Al Imam Mohammad ibn Saud University

College of Computer and Information Sciences

Computer Science Department

First semester of 1444

CS 334: Information Security

Project: Secret-Key Encryption Lab

Team Members:

Student Name:	Academic Number:
Aljowharah Aldawood	
Reem Alothman	
Teef Mohammed Aloshan	

Instructor:

Tahani Albalawi

Section:

371

Due Date:

9/11/2022

Table of Contents:

Task 2 …………………………………………………… 3

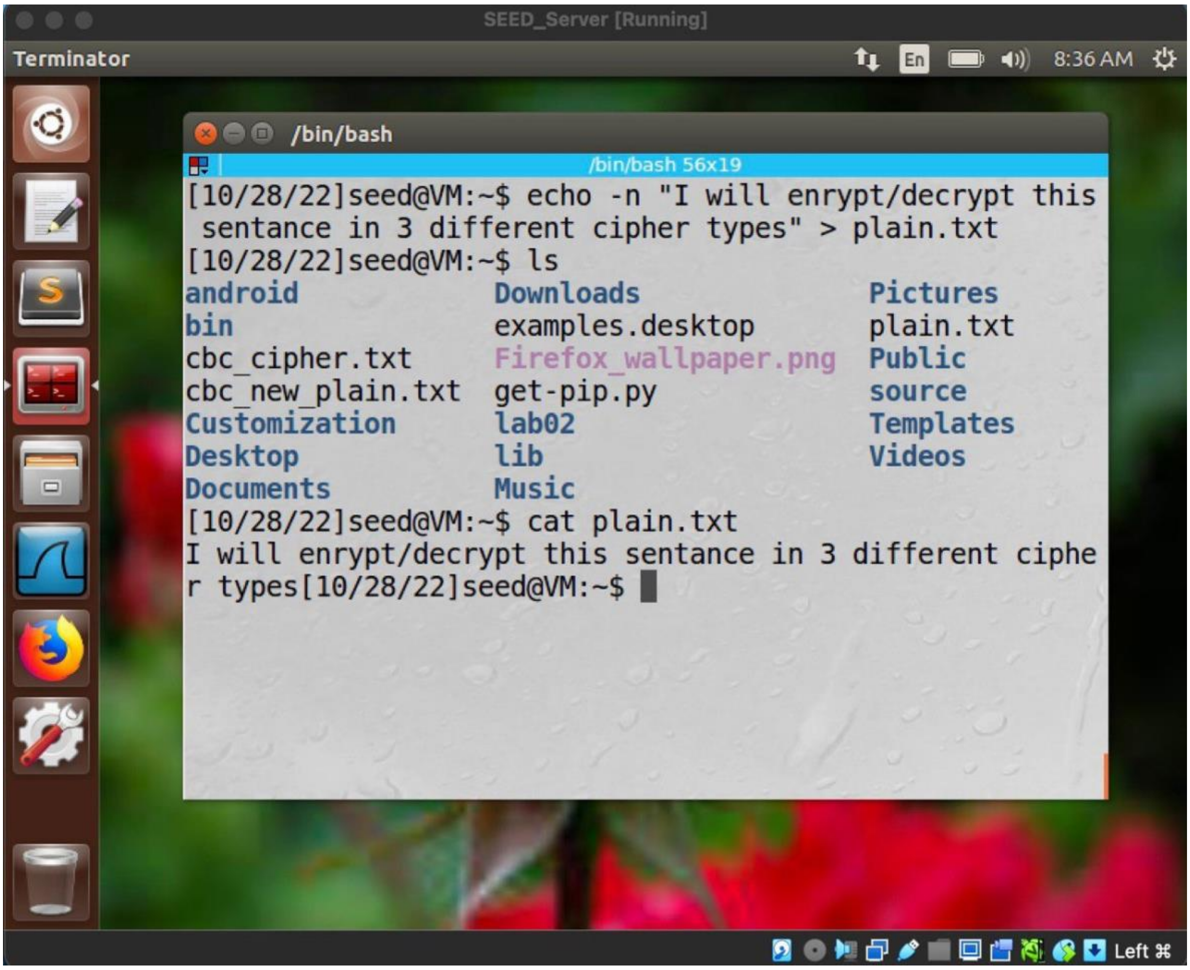
Task 3 …………………………………………………... 13

Task 7 …………………………………………………..… 20

Task 2 : Encryption using different cipher and modes:

Before ciphering the sentence “I will encrypt/decrypt this sentance in 3 different cipher types” has been entered. Each type of the 3 types will be described and explained in detail in this document. The 3 types that have been used in this lab are:

1- aes-128-cbc.
2- bf-cbc. (blow-fish cipher).
3- aes-128-cfb.



1- aes-128-cbc:

As shown in the first figure (in red) is the length of the sentence that we will use before ciphering is 63 letters. This cipher type requires a key (-K) and an initial value (-iv).

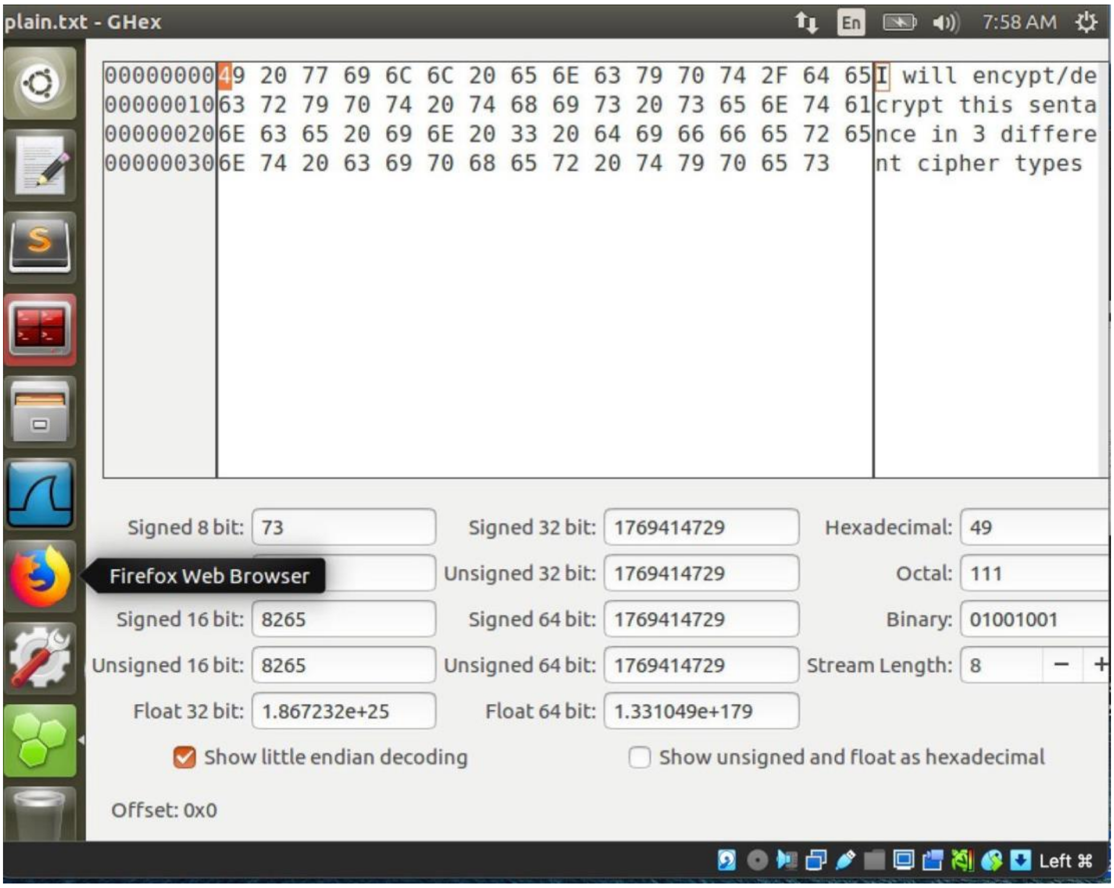
We have written the commands to cipher the sentence where we specified the cipher type (-aes-128-cbc) where -in (input) is the sentence and -out (output) is the sentence ciphered adding the key (-K) and the initial value (-iv).

The second figure shows the hex value of the original sentence before ciphering using (ghex) command.

Figure 1:

```
[10/28/22]seed@VM:~$ ls -l plain.txt
-rw-rw-r-- 1 seed seed 63 Oct 28 07:52 plain.txt
[10/28/22]seed@VM:~$ openssl enc -aes-128-cbc -e -in pla
in.txt -out cbc_cipher.txt -K 00112233445566778889aabbcc
ddeeff -iv 0102030405060708
```

Figure 2:



As seen here in the figure the sentence has been ciphered using the ciphering commands and the length of the ciphered sentence have changed to 64 letters.

The second figure shows the hex value of the ciphered sentence using (ghex) command.

Figure 1:

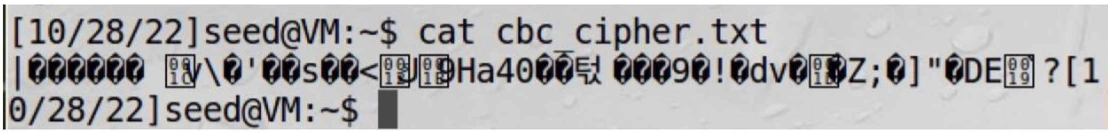


Figure 2:

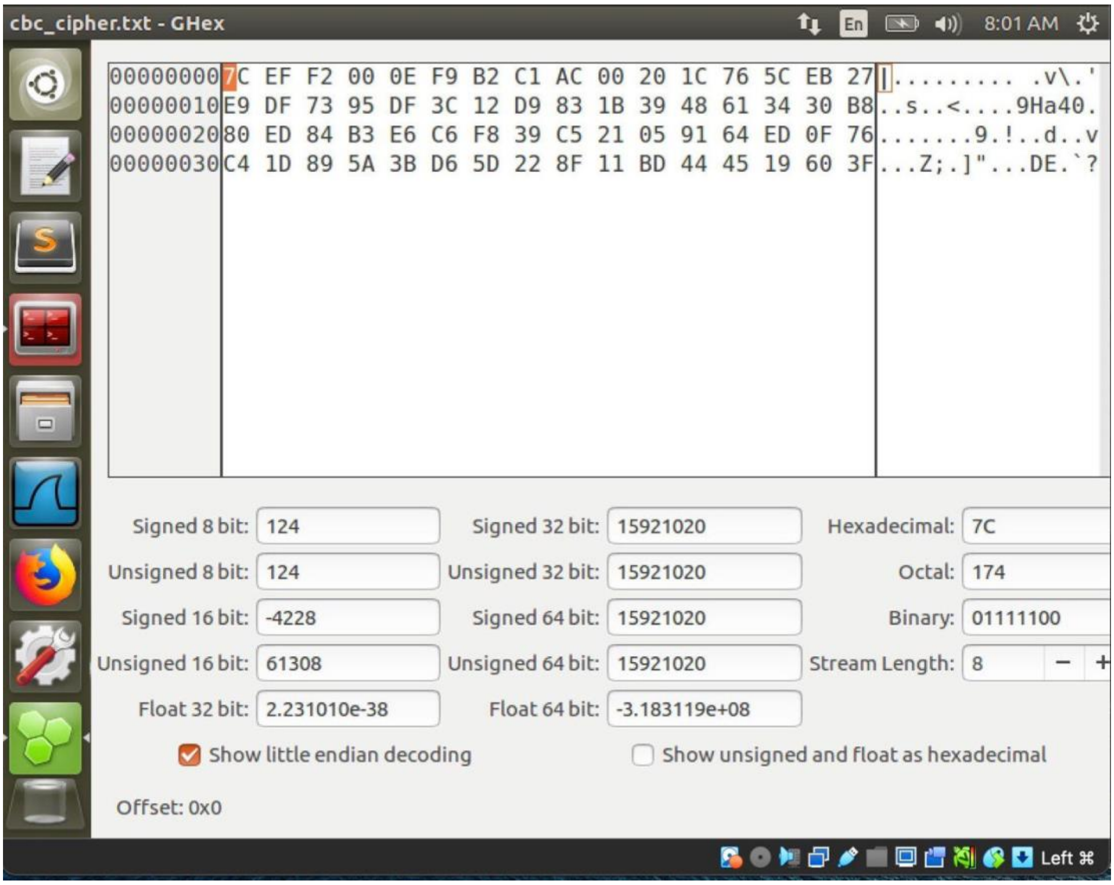
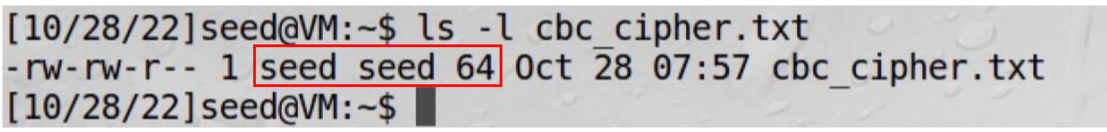


Figure 3:



To check and verify the ciphering and to make sure that it have been done correctly we have deciphered the sentence (figure 1) and checked on the length, the length of the deciphered sentence (in red) matched the original length of the original sentence (figure 2).

Original 63 letters = Deciphered 63 letters.

Figure 1:

```
[10/28/22]seed@VM:~$ openssl enc -aes-128-cbc -d -in cbc
_cipher.txt -out cbc_new_plain.txt -K 001122334455667788
89aabbccddeeff -iv 0102030405060708
```

Figure 2:

```
[10/28/22]seed@VM:~$ ls -l cbc_new_plain.txt
-rw-rw-r-- 1 seed seed 63 Oct 28 08:03 cbc_new_plain.txt
[10/28/22]seed@VM:~$
```

2- bf-cbc (blow-fish cipher):

The first steps has been repeated where the same sentence has been used again. we checked for the length shown in the first figure. This cipher type requires a key (-K) and an initial value (-iv).

We have written the commands to cipher the sentence where we specified the cipher type (-bf-cbc) and where -in (input) is the sentence and -out (output) is the sentence ciphered adding the key (-K) and the initial value (-iv) (figure 1).

The second figure shows the hex value of the original sentence using (ghex) command (figure 2).

Figure 1:

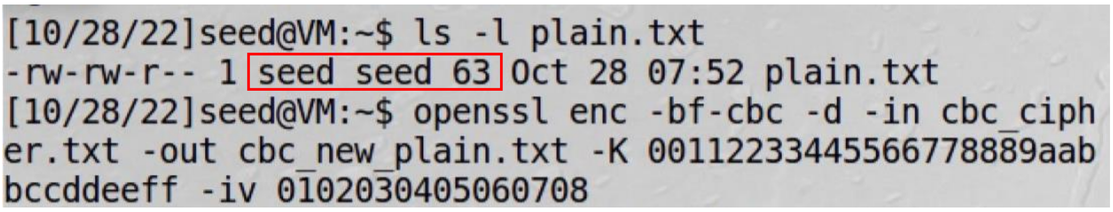
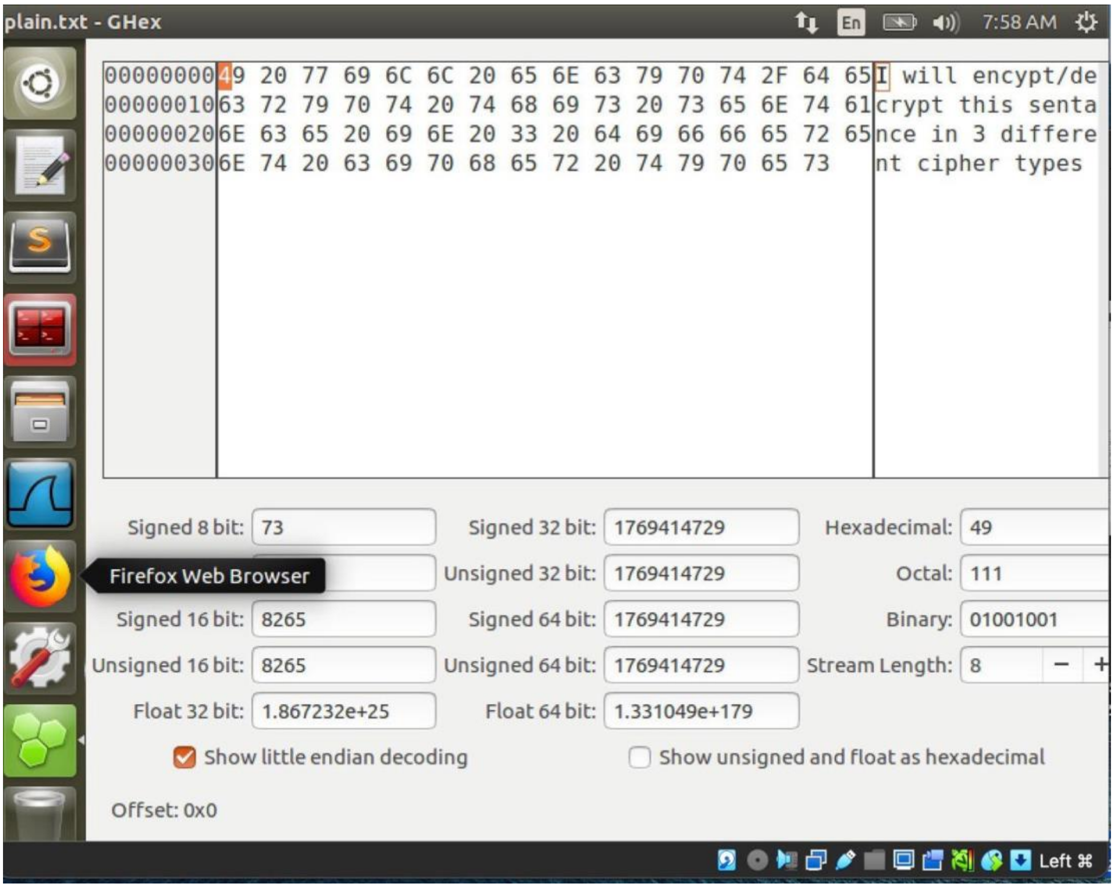


Figure 2:



As seen here in the figure the sentence has been ciphered using the ciphering commands (figure 1) and the length of the ciphered sentence have changed to 64 letters (in red) (figure 3).

The second figure shows the hex value of the ciphered sentence using (ghex) command (figure 2).

Figure 1:

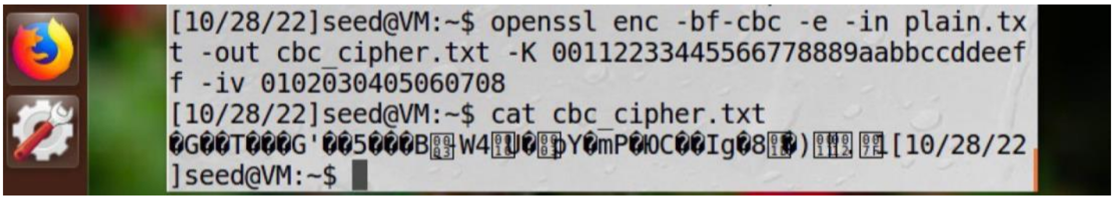


Figure 2:

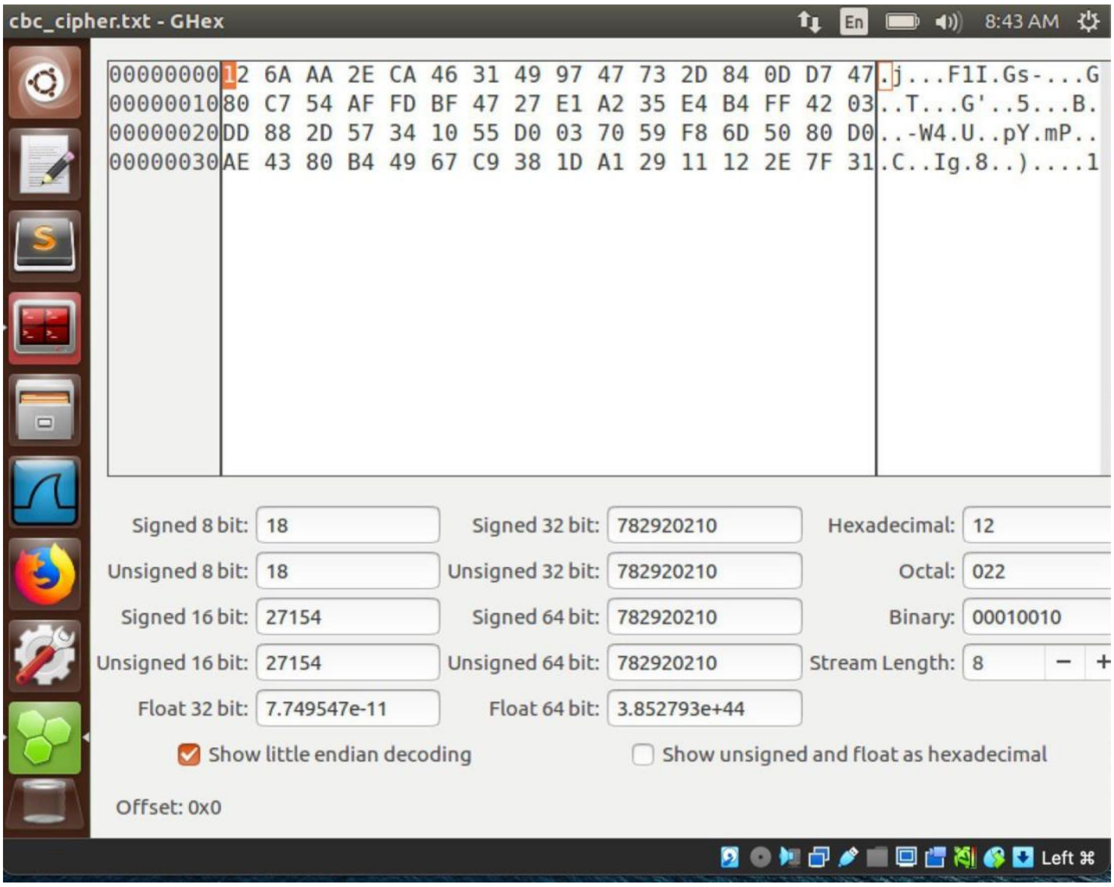
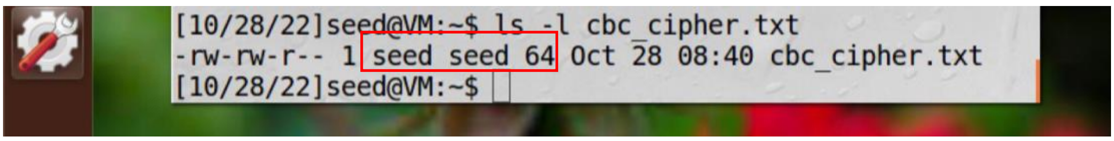


Figure 3:



To check the ciphering and to make sure that it have been done correctly we have deciphered the sentence (figure 1) and checked on the length, the length of the deciphered sentence (in red) matched the original length of the original sentence (figure 2).

Original 63 letters = Deciphered 63 letters.

Figure 1:

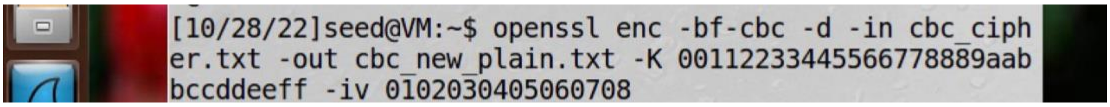
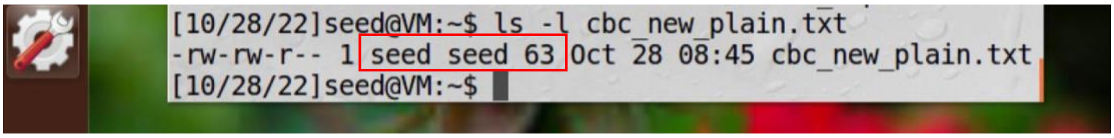


Figure 2:



3- aes-128-cfb:

The first steps has been repeated where the same sentence has been used again. we checked for the length shown in the first figure. This cipher type requires a key (-K) and an initial value (-iv).

We have written the commands to cipher the sentence where we specified the cipher type (-aes-128-cfb) and where -in (input) is the sentence and -out (output) is the sentence ciphered adding the key (-K) and the initial value (-iv) (figure 1).

The second figure shows the hex value of the original sentence using (ghex) command (figure 2).

Figure 1:

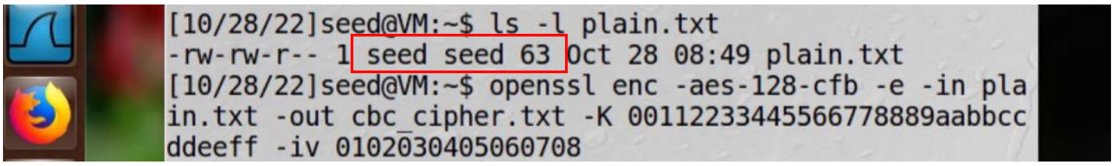
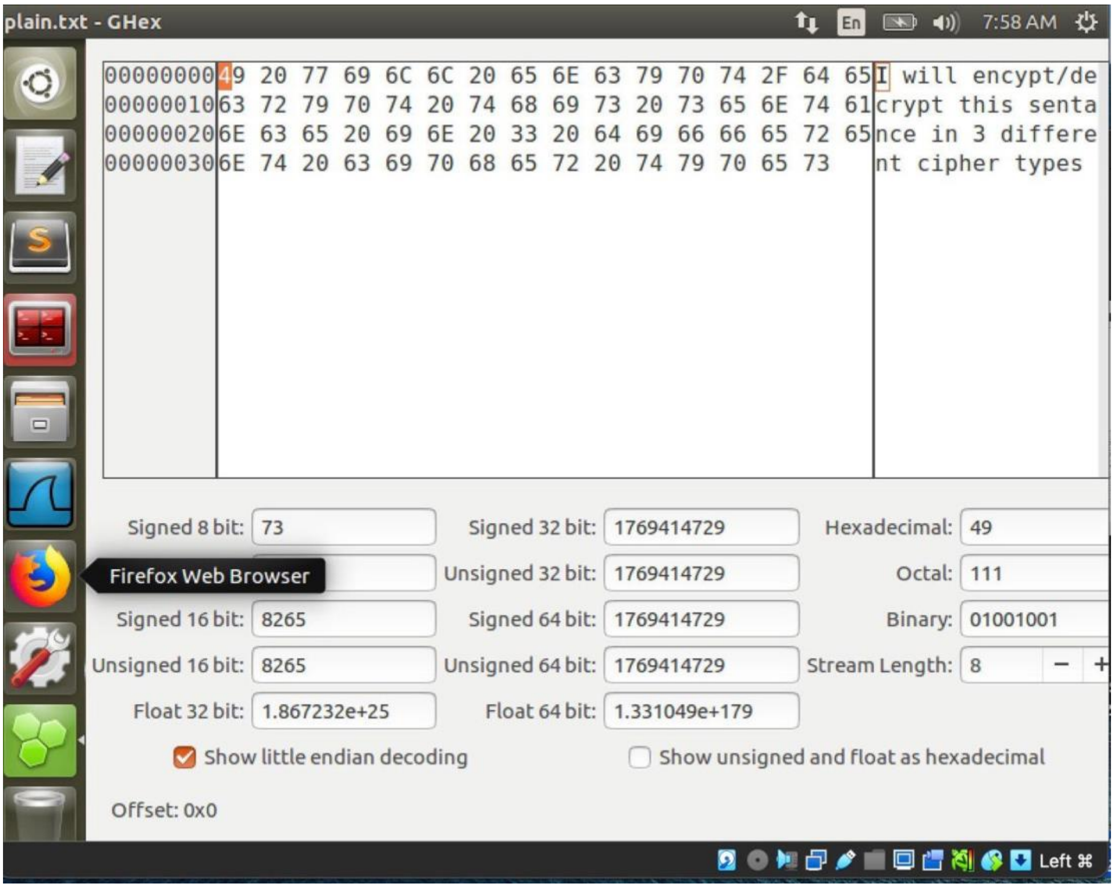


Figure 2:



As seen here in the figure the sentence has been ciphered using the ciphering commands (figure 1) and the length of the ciphered sentence have not changed (in red) (figure 3).

The second figure shows the hex value of the ciphered sentence using (ghex) command (figure 2).

Figure 1:

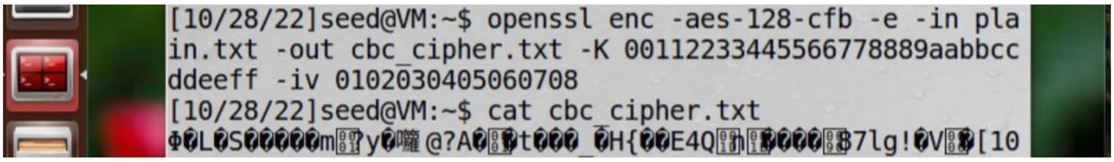


Figure 2:

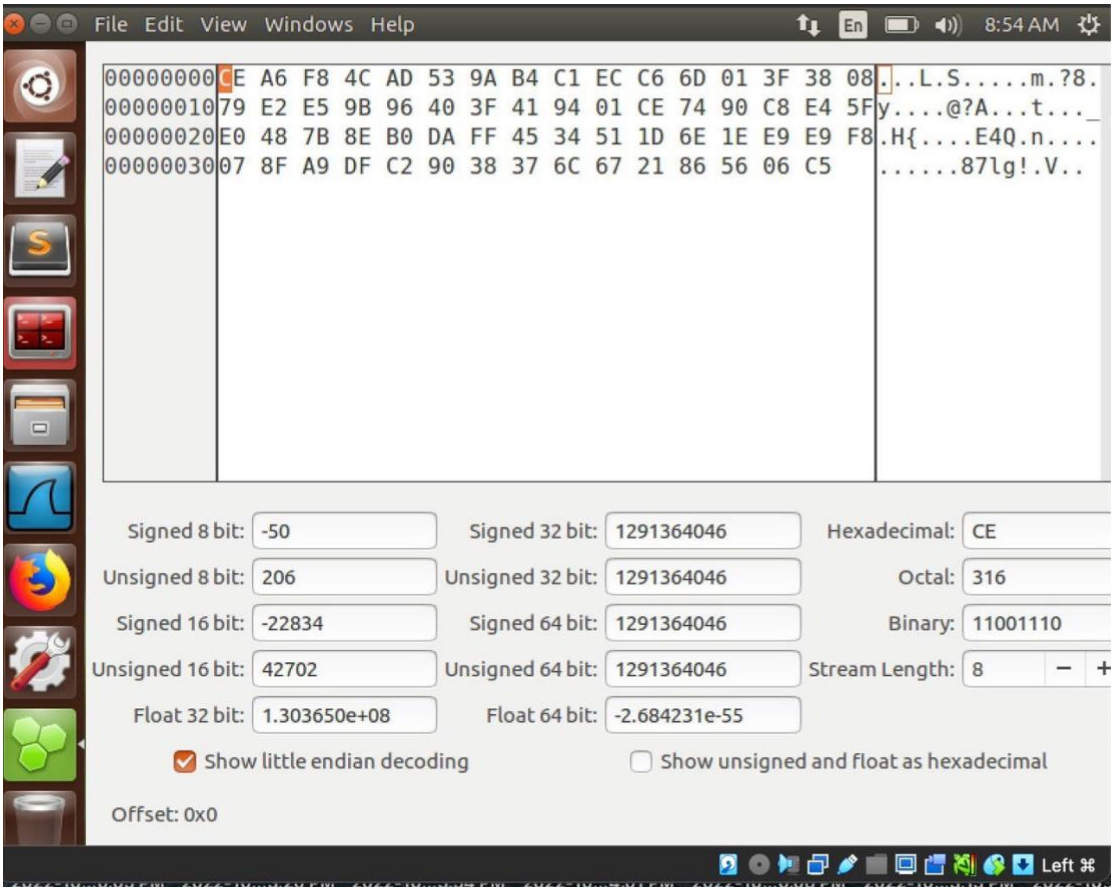
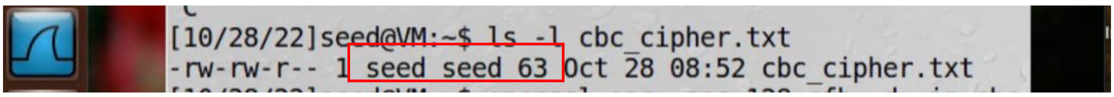


Figure 3:



To check the ciphering and to make sure that it have been done correctly we have deciphered the sentence (figure 1) and checked on the length, the length of the deciphered sentence (in red) matched the original length of the original sentence (figure 2).

Original 63 letters = Deciphered 63 letters.

Figure 1:

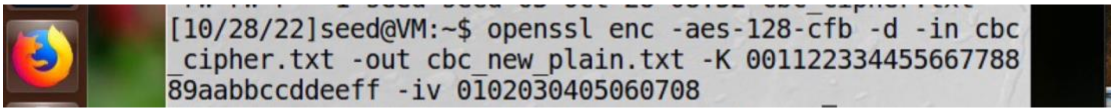
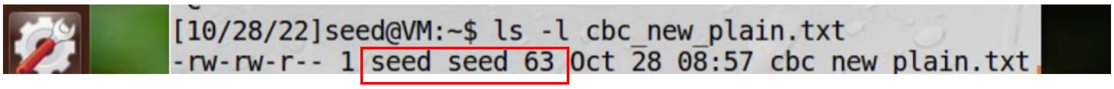


Figure 2:



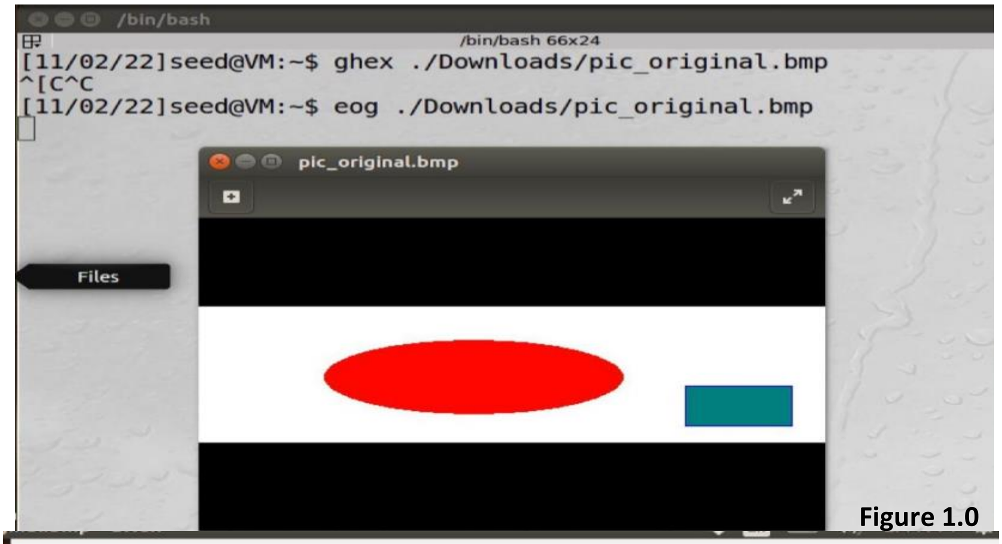
Task 3: Encryptions Mode -ECB VS. CBC:

In this task we have an image that we want to encrypt using ECB (Electronic Code Book) and CBC (Cipher Blocks Chaining) modes. On each mode, we have to follow certain steps. the first one is encrypting the header information c is consists of the first 54 bytes of the image. Then we encrypt the body which consist of +55 bytes. Following the step to combine them together.

Before encrypting the image in a bmp form in two different modes. Each of these two modes will be described and explained in detail. The two modes as mentioned above are:

1. ECB (Electronic Code Book).
2. CBC (Cipher Blocks Chaining).

In figure (1.0) is the image in bmp form before encrypting it, used ego command to view the image. As well, in figure (1.1) is the hex for the original image before encrypting it using the (ghex) command.



CBC –

Figure 1.0

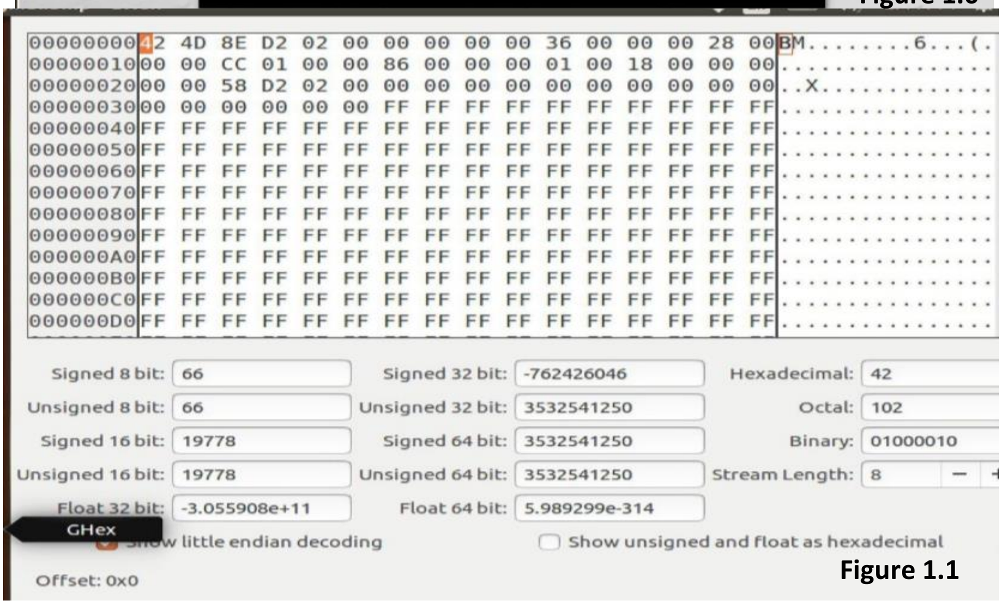


Figure 1.1

1. CBC - Cipher Block Chaining

For CBC, step one we encrypted the image as shown in figure (2.1) using the specified cipher type aes-128-cbc. Where we entered the commands to cipher only the header which consists of the first 54 bytes. Where (-in) is the original image in bmp form is the input, and out is the output of the image after encrypting the header only. Which also needs the key (-k) and the initial value (-iv).

In figure (2.0) where we entered the command to encrypt the header.

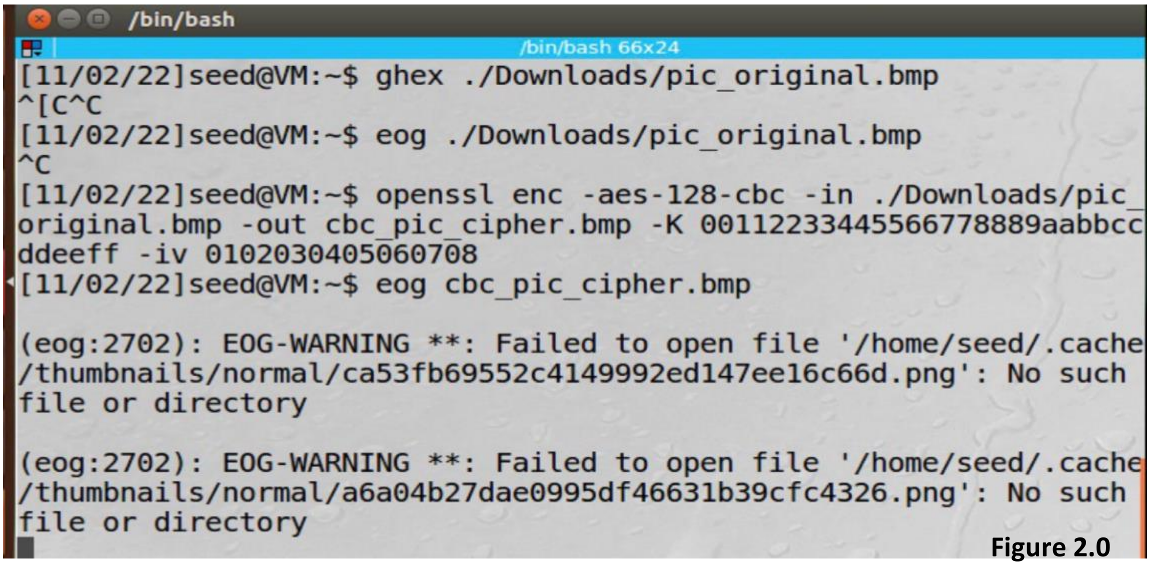


Figure 2.0

(ghex) command used to output the hexadecimal after the whole image is ciphered. shown in the following figure (2.1)

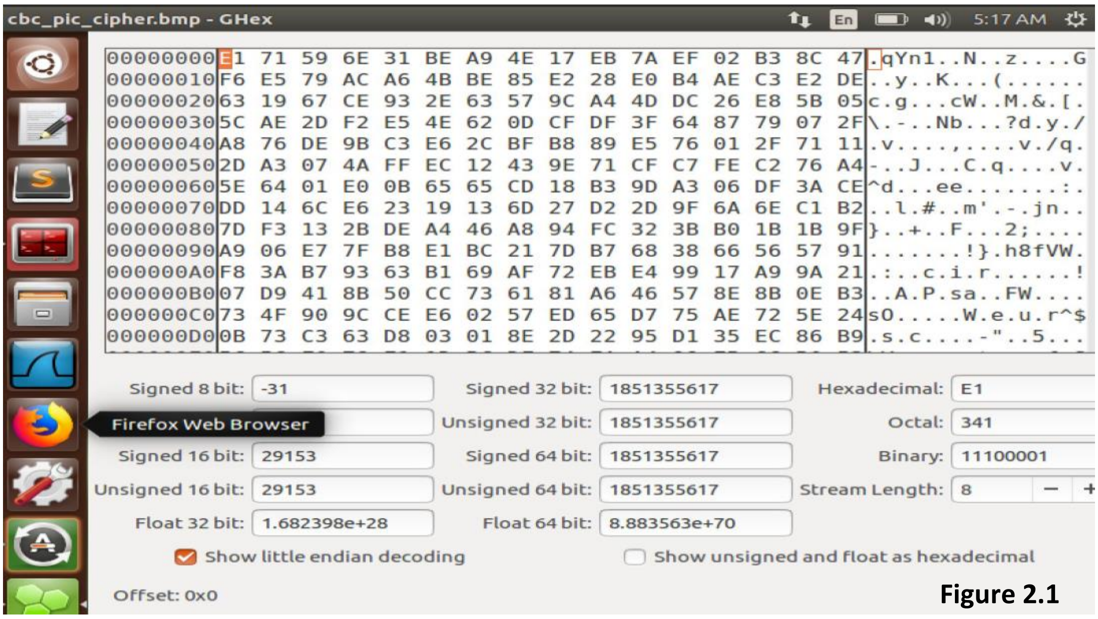


Figure 2.1

in figure (2.2) is after the header of 54 bytes only have been encrypted.

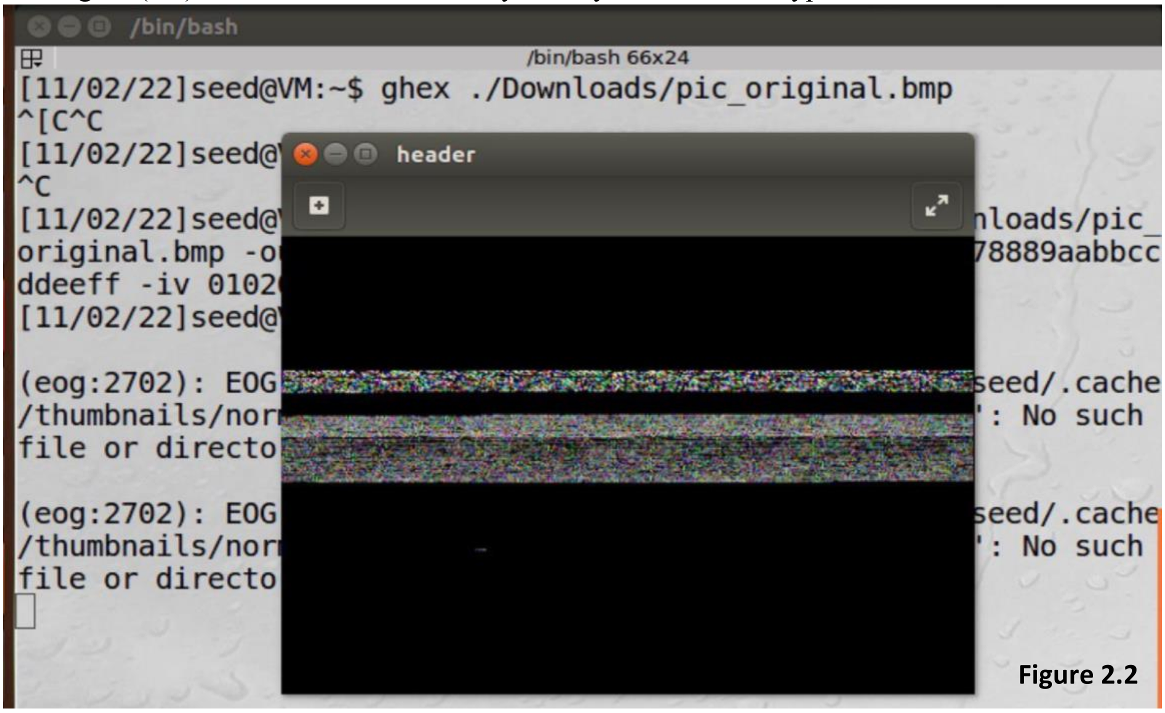


Figure 2.2

Step two, we redefined the encrypted header of 54 bytes as the original image. in order for it to encrypt the body of +55 bytes of the original image. which the first 54 bytes of the original image is already encrypted. In this case, after encrypting the body of +55 bytes it would result as the whole image being ciphered.

We used the following command to redefine the header as though it's the original image. As shown in figure (2.3) in red.

- Head -c 54 ./Downloads/ pic_original.bmp > header

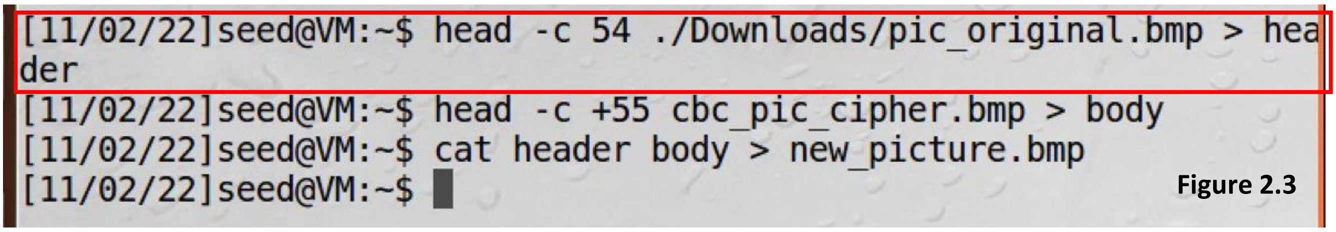


Figure 2.3

Continuing on step 2 is where we encrypt the body of +55 bytes. As shown in figure (2.4) for the whole image being ciphered. And that means we have reached the goal that we wanted to achieve for CBC mode.

In order to view the image, use eog image viewer.

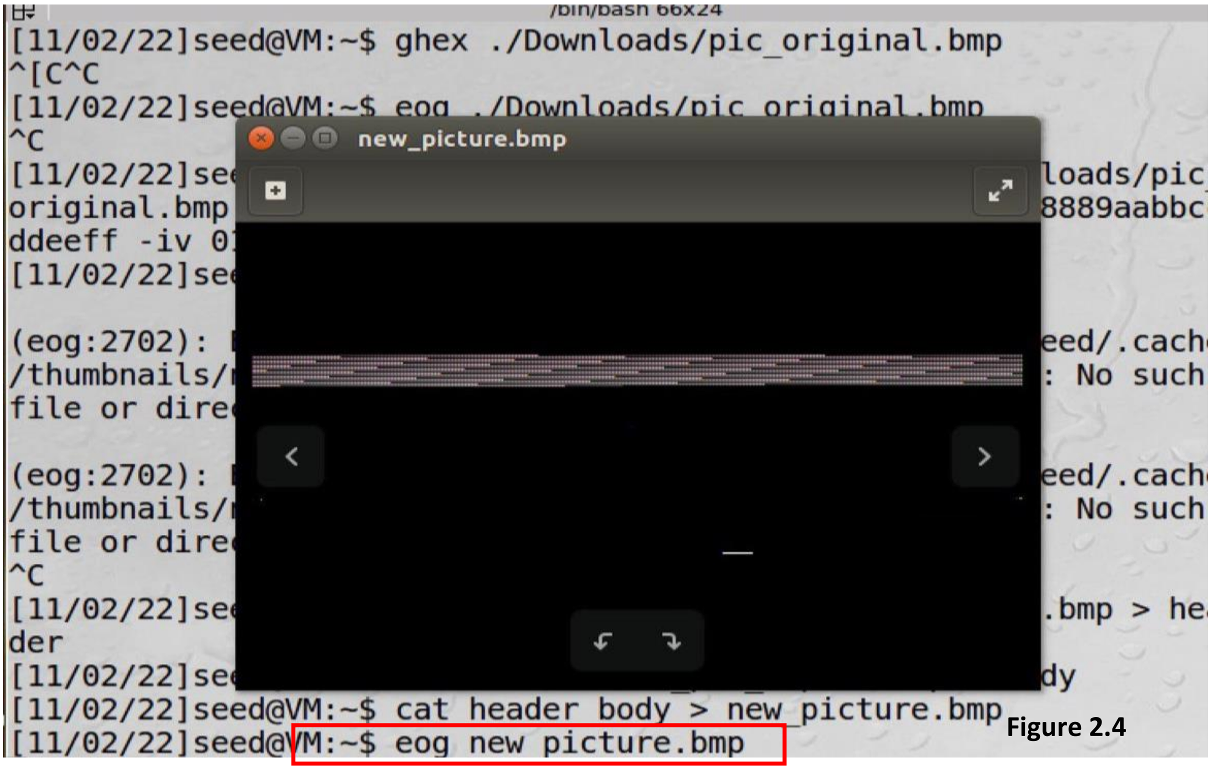


Figure 2.4

In following figure (2.5) after the ghex command. If you noticed that the header of the first 54 bytes is aligned with the header of the original image after encryption and defined it as new_picture.

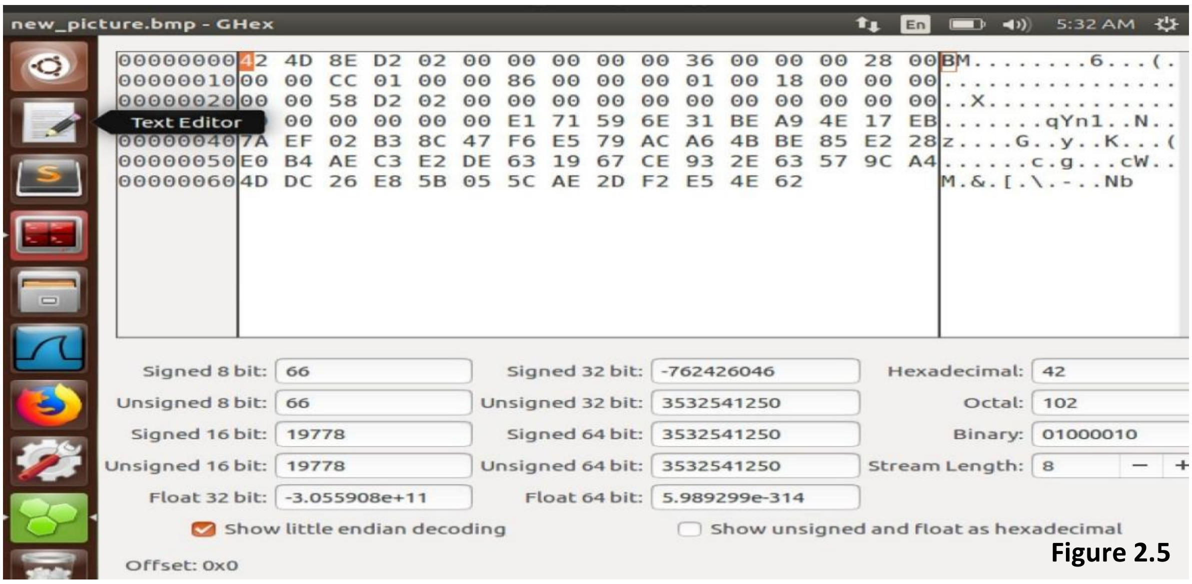


Figure 2.5

2. ECB - Electronic Code Book

For ECB, we repeat the same process with a different cipher type. Step one we encrypted the image as shown in figure (3.0) using the specified cipher type aes-128-ecb. Then we entered the commands to encrypt only the header which consists of the first 54 bytes. Where (-in) is the input consists of the original image in a bmp form, and (-out) is the output of the image after encrypting the header only. Which requires only the key (-k).

In figure (3.0) where we entered the command to encrypt the header. In figure (3.1) is after the header of the first 54 bytes have been encrypted.

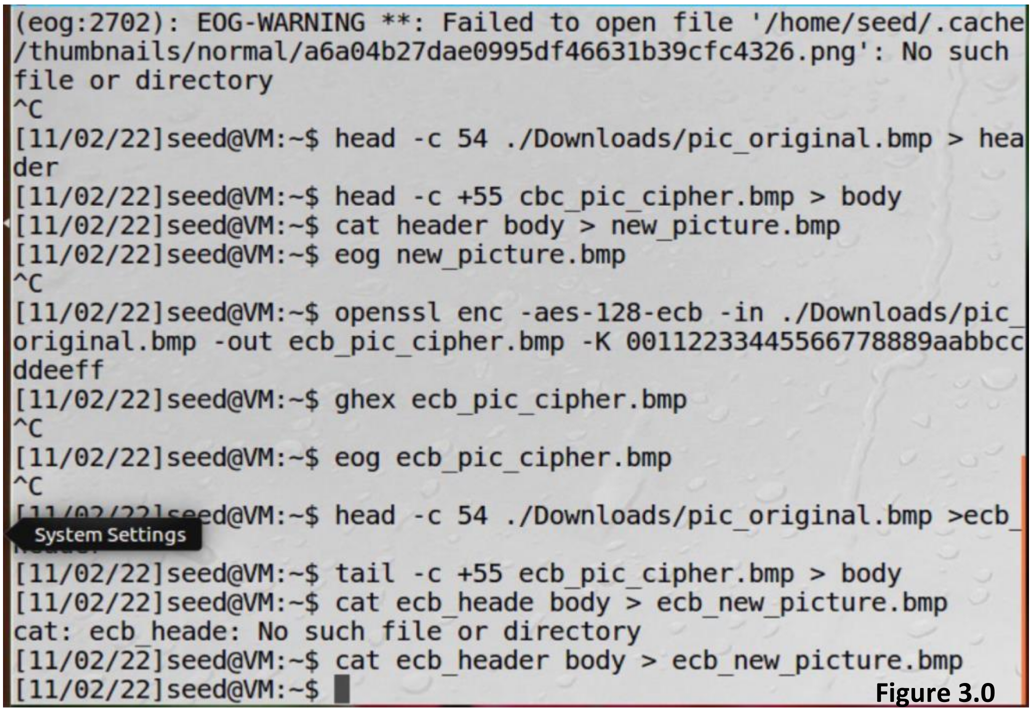


Figure 3.0

After we wrote the following command shown in figure (3.1) in red box, to encrypt it resulted differently, that it is not able to load the image shown in figure (3.1).

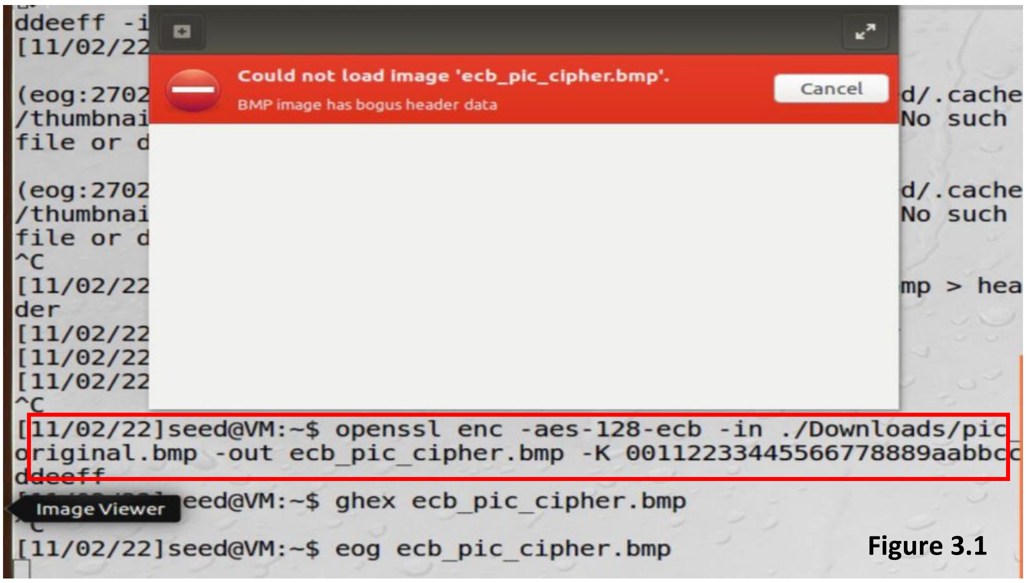


Figure 3.1

On step 2, we encrypt the body of +55 bytes. As shown in figure (3.2) for the whole image being ciphered. And that means we have reached the goal that we wanted to achieve for CBC mode. We used the eog to view the image.

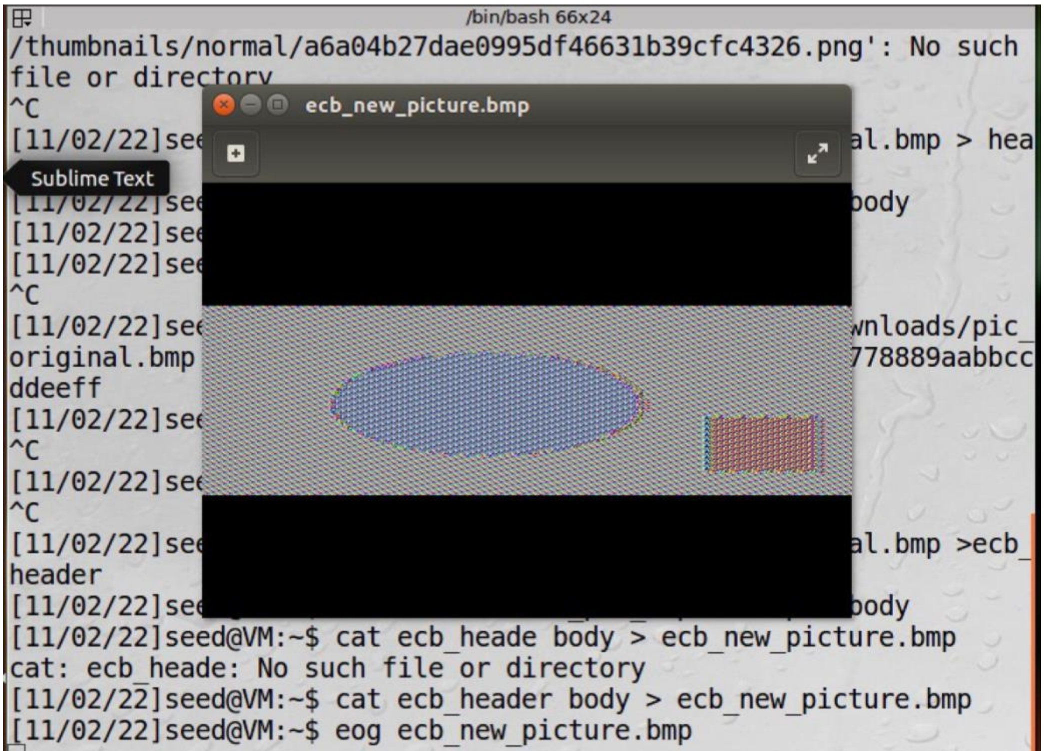


Figure 3.2

(ghex) command used to output the hexadecimal after the whole image is ciphered. shown in the following figure (3.3)

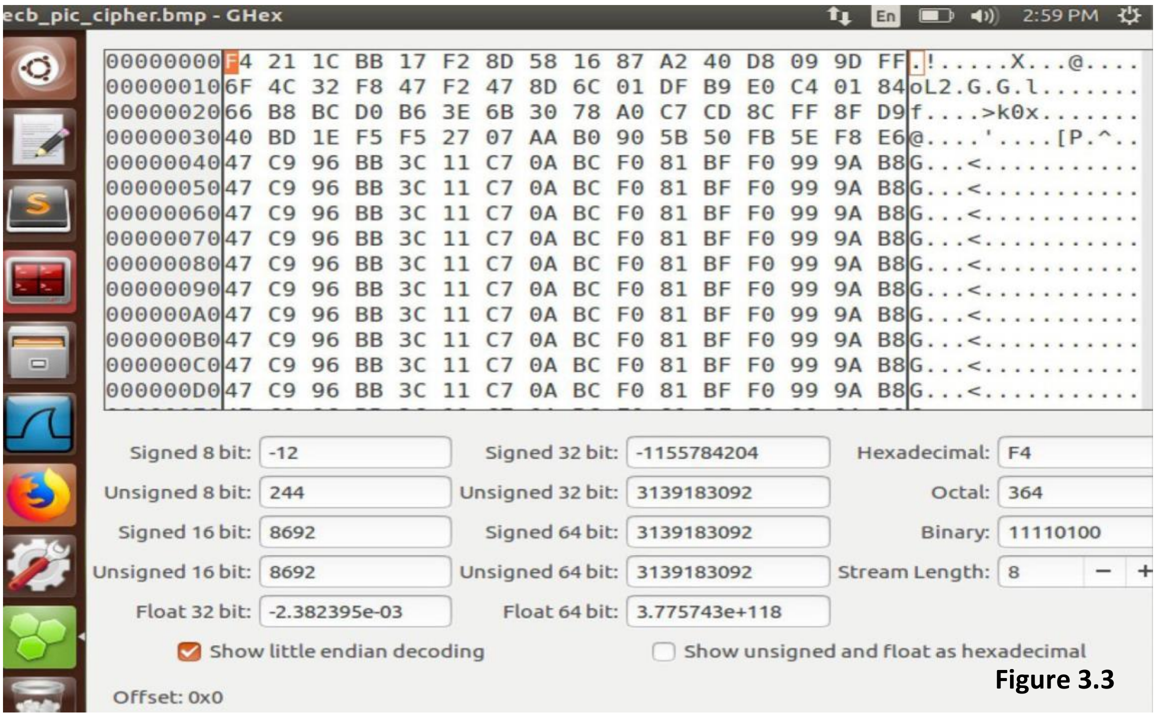
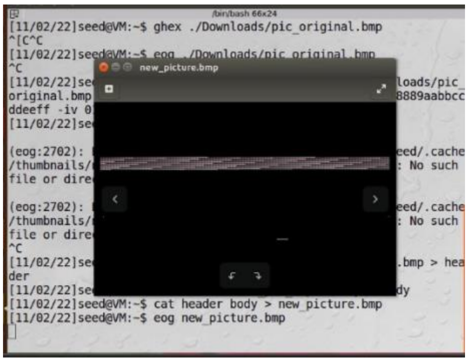


Figure 3.3

CBC stands for Cipher Block Chaining while ECB stands for Electronic Code Book as mentioned before. For more understanding, ECB is suitable for encrypting small messages, while CBC is used to encrypt large messages. As you see below ECB and CBC are a bit different from each other. CBC after encrypting it, you will not notice what the image is. While, ECB the image would be slightly noticeable, and could see slight detail of the image and its lines after ciphering it. additionally, CBC requires the key and initial value other than ECB which only requires the key.

ECB



CBC

Task 7: Programming using the Crypto Library:

The goal of this task is to build a program that is able to find the encryption key, as a .txt file has been provided for us to use that consists of different words as well as the key and the initial value (iv) that was available in the PDF. The program will read each word from the .txt file and guess which word is the correct key to the given message.

The Code:

1- Imports:

For starters we have imported all the necessary tools that we will be using to build the program, The imports that we will use to read from file are: the io.File and util.Scanner, As for the ones that we will need for the Encryption, Decryption are the Crypto tools for ciphering, iv and the key.

2- HexToByte:

This method converts the hexadecimal values into byte values by accepting the hexadecimal value as String. Defined an array value byte[] data which will create a new byte array by calculating the length of the hexadecimal string / 2, After that we need to move along the length of the hexadecimal String to convert each hex value (Using the for loop). The loop will start implementing the formula of converting the hexadecimal value to a byte value Finally returning the byte value (return data).

The formula:

Figure 1:

```
data[i/2] = (byte) ((Character.digit(hex.charAt(i), 16) << 4)+Character.digit(hex.charAt(i+1), 16));
```

*This formula is fixed to converting hexadecimal values to byte.

Figure 2:

```java
// All the imports that we need for this project:
import java.io.File;
import java.util.Scanner;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;


public class Task7 {


    public static byte[] HexToByte(String hex){ // Turn Hex to Bytes value.

        byte[] data = new byte[hex.length()/2];
        for (int i = 0; i < hex.length(); i += 2){
            data[i/2] = (byte) ((Character.digit(hex.charAt(i), radix: 16) << 4)+Character.digit(hex.charAt(i+1), radix: 16));
        }
        return data;
    }

```

3- BytesToHex:

This method converts the byte values to hex values by accepting the byte value as an array. Defined a char value char[] HexChars which will create a new char array by calculating the bytes length * 2, Another array has been defined char[] HexArrayValues where it will take the hexadecimal values (0-9 and A-F) and turn them to a charArray,After that we need to move along the length of the bytes to convert each byte value (Using the for loop).

The loop will start implementing the formula of converting the Byte values to a hexadecimal value Finally returning the String of the Hexadecimal value (return new String(HexChars);).

The formula:

Figure 3:

```
int x = bytes[i] & 0xFF;

        HexChars[i*2] = HexArrayValue[x >>> 4];

        HexChars[i*2+1] = HexArrayValue[x & 0x0F];
```

*This formula is fixed to converting byte values to hexadecimal.

Figure 4:

```
// Turning Byte to hex values:
public static String BytesToHex(byte[] bytes){

    char[] HexChars = new char[bytes.length * 2];
    char[] HexArrayValue = "0123456789ABCDEF".toCharArray();

    for (int i = 0; i < bytes.length; i++){
        int x = bytes[i] & 0xFF;
        HexChars[i*2] = HexArrayValue[x >>> 4];
        HexChars[i*2+1] = HexArrayValue[x & 0x0F];
    }
    return new String(HexChars); // return new hex char.
}
```

4- encrypt:

In this method we have started the encryption process where this method will accept the plaintext, initial value and the key. The process begins by a try{} statement that will allow us to define the values that we inserted and test for any errors. We will be using the crypto tool here as we will be needing the IvparameterSpec, SecrectKeySpec and Cipher.

- **IvparameterSpec:** specifies the initialization vector, where it will take the value of the initiation vector that have been given (invex).
- **SecretKeySpec:** constructs the given key (K) that will be represented as bytes (K.getBytes()) using the aes algorithm (,“AES’’).
- **Cipher:** providing the functionality of cryptographic cipher used in encryption and decryption, It will get the Instance (.getInstance()) of the defult values (“AES/CBC/PKCS5Padding’’).

  After that the cipher will initialize (.init()) the value by taking the Cipher and using the Encrypt_Mode tool (Cipher.ENCRYPT_MODE) , the value of the SecretKeySpec(skySpec) and the IvParameterSpec (iv).

  Finally returning the cipher text value (return cipherText). Catch exception to end the try{}, returning null if nothing (return null).

figure 5:

```java

    //The Encryption Method:
    public static byte[] encrypt(String PT, byte[] invex, String K){ // PT: plaintext, invex: initial vector, K: key.

        try{

            IvParameterSpec iv = new IvParameterSpec(invex);
            SecretKeySpec skeySpec = new SecretKeySpec(K.getBytes(), "AES");
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv);
            byte[] cipherText = cipher.doFinal(PT.getBytes());

            return cipherText;
        } catch (Exception exp){

        }
        return null;
    }

```

5- decrypt:

In this method we have started the decryption process where this method will accept the encrypted message, initial value and the key. The process begins by a try{} statement that will allow us to define the values that we inserted and test for any errors. We will be using the crypto tool here as we will be needing the IvparameterSpec, SecrectKeySpec and Cipher.

- **IvparameterSpec:** specifies the initialization vector, where it will take the value of the initiation vector that have been given (invex).
- **SecretKeySpec:** constructs the given key (K) that will be represented as bytes (K.getBytes()) using the aes algorithm (,"AES'').
- **Cipher:** providing the functionality of cryptographic cipher used in encryption and decryption, It will get the Instance (.getInstance()) of the defult values ("AES/CBC/PKCS5Padding'').

After that the cipher will initialize (.init()) the value by taking the Cipher and using the Decrypt_Mode tool (Cipher.DECRYPT_MODE) , the value of the SecretKeySpec(skySpec) and the IvParameterSpec (iv).
A byte array is defined as it will contain the original text (byte[] original) as it will take the cipher value and finishes the decryption (.doFinal()) of the encrypted text that is given (encrypt).

Finally returning the cipher text value (return new String(original);). Catch exception to end the try{}, returning null if nothing (return null).

Figure 6:

```
    //The Decryption Method:
    public static String decrypt(byte[] encrypt, byte[] invex, String K){ // encrypt: encrypted message, invex: initial value, K: key.

        try{

            IvParameterSpec iv = new IvParameterSpec(invex);
            SecretKeySpec skeySpec = new SecretKeySpec(K.getBytes(), "AES");
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, skeySpec, iv);
            byte[] original = cipher.doFinal(encrypt);

            return new String(original);
        } catch (Exception exp){

        }
        return null;
    }
```

6- GuessTheKey:

In this method is where everything comes together, this method will call all the previous methods as well as start the final calculations for this project. The method starts by the try{} statement that will allow us to define and test for any errors. After that we used the File and gave it the .txt directory and Scanner to scan the directory to be able to read the.txt file that was given.

A while loop have been created to read through the file scanning each line and storing each vale as a String key (K). where the key will be replace the target line (\n) with a white space (“”).

If the key length was bigger than 16 then replace what is left of it as # shown bellow:

Figure 7:

```
if (K.length() < 16){
    for (int i = K.length(); i < 16; i++) {
        K += "#"; // Replace the rest of the key that is < 16 with hashes.
    }
```

After that we will begin with the inputs, specifying our plaintext “This is a top secrect.” As String, The initial value “aabbccddeeff00112233445566778899” converting from hexadecimal to bytes (Using HexToByte) and the encrypting it (using encrypt) taking the plaintext, initial value (iv), and key (K).

Figure 8:

```java

    //Guess The Key Method:
    static public void GuessTheKey() {

        try{

            File file = new File(pathname: "/Users/XI/Desktop/words.txt"); // using file IO.
            Scanner scan = new Scanner(file); // Scanner to scan document.

            while (scan.hasNextLine()){ // Loop to read from file.
                String K = scan.nextLine(); //K: key.
                K = K.replace(target: "\n", replacement: "");

                if (K.length() < 16){
                    for (int i = K.length(); i < 16; i++) {
                        K += "#"; // Replace the rest of the key that is < 16 with hashes.
                    }
                }

                //The inputs for GuessTheKey:
                String plaintext = "This is a top secret.";//PlainText
                byte[] iv = HexToByte(hex: "aabbccddeeff00998877665544332211"); //IV
                byte[] encryptText = encrypt(plaintext, iv, K);

```

Continuing we will create an if statement in the event of an encrypted text available and not null (if (encryptText != null)) We will convert the encrypted text from bytes to a hexadecimal value, if the encrypted text turned to lowercase (.toLowCase()) is equal to "764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2" and a message will be displayed for the user letting them know the search is now done.

The encrypted massage will be decrypted (decrypt()) defined as plain, where it will take the encrypted message value (encryptedText), initial value (iv) and the key (K). if the decrypted text is available and not null (if (plain != null)) the program will start printing the results displaying the plaintext, Key used in ASCII, Key used in HEX and the encryption text in hex, else the key has not been found.

Finally Catch exception to end the try{}.

Figure 9:

```java
                if (encryptText != null){

                    //Turn Bytes to Hex:
                    String encryptedHex = BytesToHex(encryptText);
                    if (encryptedHex.toLowerCase().equals(anObject: "764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2")){
                        System.err.println(x: "Key Search Is Now Done! The Results >>>");

                        String plain = decrypt(encryptText, iv, K); //Decryption of the text:
                        if (plain != null){ //Print the final results.
                            System.out.println("Plain Test is: " + plain);
                            System.out.println("Key Used in ASCII: " + K);
                            String keyHex = BytesToHex(K.getBytes());//Key in hex
                            System.out.println("Key Used in Hex: " + keyHex);
                            System.out.println("Encrypted Text in Hex: " + encryptedHex);

                        }else{
                            System.err.println(x: "No Key Found"); // Error message if key not found.
                        }

                    }
                }
            }

        }
        catch (Exception exp){
            exp.printStackTrace();
        }
    }

```

7- Main:

Reaching the end of our program where the main method will call the GuessTheKey method as all the calculations have been done there.

Figure 10:

```java


    Run | Debug
    public static void main(String[] args){
        GuessTheKey();
    }
}

```

The Final Output:

Figure 11:

```
Key Search Is Now Done! The Results >>>
Plain Test is: This is a top secret.
Key Used in ASCII: Syracuse########
Key Used in Hex: 53797261637573652323232323232323
Encrypted Text in Hex: 764AA26B55A4DA654DF6B19E4BCE00F4ED05E09346FB0E762583CB7DA2AC93A2
```