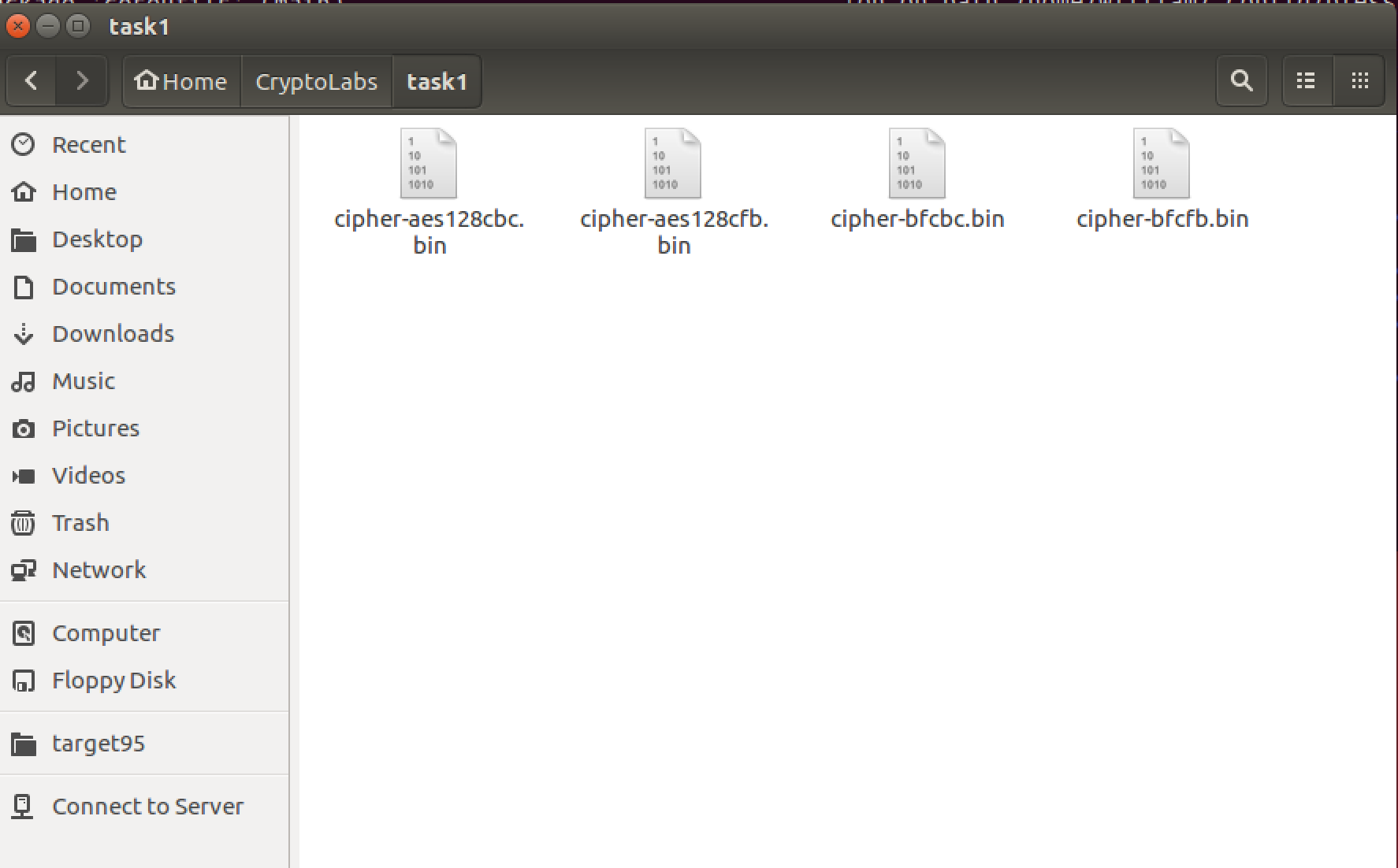
Crypto Lab Report

Environment:

I prepare the environment as the instructions. The OS is Ubuntu and I installed the bless as the hex editor. Moreover, I installed the OpenSSL.

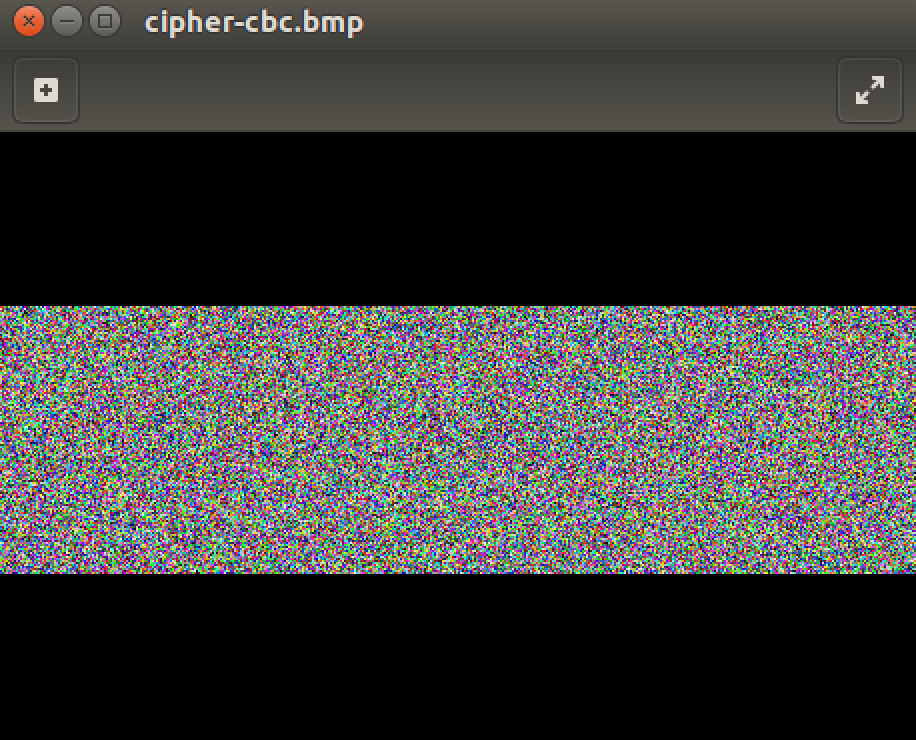
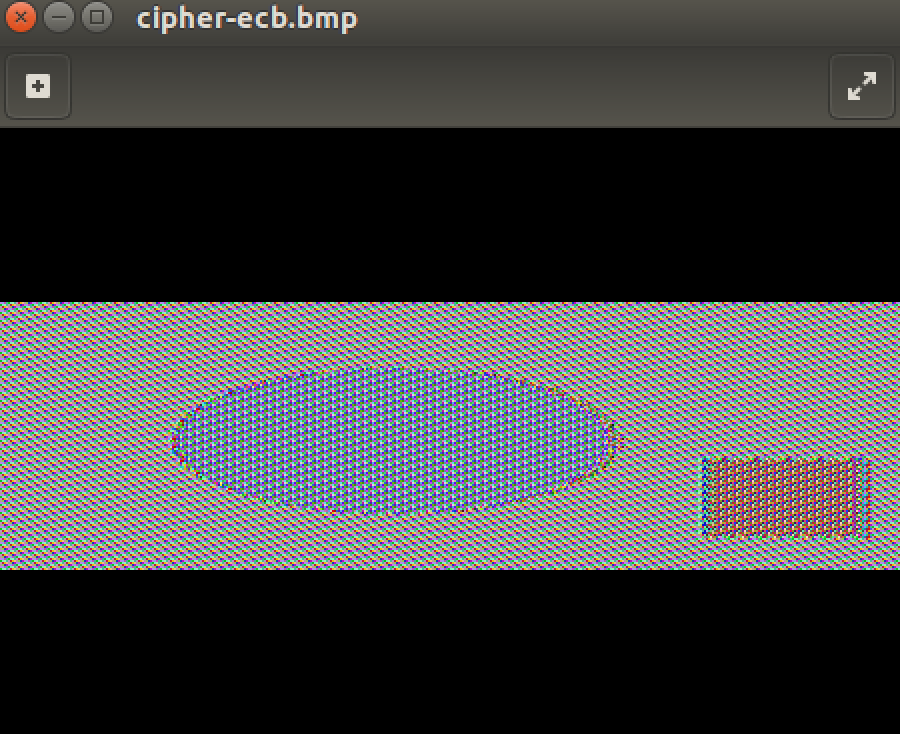
Task 1: Encryption using different ciphers and modes

In this task, I use two cipher types: Base 64 and AES 128. And two modes are CBC and ECB. And then I use the given commands and get the following encrypted files:



Task 2: Encryption Mode – ECB vs. CBC

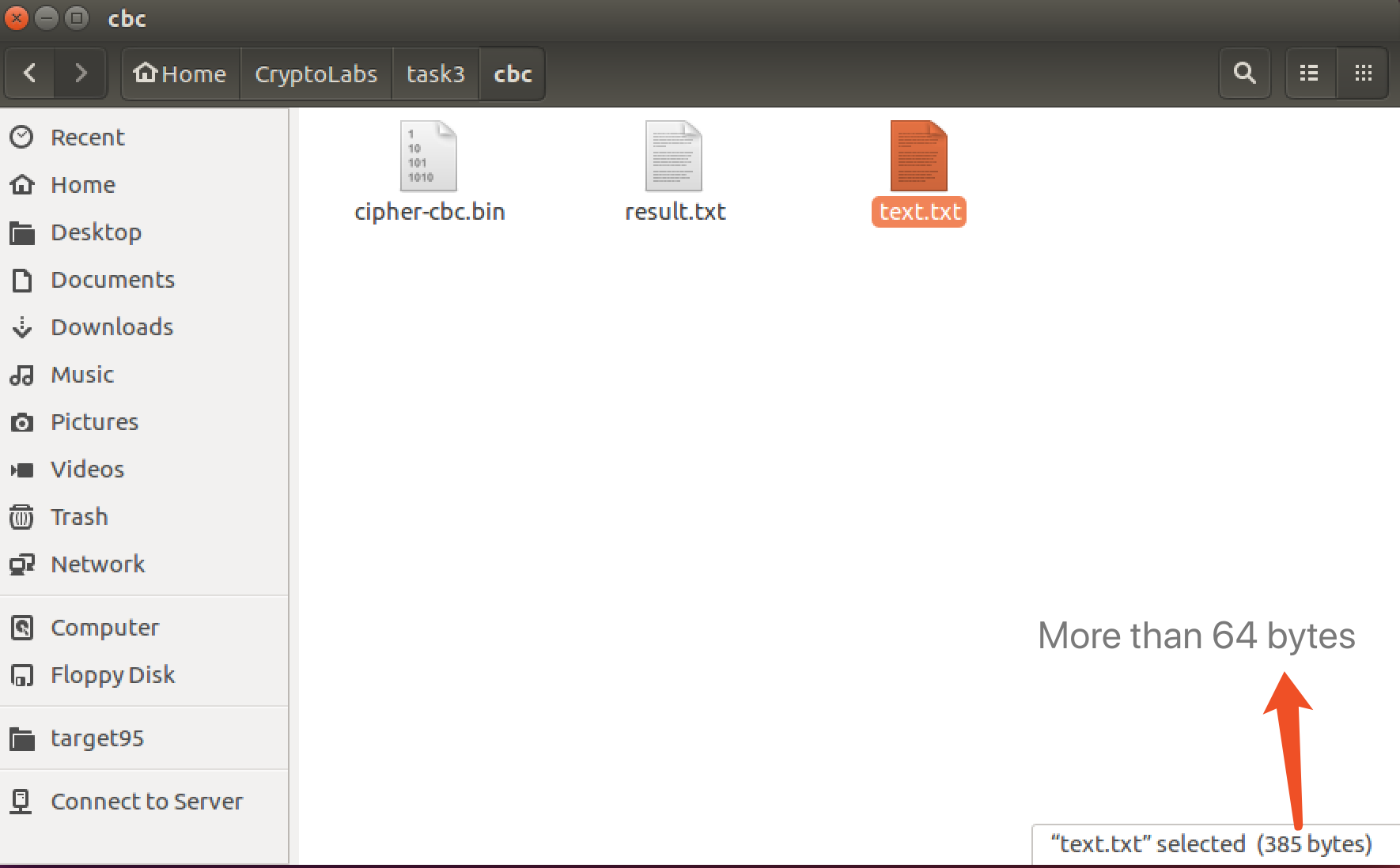
In this task, I used the AES 128 cipher and two modes. The modes are CBC and ECB. I encrypted the original file pic\_original.bmp and get the encrypted file. In my task, I named these files cipher-cbc.bmp and cipher-ecb.bmp. After that, I use bless to open these encrypted files and copy the first 54 bytes from original file, pasted them into the two encrypted files. After these steps, I opened the two files as below:



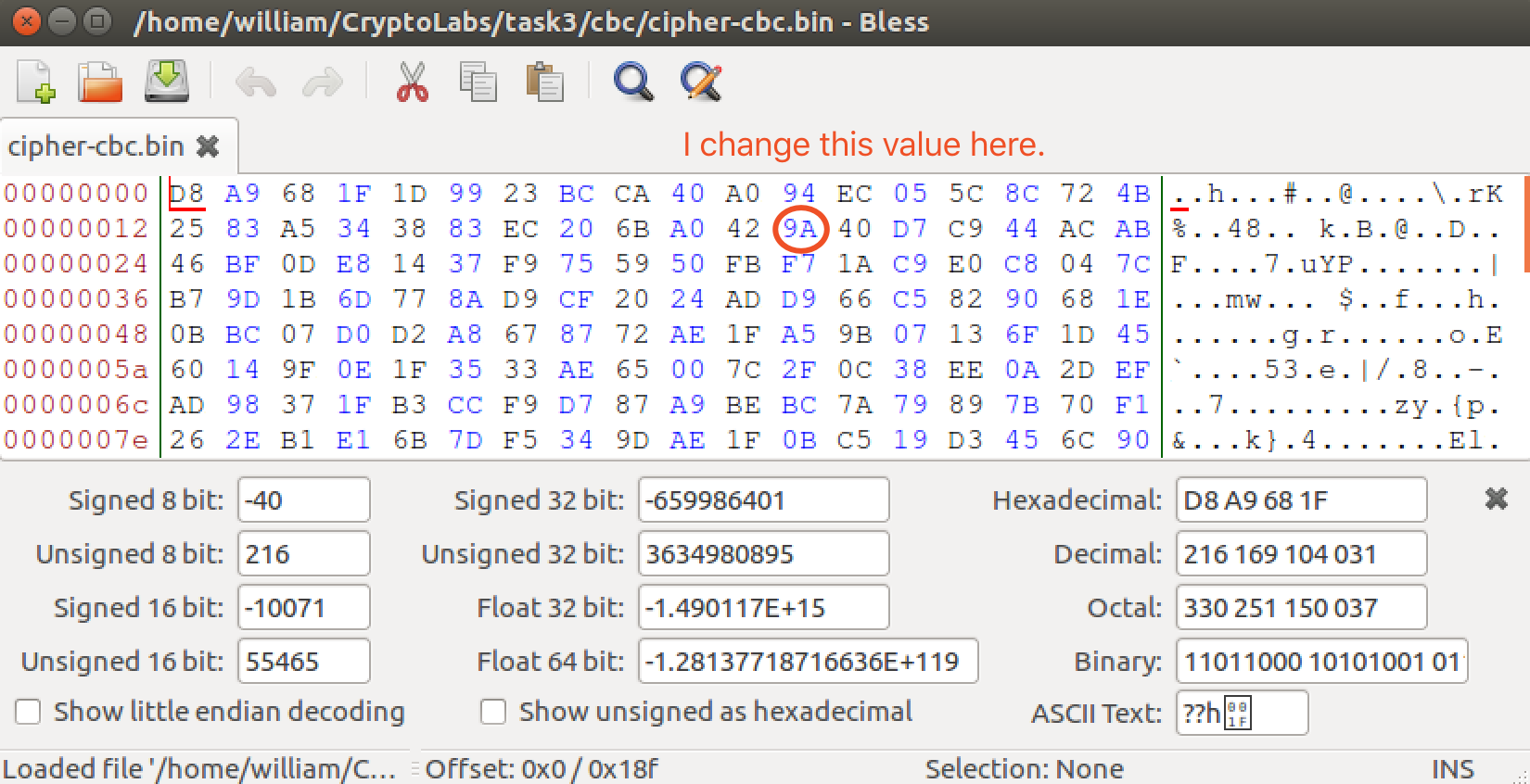
According to the pictures, we can get the information from ECB mode but we cannot get any information form CBC mode. This is because ECB mode generates same cipher text for repeating plain text. Even though the cipher image is different from the original image, most of the original image’s information can be obtained from the cipher image.

Task 3: Encryption Mode – Corrupted Cipher Text

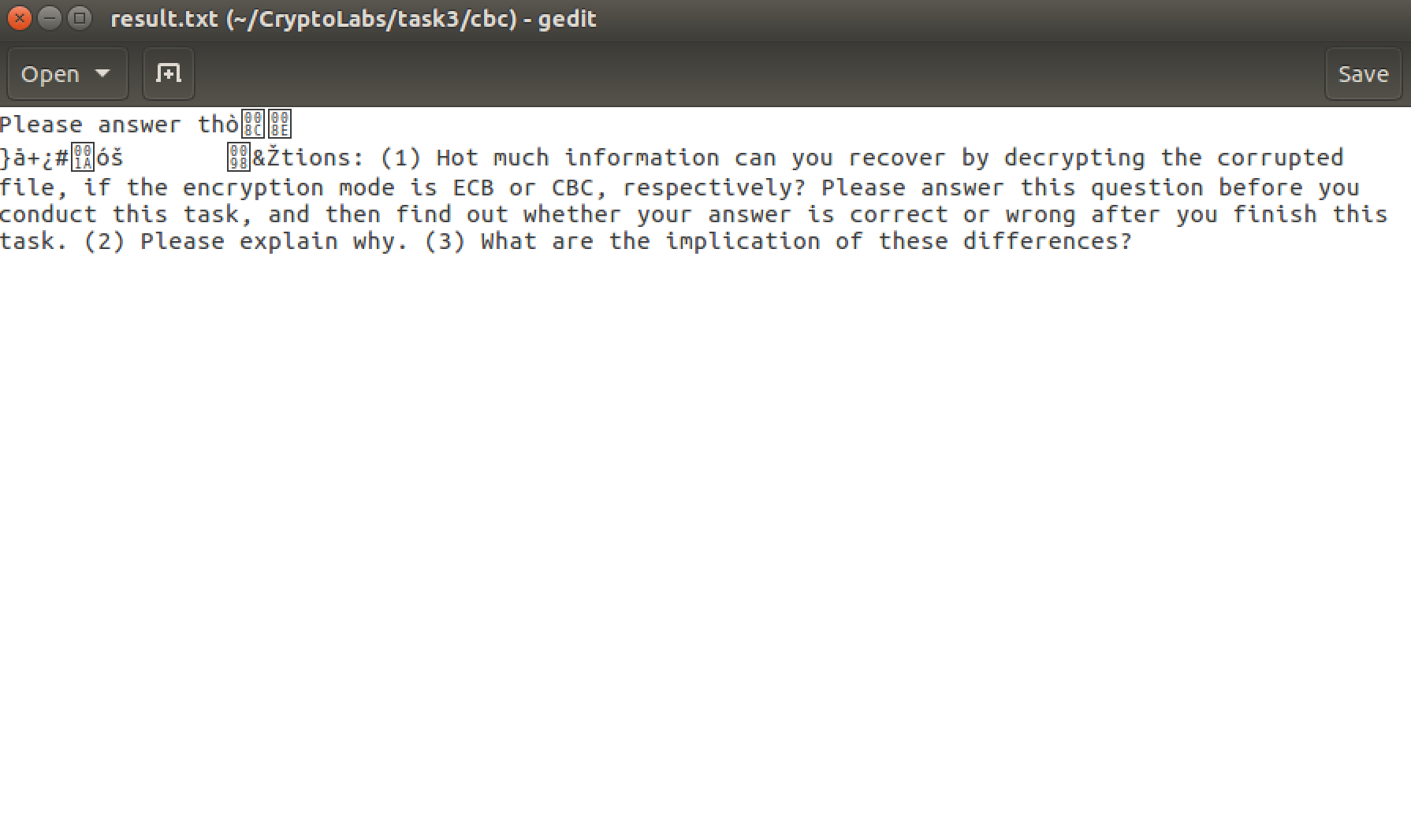
In this part, firstly, I created a 385 bytes file as below:



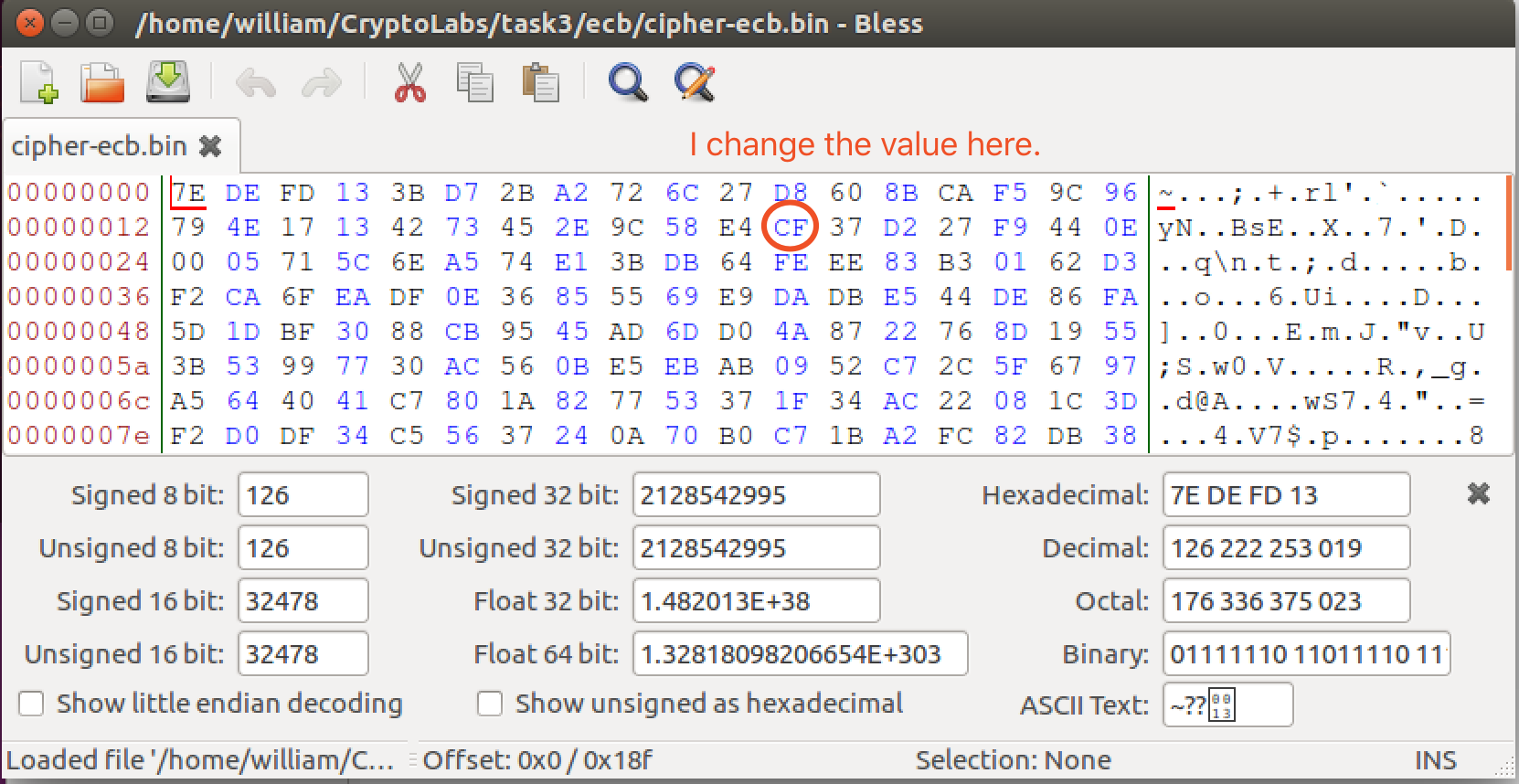
Secondly, I use AES-128 cipher and CBC mode to encrypt this file and I got the encrypted file: cipher-cbc.bin. After that, I use the hex editor, bless, modified the 30th byte in the encrypted file:

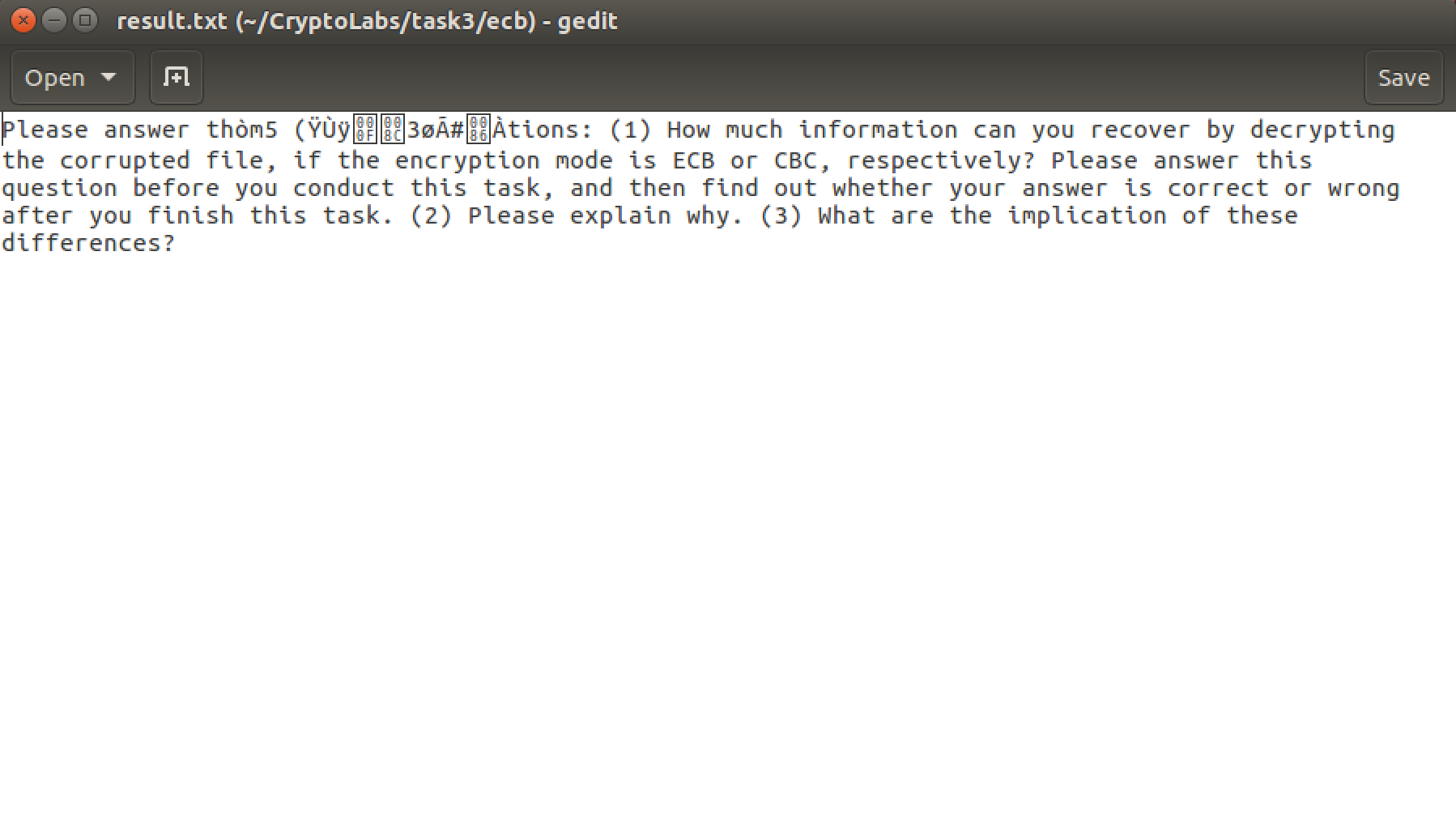


Finally, I decrypted the file and got the corrupted plaintext. It is different with the original one.



Change the mode to ECB and repeat the steps above. I got some pictures as below:

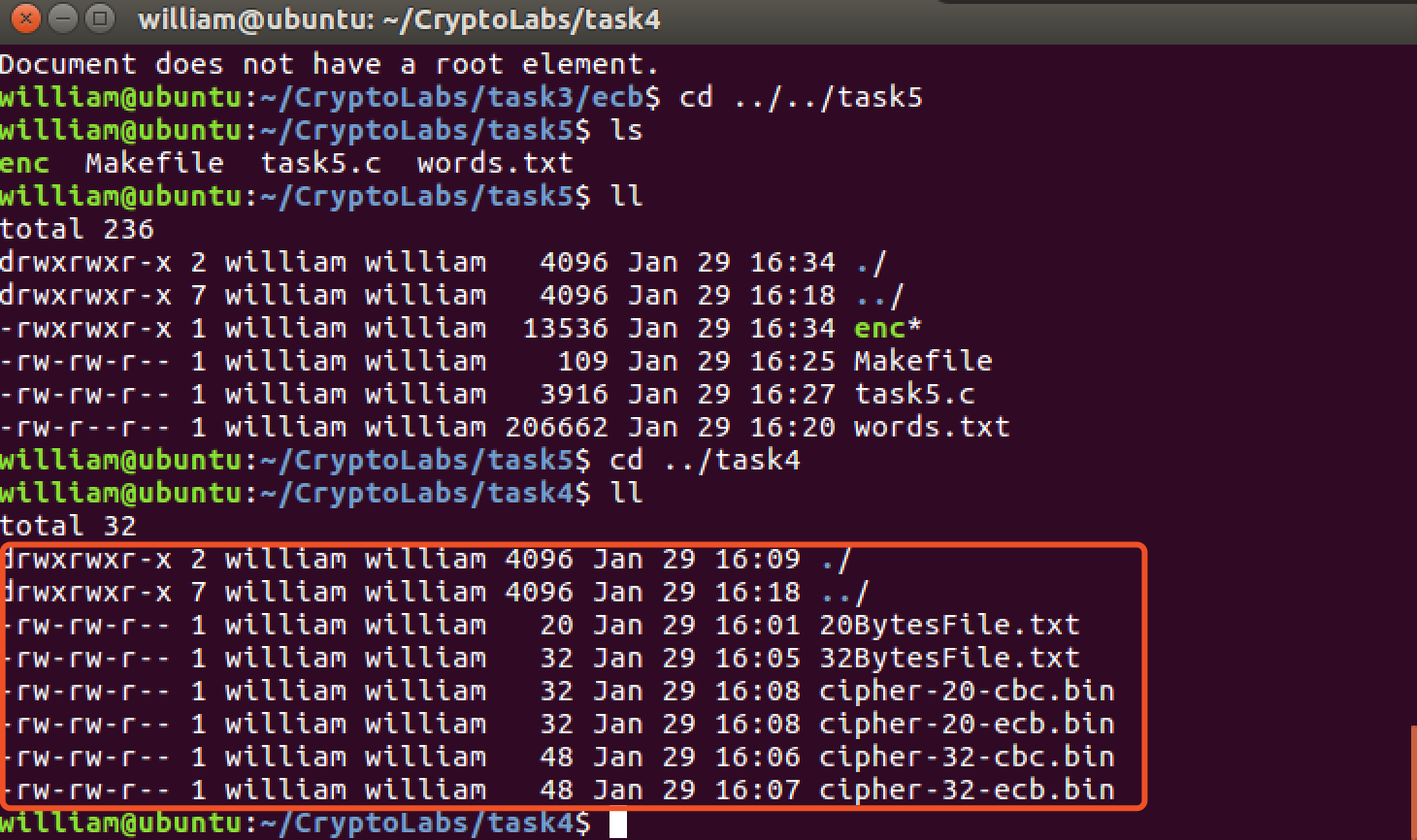




Before the task, I thought that ECB is better to recover the information than CBC. Because it is based on the second task: the ECB picture is easy to see. According to the two modes, I can conclude the questions in the instructions. In the ECB mode, only one block is affected when any problem in a cipher text happens; moreover, each block is decrypted independently. However, the corrupted bit of the 30th byte in cipher text block 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time. And in the CBC mode, there was affect in two blocks.

Task 4: Padding

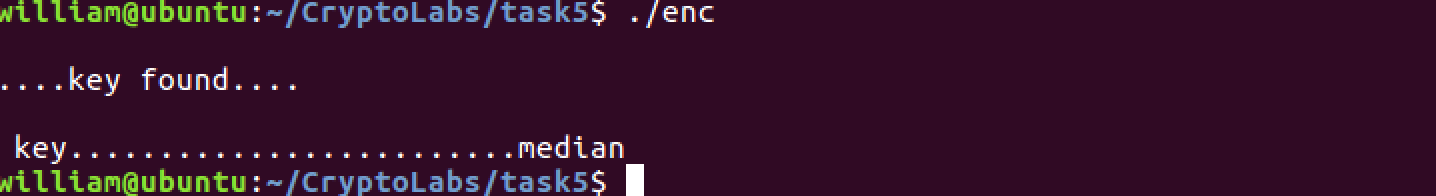
In this task, I created two files: one is 20 bytes size, and the other one is 32 bytes size. And then I used AES 128 to encrypted them in two modes. I got these files as below:



We can see that in the ECB and CBC mode, padding is needed. Because their inputs contain number of blocks, thus padding could ensure that. Block size depends on the algorithm: AES uses 16 byte blocks while 3DES uses 8-byte blocks.

Task 5: Programming using the Crypto Library

In the last part, I used the EVP interface and linked it to the OpenSSL libraries. In the libraries, I can call many encryption algorithms.



Based on this picture, we can get the key from the word list that can generate that cipher text. For this task, the key is “median”.

The code as below:

#include <openssl/conf.h>

#include <openssl/evp.h>

#include <openssl/err.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

int hex2int(char c)

{

int one = (c / 16) - 3;

int two = c % 16;

int r = one\*10 + two;

if(r > 9) r--;

return r;

}

int hex2ascii(char c, char d)

{

int h = hex2int(c) \* 16;

int l = hex2int(d);

return h+l;

}

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

{

int outlen;

int tmplen;

int l;

int i;

int length;

int count;

int found =0;

int k = 0;

unsigned char outbuf[1024];

unsigned char cipher[1024];

unsigned char temp, key[16];

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] = hex2ascii(buf, st[i]);

k++;

}

else

{

buf = st[i];

}

}

cipher[k] = '\0';

in = fopen("/home/william/CryptoLabs/task5/words.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)))

return 0;

if(!EVP\_EncryptFinal\_ex(&ctx, outbuf + outlen, &tmplen))

return 0;

outlen += tmplen;

EVP\_CIPHER\_CTX\_cleanup(&ctx);

count = 0;

for(i = 0; i < 32; i++)

{

if(cipher[i] == outbuf[i])

count++;

}

if(count == 32)

{

printf("\n....key found....\n");

printf("\n key.........................%s",key);

printf("\n");

found = 1;

break;

}

}

fclose(in);

if(found == 0)

printf("\n\n key cannot be found for the above cipher text\n");

return 0;

}