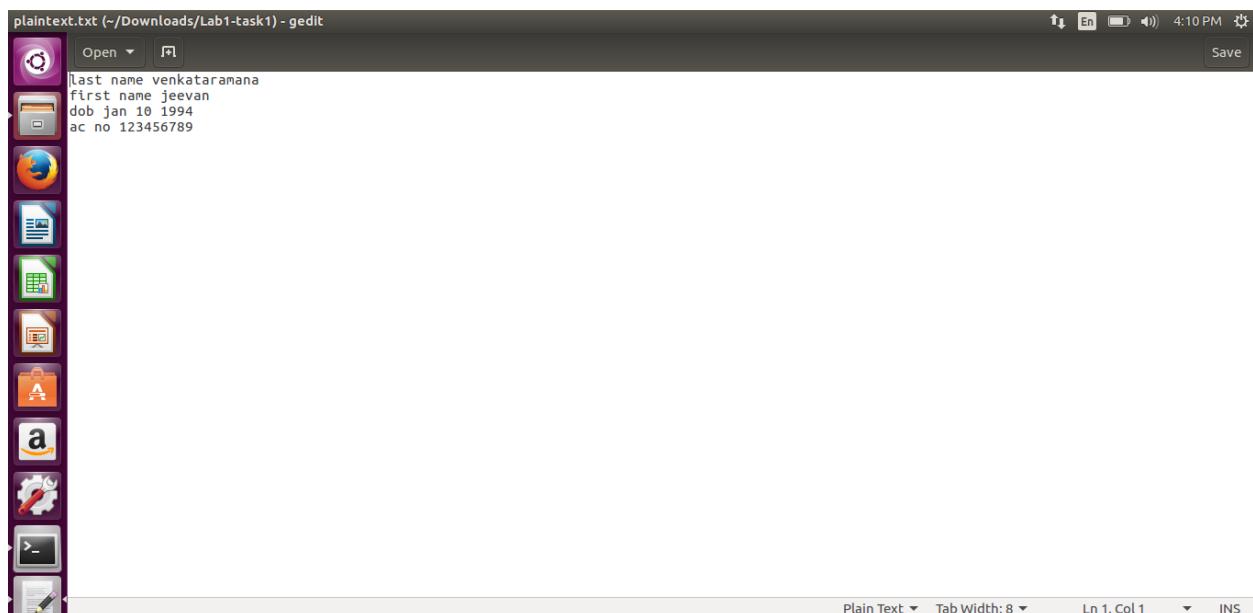


Lab 1
Network Security – Crypto_Encryption
Jeevan Venkataramana
SID : 011917477

Task 1:

Input file: plaintext.txt



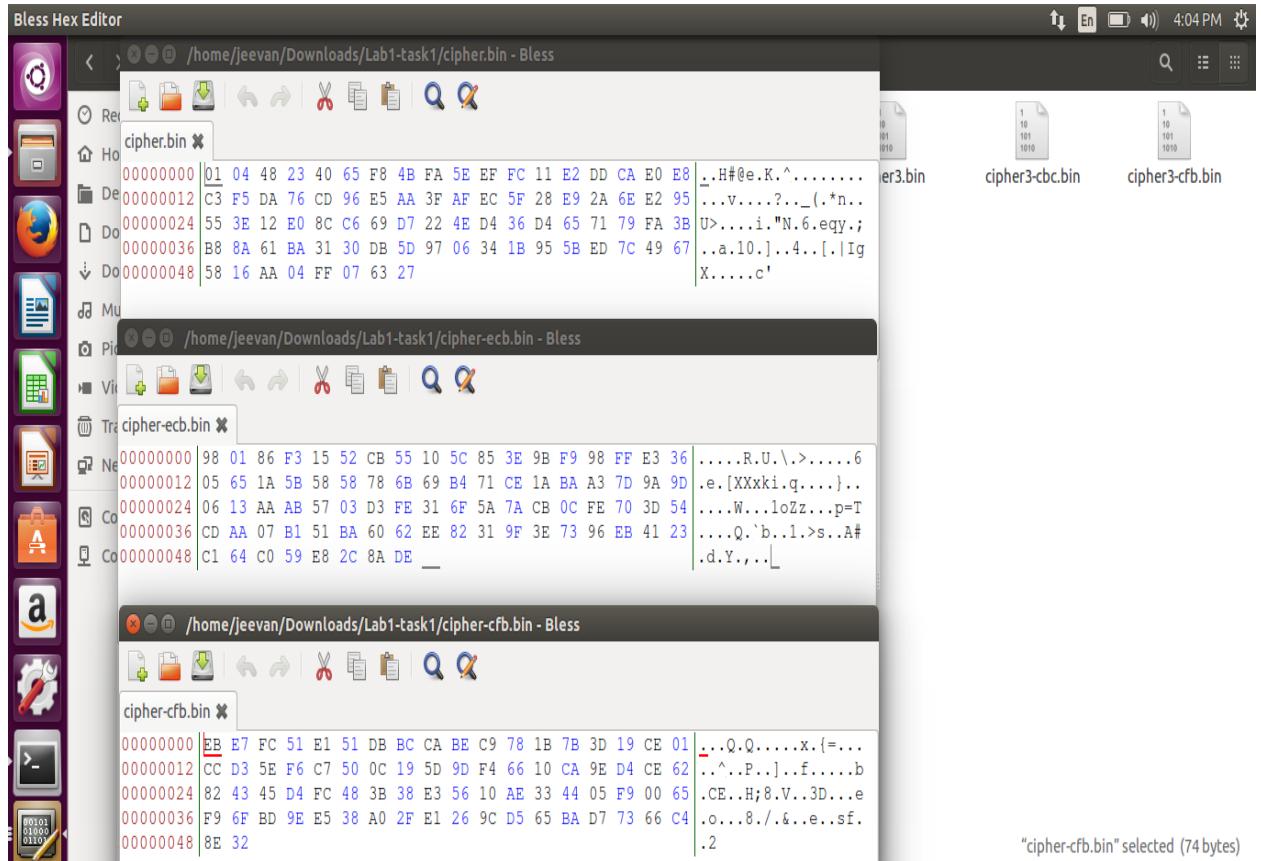
A screenshot of a Gedit text editor window. The title bar reads "plaintext.txt (~/Downloads/Lab1-task1) - gedit". The main text area contains the following four lines of text:
last name venkataramana
first name jeevan
dob jan 10 1994
ac no 123456789

AES Encryption:

Key : 00112233445566778889aabbccddeeff

lv : 0102030405060708

AES-128 Encryption will all 3 modes.



Commands Executed :

```
openssl enc -aes-128-cbc -e -in plaintext.txt -out cipher.bin -K 00112233445566778889aabcccddeeff -iv 0102030405060708
```

```
openssl enc -aes-128-ecb -e -in plaintext.txt -out cipher-ecb.bin -K 00112233445566778889aabcccddeeff
```

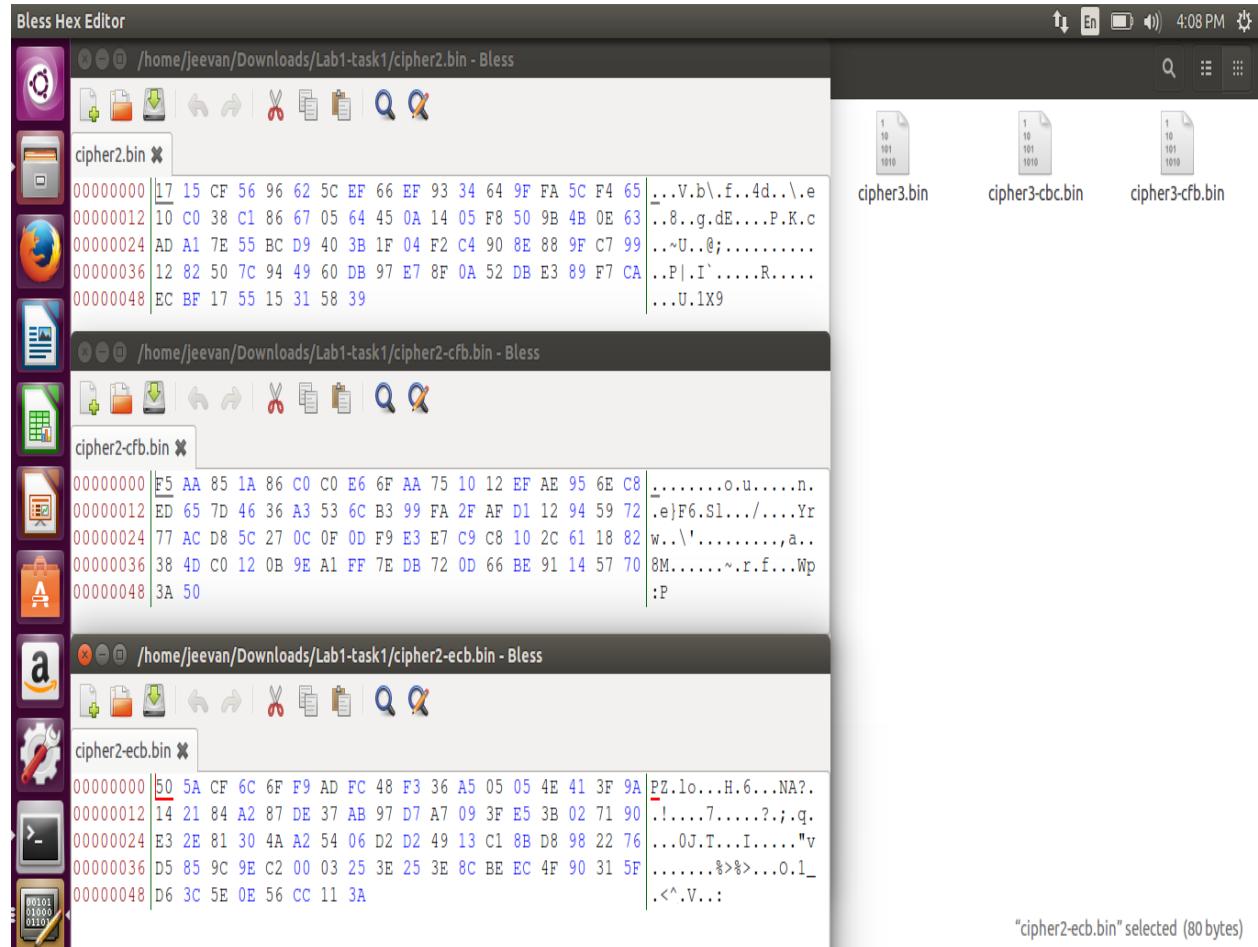
```
openssl enc -aes-128-cfb -e -in plaintext.txt -out cipher-cfb.bin -K 00112233445566778889aabcccddeeff -iv 0102030405060708
```

DES Encryption:

Key : 00112233445566

Iv : 0102030405060708

DES Encryption with all 3 modes.



Commands Executed:

```
openssl enc -des-cbc -e -in plaintext.txt -out cipher2.bin -K 00112233445566 -iv 0102030405060708
```

```
openssl enc -des-ecb -e -in plaintext.txt -out cipher2-ecb.bin -K 00112233445566
```

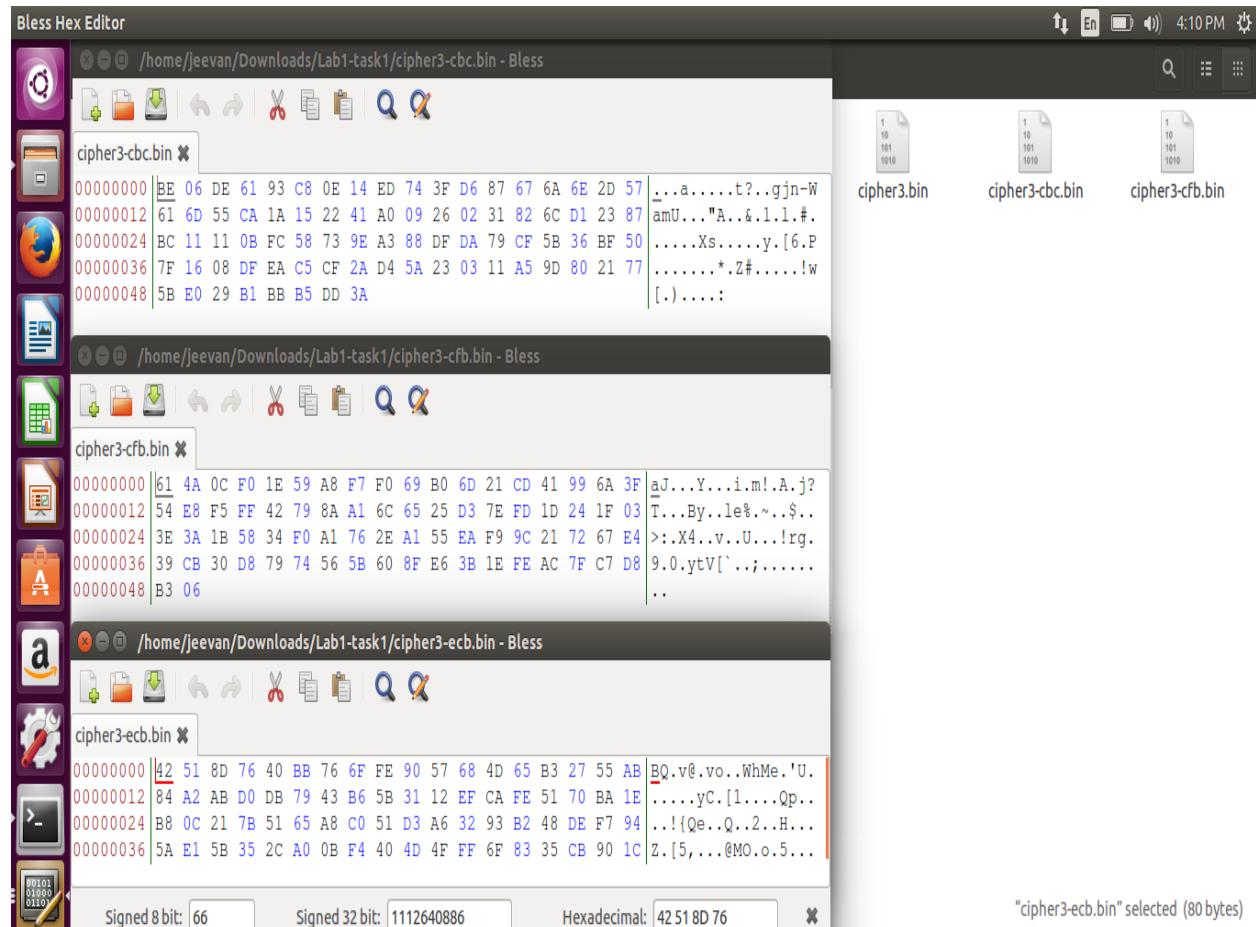
```
openssl enc -des-cfb -e -in plaintext.txt -out cipher2-cfb.bin -K 00112233445566 -iv 0102030405060708
```

RC2 Encryption :

Key : 00112233445566

Iv : 0102030405060708

RC2 Encryption with all 3 modes.



Commands Executed:

```
openssl enc -rc2-cbc -e -in plaintext.txt -out cipher3-cbc.bin -K 00112233445566778889aabbccddeeff -  
iv 0102030405060708
```

```
openssl enc -rc2-ecb -e -in plaintext.txt -out cipher3-ecb.bin -K 00112233445566778889aabbccddeeff
```

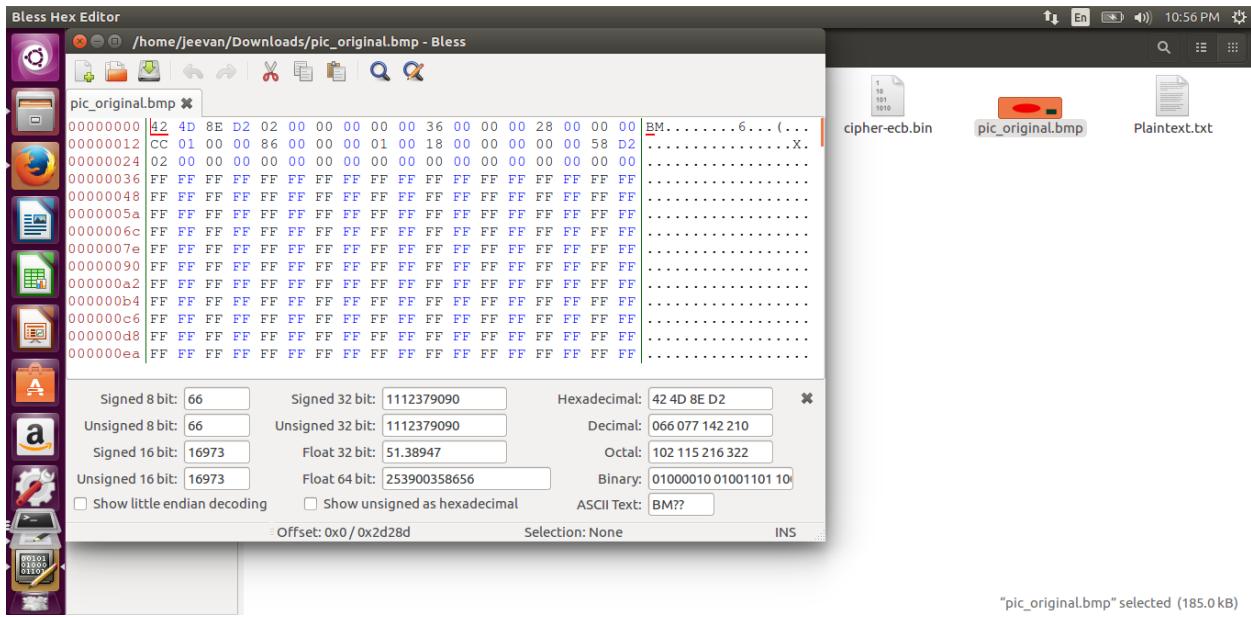
```
openssl enc -rc2-cfb -e -in plaintext.txt -out cipher3-cfb.bin -K 00112233445566778889aabbccddeeff -  
iv 0102030405060708
```

Task 2:

Original Image: pic_original.bmp



Binary file of pic_original.bmp

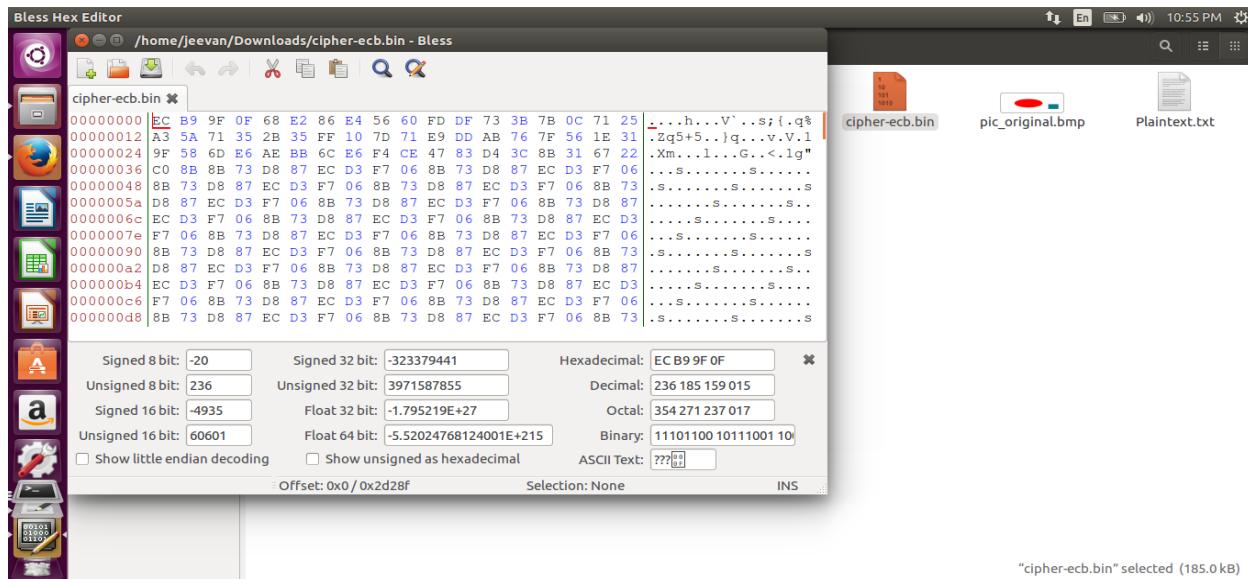


Encryption:

Key: 00112233445566

iv: 01020304050607

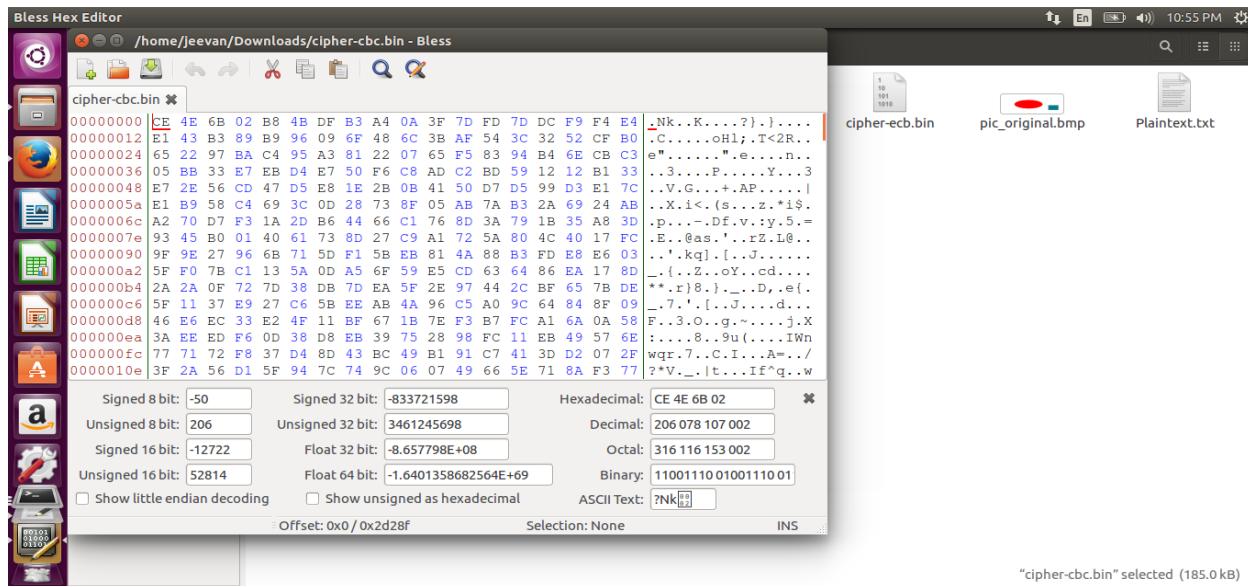
DES – ECB mode:



Command Executed :

```
openssl enc -des-ecb -e -in pic_original.bmp -out cipher-ecb.bin -K 00112233445566
```

DES – CBC mode:

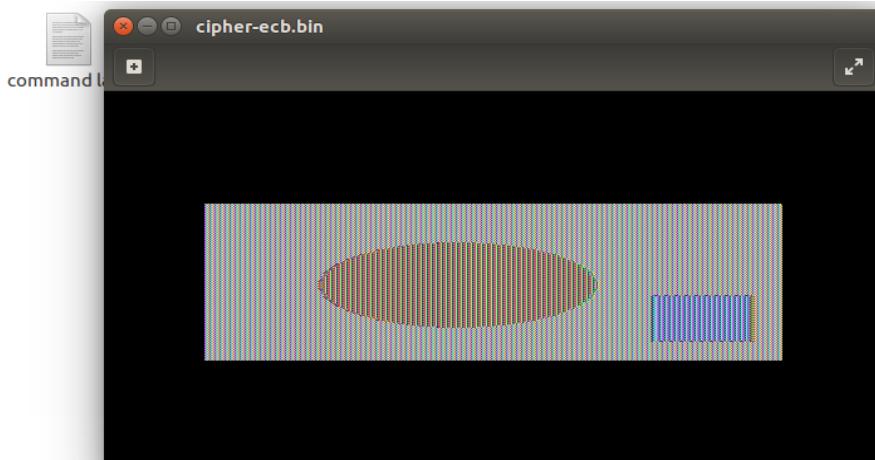


Command Executed:

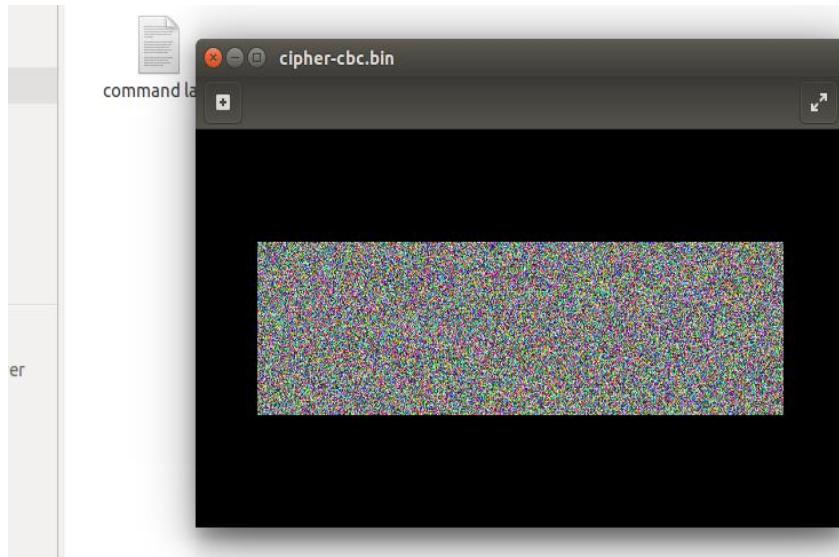
```
openssl enc -des-cbc -e -in pic_original.bmp -out cipher-cbc.bin -K 00112233445566 -iv 01020304050607
```

1. Image after replacing header field first 54 bytes of cipher-cbc.bin and cipher-ecb.bin with the first 54 bytes header field of original file pic_original.bmp

cipher-ecb.bin



cipher-cbc.bin



Cannot derive any useful information from the encrypted file in CBC mode. However In ECB mode the overall image may still be discerned as the pattern of identically colored pixels in the original remains in the encrypted version.

Task 3:

1. Text file more than 64 bytes long.

text file: Task3-input.txt



2. Encryption using AES-128 with all 4 modes.

Commands Executed:

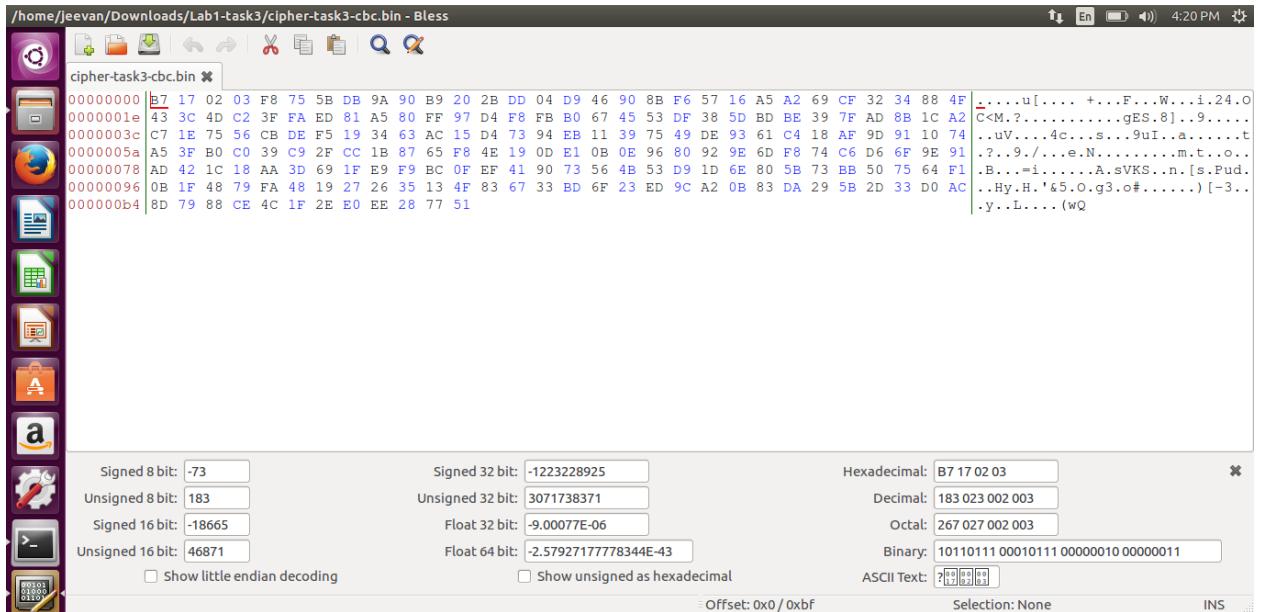
```
openssl enc -aes-128-ecb -e -in Task3-input.txt -out cipher-task3-ecb.bin -K 1234512345
```

```
openssl enc -aes-128-cbc -e -in Task3-input.txt -out cipher-task3-cbc.bin -K 1234512345 -iv 5432154321
```

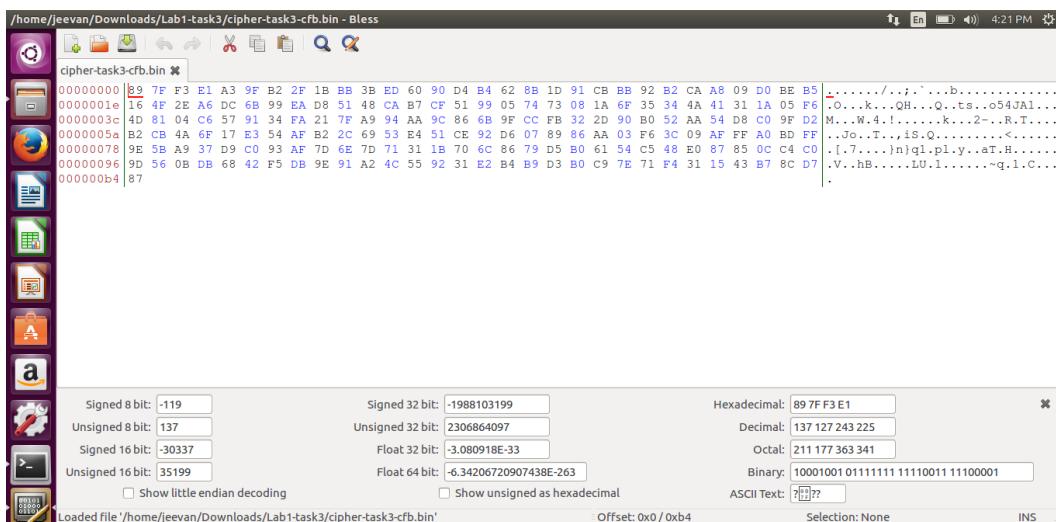
```
openssl enc -aes-128-cfb -e -in Task3-input.txt -out cipher-task3-cfb.bin -K 1234512345 -iv 5432154321
```

```
openssl enc -aes-128-ofb -e -in Task3-input.txt -out cipher-task3-ofb.bin -K 1234512345 -iv 5432154321
```

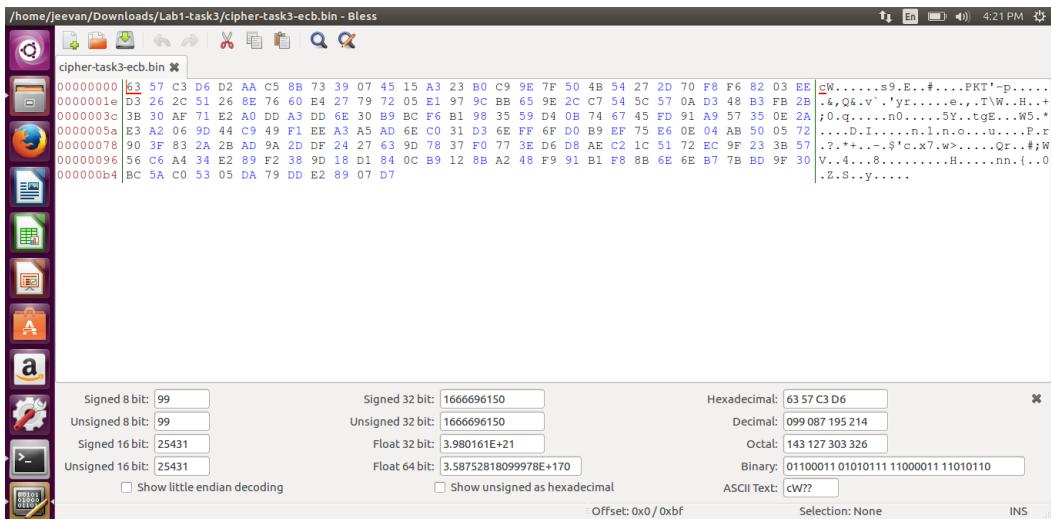
AES-128-CBC



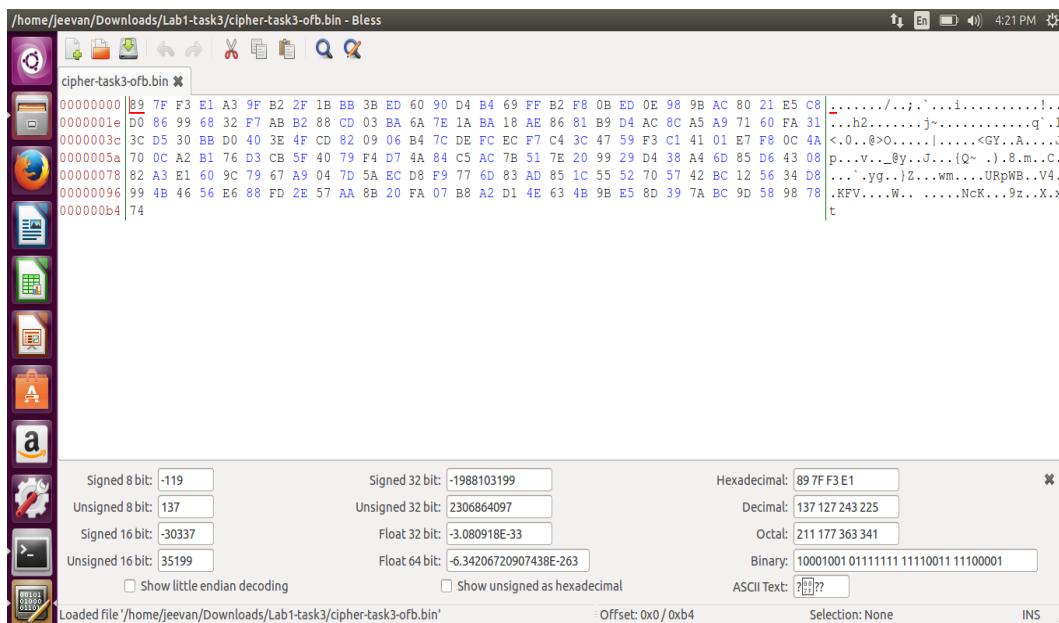
AES-128-CFB



AES-128-ECB

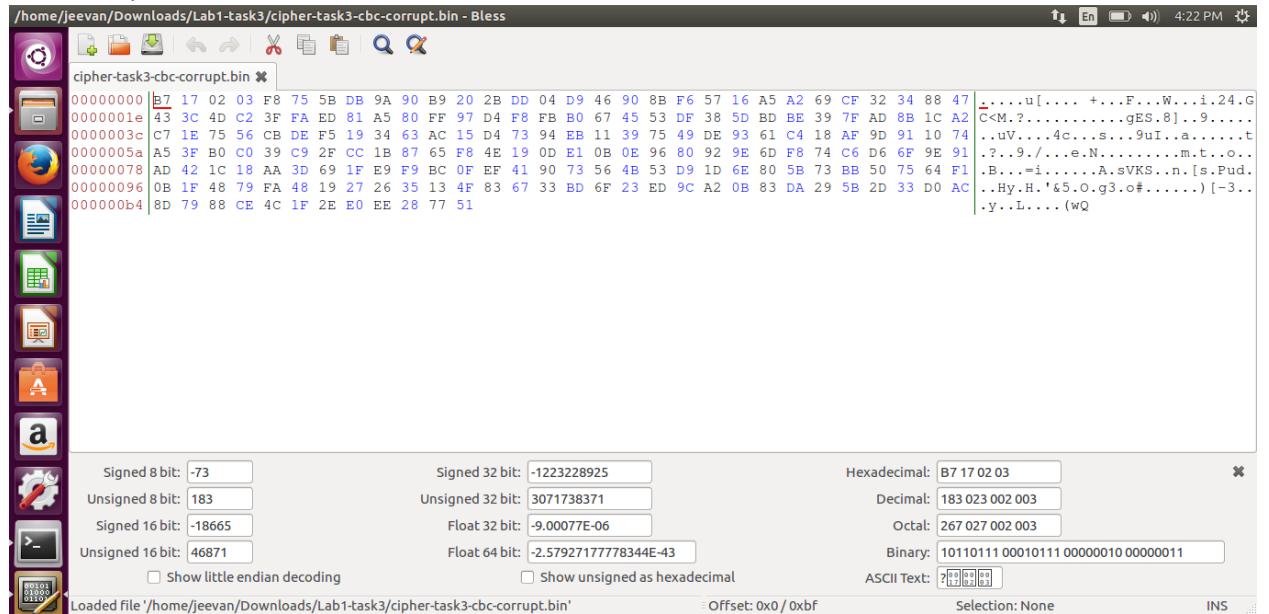


AES-128-OFB

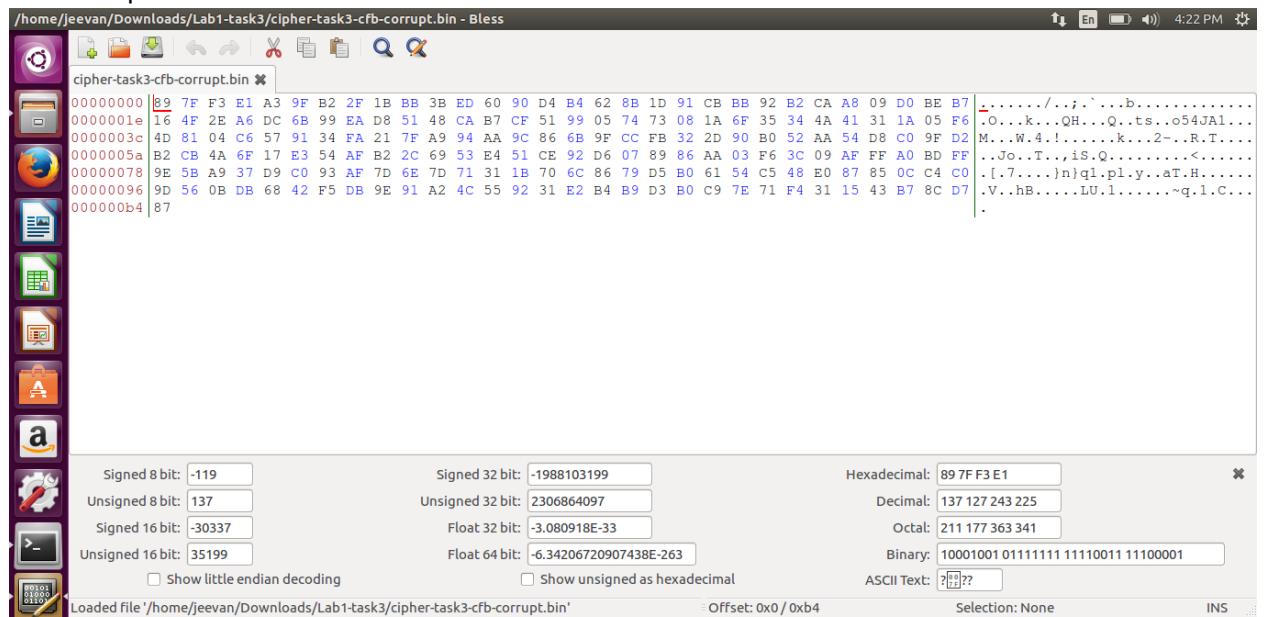


3. Corrupting 30th byte of all encrypted files.

CBC-corrupt



CFB-corrupt



ECB-corrupt

The screenshot shows the Bless hex editor interface with the file 'cipher-task3-ecb-corrupt.bin' loaded. The left pane displays the raw binary data in hex format, with the first few bytes being 63 57 C3 D6 D2 AA C5 8B 73 39 07 45 15 A3 23 B0 C9 9E 7F 50 4B 54 27 2D 70 F8 F6 82 03 EF. The right pane shows the ASCII representation of the data, which includes non-printable characters like carriage returns and line feeds. Below the panes are various conversion tools for different data types (Signed 8-bit, Unsigned 8-bit, Signed 16-bit, Unsigned 16-bit, etc.) and a text input field for ASCII text.

OFB-corrupt

The screenshot shows the Bless hex editor interface with the file 'cipher-task3-ofb-corrupt.bin' loaded. The left pane displays the raw binary data in hex format, with the first few bytes being 89 7F F3 E1 A3 9F B2 2F 1B BB 3B ED 60 90 D4 B4 69 FF B2 F8 0B ED 0E 98 9B AC 80 21 E5 C9. The right pane shows the ASCII representation of the data, which includes non-printable characters like carriage returns and line feeds. Below the panes are various conversion tools for different data types (Signed 8-bit, Unsigned 8-bit, Signed 16-bit, Unsigned 16-bit, etc.) and a text input field for ASCII text.

4. Decrypt all above encryption with their correct key and nonce values.

CBC-Decryption



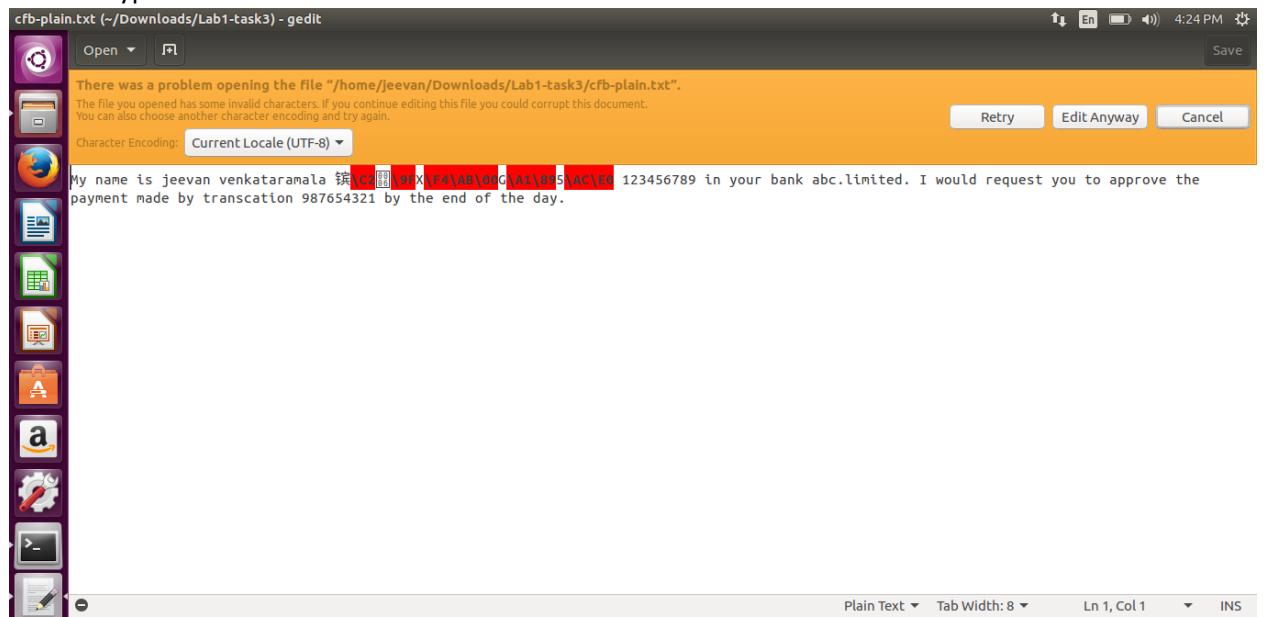
cbc-plain.txt (~/Downloads/Lab1-task3) - gedit

Open Save

Plain Text Tab Width: 8 Ln 1, Col 1 INS

```
My name is jeevan venkataramala holding an ac(no 123456789 in your bank abc.limited. I would request you to approve the payment made by transcation 987654321 by the end of the day.
```

CFB-Decryption



cfb-plain.txt (~/Downloads/Lab1-task3) - gedit

Open Save

Plain Text Tab Width: 8 Ln 1, Col 1 INS

There was a problem opening the file "/home/jeevan/Downloads/Lab1-task3/cfb-plain.txt".
The file you opened has some invalid characters. If you continue editing this file you could corrupt this document.
You can also choose another character encoding and try again.

Character Encoding: Current Locale (UTF-8) Retry Edit Anyway Cancel

```
My name is jeevan venkataramala 123456789 in your bank abc.limited. I would request you to approve the payment made by transcation 987654321 by the end of the day.
```



```
openssl enc -aes-128-ofb -d -in cipher-task3-ofb-corrupt.bin -out ofb-plain.txt -K 1234512345 -iv 5432154321
```

1. How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?

Sol.

Before Task:

ECB: Data will be encrypted in blocks. The encrypted block whose bit is corrupted should map to some other plaintext. Resulting in failure to recognize only that plaintext block.

CBC: Bit change to the cipher text should cause complete corruption of the corresponding block of plaintext and also invert the corresponding bit in the following block of plaintext, but rest of the blocks should remain intact.

CFB: One-bit change in the ciphertext should affect two plaintext blocks: a one-bit change in the corresponding plaintext block, and complete corruption of the following plaintext block. Rest of the blocks intact.

OFB: One-bit change in ciphertext should affect only corresponding single bit in plaintext.

After Task:

ECB: Satisfies as stated. Each block is of size 16 bytes and the block containing corrupted bit has been mapped to a random plaintext.

CBC: Satisfies as stated. The corresponding block in plaintext is corrupted as well as the corresponding bit in next plaintext block. Where a space is mapped to symbol "(".

CFB: Satisfies as stated. The corresponding corrupt bit is mapped to a different character in plaintext and the next block is also mapped to some other plaintext.

OFB: Only one bit has changed also satisfies as expected.

2) Reasons:

ECB: It decrypts identical cipher blocks into identical plaintext blocks and each block is independent of other block.

CBC: Corrupt bit block should map to some other plain text. And the corresponding bit in next block also corrupted because the corrupted ciphertext block is fed as input.

CFB: Next block of corresponding corrupted block maps to some other plaintext because the corrupted ciphertext block is fed as input to encryption along with key.

OFB: Error does not propagate because each decryption block is independent of ciphertext of previous block. Which results in error of the only bit that is corrupted.

3) Implications:

Single bit error in ciphertext would result in 1 corrupted block in ECB, 2 blocks in CBC and CFB(1 whole block and 1 bit in other block). And only a single bit in OFB.

Error propagation: ECB and OFB do not propagate error. CBC and ECB does.

Lab 2

Crypto Lab – One-Way Hash Function and MAC

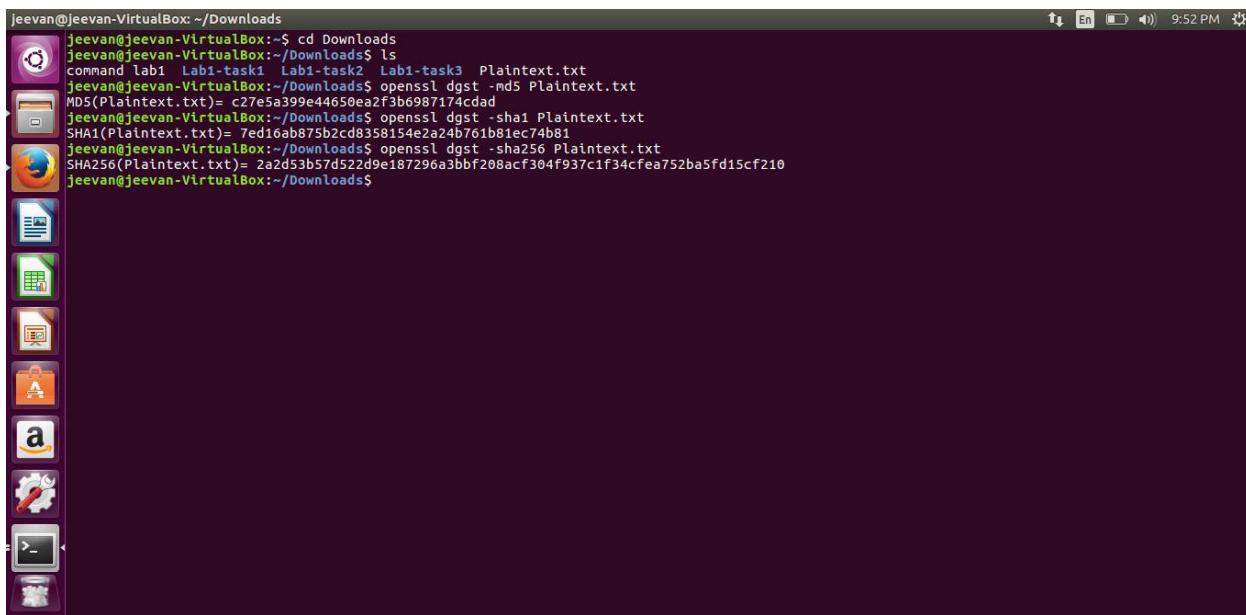
Task 1: Generating Message Digest and MAC

In this task, we will play with various one-way hash algorithms. You can use the following openssl dgst command to generate the hash value for a file. To see the manuals, you can type man openssl and man dgst.

```
% openssl dgst dgstype filename
```

Please replace the dgstype with a specific one-way hash algorithm, such as -md5, -sha1, -sha256, etc. In this task, you should try at least 3 different algorithms, and describe your observations. You can find the supported one-way hash algorithms by typing "man openssl".

Observations



```
jeevan@jeevan-VirtualBox: ~/Downloads
jeevan@jeevan-VirtualBox:~$ cd Downloads
jeevan@jeevan-VirtualBox:~/Downloads$ ls
command lab1 Lab1-task1 Lab1-task2 Lab1-task3 Plaintext.txt
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 Plaintext.txt
MD5(Plaintext.txt)= c27e5a399e44650ea2f3b6987174cdad
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 Plaintext.txt
SHA1(Plaintext.txt)= 7ed16ab875b2cd8358154e2a24b701b81ec74bb1
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 Plaintext.txt
SHA256(Plaintext.txt)= 2azd53b57d522d9e187296a3bbf208acf304f937cf34cfea752ba5fd15cf210
jeevan@jeevan-VirtualBox:~/Downloads$
```

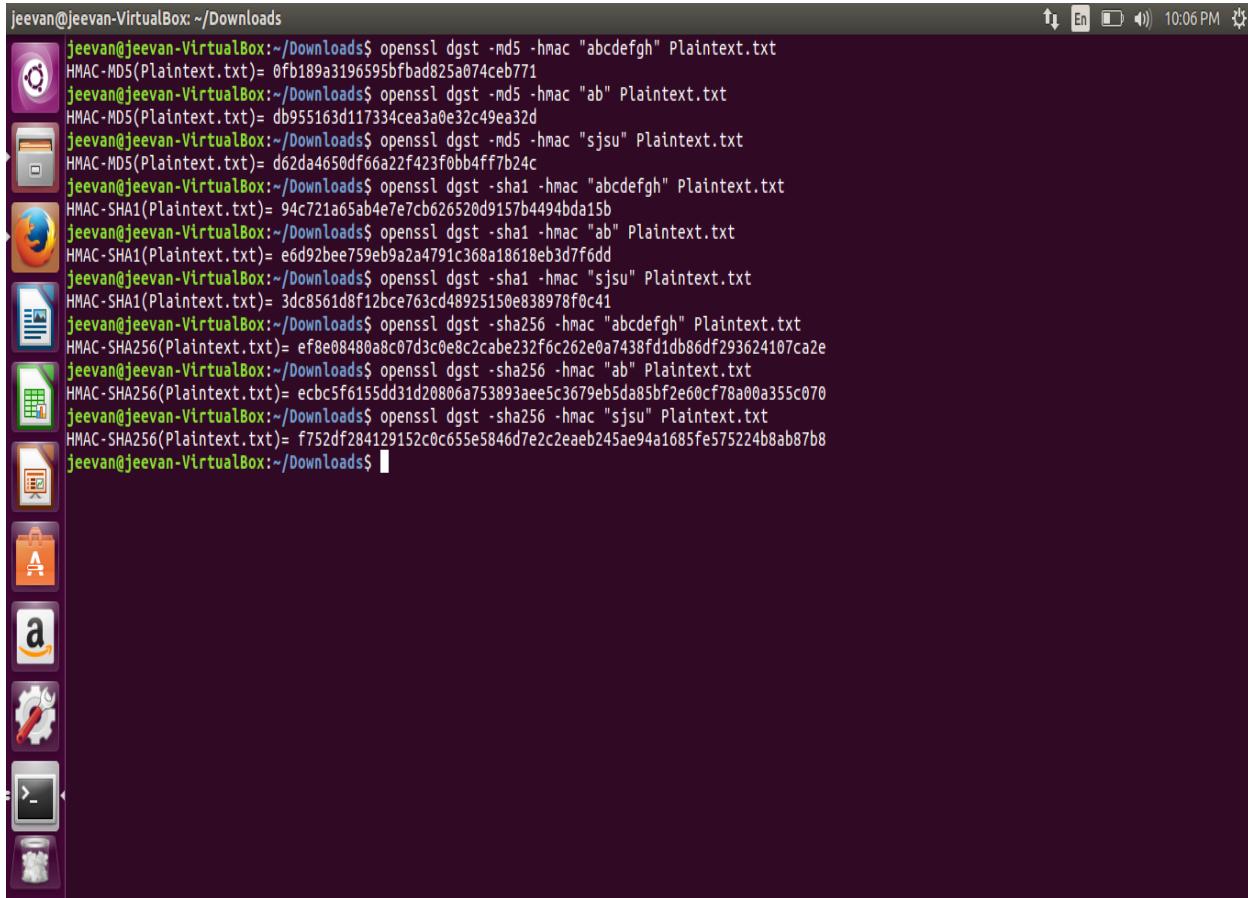
I have created a file called “Plaintext.txt” in Downloads directory. After that, we’ve implemented the three algorithms, md5, sha-1 and sha-256. The results can be seen in the figure.

From the results, it can be inferred that the length of hash value generated by md5, sha-1 and sha-256 are 32, 40 and 64 respectively. It follows from the fact that hash value of md5, sha-1 and sha-256 are 128, 140 and 256 bits each. It can also be inferred that SHA-256 algorithm is the most secure of all the three algorithms.

Task 2: Keyed Hash and HMAC

Please generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 for any file that you choose. Please try several keys with different length. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

Observation:



```
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 -hmac "abcdefg" Plaintext.txt
HMAC-MD5(Plaintext.txt)= 0fb189a3196595bfbad825a074ceb771
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 -hmac "ab" Plaintext.txt
HMAC-MD5(Plaintext.txt)= db955163d117334cea3a0e32c49ea32d
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 -hmac "sjsu" Plaintext.txt
HMAC-MD5(Plaintext.txt)= d62da4650df66a22f423f0bb4ff7b24c
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 -hmac "abcdefg" Plaintext.txt
HMAC-SHA1(Plaintext.txt)= 94c721a65ab4e7e7cb626520d9157b4494bda15b
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 -hmac "ab" Plaintext.txt
HMAC-SHA1(Plaintext.txt)= e0d92bee759eb9a2a4791c368a18618eb3d7f6dd
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 -hmac "sjsu" Plaintext.txt
HMAC-SHA1(Plaintext.txt)= 3dc8561d8f12bce763cd48925150e838978f0c41
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 -hmac "abcdefg" Plaintext.txt
HMAC-SHA256(Plaintext.txt)= ef8e08480a8c07d3c0e8c2cabef232f6c262e0a7438fd1db86df293624107ca2e
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 -hmac "ab" Plaintext.txt
HMAC-SHA256(Plaintext.txt)= ecbc5f6155dd31d20806a753893aee5c3679eb5da85bf2e60cf78a00a355c070
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 -hmac "sjsu" Plaintext.txt
HMAC-SHA256(Plaintext.txt)= f752df284129152c0c655e5846d7e2c2eaeb245ae94a1685fe575224b8ab87b8
jeevan@jeevan-VirtualBox:~/Downloads$
```

For this task, we created a keyed hash for Plaintext.txt file. We generated several keyed hash values using HMAC –md5, HMAC –sha1 and HMAC -256 with several different keys. The results of this can be seen in the figure above.

From the results above, it is clear that HMAC values length is same with the corresponding hash code length. Based on this, it is easily established that we do not need to use a key with fixed size and in any way the HMAC algorithm is quite flexible.

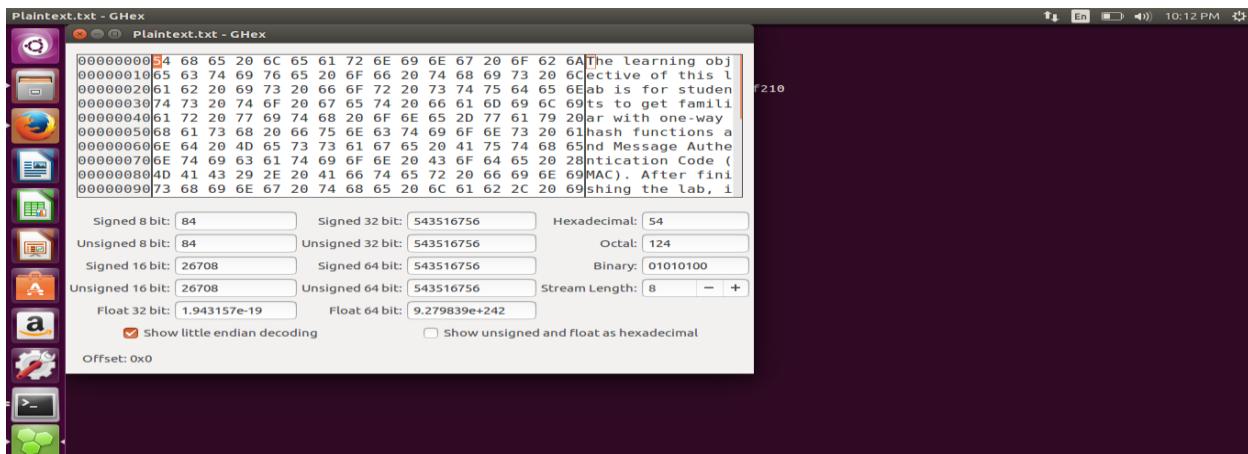
Task 3: The Randomness of One-way Hash

To understand the properties of one-way hash functions, we would like to do the following exercise for MD5 and SHA256

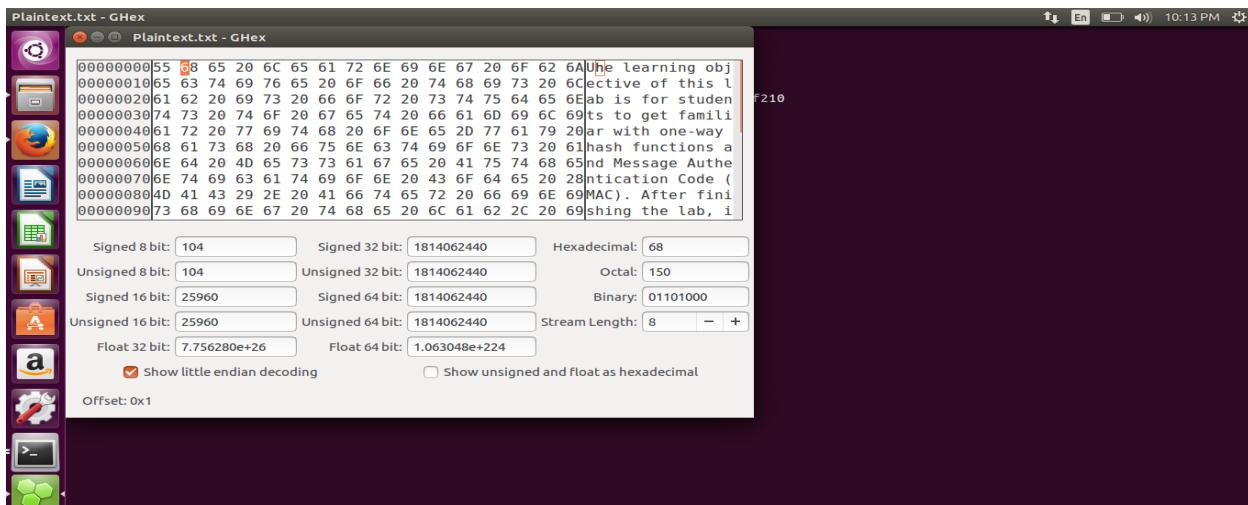
1. Create a text file of any length.
2. Generate the hash value H1 for this file using a specific hash algorithm.
3. Flip one bit of the input file. You can achieve this modification using ghex or Bless.
4. Generate the hash value H2 for the modified file.
5. Please observe whether H1 and H2 are similar or not. Please describe your observations in the lab report. You can write a short program to count how many bits are the same between H1 and H2.

Observation:

Original file:



After flip:



Comparison before and after flip

```
jeevan@jeevan-VirtualBox: ~/Downloads
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 Plaintext.txt
MD5(Plaintext.txt)= c27e5a399e44650ea2f3b6987174cdad
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 Plaintext.txt
SHA1(Plaintext.txt)= 7ed16ab875b2cd8358154e2a24b761b81ec74b81
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 Plaintext.txt
SHA256(Plaintext.txt)= 2a2d53b57d522d9e187296a3bbf208acf304f937c1f34cfea752ba5fd15cf210
jeevan@jeevan-VirtualBox:~/Downloads$ ghex Plaintext.txt
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -md5 Plaintext.txt
MD5(Plaintext.txt)= b175ee556d844afa584daf30f2e8a1df
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha1 Plaintext.txt
SHA1(Plaintext.txt)= 1060709d19de0d4810a892a3826a3d323e0fd383
jeevan@jeevan-VirtualBox:~/Downloads$ openssl dgst -sha256 Plaintext.txt
SHA256(Plaintext.txt)= 0e1346418f8f01f0c599fae8e7f4e683eec70ee37bd5ed1e1a15eb1312dfb256
jeevan@jeevan-VirtualBox:~/Downloads$
```

Generated hash values for plaintext.txt. Flipped a single bit of first byte and again generated hash value. It can be concluded a single bit change produces a completely different hash value. H1 and H2 are not similar.

md5

H1: c27e5a399e44650ea2f3b6987174cdad

H2: b175ee556d844afa584daf30f2e8a1df

no of similar bits: 61

sha1

H1: 7ed16ab875b2cd8358154e2a24b761b81ec74b81

H2: 1060709d19de0d4810a892a3826a3d323e0fd383

No of similar bits: 89

sha256

H1: 2a2d53b57d522d9e187296a3bbf208acf304f937c1f34cfea752ba5fd15cf210

H2: 0e1346418f8f01f0c599fae8e7f4e683eec70ee37bd5ed1e1a15eb1312dfb256

No of bits similar: 125

Program: To count similar bits.

```
/* Created by Jeevan Venkataramana */

#include<iostream>
#include<string>
int to_decimal(char a);
using namespace std;
int main()
{
    string a, b;
    cout<<endl<<"enter hash 1: ";
    cin>>a;
    cout<<endl<<"enter hash 2: ";
    cin>>b;
    int n= a.length();
    int count=0;
    for(int i=0; i<n; i++)
    {
        int x= to_decimal(a[i]);
        int y= to_decimal(b[i]);
        int xor_result = x^y;

        for(int j=0; j<4; j++)
        {
            if(xor_result % 2 == 0)
            {
                count++;
            }
            xor_result = xor_result/2;
        }
    }
    cout<<endl<<" No of bits similar: "<<count;
    return 0;
}

int to_decimal(char a)
{
    if(a =='0')return 0;
    if(a =='1')return 1;
    if(a =='2') return 2;
    if(a =='3') return 3;
    if(a =='4') return 4;
    if(a =='5') return 5;
    if(a =='6') return 6;
    if(a =='7') return 7;
    if(a =='8') return 8;
    if(a =='9') return 9;
    if(a =='a') return 10;
    if(a =='b') return 11;
    if(a =='c') return 12;
    if(a =='d') return 13;
    if(a =='e') return 14;
    if(a =='f') return 15;
}
```

Task 4: One-Way Property versus Collision-Free Property

- 1) How many trials it will take you to break the one-way property using the brute-force method?
You should repeat your experiment for multiple times and report your average number of trials.

For any given h , computationally infeasible to find y such that $H(y) = h$ With m -bit hash code, effort required proportional to 2^m We have 24 bit hash code, effort required is 2^{24} (theoretically)

- 2) How many trials it will take you to break the collision-free property using the brute-force method? Similarly, you should report the average.

It is hard to find two inputs that hash to the same output; that is, two inputs a and b such that $H(a) = H(b)$ With m -bit hash code, effort required proportional to $2^{(m/2)}$. We have 24 bit hash code, effort required is 2^{12} (theoretically).

- 3) Based on your observation, which property is easier to break using the brute-force method?

collision resistant is easier to break then one way property

The screenshot shows the Spyder IDE interface. On the left, a code editor displays a Python script for generating hash collisions. The script uses numpy to read bytes from a file, converts them to binary strings, and then tries to find collisions by comparing them. A tooltip 'Usage' is shown over the code editor, explaining how to get help for objects. On the right, an IPython console window is open, showing the execution of the script and its output. The output includes the version of Python and Anaconda, copyright information, and the results of the collision search.

```
1 import numpy as np
2
3
4 number = 0
5 hashfile = []
6
7
8 # initializing with m zeros
9 h = np.binary_repr(0).rfill(24)
10 for m in range(0,1000):
11     for l in range(0, m):
12         # converting
13         a = int(h[16:], 2)
14         b = np.binary_repr(int(h[:16]), 2)
15         # conversion integer into binary string
16         b2 = "{0:b}".format(b)
17         # padding with 0's on left until 8 bits
18         c = b2.rfill(8)
19         # adding the converted bit to
20         h = h[0:16] + c
21
22         # left circular shift by 4 bits
23         h = h[4:] + h[0:4]
24         hashfile.append(h)
25     number = number + 1
26
27 # counting for collisions
28 count=0
29 for k in range(0, 1000):
30     for l in range(k + 1, 1000):
31         if hashfile[k] == hashfile[l]:
32             count = count + 1
33
34 print("number of two way collision")
35 print(count)
36
37 count1=0
38 for l in range(1, 1000):
39     if hashfile[55] == hashfile[l]:
40         count1 = count1 + 1
41 print("number of one way collision")
42 print(count1)
43
44
```

Numbers are from 0 to 1000, Two-way collision are 43. Thus, one-way property is satisfied on four instances.

Lab 3

RSA Public-Key Encryption and Signature Lab

Task1: Deriving the Private Key

The screenshot shows a Windows desktop environment. In the center is a Python 3.6.4 Shell window titled "Python 3.6.4 Shell". The code being run is a Python script named "lab3-Task1.py". The script contains functions for calculating the GCD and finding the modular inverse, and it uses these to calculate the private key "d" from given public key components "p", "q", and "e". The output of the script shows the calculated value of "d" in both decimal and hexadecimal formats.

```
lab3-Task1.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task1.py (3.6.4)
File Edit Format Run Options Window Help
def gcd(a,b):
    if a==0:
        return(b,0,1)
    else:
        g,y,x = gcd(b%a,a)
        return (g, x-(b//a)*y, y)

def find(a,m):
    g,x,y=gcd(a,m)
    if g!=1:
        return None
    else:
        return x*m

p="F7E75FDC469067FFDC4E847C51F452DF"
p= int(p,16)
q="E85CED54AF57E53E092113E62F436F4F"
q=int(q,16)
phi= (p-1)*(q-1)
e="DDB8C3"
e=int(e,16)

d=find(e,phi)
print(d)

hexofd=hex(d)
print(hexofd)

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task1.py
24212225287904763939160097464943268930139828978795606022583874367720623008491
0x3587a24598e5f2a21db007d89d18cc50aba5075ba19a33890fe7c28a9b496aeb
>>> |
```

Private key:

Integer

d = 24212225287904763939160097464943268930139828978795606022583874367720623008491

Hexadecimal

d = 0x3587a24598e5f2a21db007d89d18cc50aba5075ba19a33890fe7c28a9b496aeb

Task 2: Encrypting a Message

The screenshot shows a Windows desktop environment. In the center is a Python 3.6.4 Shell window titled "Python 3.6.4 Shell". The window displays the following content:

```
lab3-Task2.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task2.py (3.6.4)
File Edit Format Run Options Window Help
def crypto(F,z):
    if z==0:
        return 1
    if z==1:
        return F
    w=z//2
    r=z%2
    if r==1:
        return crypto(F*F%n, w)*F%n
    else:
        return crypto(F*F%n, w)

p="E7E75FDC469067FFDC4E847C51F452DF"
p=int(p,16)
q="E85CED54AF57E53E092113E62F436F4F"
q=int(q,16)
n="DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"
n=int(n,16)
e="010001"
d="74B006F9F3A62BAE331FFE3F0A68AFE35B3D2E4794140AACBC26AA381CD7D30D"
phi=(p-1)*(q-1)
e=int(e,16)
print(e)
d1=int(d,16)
m="A top secret!".encode('utf-8')
en=m.hex()
print(en)
en1=int(en,16)
encrypted=crypto(en1,e)
encrypted1=hex(encrypted)
print(encrypted1)
encrypted1=int(encrypted1,16)
decrypted=crypto(encrypted1,d1)
decrypted=hex(decrypted)
print(decrypted)
decrypted1=decrypted[2:]
print(bytes.fromhex(decrypted1))
```

The output of the script is displayed in the shell window:

```
>>> 65537
4120746f702073656372657421
0x6fb078da550b2650832661e14f4f8d2cfaef475a0df3a75cacdc5de5fcf5fad
0x4120746f702073656372657421
b'A top secret!'
>>> |
```

Below the shell window is a Notepad window with the same content as the shell window, showing the Python script code.

Ciphertext = 0x6fb078da550b2650832661e14f4f8d2cfaef475a0df3a75cacdc5de5fcf5fad

Decryption: A top secret! got the plaintext back.

Task 3: Decrypting a Message

The screenshot shows a Windows desktop environment. In the center is a Python 3.6.4 Shell window titled "Python 3.6.4 Shell". The shell window displays the following text:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task3.py

0x50617373776f72642069732064656573
50617373776f72642069732064656573
>>> |
```

To the left of the shell window is a code editor window titled "lab3-Task3.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task3.py (3.6.4)". The code editor contains the following Python script:

```
def gcd(x,y):
    if y==0:
        return x
    else:
        return gcd(y,x%y)

def find(a,m):
    g,x,y=gcd(a,m)
    if g!=1:
        return None
    else:
        return x*m

def decrypt(F,z):
    if z==0:
        return 1
    if z==1:
        return F
    w=z//2
    r=z%2
    if r==1:
        return decrypt(F*F%n, w)*F%n
    else:
        return decrypt(F*F%n, w)

p="E7E75FDC469067FFDC4E847C51F452DF"
p= int(p,16)
q="E85CED54AF57E53E092113E62F436F4F"
q=int(q,16)
n="1C9FE351F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"
n=int(n,16)
c="5C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F"
c=int(c,16)
d="74B806F93A62BAE331FFE3F0A68AFE35B3D2E4794148ACBC26AA381CD7D30D"
d=int(d,16)
decrypted=decrypt(c,d)
decrypted1=hex(decrypted)
print(decrypted1)
decrypted1=decrypted1[2:]
print(decrypted1)
```

The task involves decrypting a message using the provided Python script. The decrypted message is shown in the shell window.

Plaintext hexadecimal string: 0x50617373776f72642069732064656573

Converting in ASCII format:

Plaintext: "Password is dees"

Task 4: Signing a Message

The screenshot shows a Windows desktop environment with two open windows. The top window is titled "Python 3.6.4 Shell" and displays a Python script named "lab3-Task4.py". The script contains code for a modular exponentiation function and a main block that calculates a hash of a message "I owe you \$2000.", converts it to hex, and then encrypts it using a public key. The resulting encrypted message is printed. The bottom window is also a "Python 3.6.4 Shell" window, showing the same process but with a different hash value ("I owe you \$3000.") and a different encrypted message.

```
lab3-Task4.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task4.py (3.6.4)
File Edit Format Run Options Window Help
    return None
else:
    return x%m

def crypto(F,z):
    if z==0:
        return 1
    if z==1:
        return F
    w=z//2
    r=z%2
    if r==1:
        return crypto(F*F%n, w)*F%n
    else:
        return crypto(F*F%n, w)

p="F7E75FDC469067FFDC4E847C51F452DF"
p=int(p,16)
q="85CED54AF57E53E092113E62F436F4F"
q=int(q,16)
n="DCEFF3E51F62E09CE7032E2677A78946A849DC4CDDE3A4DOC81629242FB1A5"
n=int(n,16)
phi=(p-1)*(q-1)
e="10001"
e=int(e,16)
print(e)
d="4D806F9F3A62BAE331FFE3FOA68AFE35B3D2E4794148AACBC26AA381CD7D30D"
d1=int(d,16)
m="I owe you $2000.".encode('utf-8')
en=m.hex()
en1=int(en,16)
encrypted= crypto(en,d1)
encrypted=hex(encrypted)
print(encrypted)
encrypted1=int(encrypted,16)
decrypted=crypto(encrypted1,e)
decrypted=hex(decrypted)
decrypted=decrypted[2:]
print(bytes.fromhex(decrypted))

RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task4.py
65537
0x5a4e7f17f04ccfe2766eleb32adddba890bbe92a6fbe2d785ed6e73ccb35e4cb
b'I owe you $2000.'
>>>
```

```
lab3-Task4.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task4.py (3.6.4)
File Edit Format Run Options Window Help
    return None
else:
    return x%m

def crypto(F,z):
    if z==0:
        return 1
    if z==1:
        return F
    w=z//2
    r=z%2
    if r==1:
        return crypto(F*F%n, w)*F%n
    else:
        return crypto(F*F%n, w)

p="F7E75FDC469067FFDC4E847C51F452DF"
p=int(p,16)
q="85CED54AF57E53E092113E62F436F4F"
q=int(q,16)
n="DCEFF3E51F62E09CE7032E2677A78946A849DC4CDDE3A4DOC81629242FB1A5"
n=int(n,16)
phi=(p-1)*(q-1)
e="10001"
e=int(e,16)
print(e)
d="4D806F9F3A62BAE331FFE3FOA68AFE35B3D2E4794148AACBC26AA381CD7D30D"
d1=int(d,16)
m="I owe you $3000.".encode('utf-8')
en=m.hex()
en1=int(en,16)
encrypted= crypto(en,d1)
encrypted=hex(encrypted)
print(encrypted)
encrypted1=int(encrypted,16)
decrypted=crypto(encrypted1,e)
decrypted=hex(decrypted)
decrypted=decrypted[2:]
print(bytes.fromhex(decrypted))

RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task4.py
65537
0xbcc20fb7568e5d48e434c387c06a6025e90d29d848af9c3ebac0135d99305822
b'I owe you $3000.'
>>> |
```

The difference in signature is 253 bits as hash function is used.

Task 5: Verifying a Signature

The image shows a Windows desktop environment with two Python 3.6.4 Shell windows open. Both windows display the same code and command-line interaction. The taskbar at the bottom shows various pinned icons and the date/time as 5/12/2018 4:08 AM.

```
lab3-Task5.py - C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task5.py (3.6.4)
File Edit Format Run Options Window Help
def find(a,m):
    g,x,y=ggcd(a,m)
    if g!=1:
        return None
    else:
        return x*m

def crypto(F,z):
    if z==0:
        return 1
    if z==1:
        return F
    w=z//2
    r=z%2
    if r==1:
        return crypto(F*F%n, w)*F%n
    else:
        return crypto(F*F%n, w)

p="F7E75FDC469067FFDC4E847C51F452DF"
p=int(p,16)
q="885CED54AF57E53E092113E62F436F4F"
q=int(q,16)
n="A1CD4DC432788D933779FBBD46C6E1247F0CF1233595113AA51B450F18116115"
n=int(n,16)
phi=(p-1)*(q-1)
e="010001"
e=int(e,16)
d="4B006F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D"
d1=int(d,16)
m="Launch a missile."
s="43DF34902D8C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CDBB6802F"
s=int(s,16)
decrypted=crypto(s,e)
decrypted=hex(decrypted)
decrypted1=decrypted[2:]
print(bytes.fromhex(decrypted1))

b'Launch a missile.'
```

Python 3.6.4 Shell

```
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task5.py

b'Launch a missile.'
```

Python 3.6.4 Shell

```
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/jeeva/AppData/Local/Programs/Python/Python36-32/lab3-Task5.py

b'\x91G\x19\xc8\r\xf1\xe4,\x150\xb4c\x8c\xe8\xbcrm=\f\xc8:N\xb6\xb7\xbe\x02\x03\xb4\x1a\xc2\x94'
```

When signed with 2F it gives back plaintext. When signed with 3F it gives a different string of different length.

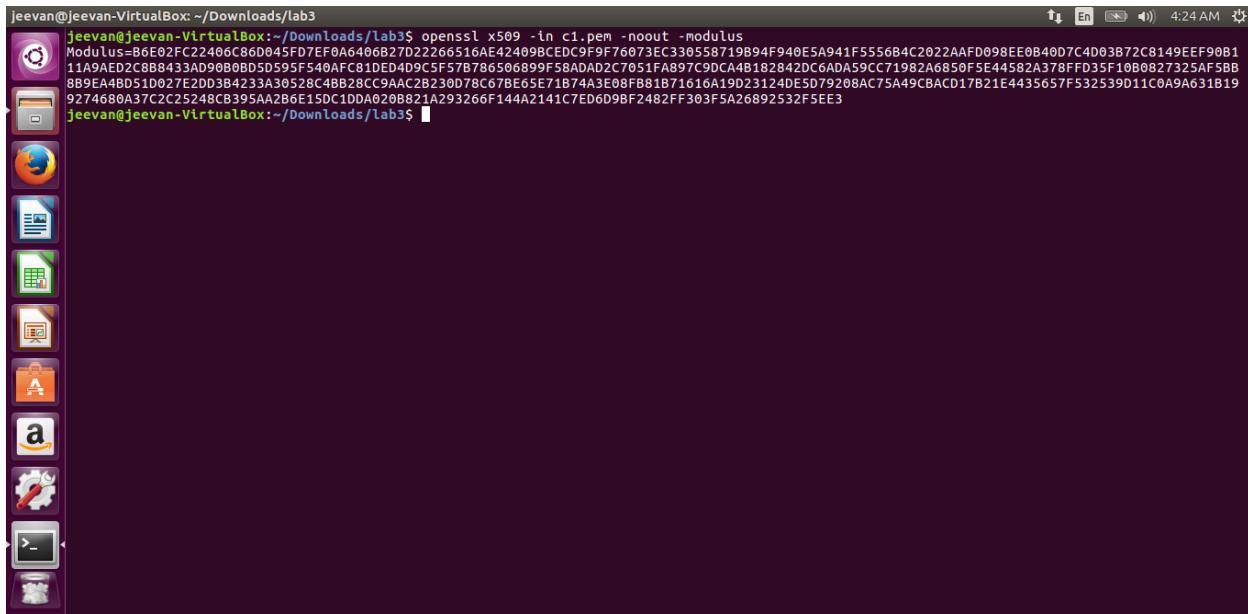
Task 6: Manually Verifying an X.509 Certificate

Step 1: Download a certificate from a real web server

```
jeevan@jeevan-VirtualBox:~$ openssl s_client -connect www.example.org:443 -showcerts
CONNECTED(0x000000)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
Verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
Verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet Corporation for Assigned Names and Numbers, OU = Technology, CN = www.example.org
Verify return:1
Certificate chain
0 s:/US/ST=California/L=Los Angeles/O=Internet Corporation for Assigned Names and Numbers/OU=Technology/CN=www.example.org
i:/US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High Assurance Server CA
-----BEGIN CERTIFICATE-----
MIIF8jCBNnqAwIBAgIQDmTF+8I2reFLFyrrQceMsDANBgkqhkiG9w0BAQsFADBw
M0swCQDVQQEwJVUzEVMBMGaIUEChMMRGLnaULncnQg5W5jMRkwFwDVQQLExB3
d3cuzGlnaWNlcnQuY29tMSwLQYDVQDQEZEawdpz2VycDBTSEEyIhpZggQXNz
dJhbhNlIxFNcnZlcLB0TDaEfFw0NTExMDwMDAwDBafw0x0DEXhjgXjAwMDba
MIGLMQswCQDVQGEwJVUzETMBEGA1UECBMKQ2saWzvcm5pYTEUMBIGA1UEBXML
TG9zIEfu2zVsZXMPDA6bgNVBAoTM0LuGvbyvW01ENvcnBvcnF0aW9uIGZvcLB1
c3NpZ2slZC80Y11cybhbhM0gTnVtmycZETMBEGA1UECxMKVGVjaG5vbG9neTEY
MBYGA1uEaxMPd3d3lm4Y1wbGuub3jnM1IBI1ANBgkqhkiG9w0BAQFAOA08A
MIIBGCKCAQEA0sCNL2Fj01Xb1l1RfFv0E0kzLjmg9LAC3bcBjgSH6N1VV0zdt6U
Xfz15btm7F3K7rfuBykL078nraM9q1zrHoIeyoFrV/n+PZZJa0sPjCPxMEJnRo
D8Z4KpWK0LyDu1SputoI4n0l/hEtHt1nuoBfNzxF7WxcxGwEsZuSiKxIkH15V
RJ0reKFHTaCxB1qcZ/RAbIv6yhvxKvTwddT4cl6gfHccexgMaSl328Fgs3
jYrvG29Pue6V3i/tbbPu6qTfp/Hibrqdjh29n52Bhb0fJKM9DhxCP/Cattcc7a
zBEtCvFu2zVsZXMPDA6bgNVBAoTn0IAQBo41UCDCAAkwhWvDVR0jBBgw
FoAUUWj/KKC8B3U8zNLzGKtErhZcjswhQDYVR00DBYEFKZPb4fdhIn850gKpUw
50ia0n5IMICB8gNVHREejB4gg93d3cu2XhhbxBSzSSvcmcC2V4Y1wbGUuY29t
ggteGftcGx1LmVkdY1LzXhhbxBSzSSu2XSC2V4Y1wbGUuB3jngg93d3cuZkh
bxBSzSSjB22C0d3d3dy5leGftcGxLmVkdY1Pd3d3lm4Y1wbGUuBmV0MA4GA1UD
DwEB/wQEAwIf0adBgnVHSUeFjAUUbgrBgfEB0cDAQYIKwYBQUHawTwdQYVR0f
BG4wbDA0OdkgM1YuahR0cDovl2NybdQuzGlnaWNlcnQuY29tL3NoYTIt
dmVtLwc0lMybAooDkgM1YuahR0cDovl2NybdQuzGlnaWNlcnQuY29tL3NoYTIt
aEtc2VydmyVtLwc0lMybDBMgBvNHSAsERTBMDcGCWCGSAGG/wBaTAqMcGCCSg
AQUFBwIBfxodhRwczoV1d3dy5kaWdpY2VydcsjB20vQ1BTMAgBmeBDAECAjCB
gwYIKwYBBQHQAQEEedZb1MCQCCsGAUFBzABhhodHRw0t8vzb2zcCSkaWdpY2V
dc5jB20WtQY1KwYBBQHMAKQWhdHA6lY9jYWNlcnRzlMrpZ2lJzX0lMnv59E
awdpQZydfNIQTJtaidoQnxndJhbhnlU2VydmyVq0eEu3J0MawGA1udEWB/wQ
MAAWDQYJKoZtIvhcNAQELB0AdggEBAsISohnGn2L0lJnSSJhuV23qM1RCIdvqoE
61s+C8ctrwR03U3x8q80H+2ahx1ompzd5al4XqzJLlji201p+hubBF1mNP
PZjbt2z2mz1pwVuMRM+2ppRMv6nV39F3vfUSHob4/jSeIUvPwYd8/krc+kP0wLvy
ieqrbcuFjmqfyPmuV1u90o14T0ikpwvTZUo2YZANP4c/gj4ry48/znmUaRvy2kvI
l7gRQ21qJTKSsu01yoNo3J9TpXPGU7Lydz/HwWw0DpArtAaukI8aNx4ohFUKS
wDSIIWIWj1GbEe100TfWvEWT0nbN1/faPxpk5IRxicapq1II=
-----END CERTIFICATE-----
1 s:/US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High Assurance Server CA
```

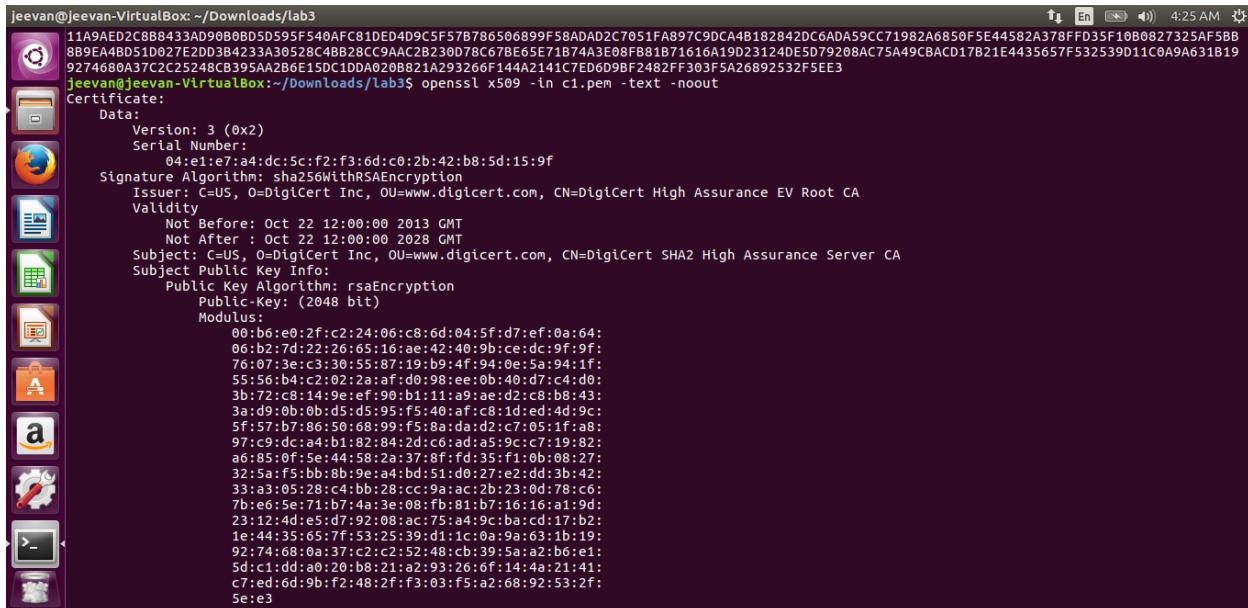
```
jeevan@jeevan-VirtualBox:~/Downloads/lab3
0 s:/US/ST=California/L=Los Angeles/O=Internet Corporation for Assigned Names and Numbers/OU=Technology/CN=www.example.org
i:/US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High Assurance Server CA
-----BEGIN CERTIFICATE-----
MIIF8jCBNnqAwIBAgIQDmTF+8I2reFLFyrrQceMsDANBgkqhkiG9w0BAQsFADBw
M0swCQDVQQEwJVUzEVMBMGaIUEChMMRGLnaULncnQg5W5jMRkwFwDVQQLExB3
d3cuzGlnaWNlcnQuY29tMSwLQYDVQDQEZEawdpz2VycDBTSEEyIhpZggQXNz
dJhbhNlIxFNcnZlcLB0TDaEfFw0NTExMDwMDAwDBafw0x0DEXhjgXjAwMDba
MIGLMQswCQDVQGEwJVUzETMBEGA1UECBMKQ2saWzvcm5pYTEUMBIGA1UEBXML
TG9zIEfu2zVsZXMPDA6bgNVBAoTM0LuGvbyvW01ENvcnBvcnF0aW9uIGZvcLB1
c3NpZ2slZC80Y11cybhbhM0gTnVtmycZETMBEGA1UECxMKVGVjaG5vbG9neTEY
MBYGA1uEaxMPd3d3lm4Y1wbGuub3jnM1IBI1ANBgkqhkiG9w0BAQFAOA08A
MIIBGCKCAQEA0sCNL2Fj01Xb1l1RfFv0E0kzLjmg9LAC3bcBjgSH6N1VV0zdt6U
Xfz15btm7F3K7rfuBykL078nraM9q1zrHoIeyoFrV/n+PZZJa0sPjCPxMEJnRo
D8Z4KpWK0LyDu1SputoI4n0l/hEtHt1nuoBfNzxF7WxcxGwEsZuSiKxIkH15V
RJ0reKFHTaCxB1qcZ/RAbIv6yhvxKvTwddT4cl6gfHccexgMaSl328Fgs3
jYrvG29Pue6V3i/tbbPu6qTfp/Hibrqdjh29n52Bhb0fJKM9DhxCP/Cattcc7a
zBEtCvFu2zVsZXMPDA6bgNVBAoTn0IAQBo41UCDCAAkwhWvDVR0jBBgw
FoAUUWj/KKC8B3U8zNLzGKtErhZcjswhQDYVR00DBYEFKZPb4fdhIn850gKpUw
50ia0n5IMICB8gNVHREejB4gg93d3cu2XhhbxBSzSSvcmcC2V4Y1wbGUuY29t
ggteGftcGx1LmVkdY1LzXhhbxBSzSSu2XSC2V4Y1wbGUuB3jngg93d3cuZkh
bxBSzSSjB22C0d3d3dy5leGftcGxLmVkdY1Pd3d3lm4Y1wbGUuBmV0MA4GA1UD
DwEB/wQEAwIf0adBgnVHSUeFjAUUbgrBgfEB0cDAQYIKwYBQUHawTwdQYVR0f
BG4wbDA0OdkgM1YuahR0cDovl2NybdQuzGlnaWNlcnQuY29tL3NoYTIt
dmVtLwc0lMybAooDkgM1YuahR0cDovl2NybdQuzGlnaWNlcnQuY29tL3NoYTIt
aEtc2VydmyVtLwc0lMybDBMgBvNHSAsERTBMDcGCWCGSAGG/wBaTAqMcGCCSg
AQUFBwIBfxodhRwczoV1d3dy5kaWdpY2VydcsjB20vQ1BTMAgBmeBDAECAjCB
gwYIKwYBBQHQAQEEedZb1MCQCCsGAUFBzABhhodHRw0t8vzb2zcCSkaWdpY2V
dc5jB20WtQY1KwYBBQHMAKQWhdHA6lY9jYWNlcnRzlMrpZ2lJzX0lMnv59E
awdpQZydfNIQTJtaidoQnxndJhbhnlU2VydmyVq0eEu3J0MawGA1udEWB/wQ
MAAWDQYJKoZtIvhcNAQELB0AdggEBAsISohnGn2L0lJnSSJhuV23qM1RCIdvqoE
61s+C8ctrwR03U3x8q80H+2ahx1ompzd5al4XqzJLlji201p+hubBF1mNP
PZjbt2z2mz1pwVuMRM+2ppRMv6nV39F3vfUSHob4/jSeIUvPwYd8/krc+kP0wLvy
ieqrbcuFjmqfyPmuV1u90o14T0ikpwvTZUo2YZANP4c/gj4ry48/znmUaRvy2kvI
l7gRQ21qJTKSsu01yoNo3J9TpXPGU7Lydz/HwWw0DpArtAaukI8aNx4ohFUKS
wDSIIWIWj1GbEe100TfWvEWT0nbN1/faPxpk5IRxicapq1II=
-----END CERTIFICATE-----
1 s:/US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High Assurance Server CA
```

Step 2: Extract the public key (e, n) from the issuer's certificate.



```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3$ openssl x509 -in ci.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7F0A6406B27D2226516AE42409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B1
11A9AE0D2C8B8433AD90B0BD595F540AFC81DE4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC719B2A6850F5E44582A378FFD35F10B0827325AF5BB
889EA4BD51D027E2D03B4233A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5079208AC75A49CBACD17B21E4435657F532539D11C0A9A631B19
9274680A37C2C25248CB395A2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
jeevan@jeevan-VirtualBox:~/Downloads/lab3$
```

Step 3: Extract the signature from the server's certificate.



```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3$ openssl x509 -in ci.pem -text -noout
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
        04:e1:e7:a4:dc:5c:f2:f3:6d:c0:2b:42:b8:5d:15:9f
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance EV Root CA
Validity
    Not Before: Oct 22 12:00:00 2013 GMT
    Not After : Oct 22 12:00:00 2028 GMT
Subject: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:b6:e0:2f:c2:24:06:c8:6d:04:f5:d7:ef:0a:64:
                06:d2:7d:22:26:65:10:ae:a2:40:9b:ce:dc:9f:9f:
                76:07:3e:c3:30:55:87:19:b9:4f:94:0e:5a:94:1f:
                55:56:b4:c2:02:2a:af:d0:98:ee:0b:40:d7:c4:d0:
                3b:72:c8:14:9e:ef:90:b1:11:a9:ae:d2:c8:b8:43:
                3a:d9:0b:0b:ds:d5:95:f5:40:af:c8:1d:ed:ad:9c:
                f5:f7:b7:86:50:68:99:f5:8a:ad:d2:c7:05:1f:a8:
                97:c9:dc:a4:b1:82:84:2d:c6:ad:a5:9c:c7:19:82:
                a6:85:0f:5e:44:58:2a:37:8f:fd:35:f1:0b:08:27:
                32:5a:f5:bb:0b:9e:a4:bd:s1:02:7:ez:dd:b4:2:
                33:a3:05:28:c4:bb:28:cc:9a:ac:2b:23:0d:78:c6:
                7b:1e:5e:71:b7:4a:3e:08:fb:81:b7:16:16:a1:9d:
                23:12:4d:e5:d7:92:08:ac:75:a4:9c:ba:cd:17:b2:
                1e:44:35:65:7f:53:25:39:d1:c0:a9:a6:31:b1:9:
                92:74:60:0a:37:c2:c2:52:48:cb:39:5a:a2:b6:e1:
                5d:c1:dd:a0:20:b8:21:a2:93:6:0f:14:4a:21:41:
                c7:ed:6d:9b:f2:48:2f:f3:03:f5:a2:68:92:53:2f:
                5e:e3
```

```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3$ cat signature | tr -d '[[:space:]]'
cat: signature: No such file or directory
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ cat c0.pem | tr -d '[[:space:]]'
-----BEGIN CERTIFICATE-----MIIFBjCCBNQgAwIBAgIQDmTF+8I2reFLFyrrQceMsDANBgqhkiG9w0BAQsFADBwMQSwCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQqSW5jMRkwFwYDVQQLExB3dscuZGlnaWNlcnQuYz9tMSwLQyDVQDDEyZeaWdpq2VycBTSEEyIEhpZ2ggQXNzdXJhbmlNlFnLcnZlc1BDQTAefw0xNTExMDwMDAwMDBafW0xDExMjgxJawMDBaMIGLM0swCQYDVQGEwJvUzETMBEGA1UECBMKQ2FsaNzVcm5pYTEUMB1GA1UEBxMLTG9zIEFuZzVSzXmxPD6BvNBVAoTm0ludGvbymV0IEvncnBvcmf0Aw9UIGzvc1Bc3NpZ2S1zcB0Yw1lvBhbnQgtNvtYmVyczETMBEGA1UECxKMVGvjacSvbG9neTErMYGA1UEAMp3d3lmV4Yw1wbGUub3JnM1IBIJAnBgkqhkiG9w0BAQEFAAOCQA8AM1BcQKAQEAs0CWL2FjP1xB1L61RFvvEOKzLJmG9LWAC3bCbjgsH6NIvVo2dt6uFxz15bt7m7F3K7srfuBykLo78mraM9q1zrHoIeyoFrV/n+pZZJausPjCPxMEJnR0d8Z4KpWKX0LybuIsputoIn4lq/htEhtiOnuoBfNxZf7WccxGwEsZuS1KxCKH15VRJ0rekFHTaxC1b1qcZ/QRaB1v0yhvxk1ybTwddT4cl16fHce3xGmSL328Fgs3jYrvG29PueB6VjI/tbbpuoqtFwp/H1brqdjh29u52Bhb0fJKM9DWxCP/Catccc7az8ExnC0/Lk8vkhw/kai1JWPkx4R8vgy73nwIdAQAb0iCUDCAkkwHwYDVR0jB8gwFoAUUWj/kk8CB3U8nZlZGK1ErhZcjswHQYDV0R0OBVYEFKZPv84fl.dhn850gkpW50ta65I51M1GBBqNVHREEEjB4gg93d3cuZXhhbx8sZ5vcmcCC2V4Yw1wbGUuYz9tggteGftGxLmVkyIDPd3d3lmV4Yw1wbGUub0v0MA4GA1UDWEB/wQEAWIFoAdBqNHVSUEfjAUBggRgF8QcDAQYIKwYBBQUHAwIwdQYDVRF0FBG4wbDA0oDKgM1YuaHR0cDovL2NyBDMuZClnaWNlcnQuYz9tL3NoYTTtaGtc2VydmyLwC0LmNyvDBBmGvNHSAAERTBDMDcGCWGSAGG/wWBATAqCgGCCsAQUBFBw1BFhxodRwczovl3d3dykskaWdpYVdyc5jb20vQ1BTAGBmEBDEACAJCbwgVIKyYBBQUHAQEEzB1MCQGCCsAQUBFBzABhhodHRw0i8vb2NzCSkawdpY2Vdc5j26w1QY1KwYBBQHNMKGQwh0dHA6ly9JYWNlcnRzlMpRzz2lJzX0lNmNb59EWdwpQ2VydFNlQTJiawdQXNzdXJhbmlNU2vydmyV0EUY330AwGA1UdeWeB/WQCMIAwDQYJKoZIhvCNAQELBQADggEBAlSomhGn2L0LJn5SJHuyVz3qMlIRClvdqe0061s+C8ctRwR03Uu3x8q80H+2ahxl0mpzdCsAl4XQzJLlJj1ZQ1p+hub8NfIMmVPPZjb2tZm2ipWvMRM+zgpRVm6VJ9F3vFfUSh0b4/jxEIUVPy+d8/Krc+kPQwLyieqRbcuFjmqfyPmuv1u90oI4To1kpwTZU0zYZANP4C/gj4Ry48/znu0Arvy2kv1L7gRQ21qJTK5suoiYoYn0339t+pxPGU7lydz/Hw+w0DpArtAaukI8aXN4ohFUKSwDsIIWIWJlJGbEeI00TIFwEVWTOnbNL/fapPk5IRXlcapqiII-----END CERTIFICATE-----jeevan@jeevan-VirtualBox:~/Downloads/lab3$
```

Step 4: Extract the body of the server's certificate.

```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3$ openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1522 cons: SEQUENCE
 4:d=1 hl=4 l=1242 cons: SEQUENCE
    8:d=2 hl=2 l= 3 cons: cont [ ... ]
      10:d=3 hl=2 l= 1 prim: INTEGER :92
      13:d=2 hl=2 l= 16 prim: INTEGER :0E64CSFBC236ADE14B172AEB41C78CB0
      31:d=2 hl=2 l= 13 cons: SEQUENCE
        33:d=3 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
        44:d=3 hl=2 l= 0 prim: NULL
      46:d=2 hl=2 l= 112 cons: SEQUENCE
        48:d=3 hl=2 l= 11 cons: SET
          50:d=4 hl=2 l= 9 cons: SEQUENCE
            52:d=5 hl=2 l= 3 prim: OBJECT :countryName
            57:d=5 hl=2 l= 2 prim: PRINTABLESTRING :US
        61:d=3 hl=2 l= 21 cons: SET
          63:d=4 hl=2 l= 19 cons: SEQUENCE
            65:d=5 hl=2 l= 3 prim: OBJECT :organizationName
            70:d=5 hl=2 l= 12 prim: PRINTABLESTRING :DigiCert Inc
        84:d=3 hl=2 l= 25 cons: SET
          86:d=4 hl=2 l= 23 cons: SEQUENCE
            88:d=5 hl=2 l= 3 prim: OBJECT :organizationalUnitName
            93:d=5 hl=2 l= 16 prim: PRINTABLESTRING :www.digicert.com
        111:d=3 hl=2 l= 47 cons: SET
          113:d=4 hl=2 l= 45 cons: SEQUENCE
            115:d=5 hl=2 l= 3 prim: OBJECT :commonName
        120:d=5 hl=2 l= 38 prim: PRINTABLESTRING :DigiCert SHA2 High Assurance Server CA
      160:d=2 hl=2 l= 30 cons: SEQUENCE
        162:d=3 hl=2 l= 13 prim: UTCTIME :151103000000Z
        177:d=3 hl=2 l= 13 prim: UTCTIME :181128120000Z
      192:d=2 hl=3 l= 165 cons: SEQUENCE
        195:d=3 hl=2 l= 11 cons: SET
          197:d=4 hl=2 l= 9 cons: SEQUENCE
            199:d=5 hl=2 l= 3 prim: OBJECT :countryName
            204:d=5 hl=2 l= 2 prim: PRINTABLESTRING :US
        208:d=3 hl=2 l= 19 cons: SET
        210:d=4 hl=2 l= 17 cons: SEQUENCE
```

```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ openssl asn1parse -i -in c0.pem -strparse 4 -out co_body.bin -noout
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ ls
c0.pem  c1.pem  co_body.bin
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ sha256sum co_body.bin
902677e610fedddd34780e359692eb7bd199af35115105636aebe623f9e4dd053  co_body.bin
jeevan@jeevan-VirtualBox:~/Downloads/lab3$
```

```
jeevan@jeevan-VirtualBox: ~/Downloads/lab3
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ openssl asn1parse -i -in c0.pem -strparse 4 -out co_body.bin -noout
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ ls
c0.pem  c1.pem  co_body.bin
jeevan@jeevan-VirtualBox:~/Downloads/lab3$ sha256sum co_body.bin
902677e610fedddd34780e359692eb7bd199af35115105636aebe623f9e4dd053  co_body.bin
jeevan@jeevan-VirtualBox:~/Downloads/lab3$
```

Step 5: Verify the signature.

The screenshot shows the Immunity Debugger interface with the file 'co_body.bin' loaded. The assembly view displays memory dump data. A tooltip provides detailed information about a certificate entry:

0.....d...6..K.*.A...0..
*..H.....Op1.0...U....US1.0
..U...DigiCert Inc1.0..U...
.www.digicert.com/0...U...&Di
giCert SHA2 High Assurance Ser
ver CA0...151103000000Z..18112
812000Z0..1.0...U....US1.0..
U....Californial.0...Los
Angeles1<0...U...3Internet Cor
poration for Assigned Names an
d Numbers1.0...Technology
1.0...U...www.example.org.."
0...*.H.....0.....
@../ac%...eE...B.....v..8..
.buZ6v..].....]....P.\$,...
...z.{..._...Y%....?...&th..x*
..._B...R..h#.C.m..bB{...Y.^..
F.K..-Jqr\$..^UD..x.GM...Z.g.h
./.(q..r.<u..rX...../)...
7....o...zt.....V....

Below the assembly view, several registers are displayed with their values:

Signed 8 bit: 54	Signed 32 bit: 917365067	Hexadecimal: 36 AD E1 4B
Unsigned 8 bit: 54	Unsigned 32 bit: 917365067	Decimal: 054 173 225 075
Signed 16 bit: 13997	Float 32 bit: 5.182029E-06	Octal: 066 255 341 113
Unsigned 16 bit: 13997	Float 64 bit: 2.6169294398568E-45	Binary: 00110110 10101101 1110001 01001011

Checkboxes at the bottom include 'Show little endian decoding' and 'Show unsigned as hexadecimal'. The status bar at the bottom right shows 'Offset: 0x10 / 0x4dd' and 'Selection: None'.

We have all the information and the certificate is verified.