17/20]seed@VM:~/encryption$ echo -n "1234567890abcdefg" > f3.txt
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f1.txt -out f1-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f2.txt -out f2-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f3.txt -out f3-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ 
```

The CFB and OFB modes pad 16 bytes to the end of the file. While CBC and ECB pad until a multiple of 8 is reached.



```
[03/17/20]seed@VM:~/encryption$ echo -n "12345" > f1.txt
[03/17/20]seed@VM:~/encryption$ echo -n "1234567890" > f2.txt
[03/17/20]seed@VM:~/encryption$ echo -n "1234567890abcdefg" > f3.txt
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f1.txt -out f1-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f2.txt -out f2-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -e -in f3.txt -out f3-cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ ls -la f1
f1-cbc.txt  f1.txt
[03/17/20]seed@VM:~/encryption$ ls -la f1
f1-cbc.txt  f1.txt
[03/17/20]seed@VM:~/encryption$ ls -la f1.txt
-rw-rw-r-- 1 seed seed 5 Mar 17 22:37 f1.txt
[03/17/20]seed@VM:~/encryption$ ls -la f1-cbc.txt
-rw-rw-r-- 1 seed seed 32 Mar 17 22:38 f1-cbc.txt
[03/17/20]seed@VM:~/encryption$ ls -la f2.txt
-rw-rw-r-- 1 seed seed 10 Mar 17 22:37 f2.txt
[03/17/20]seed@VM:~/encryption$ ls -la f2-cbc.txt
-rw-rw-r-- 1 seed seed 32 Mar 17 22:39 f2-cbc.txt
[03/17/20]seed@VM:~/encryption$ ls -la f3.txt
-rw-rw-r-- 1 seed seed 17 Mar 17 22:38 f3.txt
[03/17/20]seed@VM:~/encryption$ ls -la f3-cbc.txt
-rw-rw-r-- 1 seed seed 48 Mar 17 22:39 f3-cbc.txt
[03/17/20]seed@VM:~/encryption$
```

Now, we decrypt the file with *nopad* option which gives us the padding information, to know the padding for each of the file we have to decrypt. We make use of hex tool to display the contents of the padding in hex format.



```
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -d -nopad -in f1-cbc.txt -out f1.txt
enter aes-128-cbc decryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -d -nopad -in f2-cbc.txt -out f2.txt
enter aes-128-cbc decryption password:
[03/17/20]seed@VM:~/encryption$ openssl aes-128-cbc -d -nopad -in f3-cbc.txt -out f3.txt
enter aes-128-cbc decryption password:
[03/17/20]seed@VM:~/encryption$ hexdump -C f1.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[03/17/20]seed@VM:~/encryption$ xxd f1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[03/17/20]seed@VM:~/encryption$ hexdump -C f2.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[03/17/20]seed@VM:~/encryption$ xxd f2.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
[03/17/20]seed@VM:~/encryption$ hexdump -C f3.txt
00000000  31 32 33 34 35 36 37 38  39 30 61 62 63 64 65 66  |1234567890abcdef|
00000010  67 0f 0f 0f 0f 0f 0f 0f  0f 0f 0f 0f 0f 0f 0f 0f  |g...............|
00000020
[03/17/20]seed@VM:~/encryption$ xxd f3.txt
00000000: 3132 3334 3536 3738 3930 6162 6364 6566  1234567890abcdef
00000010: 670f 0f0f 0f0f 0f0f 0f0f 0f0f 0f0f 0f0f  g...............
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$
[03/17/20]seed@VM:~/encryption$ ls -la f*.txt
-rw-rw-r-- 1 seed seed 32 Mar 17 22:38 f1-cbc.txt
-rw-rw-r-- 1 seed seed 16 Mar 17 22:51 f1.txt
-rw-rw-r-- 1 seed seed 32 Mar 17 22:39 f2-cbc.txt
-rw-rw-r-- 1 seed seed 16 Mar 17 22:51 f2.txt
-rw-rw-r-- 1 seed seed 48 Mar 17 22:39 f3-cbc.txt
-rw-rw-r-- 1 seed seed 32 Mar 17 22:51 f3.txt
[03/17/20]seed@VM:~/encryption$
```

Task 5 - Error Propagation - Corrupted Cipher Text :

In this task, we try to understand the error propagation property of various encryption modes. To achieve that we create a file of 1000 bytes long and encrypt it using AES-128 cipher. After encrypting a single bit of the 55th byte in the encrypted file got corrupted which can be achieved using the bless hex editor. We then decrypted the corrupted ciphertext file using the correct key and an initial vector. We executed it for ECB, CBC, OFB, CFB modes. And the following conclusions were drawn after seeing the results.

- In the case of ECB, when there is any problem in the ciphertext, only one block gets affected and moreover each block is decrypted independently. Since we do the decryption one block at a time, the corrupted bit of the 30th byte in Ciphertext block's 8 bytes might spread to all n bits in plaintext block's 8 bytes.
- In CBC, there was no change seen in 2 blocks.
- In CFB, there are problems in n/r number of blocks. Therefore, the error propagation criteria is poor.
- In OFB, the feedback is only in the key-generation system. In the Plaintext, only the byte or character corresponding to the single digit of the corrupted 30th byte is corrupted.

  Therefore it can be concluded that only the OFB mode shows the most promising result and all the other texts are recovered.

Task 6 - Initial Vector(IV):

Task 6.1: In this task we explain why the Initialization vector has to be unique.

IV is a fixed size input to a cryptographic primitive that is typically required to be random or pseudo random. It can only be used once.. Randomization is important for encryption schemes to achieve semantic security. This property on repeated usage of the scheme under the same key does not allow an attacker to infer relationships between segments of the encrypted message. If we use the same IV then for similar plaintexts we will get a similar cipher text making it easy for the attacker to crack it.

Task 6.2: If we replace OFB in this experiment with CFB (Cipher Feedback), how much of P2 can be revealed?

With the CFB mode, the encryption process is ~ take the most recent ciphertext block, pass it through the block cipher, and then exclusive-or that with the plain text block to generate the next ciphertext block. This brings an advantage of error recovery including the one's that add or delete ciphertext blocks. The other advantage that CFB mode has, is that the decryption process also uses the block cipher in encryption mode.

Task 7 - Programming using the Crypto Library:

In this task, we're given plaintext and its corresponding cipher text, our job is to find the key. We made use of the EVP interface and linked it to the openssl libraries where several encryption algorithms are kept in place. I first copied all the word list and pasted it in the working directory of the application. And we call this file from the C code that we wrote. Here is the code I've made use of to get the secret key.

```
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int hex_to_int(char c){
        int first = c / 16 - 3;
        int second = c % 16;
        int result = first*10 + second;
        if(result > 9) result--;
        return result;
}

int hex_to_ascii(char c, char d){
        int high = hex_to_int(c) * 16;
        int low = hex_to_int(d);
        return high+low;
}

int main(int arc, char *argv[])
```

```
  {
        unsigned char outbuf[1024];
        unsigned char cipher[1024];
        unsigned char temp, key[16];
        int outlen, tmplen, l, i, length, count, found =0, k = 0;
        size_t nread, len;
        FILE *in;
        unsigned char iv[17];

        for(i = 0; i < 17; i++)
        iv[i] = 0;
        iv[16] = '\0';

        char intext[] = "This is a top secret.";
        char st[] =
"8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9";
        i = 0;
        while(i < 64)
        {
        if(st[i] >= 'a' && st[i] <= 'z')
                st[i] = st[i] - 32;
        i++;
        }

        length = strlen(st);
        char buf = 0;
        for(i = 0; i < length; i++)
        {
        if(i % 2 != 0)
        {
                cipher[k] = hex_to_ascii(buf, st[i]);
                k++;
        }
        else
```

```
{
        buf = st[i];
}
}
cipher[k] = '\0';
in = fopen("/home/seed/encryption/wordlist.txt", "r");
if(in == NULL)
{
printf("\n cannot open file");
exit(1);
}

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);

while(fgets(key, sizeof(key), in) != NULL)
{
l = 0;
if(strlen(key) < 16)
{
        l = strlen(key)-1;
        while(l < 16)
        {
        key[l] = ' ';
        l++;
        }
        key[l] = '\0';
}
else
        key[16] = '\0';


EVP_EncryptInit_ex(&ctx, EVP_aes_128_cbc(), NULL, key, iv);
```

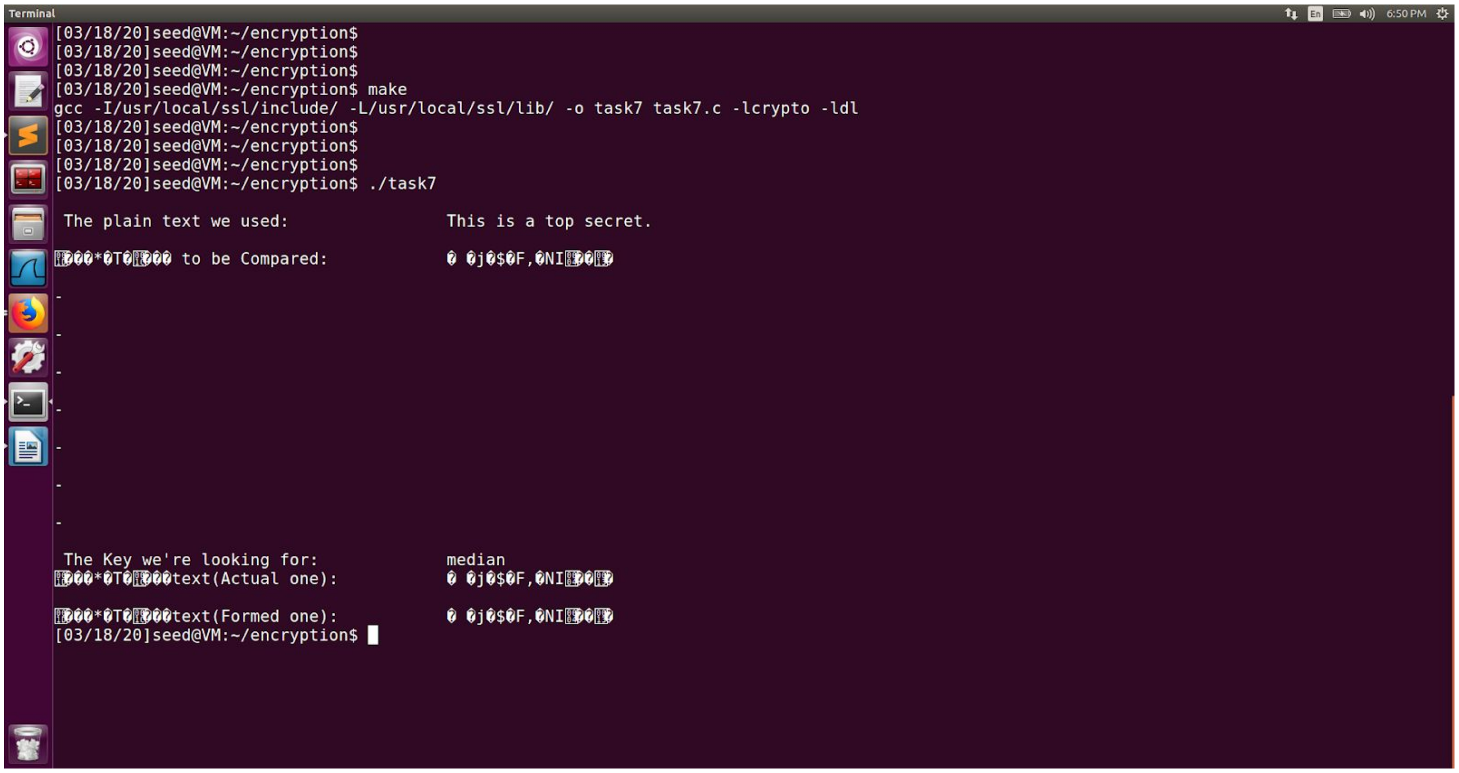
```
if(!EVP_EncryptUpdate(&ctx, outbuf, &outlen, intext, strlen(intext)))
{
        /* Error */
        return 0;
}
/* Buffer passed to EVP_EncryptFinal() must be after data just
* encrypted to avoid overwriting it.
*/
if(!EVP_EncryptFinal_ex(&ctx, outbuf + outlen, &tmplen))
{
/* Error */
return 0;
}
outlen += tmplen;
EVP_CIPHER_CTX_cleanup(&ctx);

count = 0;
for(i = 0; i < 32; i++)
{
        if(cipher[i] == outbuf[i])
        count++;
}
        //printf("\n actual cipher text..........%s", cipher);
        //printf("\n formed cipher text..........%s\n", outbuf);
if(count == 32)
{
        printf("\n The plain text we used:  \t\t%s\n",intext);
        printf("\n Cipher text to be Compared:  \t\t%s\n", cipher);
        printf("\n-\n");
        printf("\n-\n");
        printf("\n-\n");
        printf("\n-\n");
        printf("\n-\n");
        printf("\n-\n");
```

```
            printf("\n-\n");
            printf("\n The Key we're looking for:\t\t%s",key);
            printf("\n The Cipher text(Actual one): \t\t%s\n", cipher);
            printf("\n The Cipher text(Formed one):\t\t%s\n", outbuf);
            found = 1;
            break;
        }

        }
        fclose(in);
        if(found == 0)
        {
            printf("\niv..........................%s",iv);
        printf("\nplain text..................%s\n",intext);
        printf("cipher text..........%s\n", cipher);
        printf("cipher text in hex...%s\n",st);
        printf("\n\n key cannot be found for the above cipher text\n");
        }
        return 0;
        }
```

The Make file that I used for this code is:

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

all:
  gcc -I$(INC) -L$(LIB) -o task7 task7.c -lcrypto -ldl
```

The output on executing the executable task7 is shown below. This displays the secret key



```
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$ make
gcc -I/usr/local/ssl/include/ -L/usr/local/ssl/lib/ -o task7 task7.c -lcrypto -ldl
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$
[03/18/20]seed@VM:~/encryption$ ./task7

 The plain text we used:                This is a top secret.

����*�T�����  to be Compared:            � �j�$�F,�NI����

-

-

-

-

-

-

-

 The Key we're looking for:             median
����*�T�����text(Actual one):            � �j�$�F,�NI����

����*�T�����text(Formed one):            � �j�$�F,�NI����
[03/18/20]seed@VM:~/encryption$ 
```

In this output we’ve encrypted the wordlist.txt file that I copied from the lab site. I encrypted it using aes-128-cbc cipher.