ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

ITSE 4311 – FUNDAMENTALS OF IT SECURITY

Lab 1 – Part I – Symmetric-Key Cryptography

Abstract

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages

Name: Henok Gelaneh
henokgelaneh@gmail.com
ID: ATR/7217/10
Section: IT

Submission Date: 4/21/2021
Submitted to: Mr. Kabila Haile

Task 1: Encryption using different ciphers and modes

In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man enc. See lecture 2 Slide # 22 – 31 for details. You can also refer to the internet. Tip: \ tells the shell that the command continues.

```
% openssl enc ciphertype -e -in plain.txt -out cipher.bin \
-K 00112233445566778889aabbccddeeff \
-iv 0102030405060708
```

Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -aes-128-cfb,-des-cbc, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "man enc".

We include some common options for the openssl enc command in the following:

-in <file>	input file
-out <file>	output file
-e	encrypt
-d	decrypt
-K/-iv	key/iv in hex is the next argument
-[pP]	print the iv/key (then exit if -P)

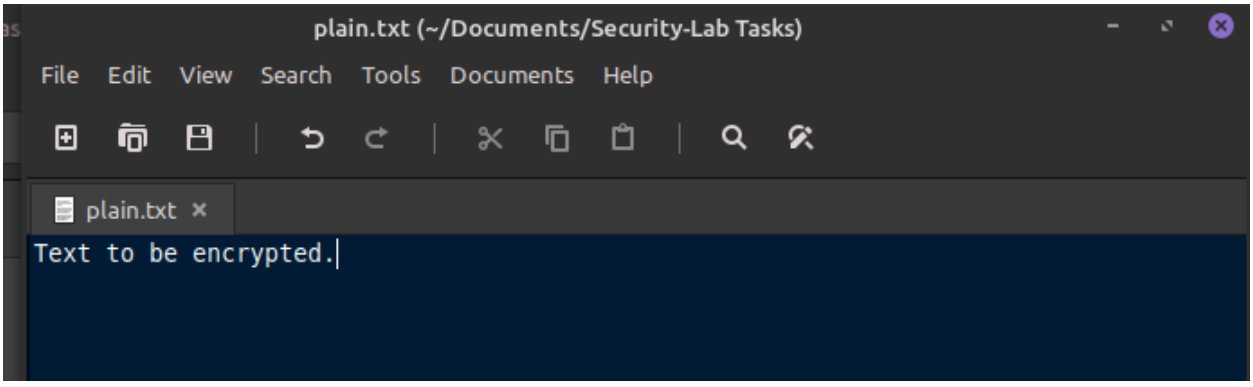

Figure 1 Created plain.txt

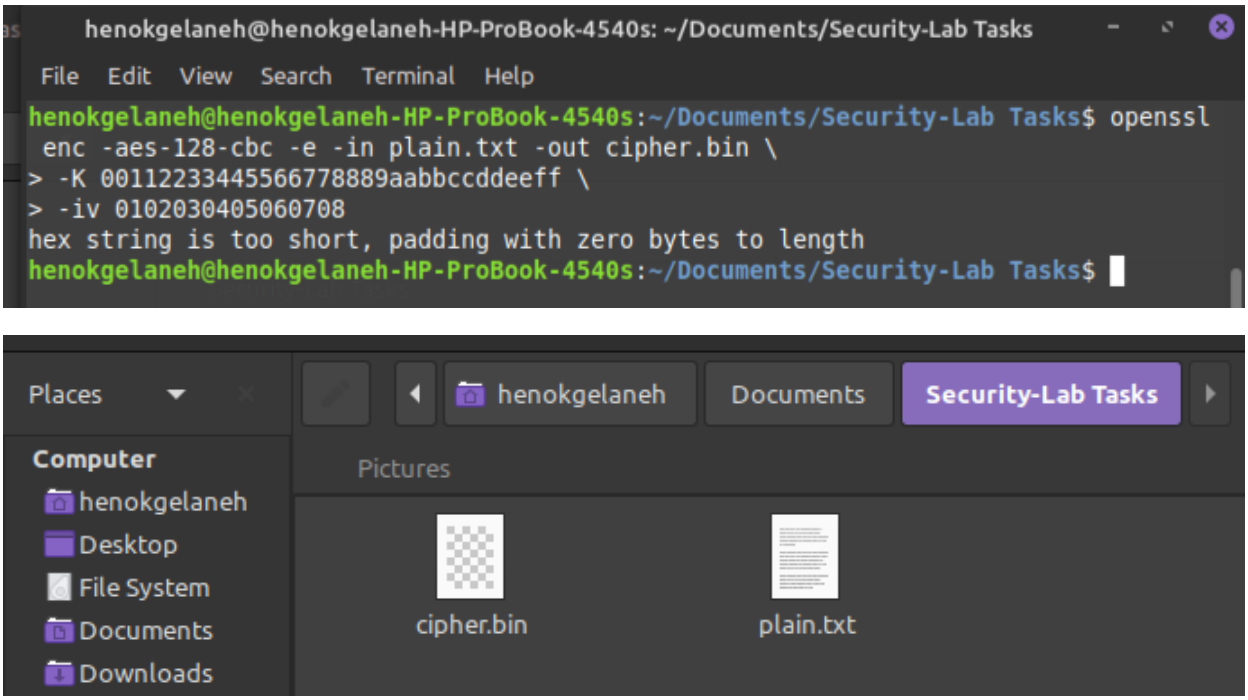

Figure 2 Created cipher.bin which is encrypted

The above error shown, 'hex sting is too short' implies the hex used in -iv is too short and the remaining bits are filled zero bytes instead.

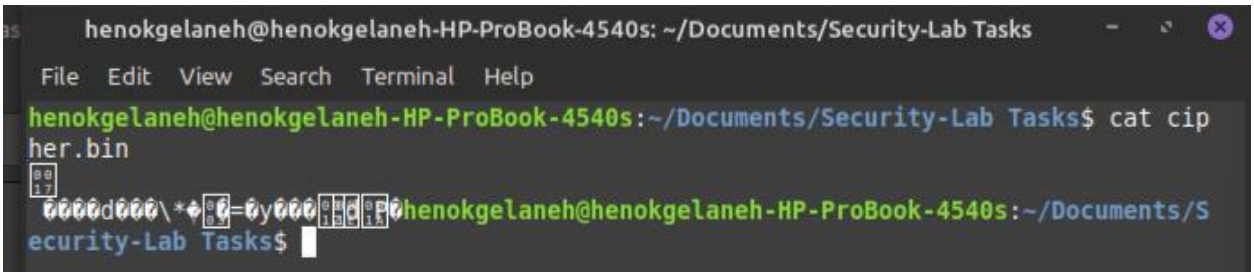

Figure 3 Contents of cipher.bin

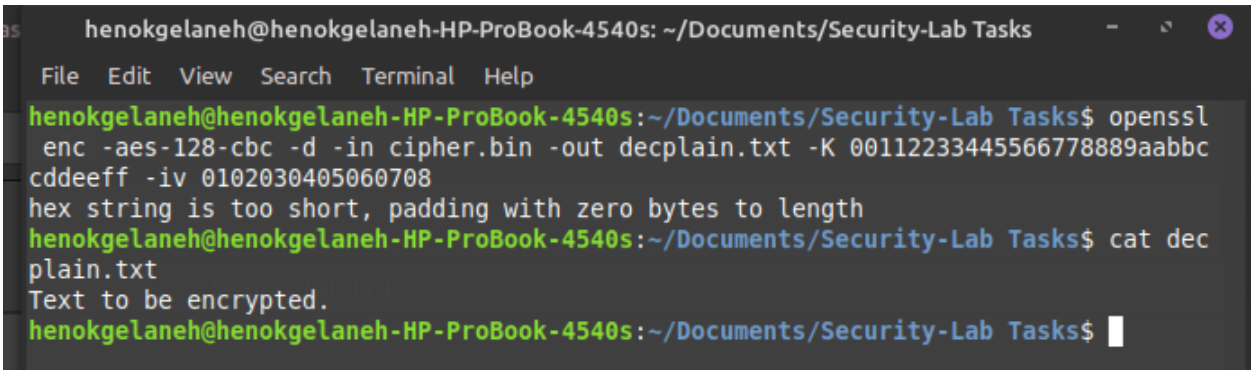

Figure 4 Decryption of cypher.bin

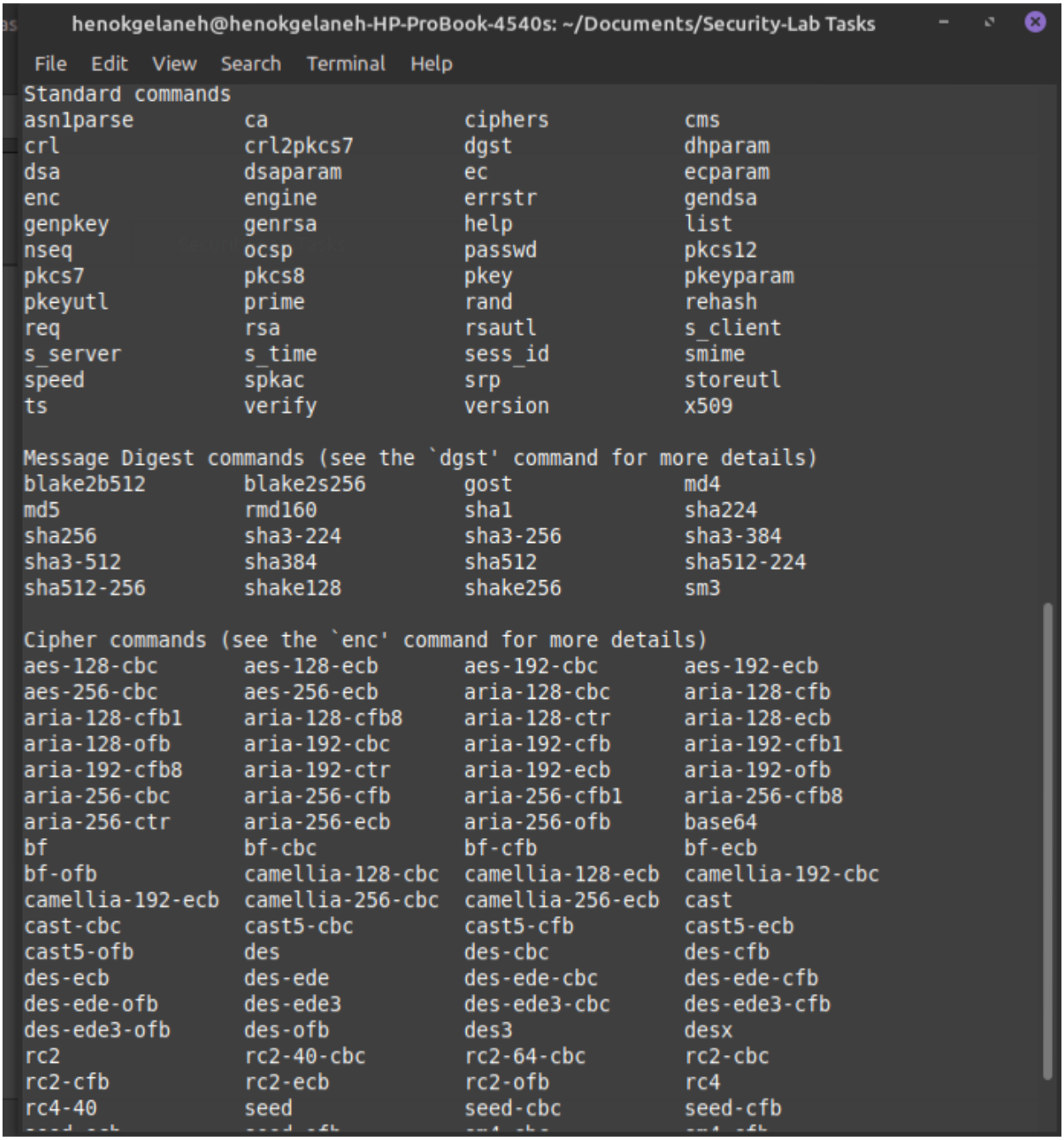

Figure 5 The commands available to openssl

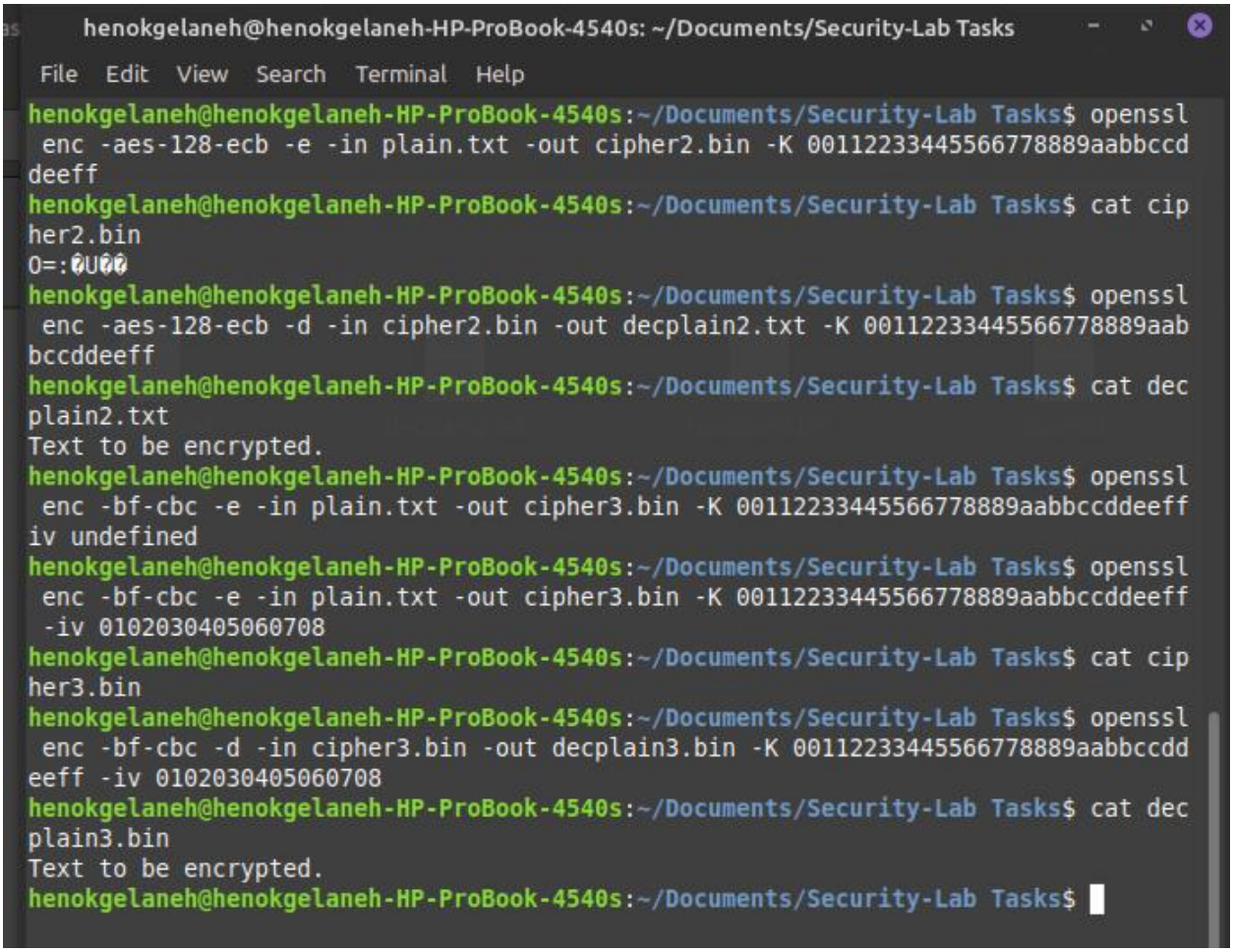

Figure 6 Encrypting using different ciphers

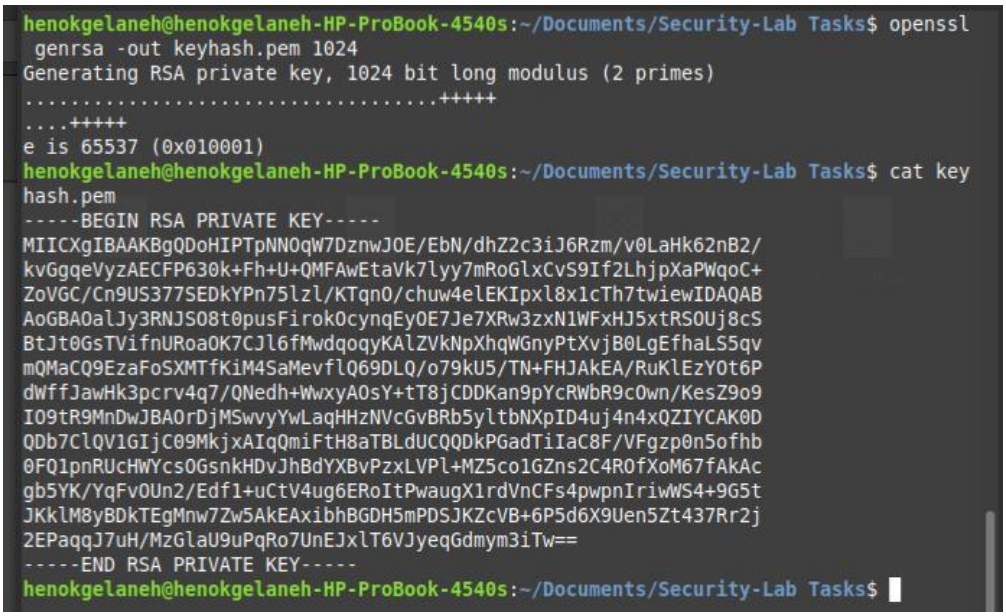

Figure 7 Generating key with rsa algorithm

The above generated key contains both a private and public key. The private key is used for decryption and as such must be secured with encryption if possible.

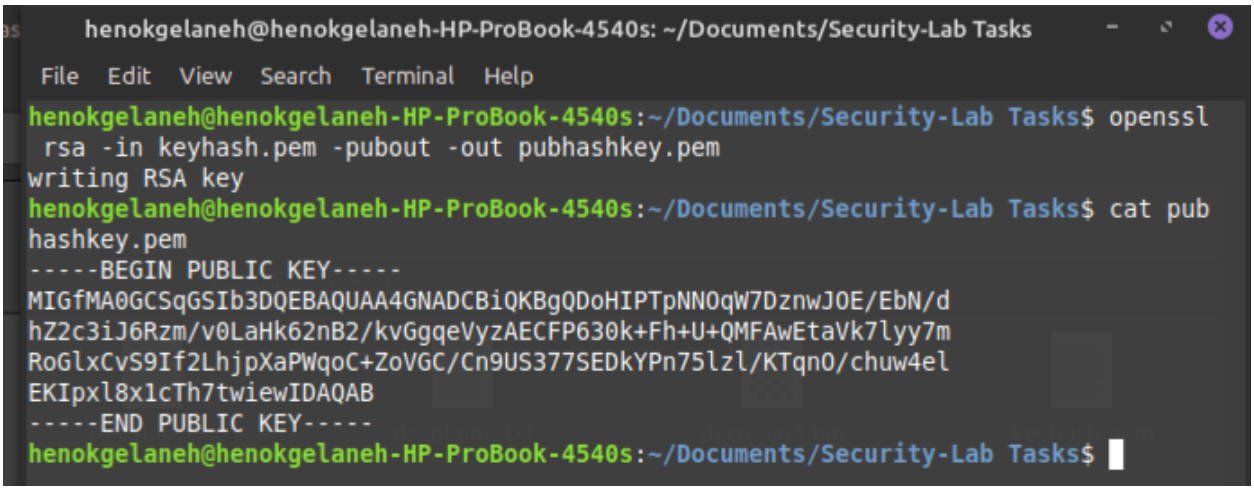

Figure 8 Extracting the public key

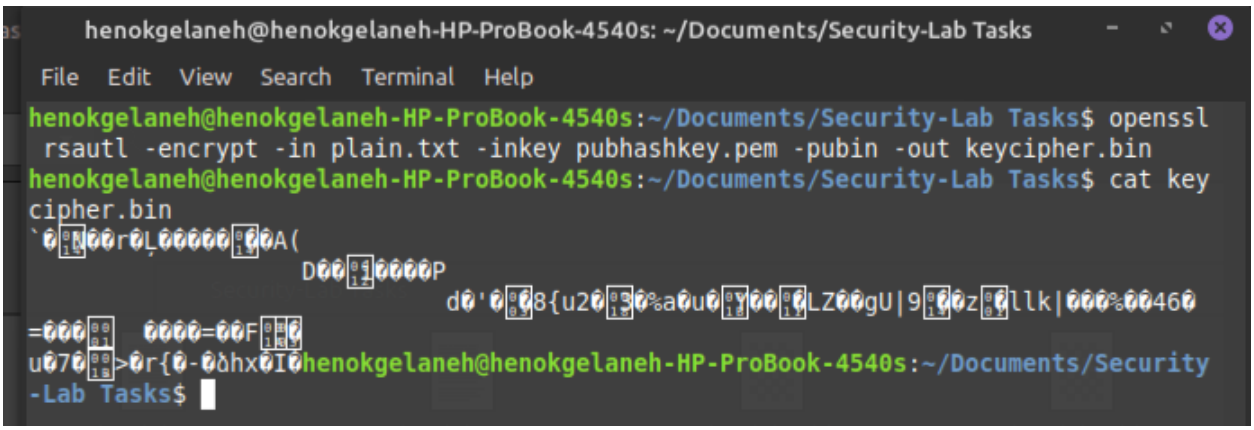

Figure 9 Using public key to encrypt our file

The **rsautl** command can be used to sign, verify, encrypt and decrypt data using the RSA algorithm. Pubin command is used when explicitly using the public key for encryption.

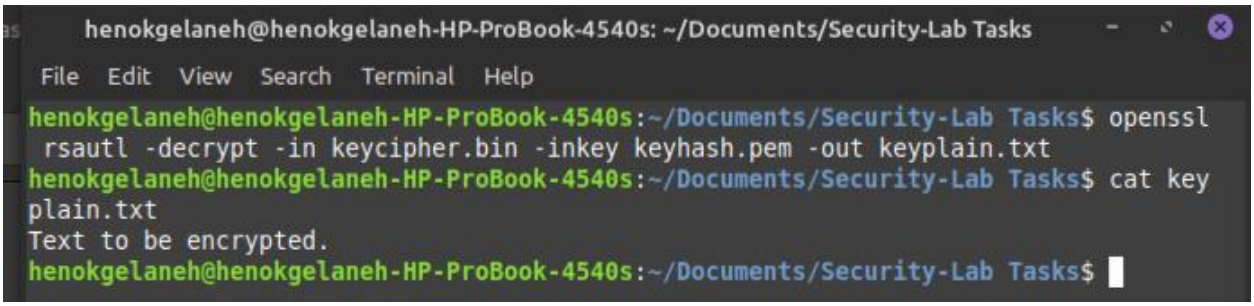

Figure 10 Decryption using key, which requires the private key

Task 2: Encryption Mode – ECB vs. CBC

The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. You can use a hex editor tool (e.g. Bless or ghex) to directly modify binary files.
   Here is how you can do it on Bless. Start Bless by typing bless on the shell, then open both the encrypted and original files on different tabs (use the New File button to create a new tab, and the Open button to open a file on that tab). Now select the first 54 bytes by clicking on the 0th byte and going through the 53rd byte, and Right click on the selection. Select copy from the context menu.After that select the first 54 bytes from the encrypted message and page what you have copied.Save the encrypted message.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

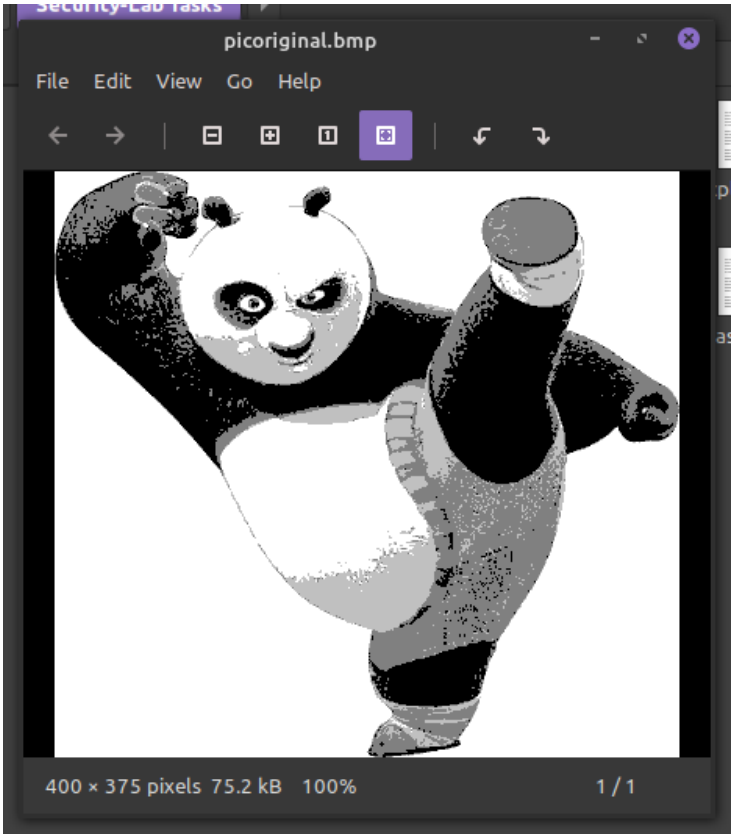

Figure 11 The original bmp image we intend to use

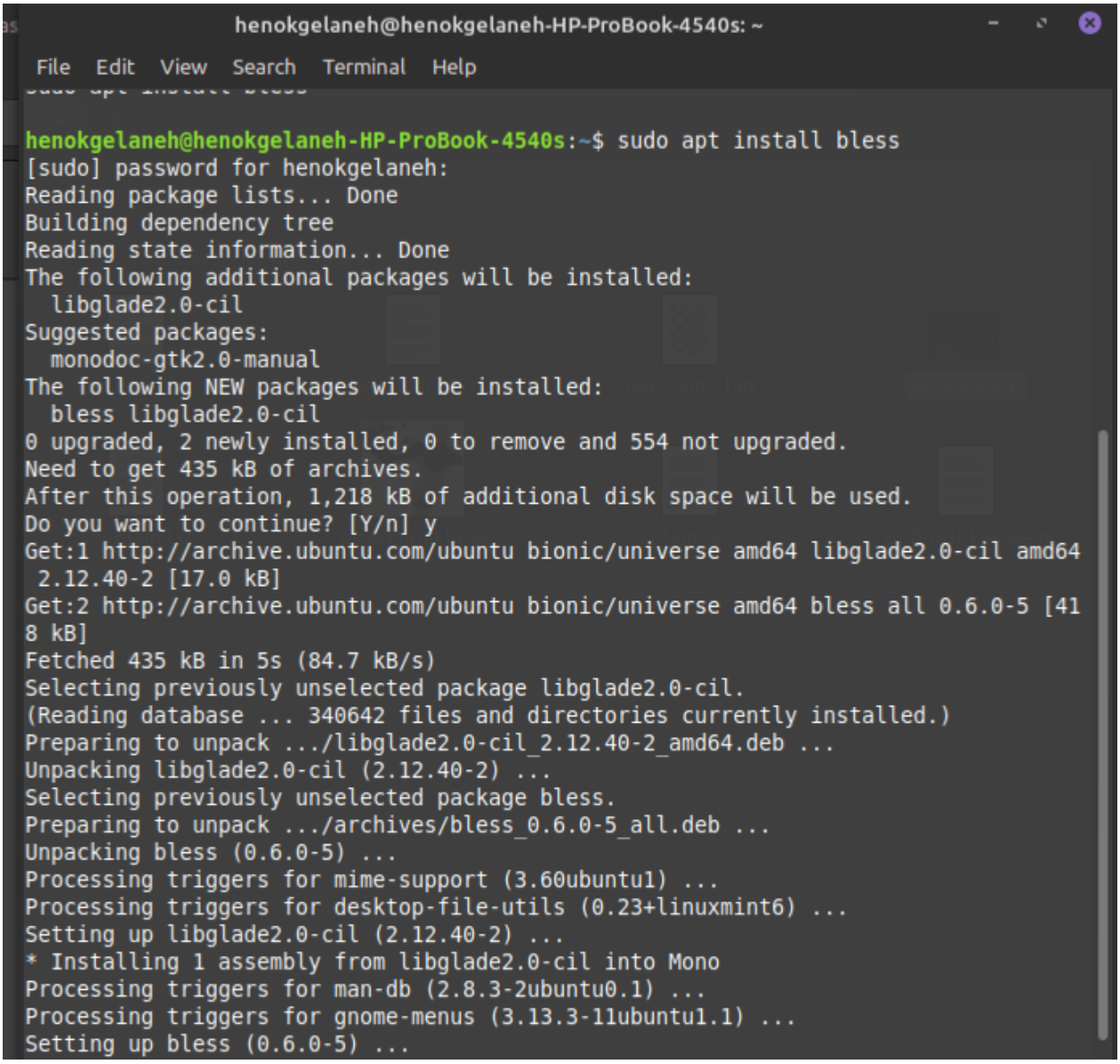

Figure 12 Installing Bless for hex editing

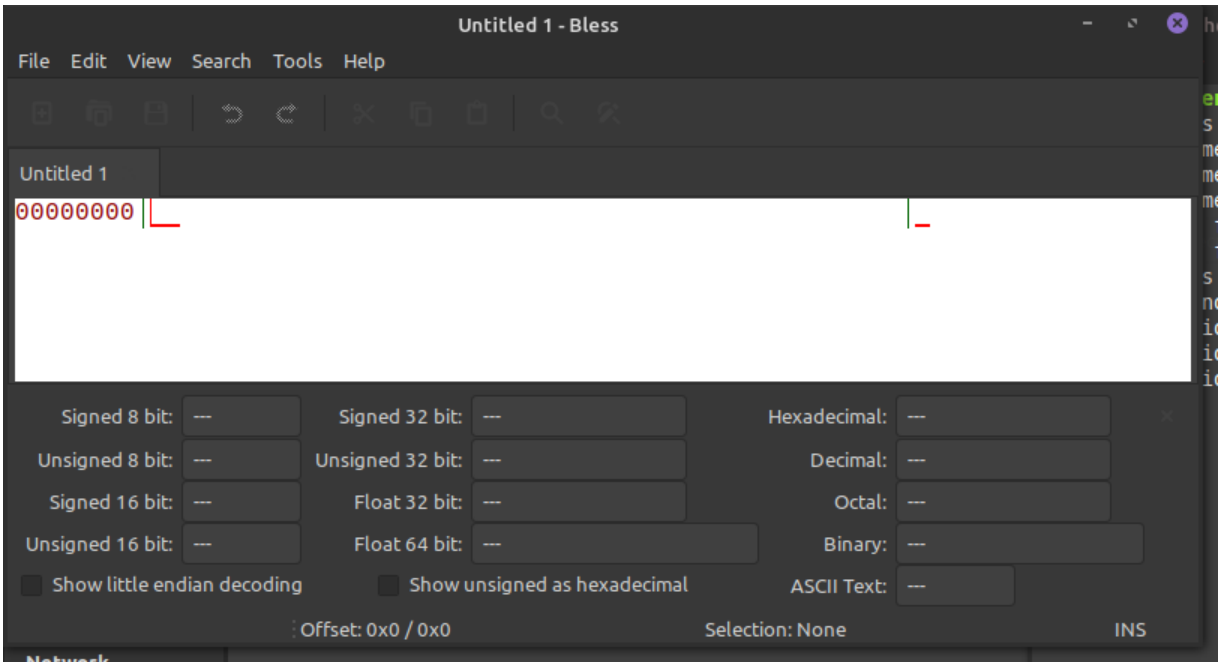

Figure 13 Bless GUI

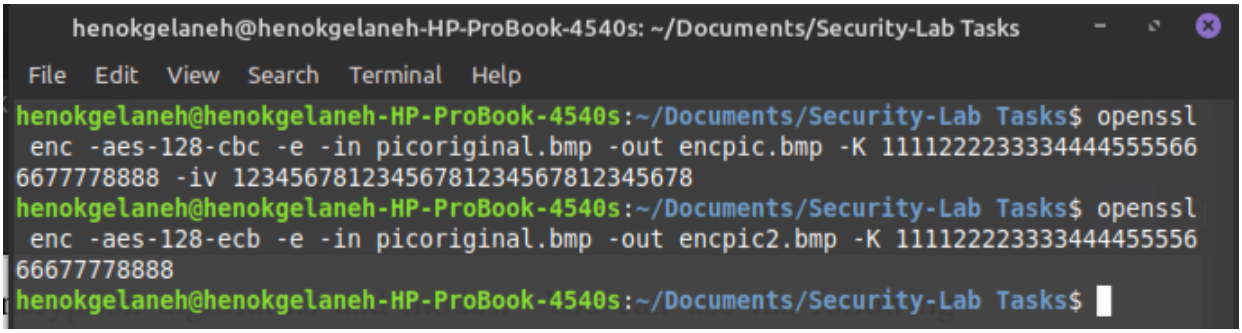

Figure 14 Encrypting picoriginal.bmp with cbc and ecb

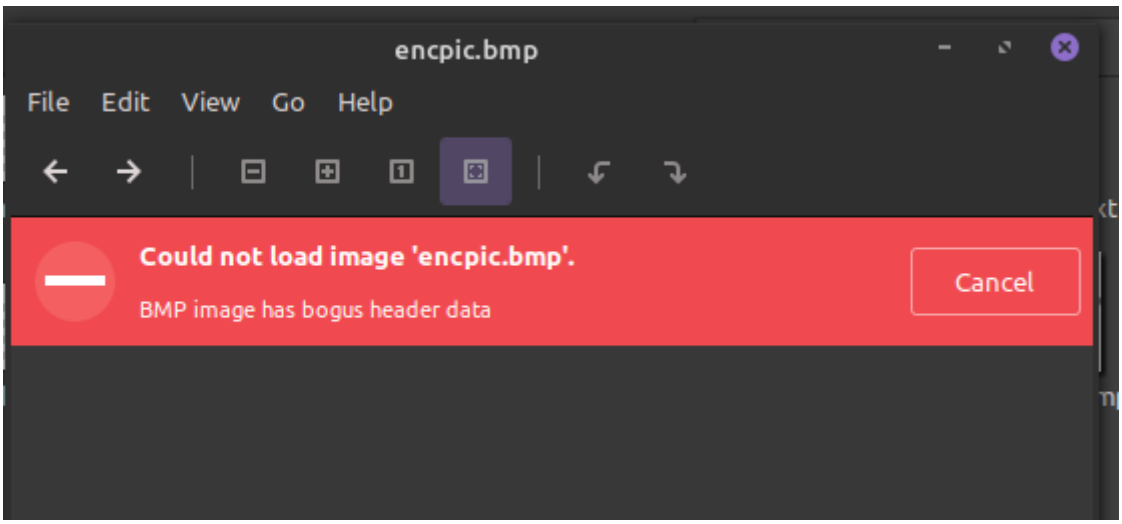

Figure 15 The encrypted bmp won't open because of incorrect header

We now have to replace the first 54 bytes of the encrypted bmp images with that of the original, we will use bless to achieve this task.

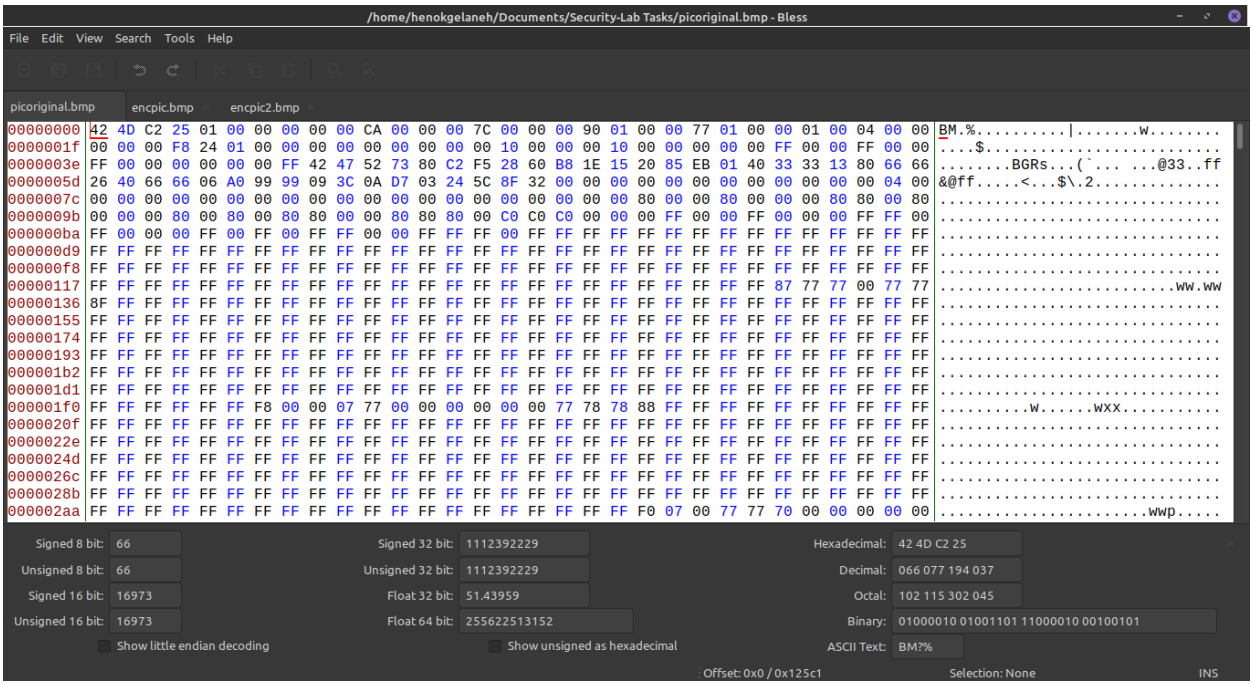

Figure 16 picoriginal.bmp opened in bless

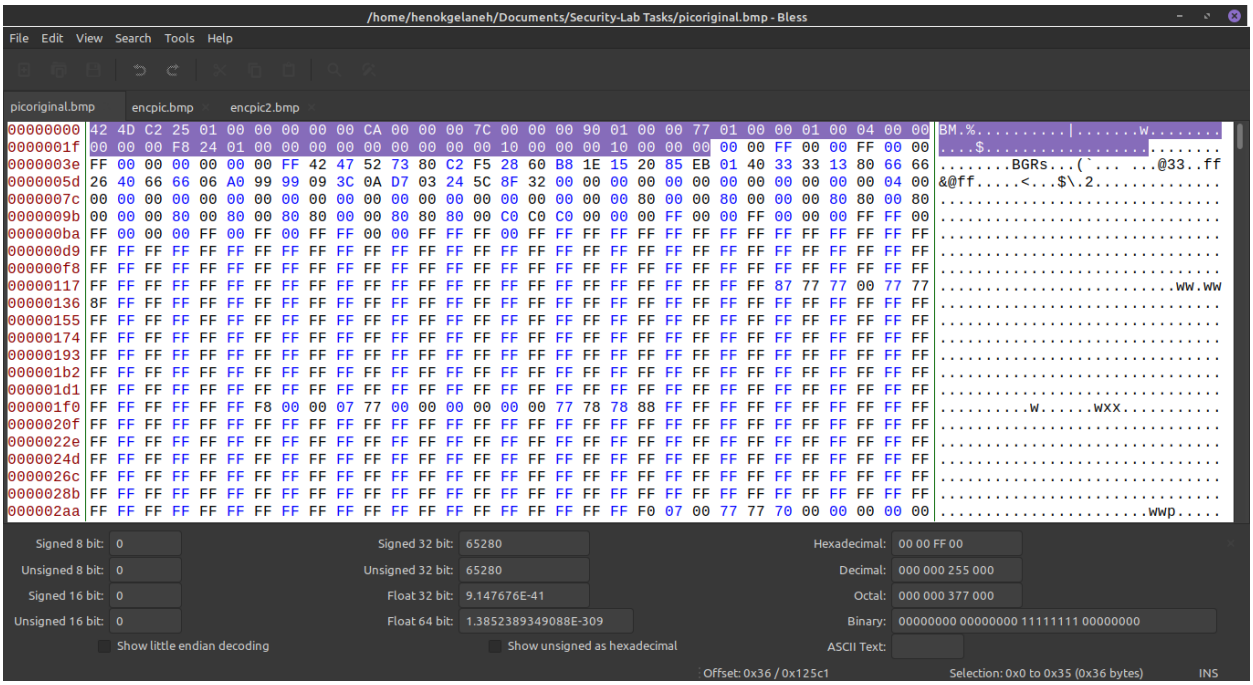

Figure 17 The first 54 bytes of picoriginal.bmp

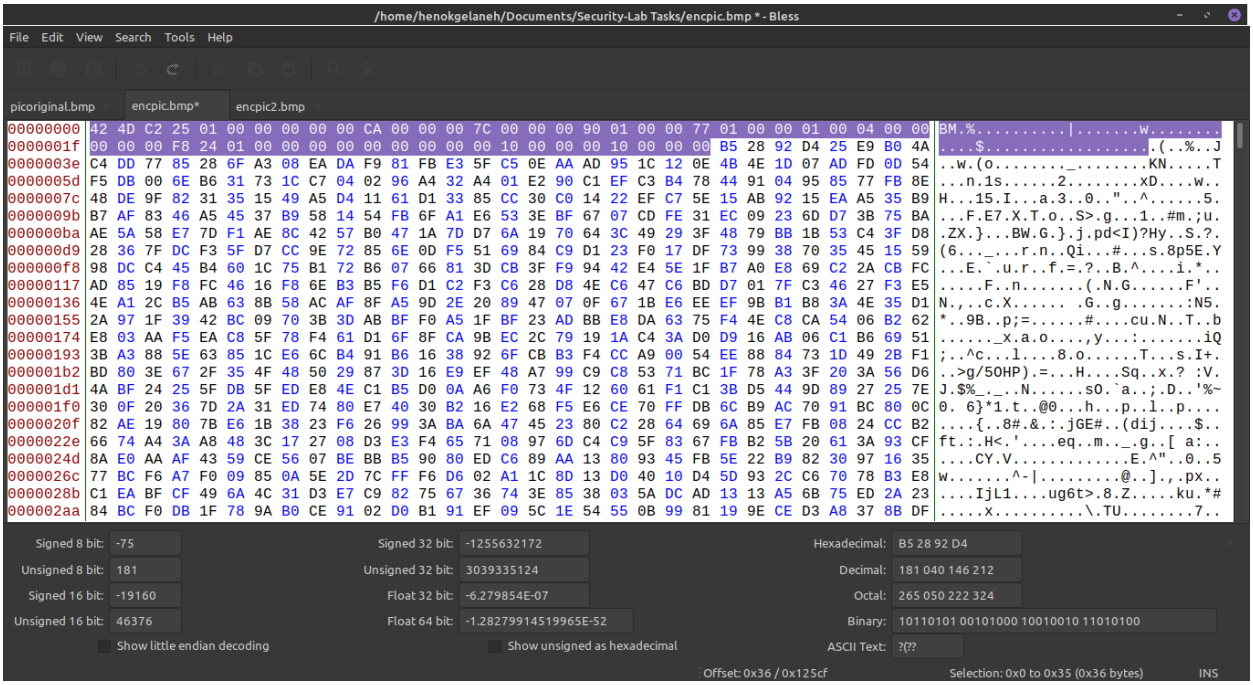

Figure 19 Replacing the headers of the encrypted bmp images

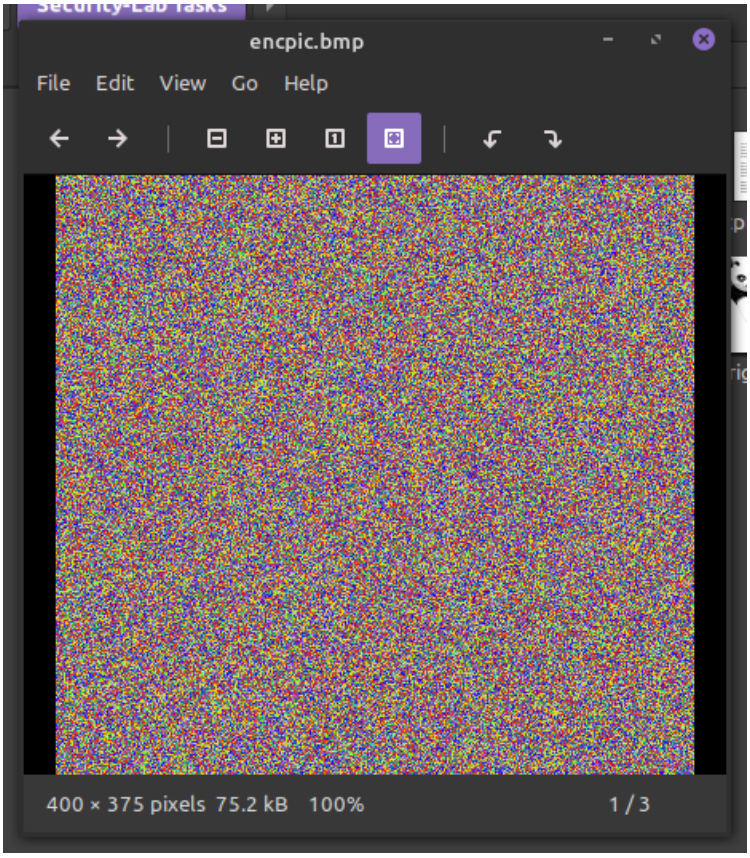

Figure 18 bmp image encrypted by cbc

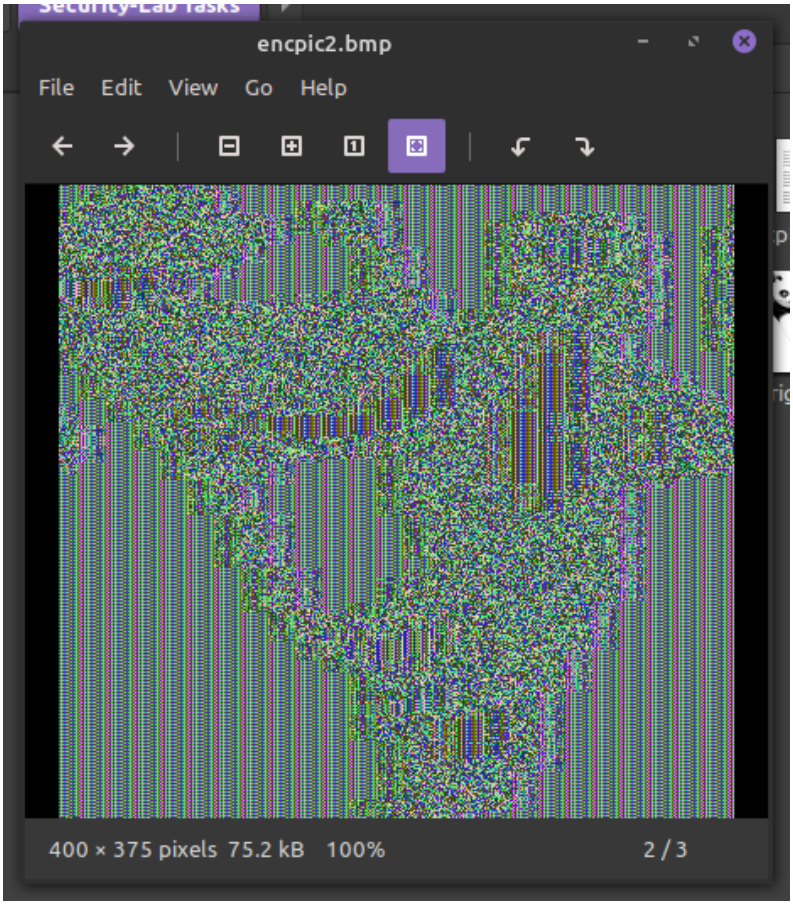

Figure 20 bmp encrypted by ecb

As we can clearly see from the above results, cbc encryption yields a better result because the resulting encrypted imaged was completely different from the original while ecb encryption still has tell tell features.

Task 3: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

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 corruption using a hex editor.
3. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please 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?

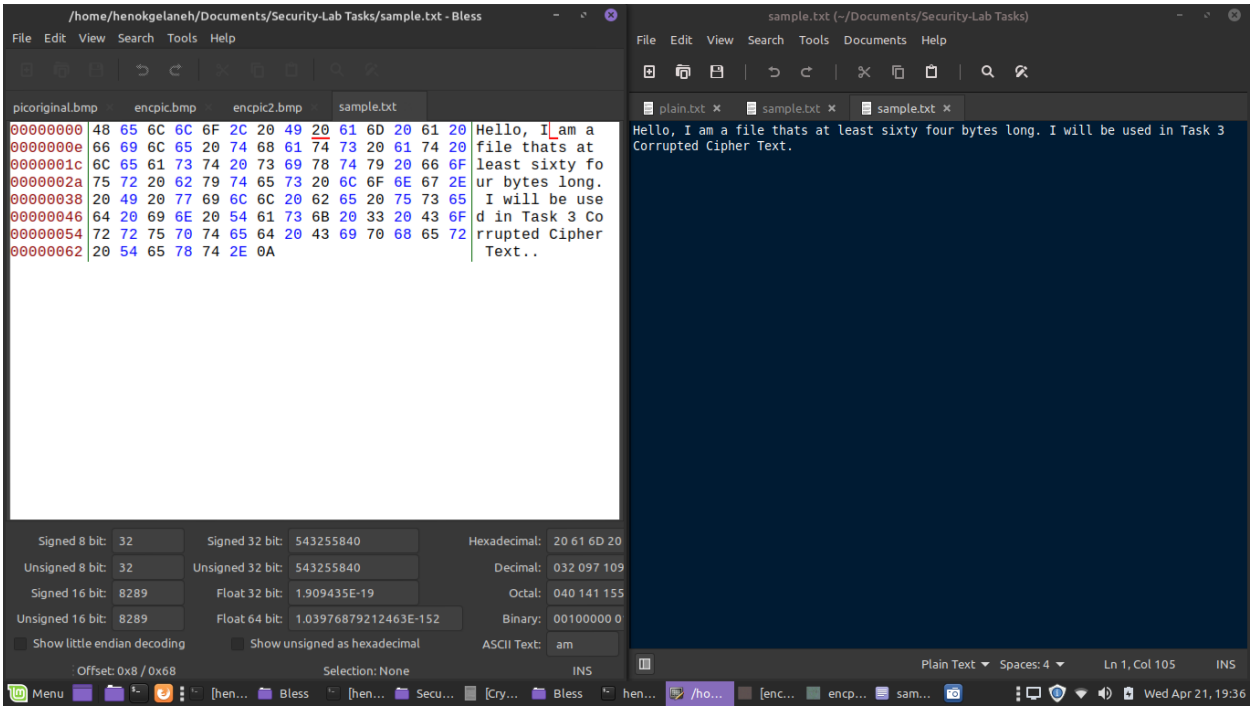

Figure 21 A file that's at least 64 bytes long

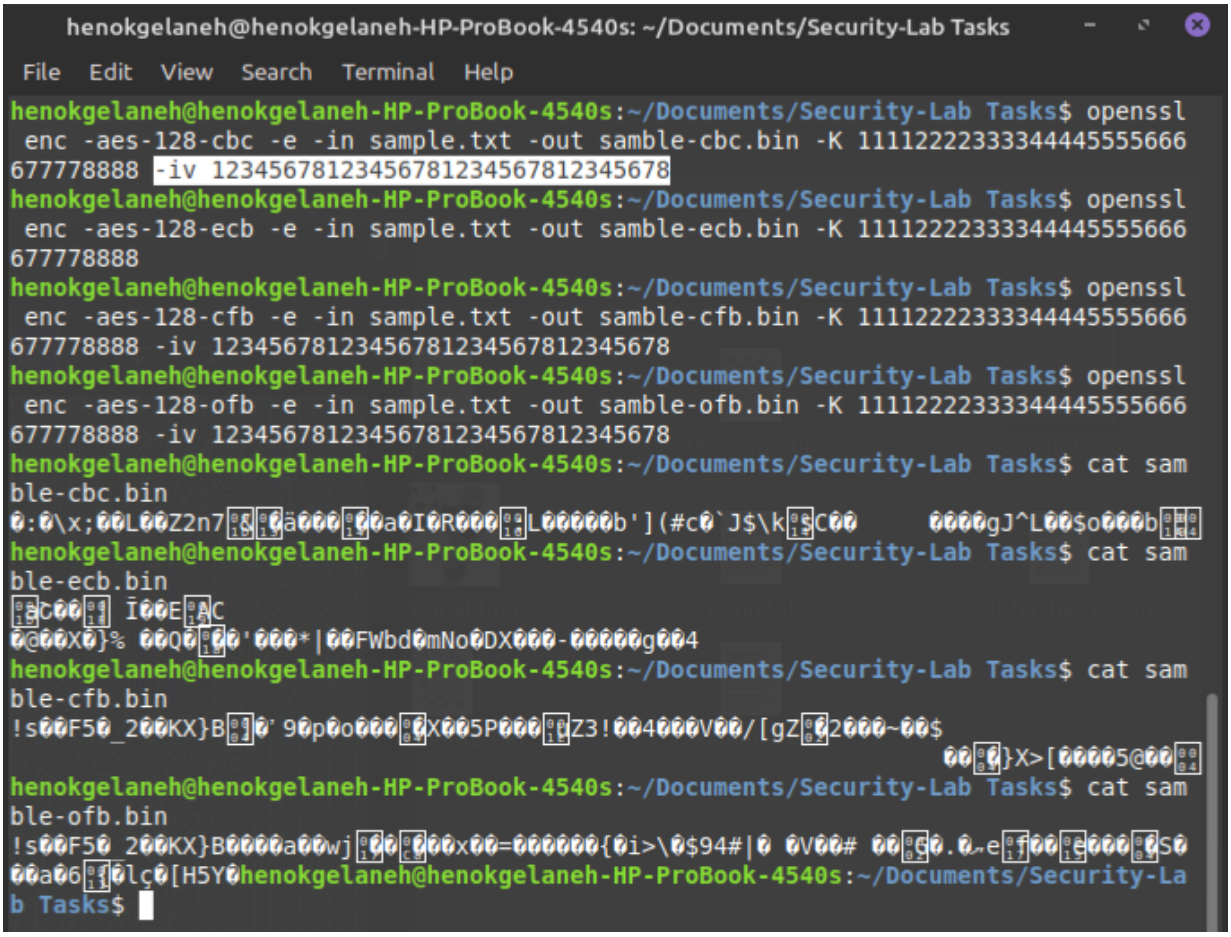

Figure 22 Encrypting the sample file with all four of the ciphers

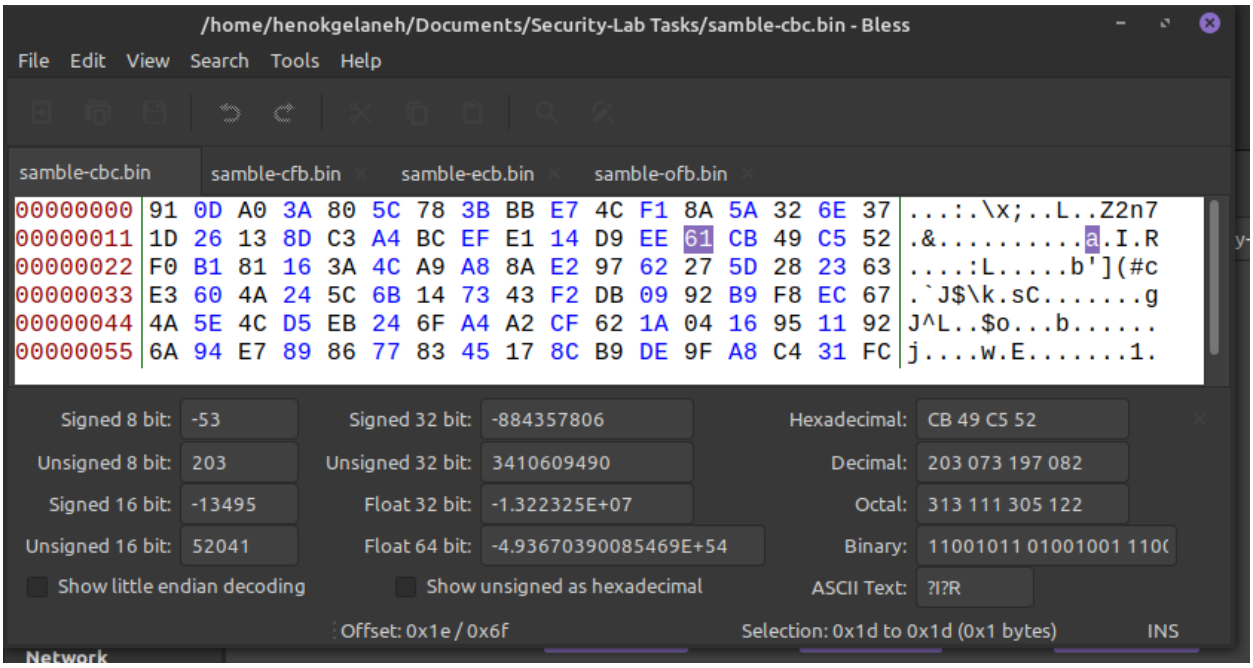

Figure 23 The 30th bit in the encrypted files

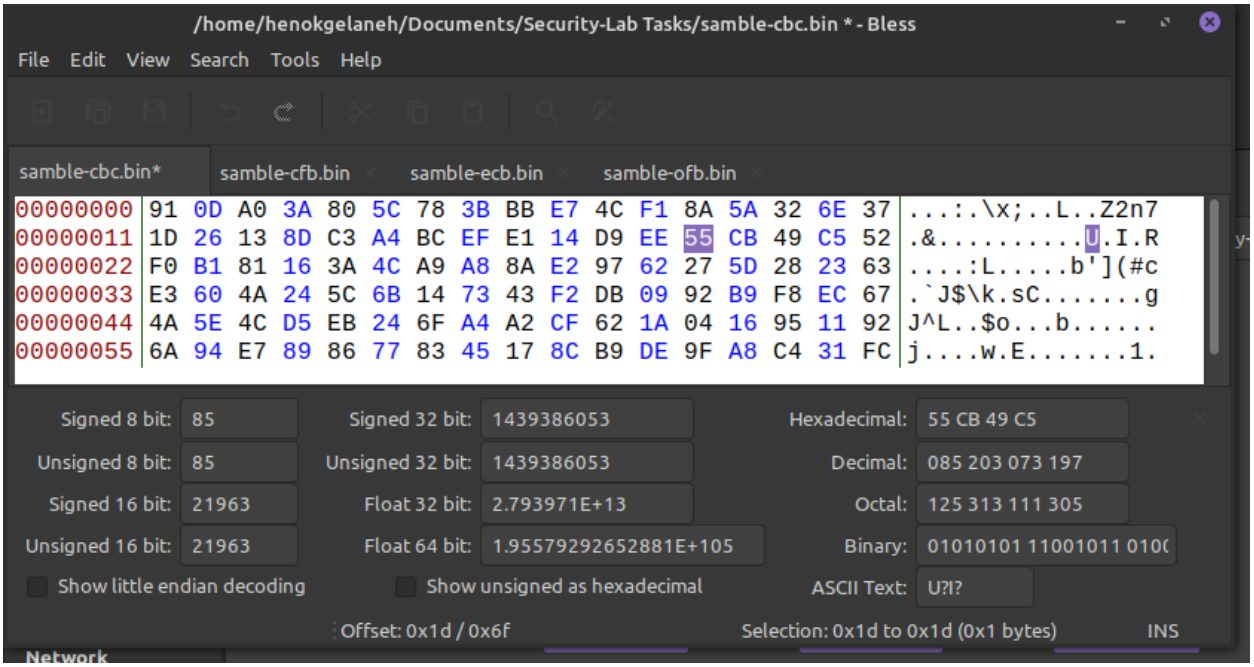

Figure 24 The 30th bit after altering it for corruption

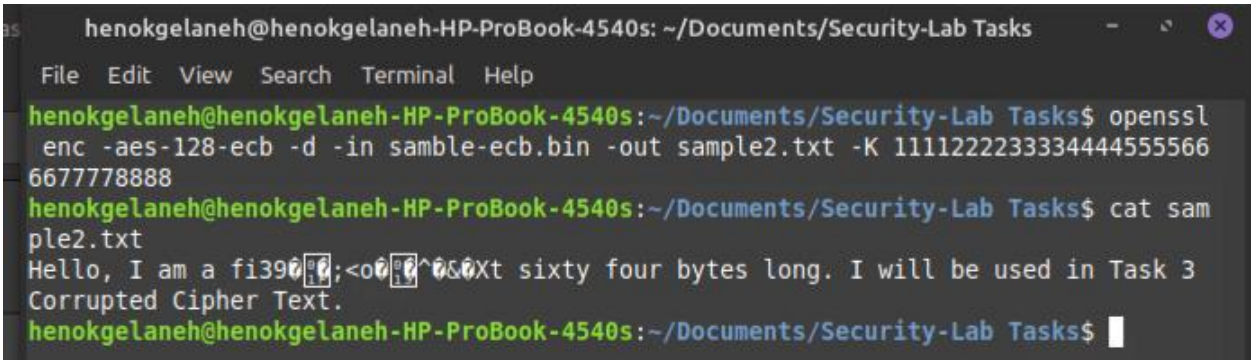

Figure 25 Decrypting the ecb encrypted sample

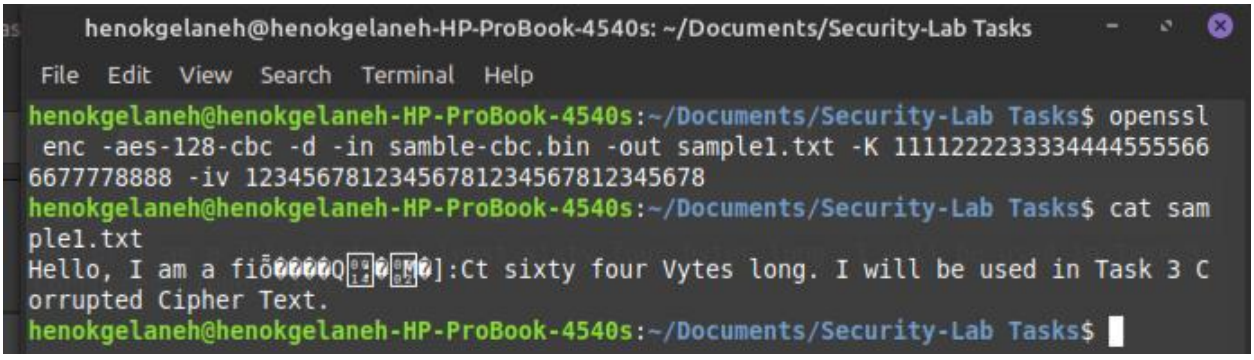

Figure 26 Decrypting the cbc encrypted sample

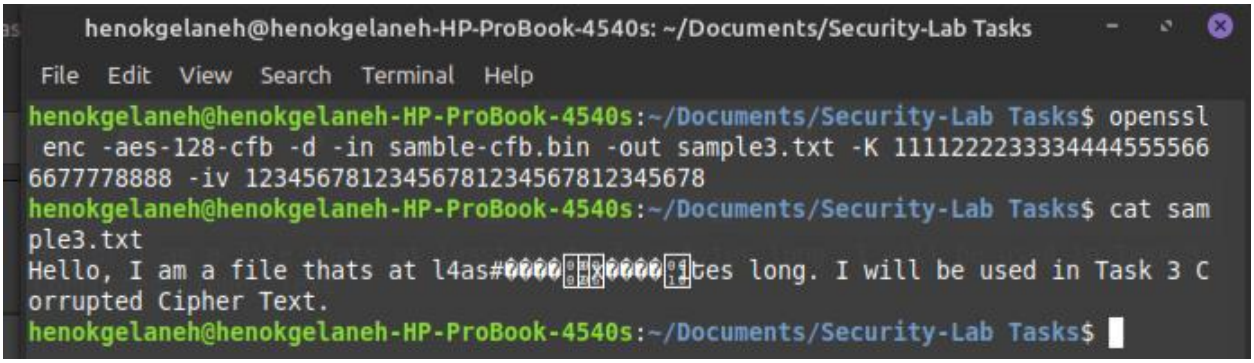

Figure 27 Decrypting the cbf encrypted sample

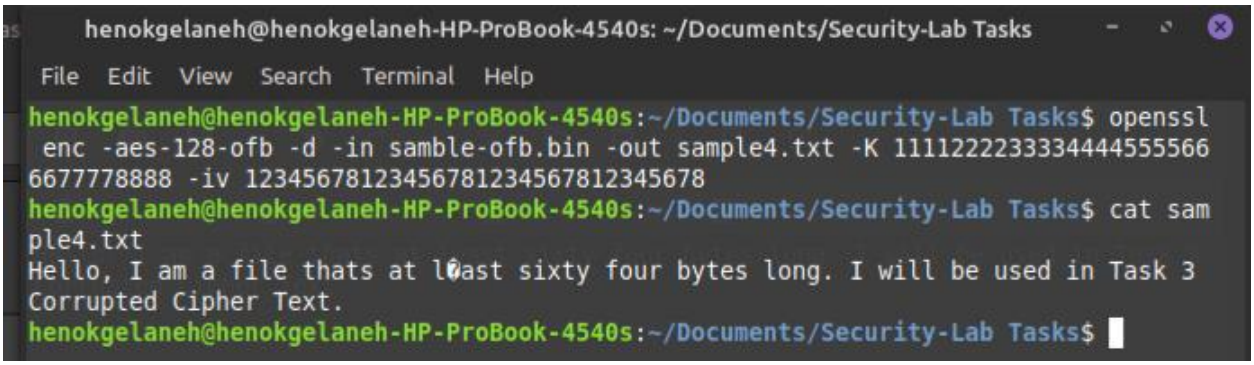

Figure 28 Decrypting the ofb encrypted sample

As we can see from the results above, ecb, cbc and cbf corruption of a single byte affected other parts of the file while the corruption of the 30th byte in the ofb encrypted file only affected the corresponding 30th byte of the decrypted(original) file.

References:

https://www.tutorialspoint.com/cryptography/block_cipher_modes_of_operation.htm

https://www.openssl.org/docs/man1.0.2/man1/

https://www.cryptogram.org/resource-area/cipher-types/

Files used can be found on my github repository:

https://github.com/henokgelaneh7217/SecurityLab.git