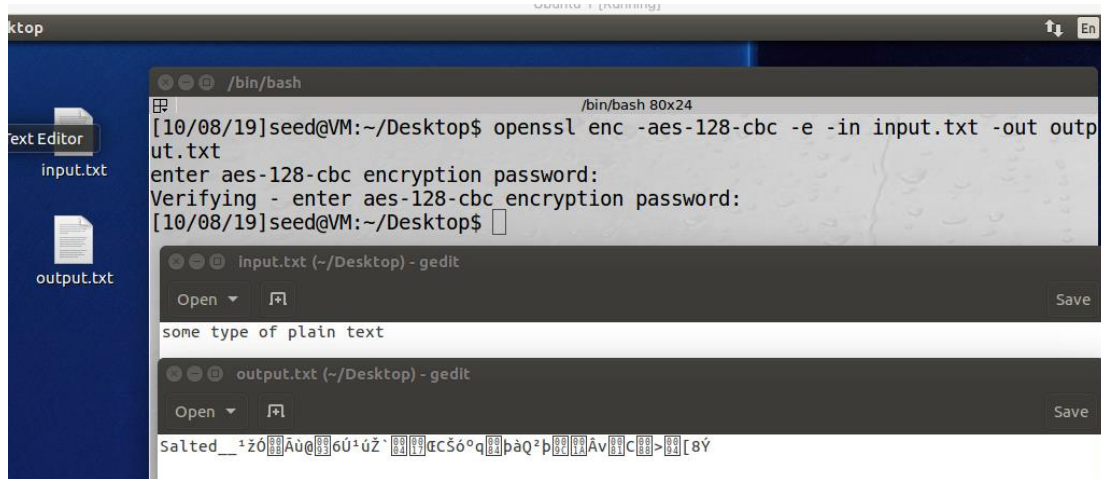


## Homework 3 Report

### 3.1 Task 1: Encryption using different ciphers and modes [6 pts]

Mode 1: cbc

Cypher type 1: -aes-128-cbc



```
[10/08/19]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -e -in input.txt -out output.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[10/08/19]seed@VM:~/Desktop$
```

Input.txt (~/Desktop) - gedit

Open Save

some type of plain text

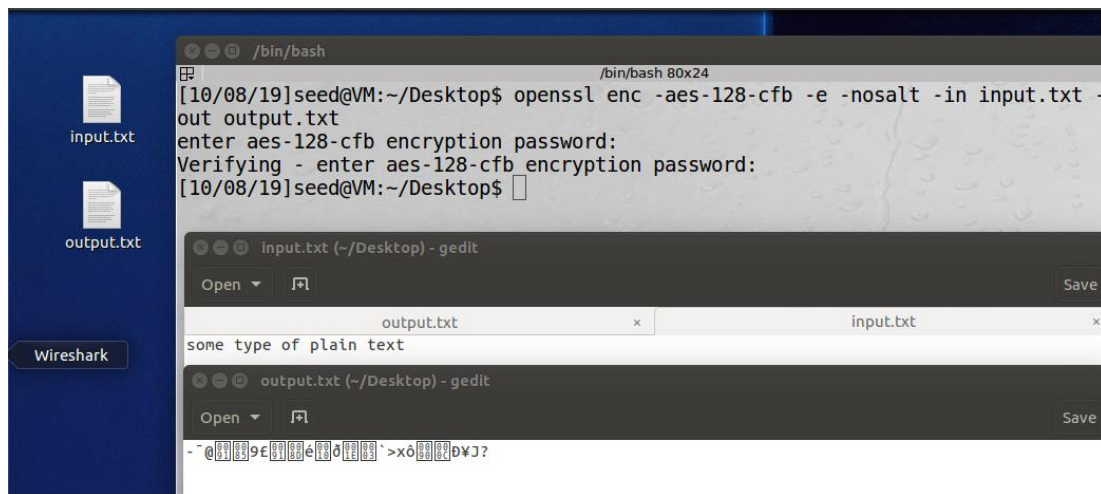
output.txt (~/Desktop) - gedit

Open Save

Salted\_\_z0Äù@6Ú'úž'ccšó°qðàQ²bÄvC>[8Y'

Mode 2: cfb

Cypher type 2: -aes-128-cfb



```
[10/08/19]seed@VM:~/Desktop$ openssl enc -aes-128-cfb -e -nosalt -in input.txt -out output.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[10/08/19]seed@VM:~/Desktop$
```

Input.txt (~/Desktop) - gedit

Open Save

output.txt x input.txt x

some type of plain text

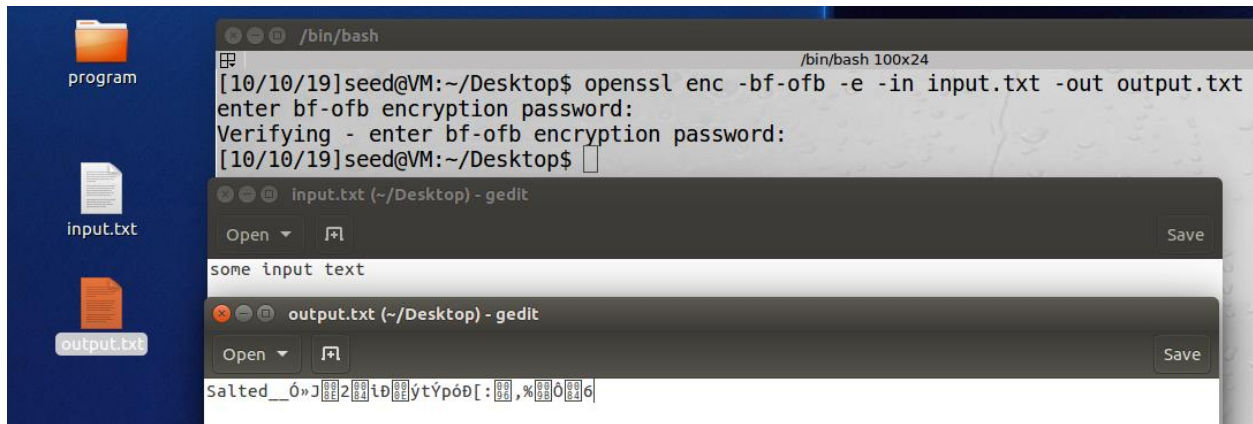
output.txt (~/Desktop) - gedit

Open Save

-@9€99é99'>xó99ðVJ?

Mode 3: ofb

Cypher type 3: bf-ofb



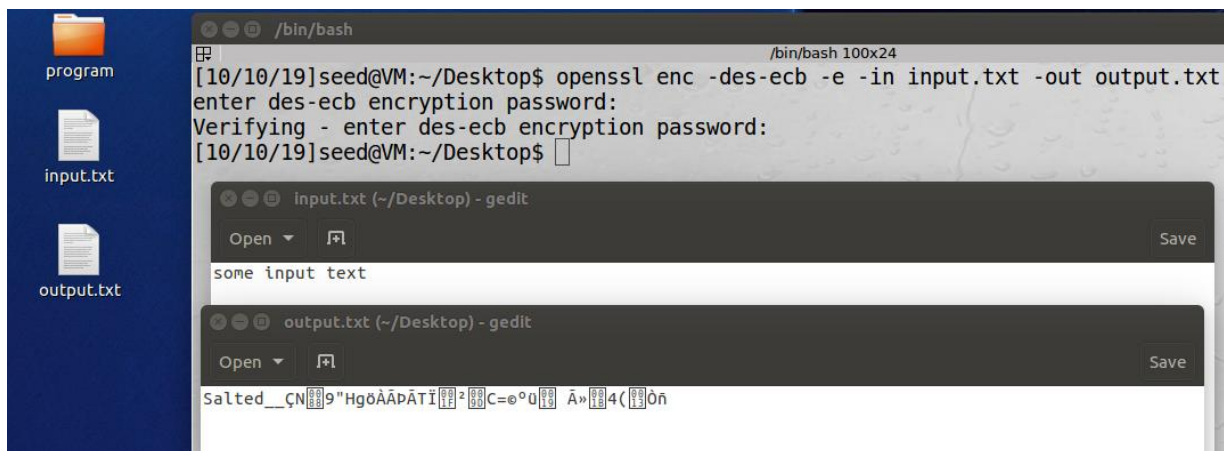
```
/bin/bash
/bin/bash 100x24
[10/10/19]seed@VM:~/Desktop$ openssl enc -bf-ofb -e -in input.txt -out output.txt
enter bf-ofb encryption password:
Verifying - enter bf-ofb encryption password:
[10/10/19]seed@VM:~/Desktop$

Input.txt (~/Desktop) - gedit
Open Save
some input text

output.txt (~/Desktop) - gedit
Open Save
Salted__0»J2i0ytYp0D[:.%,0046
```

Mode 4: ecb

Cypher type 4: -des-ecb



```
/bin/bash
/bin/bash 100x24
[10/10/19]seed@VM:~/Desktop$ openssl enc -des-ecb -e -in input.txt -out output.txt
enter des-ecb encryption password:
Verifying - enter des-ecb encryption password:
[10/10/19]seed@VM:~/Desktop$

input.txt (~/Desktop) - gedit
Open Save
some input text

output.txt (~/Desktop) - gedit
Open Save
Salted__çN9"HgöÄÄpÄTİ²C=°uÄ»4(Öñ
```

### 3.2 Task 2: Encryption Mode – ECB vs. CBC [8 pts]



Encrypting using ECB

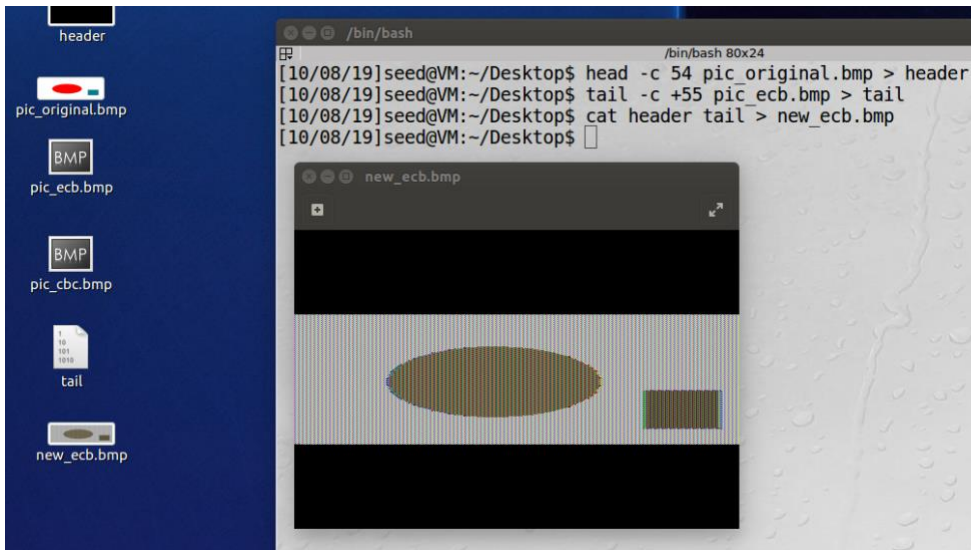
```
/bin/bash
[10/08/19]seed@VM:~/Desktop$ openssl enc -des -a -e -in input.txt -out output.txt
enter des-cbc encryption password:
Verifying - enter des-cbc encryption password:
[10/08/19]seed@VM:~/Desktop$ man enc
[10/08/19]seed@VM:~/Desktop$ openssl enc -des-ecb -e -in pic_original.bmp -out pic_ecb.bmp
enter des-ecb encryption password:
Verifying - enter des-ecb encryption password:
[10/08/19]seed@VM:~/Desktop$ man enc
[10/08/19]seed@VM:~/Desktop$
```

Encrypting using CBC

```
/bin/bash 80x24
[10/08/19]seed@VM:~/Desktop$ openssl enc -des-cbc -e -in pic_original.bmp -out pic_cbc.bmp
enter des-cbc encryption password:
Verifying - enter des-cbc encryption password:
[10/08/19]seed@VM:~/Desktop$
```

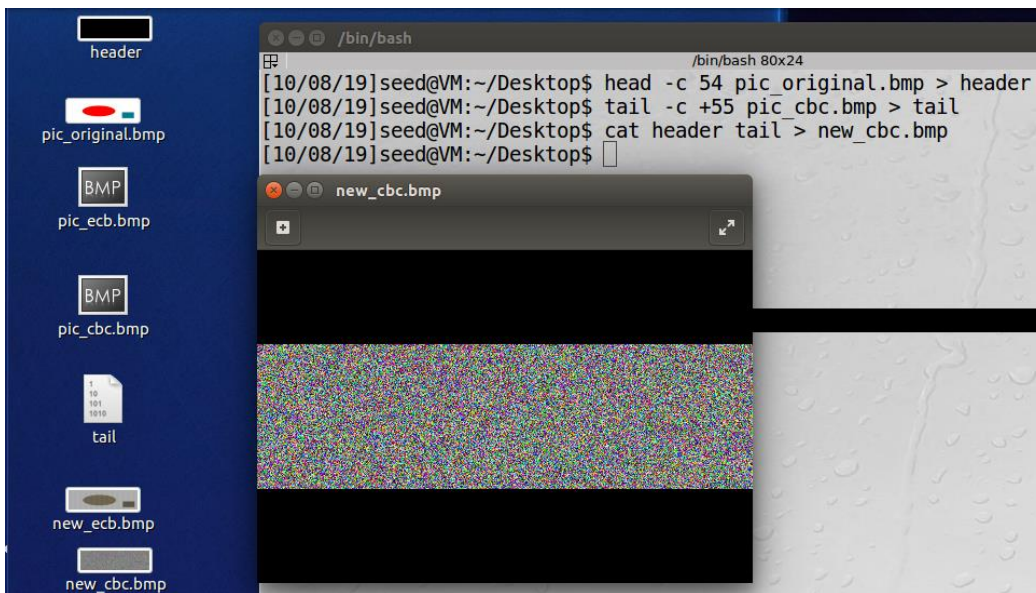
Replacing the headers for ecb encrypted bmp file:

After ebc encryption, the basic shapes within the image are distinguishable but the colors are not.



Replacing headers for the cbc encrypted file

After cbc encryption, there are virtually no distinguishable features of the original .bmp file.





Encrypting a picture of my choice using cbc encryption:

The encryption of my bmp image of choice appeared to have the same effect as the previous experiment. Cbc encryption of an image appears to me a much more affect method than ecb encryption.

```
myopic_cbc.bmp
myopic.bmp
header

/bin/bash
/bin/bash 80x24
[10/08/19]seed@VM:~/Desktop$ openssl enc -des-cbc -e -in mypic.bmp -out mypic_cbc.bmp
enter des-cbc encryption password:
Verifying - enter des-cbc encryption password:
[10/08/19]seed@VM:~/Desktop$ head -c 54 mypic.bmp > header
[10/08/19]seed@VM:~/Desktop$ tail -c +55 mypic_cbc.bmp > tail
[10/08/19]seed@VM:~/Desktop$ cat header tail > mypicnew_cbc.bmp
[10/08/19]seed@VM:~/Desktop$
```

Original

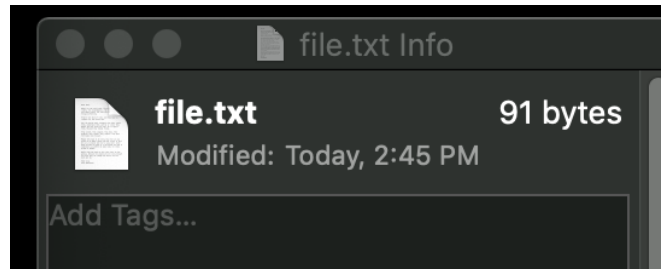


After encryption

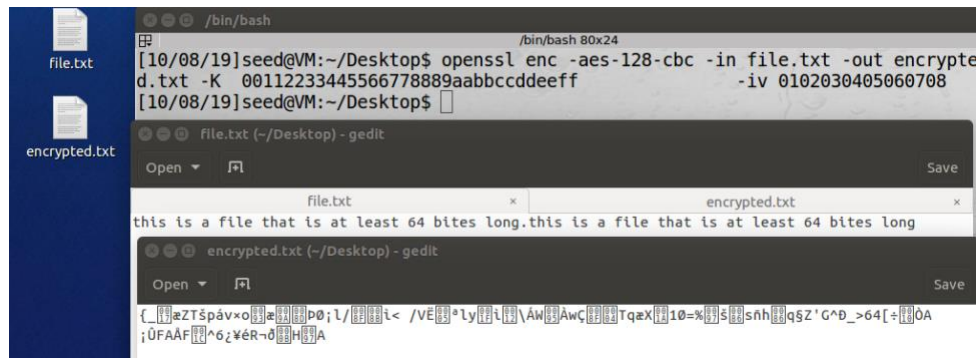


### 3.3 Task 3: Encryption Mode – Corrupted Cipher Text [10 pts]

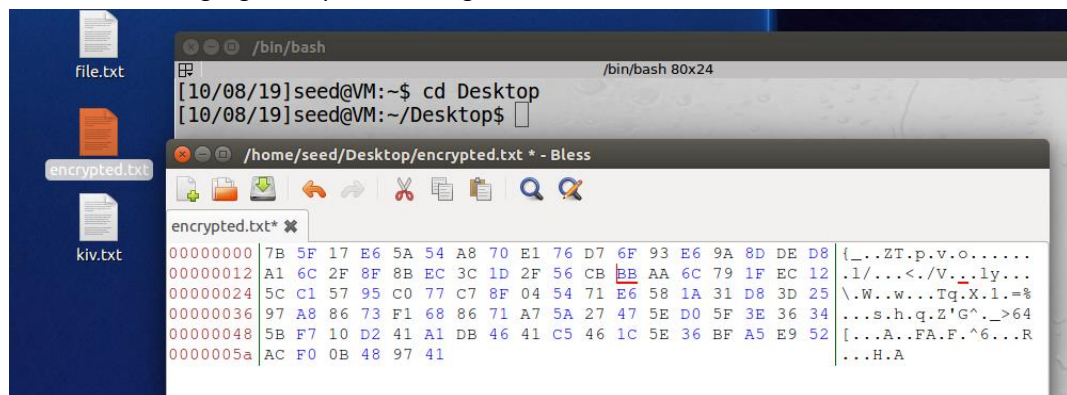
1. Create a text file that is at least 64 bytes long.



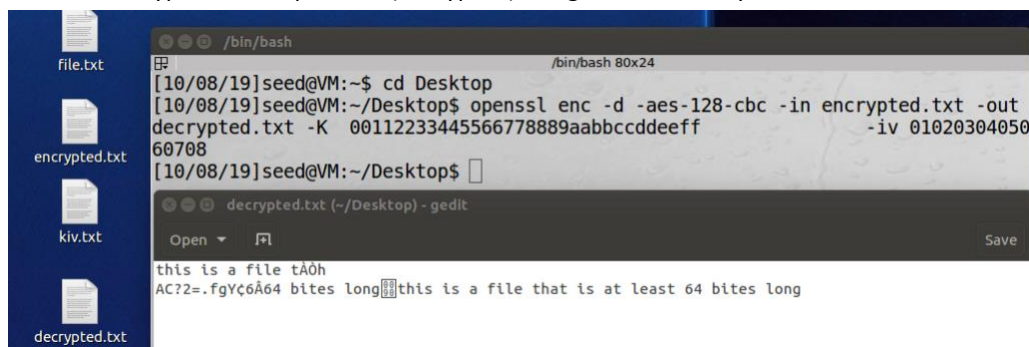
2. Encrypt the file using the AES-128 cipher.



3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this. The highlighted byte was changed from 05 to BB



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

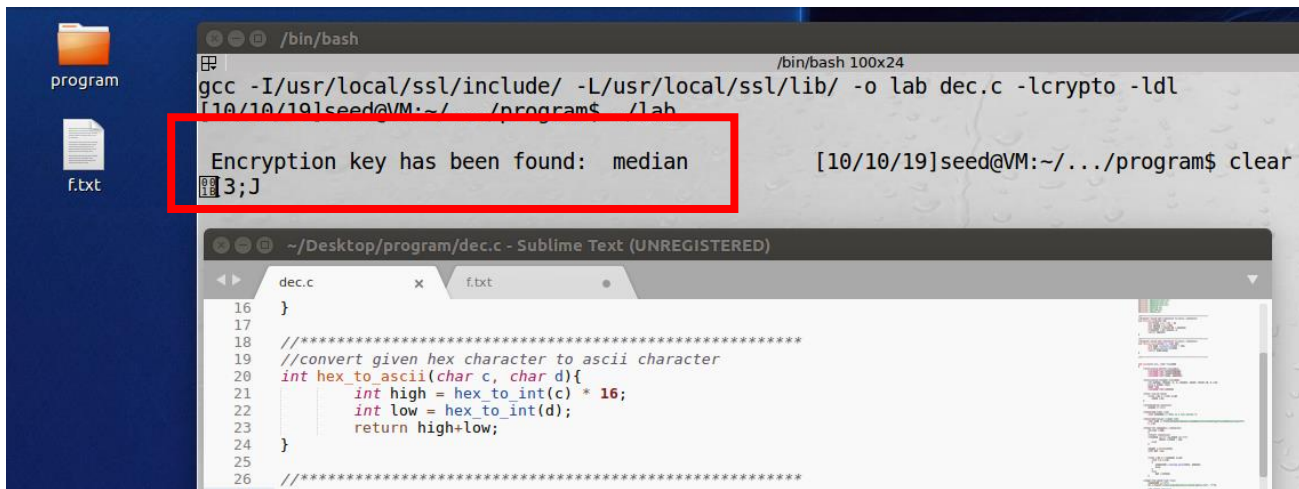


Answer the following questions

- (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task
  - a. ECB: I think that ecb will offer the greatest amount of information recovery based on the amount of information recovered in the previous task.
  - b. CBC: I think that cbc will offer the least amount of information recovered because of how this encryption method performed in the previous exercise.
  - c. CFB: I think that CFB will have the second most amount of information available after corruption of the 30<sup>th</sup> byte. This is simply a guess before testing the encryption and decryption.
  - d. OFB: I think OFB will have the third most information available after corruption of the 30<sup>th</sup> byte. Again, this is simply a guess before testing encryption and decryption using this method.
- (2) Please explain why
  - a. ECB: Earlier assumption was close to correct. Most of the information was recovered with ecb encryption, only 1 block is affected
  - b. CBC: This assumption was close to correct. The error in the corrupted byte affected several of the encrypted blocks, limiting the amount of information recovery
  - c. CFB: This assumption was close to correct. The corruption of the 30<sup>th</sup> byte will affect a reoccurring ratio of the encrypted blocks of the text.
  - d. OFB: Earlier assumption is incorrect. In OFB, only the 30<sup>th</sup> bit is corrupted making this the method allowing for most information recovery.
- (3) What are the implications of these differences?
  - a. ECB: ECB encryption results in the creation several identical cypher blocks which can be recognizable to an attacker and also rearranged without consequence. Because of these vulnerabilities, ecb encryption should be avoided.
  - b. CBC: This method is one of the more secure methods because an error in the encrypted data will propagate through several of the other encrypted blocks, and preventing the recovery of encrypted information.
  - c. CFB: Because an error in a single block will affect the other blocks, this is a much more secure method of encryption compared to OFB, and ECB
  - d. OFB: Because only an extremely limited number of bytes are affected by the corruption of the information encrypted using OFB, it should be avoided, and other methods of encryption should be used to ensure data integrity.

### 3.4 Task 4: Programming using the Crypto Library [15 pts]

Compiling and running the code



The screenshot shows a terminal window with the following commands and output:

```
/bin/bash
gcc -I/usr/local/ssl/include/ -L/usr/local/ssl/lib/ -o lab dec.c -lcrypto -ldl
[10/10/19]seed@VM:~/.../program$ ./lab
Encryption key has been found: median
[10/10/19]seed@VM:~/.../program$ clear
```

The code editor shows the following C code:

```
dec.c
16 }
17
18 //convert given hex character to ascii character
19 int hex_to_ascii(char c, char d){
20     int high = hex_to_int(c) * 16;
21     int low = hex_to_int(d);
22     return high+low;
23 }
24
25
26 //*****
```

Main elements of the Code:

```
int main(int argc, char *argv[])
{
    //initialize buffer variables
    unsigned char outbuf[1024];
    unsigned char cipher[1024];
    unsigned char temp, key[16];

    //initialize integer variabls
    int outlen, tmplen, l, i, length, count, found = 0, k = 0;
    size_t nread, len;
    FILE *in;
    unsigned char iv[17];

    //fill the IV zeros
    for(i = 0; i < 17; i++){
        iv[i] = 0;
    }

    //termination character
    iv[16] = '\0';

    //provided input text
    char intext[] = "This is a top secret.";

    //provided output cupher text
    char st[] = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9";
    i = 0;
```

```
//look for alphabetic characters
while(i < 64)
{
    //shift characters
    if(st[i] >= 'a' && st[i] <= 'z')
        st[i] = st[i] - 32;
    i++;
}

length = strlen(st);
char buf = 0;

//iterate through the cipher text and convert to ascii cahacters
for(i = 0; i < length; i++){
    if(i % 2 != 0)
    {
        cipher[k] = hex_to_ascii(buf, st[i]);
        k++;
    }
    else
        buf = st[i];
}

//open the words text file
cipher[k] = '\0';
in = fopen("/home/seed/Desktop/program/words.txt", "r");

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
```



```

//look for alphabetic characters
while(i < 64)
{
    //shift characters
    if(st[i] >= 'a' && st[i] <= 'z')
    {
        st[i] = st[i] - 32;
        i++;
    }

    length = strlen(st);
    char buf = 0;

    //iterate through the cipher text and convert to ascii cahracters
    for(i = 0; i < length; i++){
        if(i % 2 != 0)
        {
            cipher[k] = hex_to_ascii(buf, st[i]);
            k++;
        }
        else
            buf = st[i];
    }

    //open the words text file
    cipher[k] = '\0';
    in = fopen("/home/seed/Desktop/program/words.txt", "r");

    EVP_CIPHER_CTX ctx;
    EVP_CIPHER_CTX_init(&ctx);

```