## Applied

10 October 2022

## Cryptography

21:43

## Lab-05

Manual

Name : Vishwas M

SRN : PES2UG20CS390

SEC: F

DATE: 25/10/2022

Lab: 4

***Note:*** *The demo*

##### showed in

*the video for this*

*lab has*

*been performed*

*on Ubuntu*

*22.04 running on*

*wsl2. So*

*far, there have*

*been no changes*

*noted between*

*the*

##### execution of

*this lab on*

*wsl and execution on*

*the*

##### seedlabs

*Ubuntu*

*20.04*

*virtualbox*

*vm.*

##### Therefore, despite

*the differences in*

*environment,*

*all*

##### tasks in the

*lab*

##### should

*run*

##### smoothly.

**Task**

**1: A**

**Complete**

**Example of**

**BIGNUM**

The

program

below

shows

a complete

example

of BIGNUM.

This

program

uses

three

BIGNUM

variables, a,

b, and n;

and

then

compute a

**Code**

∗ b and

(a b mod n)

#include #include

<stdio.h>

<openssl/bn.h>

#define

NBITS

256

void printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for

hex

string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX BIGNUM BIGNUM BIGNUM

\*ctx = BN\_CTX\_new();

\*a = BN\_new();

\*b = BN\_new();

\*n = BN\_new();

BIGNUM \*res = BN\_new();

// Initialize

BN\_generate\_prime\_ex(a,NBITS,1,NULL,NULL,NULL); BN\_dec2bn(&b,"273489463796838501848592769467194369268"); BN\_rand(n,NBITS,0,0);

// res = a\*b

BN\_mul(res,a,b,ctx); printBN("a\*b=",res);

// res = a^b mod n

BN\_mod\_exp(res,a,b,n,ctx); printBN("a^b mod n=",res);

return 0;

}

**Commands**

$ gcc

task1.c -o

task1

-lcrypto

$ ./task1



**We can see the result of multiplication of 2 large numbers in the first row of the output and we can see the modulus of that result by a number ‘n’ in the second row of the output in the terminal.**

**Task**

**2: Deriving**

**the**

# Private Key

The

objective of

this task is

to derive

private key.

Given are

the

hexadecimal values

of p, q, e, and

public key pair (e,n)

p = F7E75FDC469067FFDC4E847C51F452DF q = E85CED54AF57E53E092113E62F436F4F e = 0D88C3

**Code**

#include <stdio.h> #include <openssl/bn.h> #define NBITS 256

void

printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for hex string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX \*ctx = BN\_CTX\_new(); BIGNUM \*p = BN\_new(); BIGNUM \*q = BN\_new(); BIGNUM \*e = BN\_new(); BIGNUM \*d = BN\_new(); BIGNUM \*res1 = BN\_new(); BIGNUM \*res2 = BN\_new(); BIGNUM \*res3 = BN\_new(); BIGNUM \*one = BN\_new();

// Initialize BN\_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF"); BN\_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");

BN\_hex2bn(&e,"0D88C3"); BN\_hex2bn(&one,"1"); BN\_sub(res1,p,one); BN\_sub(res2,q,one); BN\_mul(res3,res1,res2,ctx); BN\_mod\_inverse(d,e,res3,ctx); printBN("d=",d);

return 0;

}

**Commands**

$ gcc task2.c -o

$ ./task2

task2 -lcrypto



The decryption key found is mentioned in the above screenshot by Elgamal method.

First we calculate Euler’s totient (p-1)\*(q-1)

Then we find mod inverse to find the decryption key to find d

**Task**

**3: Encrypting a**

**Message**

The

objective of

this task is

to encrypt a

given message.

Given are

the

hexadecimal values

of n, e, M (you can

use

whatever message

you want). The value of “d” is also

given

to verify

the

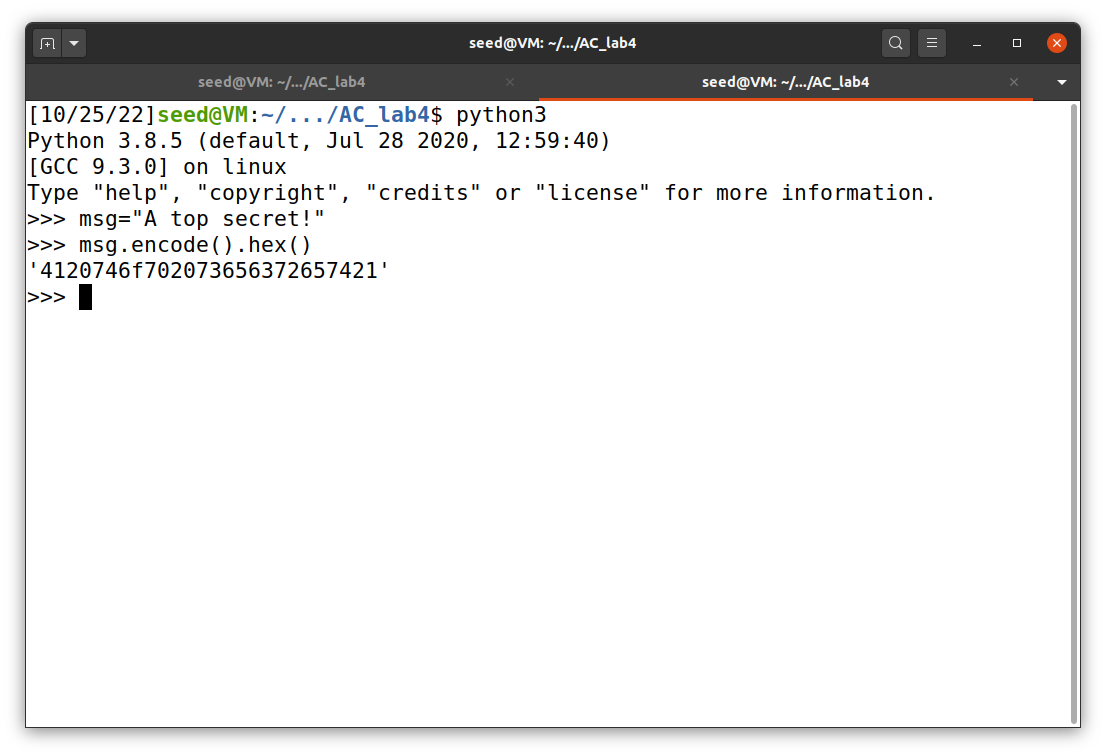
result.

**Step 1**

Convert the ASCII String message

to a hex string

python3 -c "print('A top secret!'.encode().hex())"



We are finding the hexadecimal form of the text messge and inserting it in the code as the text message.

Give your

**Step 2**

observation with

a screenshot

Execute the below program to encrypt the message M and

verify

**Code**

it by decrypting

it.

#include <stdio.h> #include <openssl/bn.h> #define NBITS 256

void

printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for hex string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX \*ctx = BN\_CTX\_new(); BIGNUM \*m = BN\_new(); BIGNUM \*e = BN\_new(); BIGNUM \*n = BN\_new(); BIGNUM \*d = BN\_new(); BIGNUM \*enc = BN\_new();

BIGNUM \*dec = BN\_new();

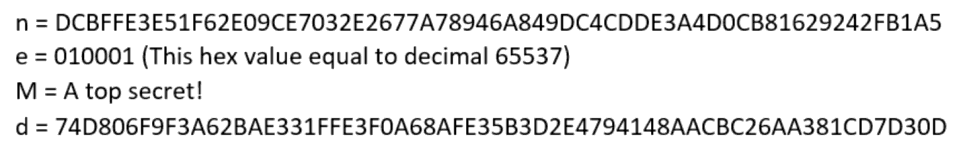
// Initialize

BN\_hex2bn(&m,"<< Enter the message in hex, obtained in BN\_hex2bn(&e,"010001");

step

1 >>");

BN\_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"); BN\_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");



// Encryption BN\_mod\_exp(enc,m,e,n,ctx); printBN("Encrypted Message =",enc);

// Decryption BN\_mod\_exp(dec,enc,d,n,ctx); printBN("Decrypted Message =",dec); return 0;

}

**Commands**

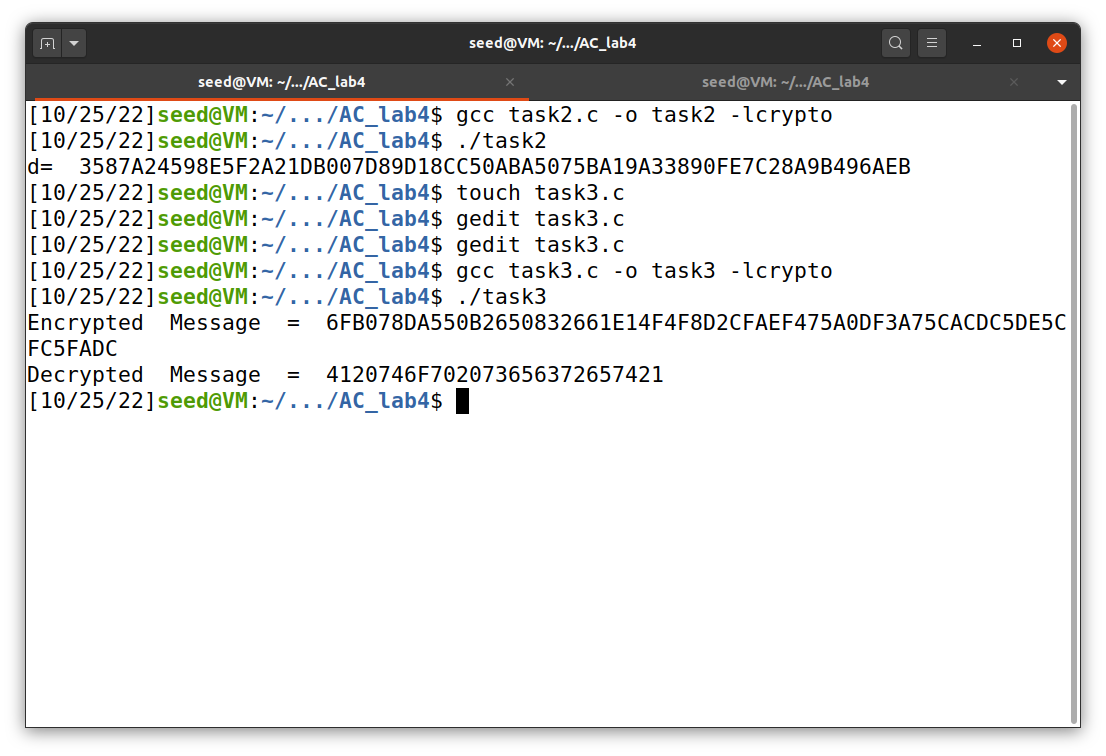
$ gcc task3.c -o

$ ./task3

First we are doing encryption: enc=m^e mod n

Then we are doing decryption:dec=enc^d mod n

task3 -lcrypto



We are getting the same text message when we decrypt the ciphertext.

**Task**

**4: Decrypting a**

**Message**

The

objective of

this task is

to decrypt a

given

ciphertext. given are the hexadecimal values of n, e, d from the above task

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567E A1E2493F

**Code**

#include <stdio.h> #include <openssl/bn.h> #define NBITS 256

void

printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for hex string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX \*ctx = BN\_CTX\_new(); BIGNUM \*m = BN\_new(); BIGNUM \*e = BN\_new(); BIGNUM \*n = BN\_new(); BIGNUM \*d = BN\_new(); BIGNUM \*enc = BN\_new(); BIGNUM \*dec = BN\_new();

// Initialize BN\_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"); BN\_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D"); BN\_hex2bn(&enc,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F"

);

// Decryption BN\_mod\_exp(dec,enc,d,n,ctx); printBN("Decrypted Message =",dec); return 0;

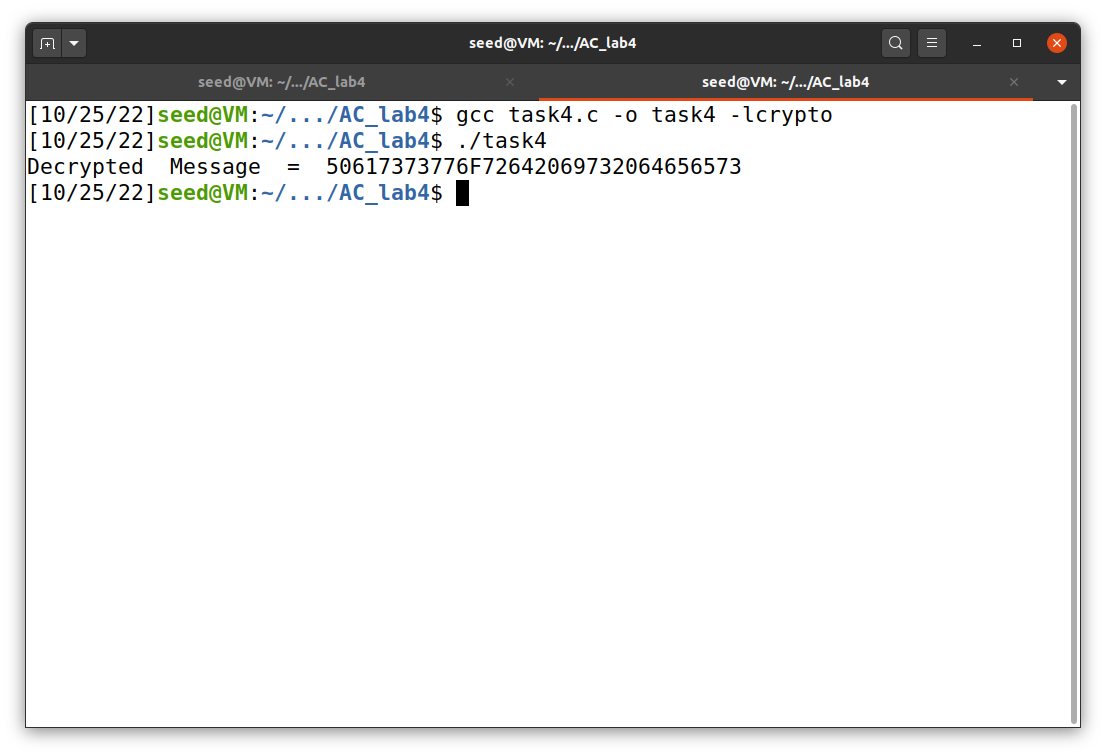
}

### Commands

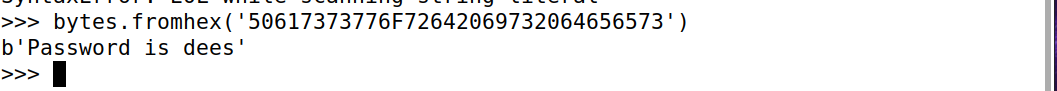
$ gcc task4.c -o

$ ./task4

task4 -lcrypto



We are finding the message given the ciphertext and decryption key



When we convert the plaintext to bytes we get the above text as the message text.

**Task 5: Signing a Message**

The

objective of

this task is

to generate a signature for

the

following

message. Use

the public/private key

set

from task3

M= I owe you $2000

**Step 1**

Generate hex for M

python3 -c "print('I owe you $2000'.encode().hex())"

**Step 2**

Execute the e

following

program to generate signature of

the

given

message. Using the signing

algorithm

M^d mod n

**Code**

#include <stdio.h> #include <openssl/bn.h> #define NBITS 256

void

printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for hex string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX \*ctx = BN\_CTX\_new(); BIGNUM \*m = BN\_new(); BIGNUM \*n = BN\_new(); BIGNUM \*d = BN\_new(); BIGNUM \*sign = BN\_new();

// Initialize

BN\_hex2bn(&m,"<< Hex value of M >>");

BN\_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"); BN\_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

// Signing BN\_mod\_exp(sign,m,d,n,ctx); printBN("Sign =",sign); return 0;

}

**Commands**

$ gcc task5.c -o

$ ./task5

**Step 3**

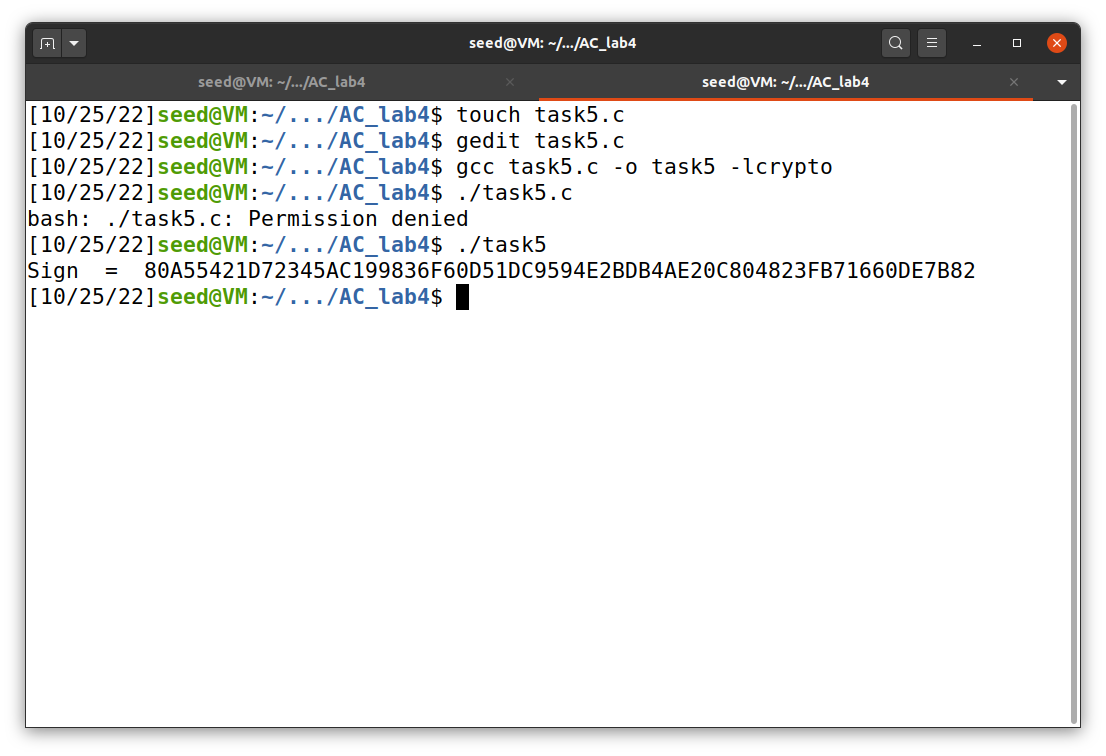
task5 -lcrypto

Execute steps

1. and
2. for the

message "I owe $3000"





We are generating the signature of the given plain text

Sign = m^d mod n

**Task 6: Verifying a signature**

The

objective of

this

task is

to verify

if the

signature

received by

Bob

is Allice’s or

not.

Given

are and

the n.

Message

M, signature S,

Allice

public

key e

**Code**

#include #include

<stdio.h>

<openssl/bn.h>

#define

NBITS

256

void printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for

hex

string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX BIGNUM BIGNUM BIGNUM

\*ctx = BN\_CTX\_new();

\*s = BN\_new();

\*n = BN\_new();

\*e = BN\_new();

BIGNUM

\*message

= BN\_new();

// Initialize

F");

5");

BN\_hex2bn(&s,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802 BN\_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F1811611

BN\_hex2bn(&e,"010001");

// Signing

BN\_mod\_exp(message,s,e,n,ctx); printBN("Message =",message);

return 0;

}

### Commands

$ gcc

task6.c -o

task6

-lcrypto

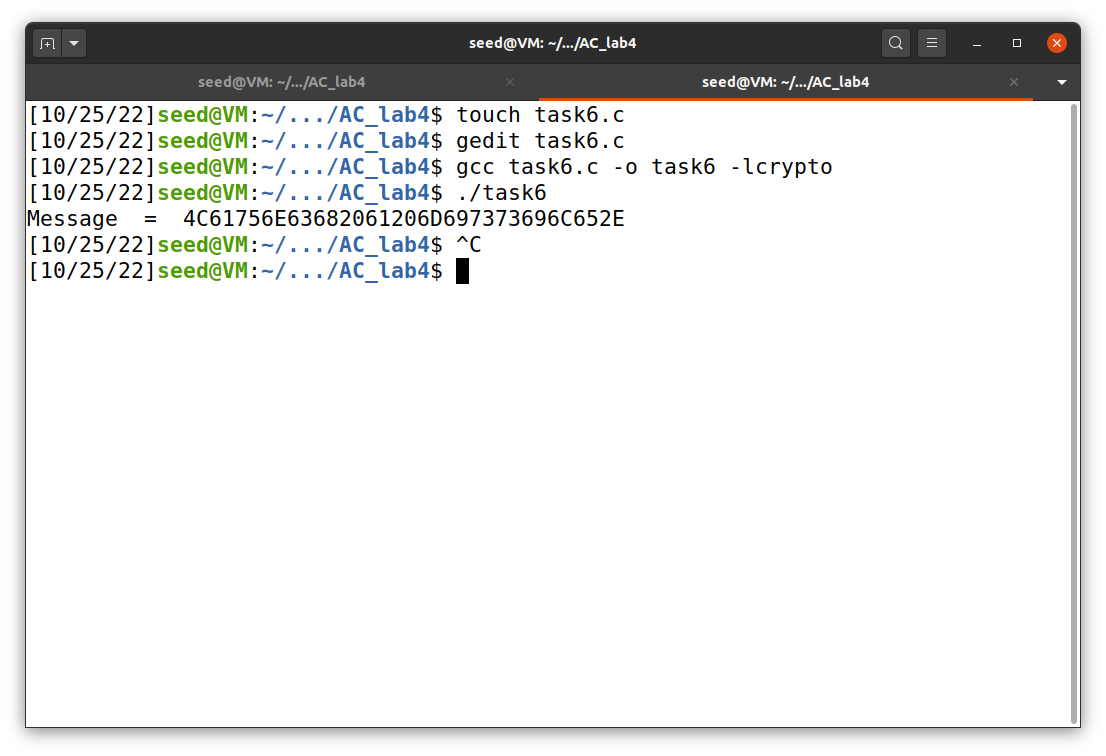
$ ./task6

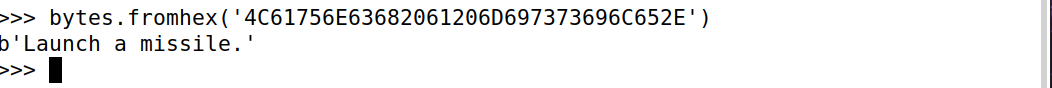
$ python3

-c "print(bytes.fromhex('<<

output of

task6 >>'))"





We are getting back the message by using the signature and the by decrypting using public key.

**Task**

**7: Manually**

**Verifying**

**X.509**

**Certificate**

The

objective of

this

task is

to verify

the

signature

of a public

key

certificate

from

a server

and

show

that

the

signature matches

To verify

that a

certificate

was

signed

by a specific

certificate

authority we

need

the following

details

1. Public

key

of the

certificate

authority

(issuer).

1. signature and

algorithm

used to generate signature

from the server’s certificate.

**Step 1**

Download the certificate from any website (each

student use a

different

website)

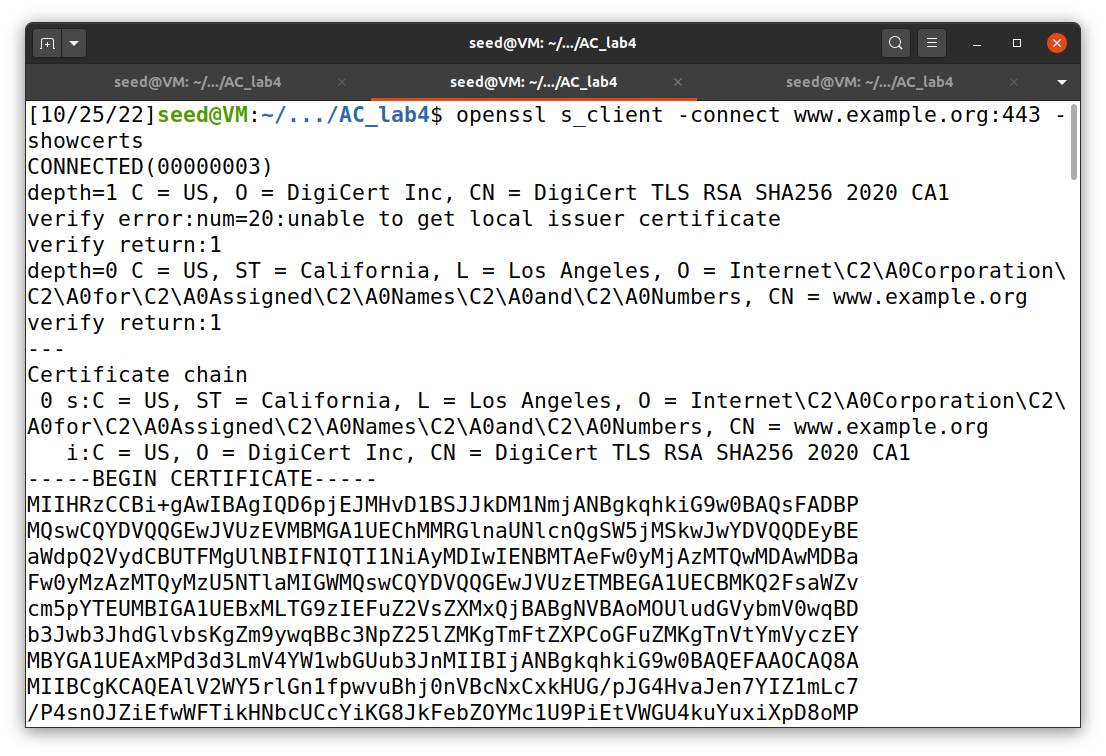
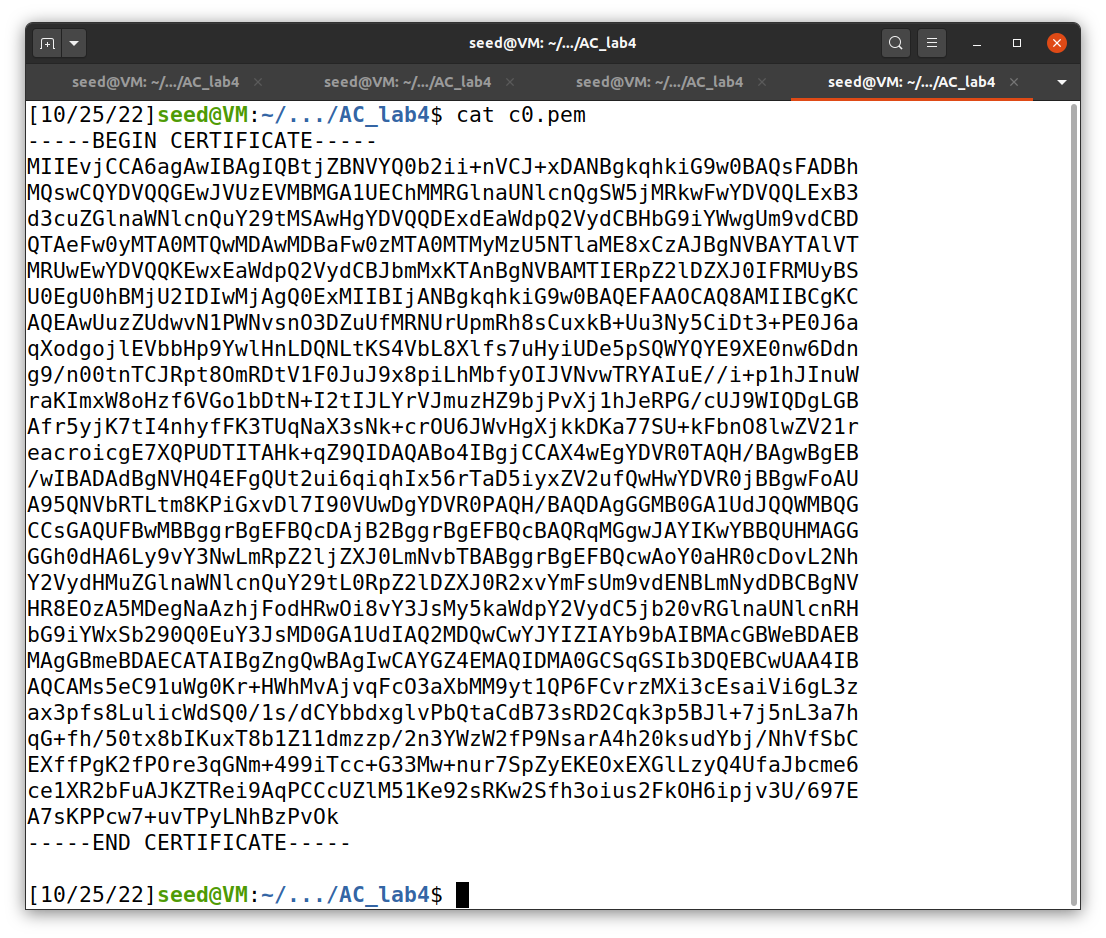
$ openssl showcerts

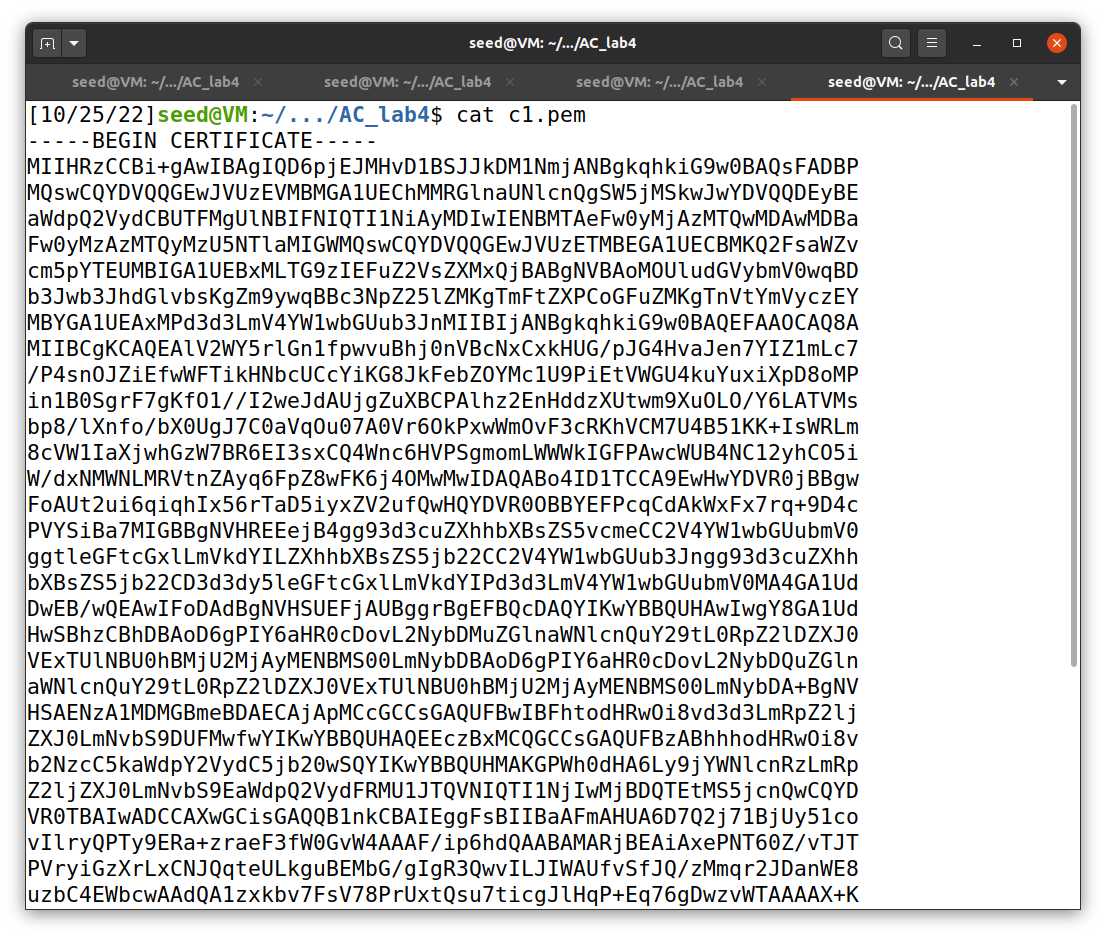
s\_client -connect www.example.org:443 -

Copy server certificate to

c0.pem file and

root

certificate of the issuer to c1.pem



**Step 2**

Extract the public key (e, n) from the issuer’s

certificate. Openssl provides commands to extract

certain attributes from

the x509

certificates. We

can

extract the value of n using -modulus. There is no

specific command to extract e, but we can print out

all

the fields and can easily

find the value of e.

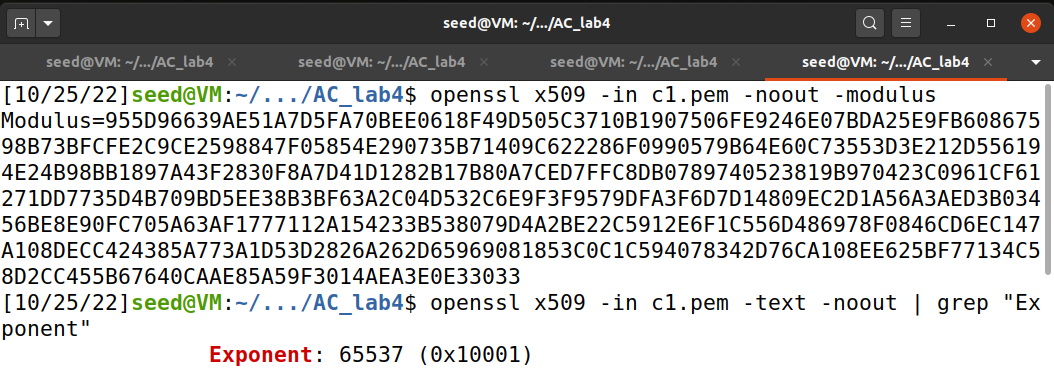
$ openssl

x509 -in c1.pem -noout

-modulus

$ openssl x509 -in c1.pem -text -noout |grep

"Exponent"



**Give your**

**Step 3**

**observation with**

**screenshot**

Extract the signature from

the server’s

certificate.

There is no specific openssl command to extract the

signature

field.

However, we can

print out

all

the

fields into a

and then copy and paste the signature block file (note: if the signature algorithm used in

the

certificate is not based on RSA,

find another

certificate).

**Commands**

$openssl x509

-in c0.pem -text -noout

//extract signature

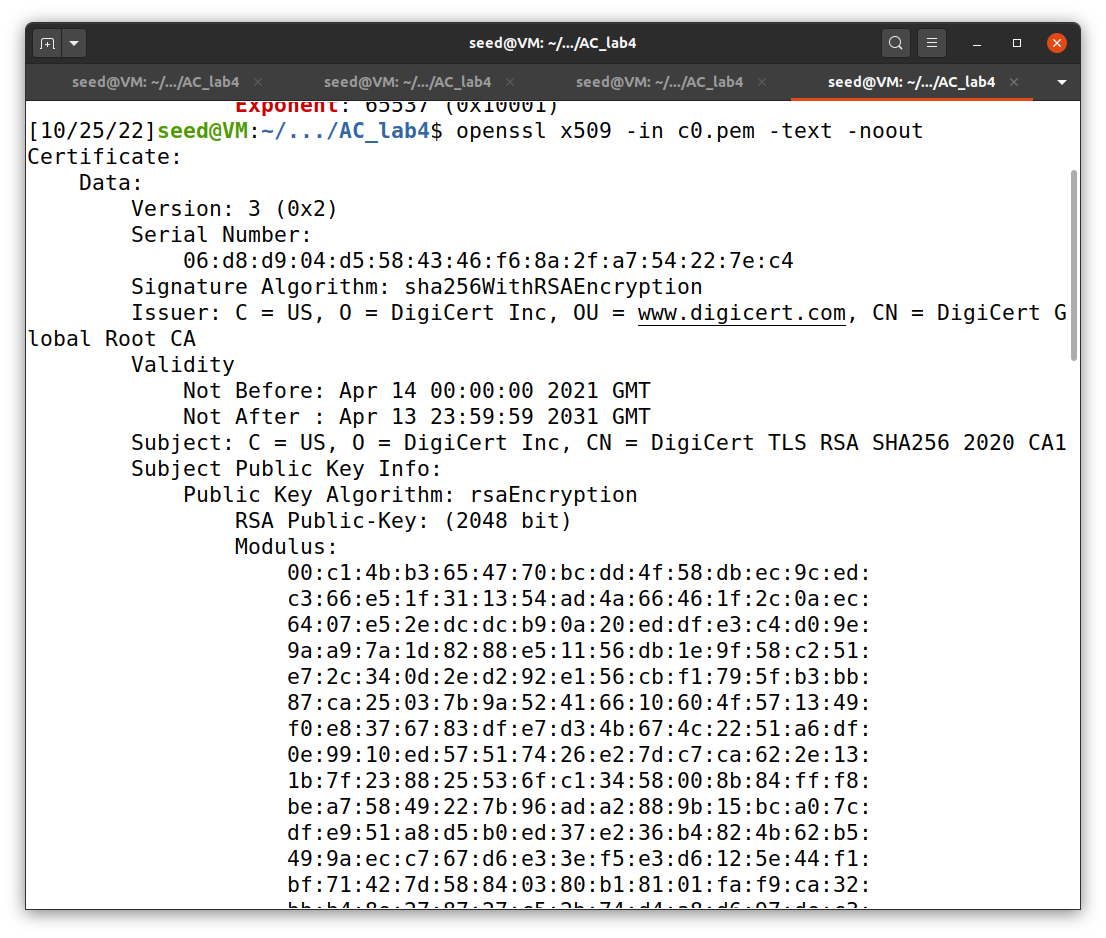
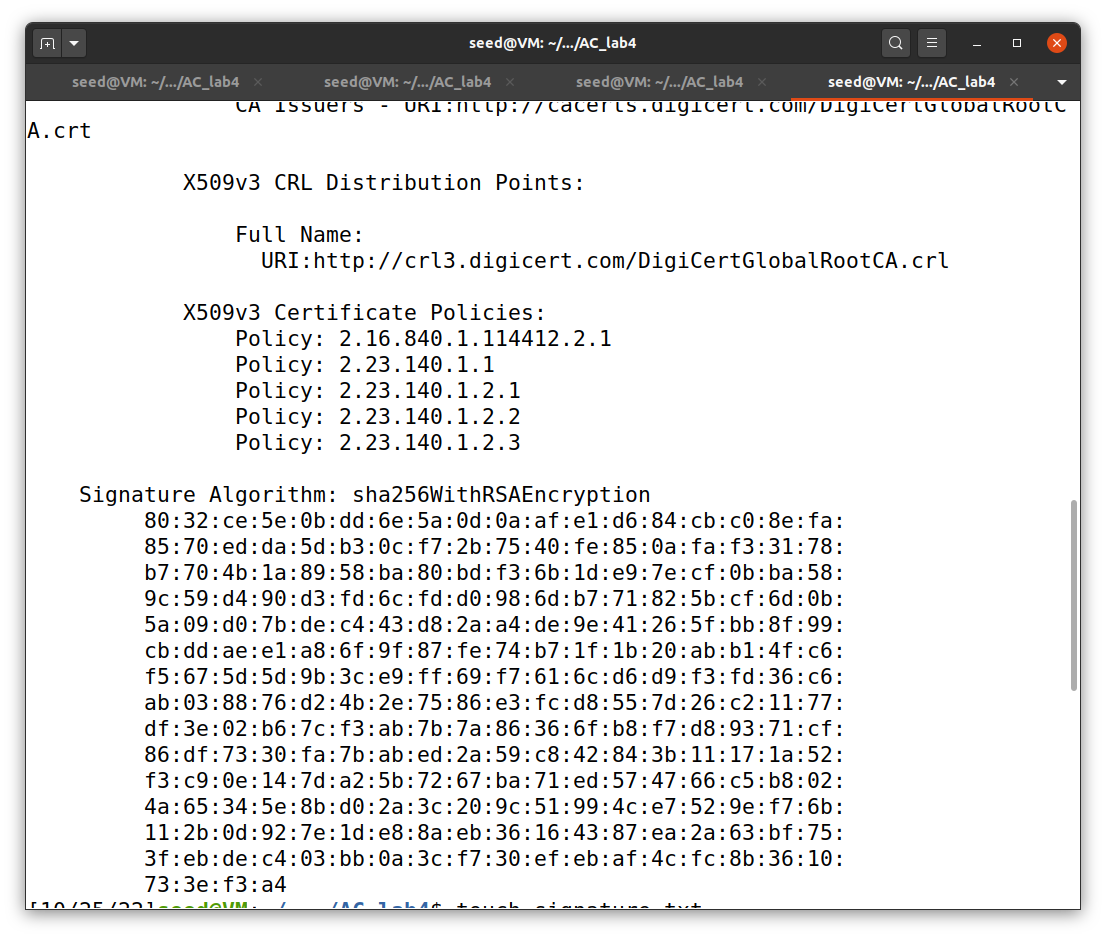
only the signature part and paste it in file

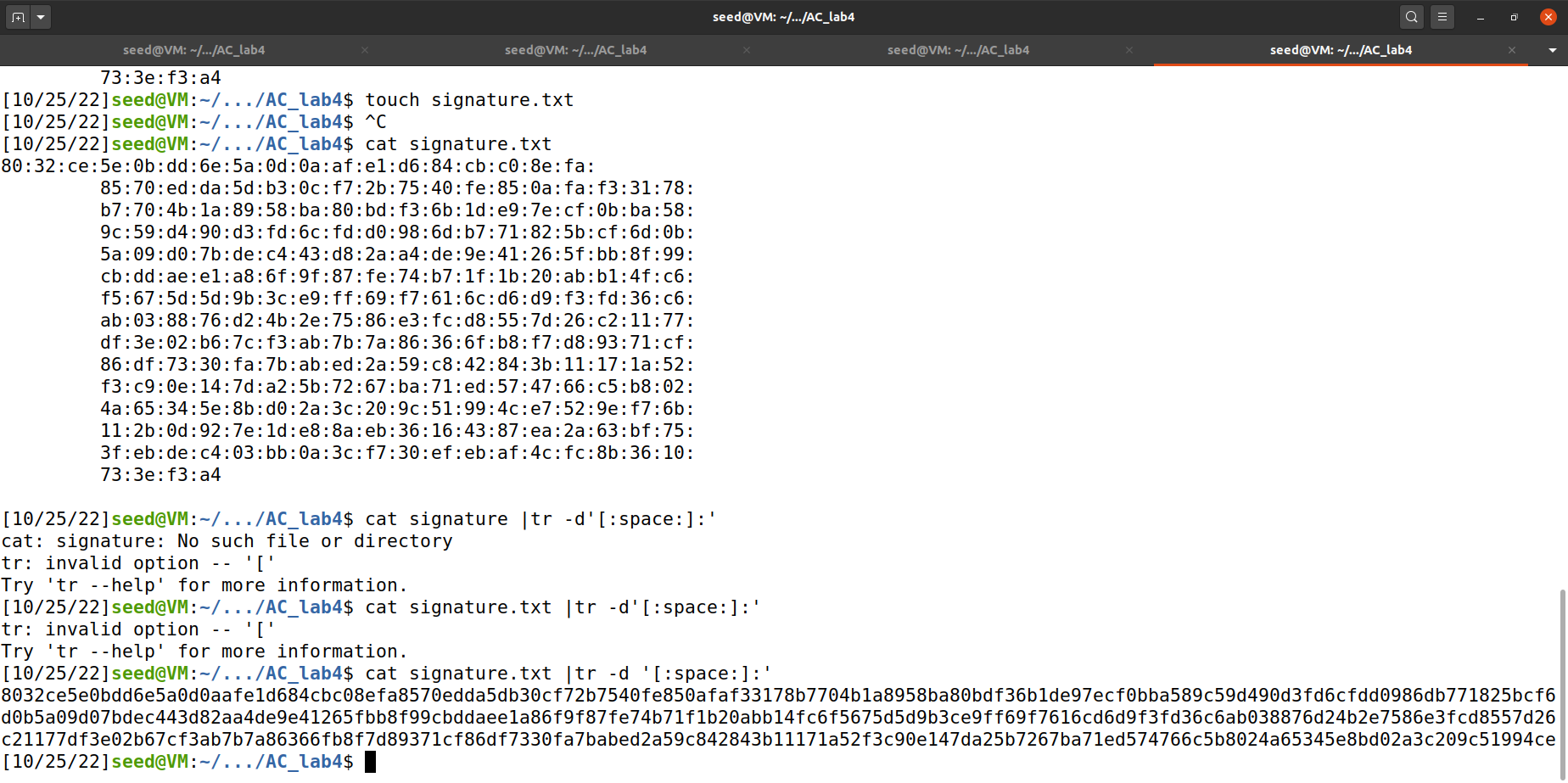
$ cat signature | tr -d ’[:space:]:’

Give your

observation with

screenshot.





**Step 4**

Verify

the signature by

substituting

the values just

found out, in

the code given below and running

it.

**Code**

#include <stdio.h> #include <openssl/bn.h> #define NBITS 256

void

printBN(char\*msg, BIGNUM\*a) {

/\* Use BN\_bn2hex(a) for hex string

Use BN\_bn2dec(a) for decimal string\*/

char\* number\_str = BN\_bn2hex(a);

printf("%s %s\n",msg,number\_str);

OPENSSL\_free(number\_str);

}

int main() {

BN\_CTX \*ctx = BN\_CTX\_new(); BIGNUM \*s = BN\_new(); BIGNUM \*n = BN\_new(); BIGNUM \*e = BN\_new();

BIGNUM \*message

// Initialize

= BN\_new();

BN\_hex2bn(&s,"<< signature >>"); BN\_hex2bn(&n,"<< modulus >>");

BN\_hex2bn(&e,"<< exponent

// Signing

>>");

BN\_mod\_exp(message,s,e,n,ctx); printBN("Message =",message); return 0;

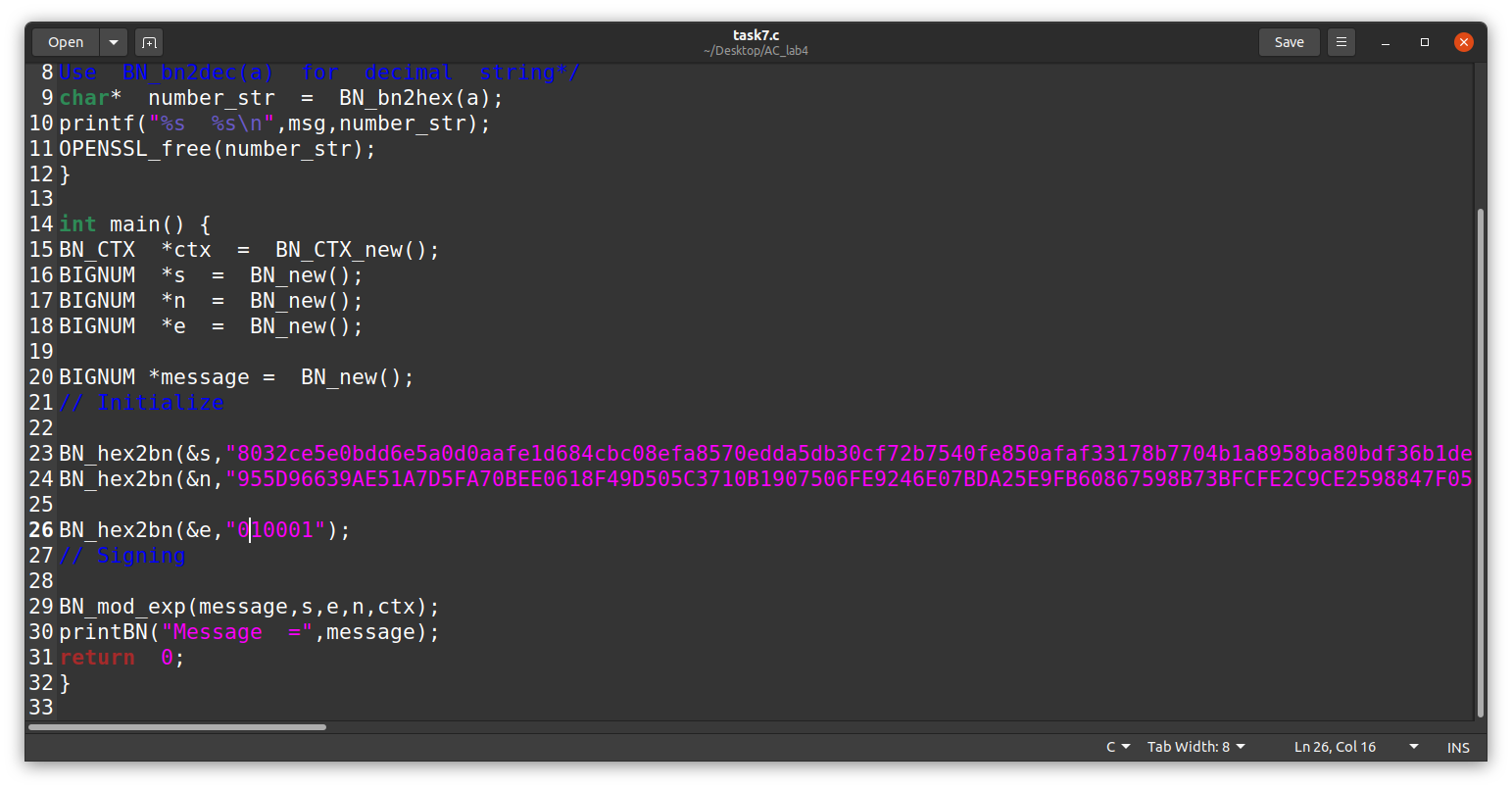
}

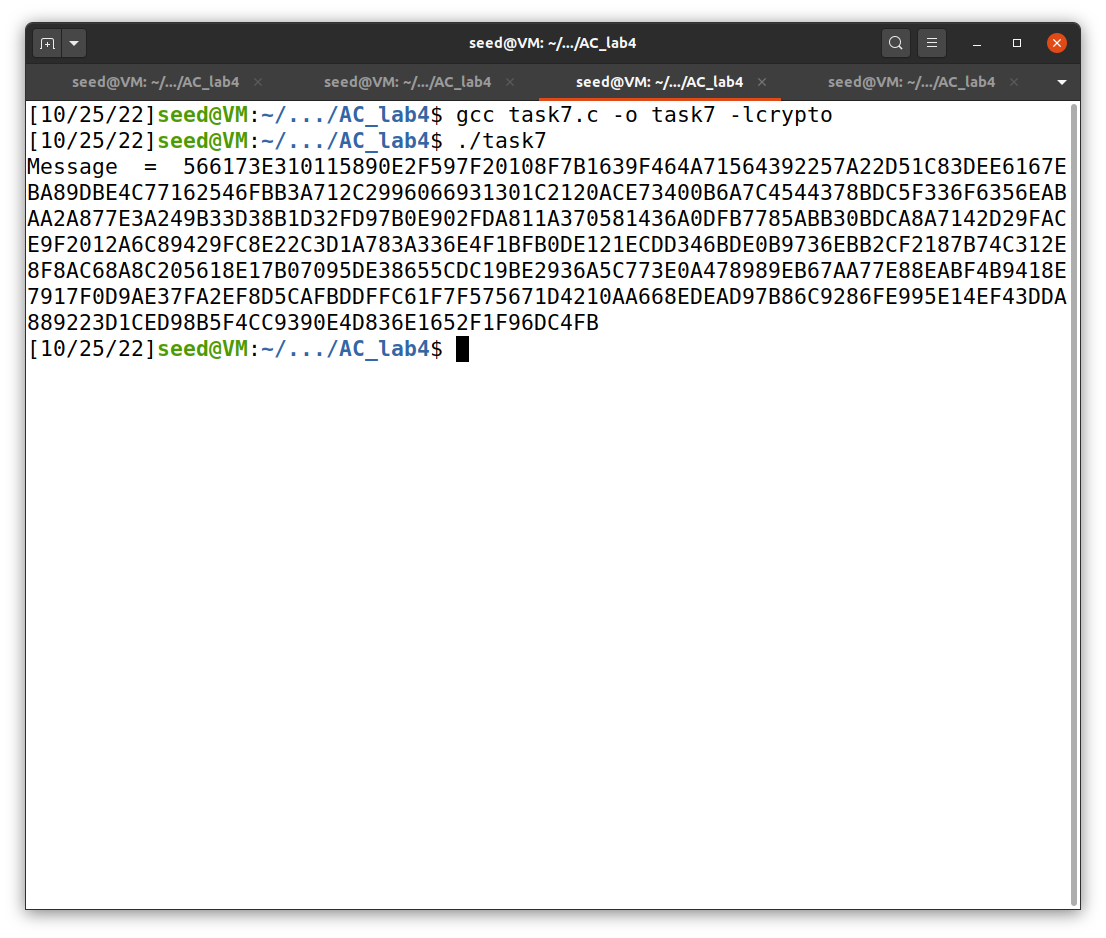
#### Commands

$ gcc task7.c -o

$ ./task7

task7 -lcrypto





We will substitute modulus, exponent and signature and find the message