

# PUBLIC KEY CRYPTOGRAPHY

Shehryar Kamran

## Table of Contents

<b>Section-1.....</b>	3
<b>Public-Key Infrastructure Lab .....</b>	4
<b>Pre-Requisite Tasks:</b> Container setup Commands and DNS setup.....	4
<b>Task 1:</b> Becoming a certificate authority .....	6
<b>Task 2:</b> Generating a certificate request for your web server.....	8
<b>Task 3:</b> Generating a certificate for your server.....	10
<b>Task 4:</b> Deploying Certificate in an Apache-Based HTTPS Website.....	10
<b>Task 5:</b> Launching a Man-in-the-Middle attack.....	15
<b>Task 6:</b> Launching a Man-in-the-Middle attack with a compromised CA.....	18
<b>Section-2.....</b>	4
<b>RSA Encryption and Signature Lab .....</b>	6
<b>Pre-Requisite Tasks:</b> Implement the BigNum example, compile and execute it.....	6
<b>Task 1:</b> Deriving the Private Key .....	7
<b>Task 2:</b> Encrypting a Message .....	9
<b>Task 3:</b> Decrypting a Message .....	11
<b>Task 4:</b> Signing a Message.....	13
<b>Task 5:</b> Verifying a Signature .....	15
<b>Task 6:</b> Manually Verifying an X.509 Certificate.....	18

# *Section-1*

## Public-Key Infrastructure Lab

**Pre-Requisite Tasks:** Container setup Commands and DNS setup

In the following, some of the commonly used commands related to Docker and Compose have been executed. Following are the screenshots.

```
docker-compose build      # Build the container image  
docker-compose up        # Start the container  
docker-compose down     # Shut down the container  
dcbuild                 # Alias for: docker-compose build  
dcup                     # Alias for: docker-compose up  
dcdown                  # Alias for: docker-compose down
```

```
root@VM:/home/seed/Downloads/Labsetup# docker-compose build
Building web-server
Step 1/7 : FROM handsonsecurity/seed-server:apache-php
apache-php: Pulling from handsonsecurity/seed-server
da7391352a9b: Pulling fs layer
14428a6d4bcd: Pulling fs layer
14428a6d4bcd: Downloading [=====>]
da7391352a9b: Downloading [>]
da7391352a9b: Downloading [=>
da7391352a9b: Downloading [==>
da7391352a9b: Downloading [==>
da7391352a9b: Downloading [==>
da7391352a9b: Downloading [====>]
da801bb9d0b6c: Downloading [==>
4.291MB/71.99MB
```

```
root@VM: /home/seed/Downloads/Labsetup
--> Using cache
--> lcc95471cf90
Step 6/7 : RUN chmod 400 /certs/bank32.key      && chmod 644 $WWWDIR/index.html
          && chmod 644 $WWWDIR/index_red.html      && a2ensite bank32_apache_ssl
--> Using cache
--> 973c18b9d731
Step 7/7 : CMD tail -f /dev/null
--> Using cache
--> 9fdaea0e4627

Successfully built 9fdaea0e4627
Successfully tagged seed-image-www-pki:latest
root@VM:/home/seed/Downloads/Labsetup# docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.80 ... done
Attaching to www-10.9.0.80
^CGracefully stopping... (press Ctrl+C again to force)
Stopping www-10.9.0.80 ... done
root@VM:/home/seed/Downloads/Labsetup# docker-compose down
Removing www-10.9.0.80 ... done
Removing network net-10.9.0.0
root@VM:/home/seed/Downloads/Labsetup#
```

```
seed@VM: ~/.../Labsetup
[10/18/22]seed@VM:~/.../Labsetup$ dockps
710a2d02cad4  www-10.9.0.80
[10/18/22]seed@VM:~/.../Labsetup$ docksh 71
root@710a2d02cad4:/#
```

### DNS Setup:

We add two entries in /etc/hosts file. Screenshot is attached

```

Activities Terminal Oct 25 08:50 root@VM: /home/seed
127.0.0.1      localhost
127.0.1.1      VM
127.0.0.1      www.norin.com
127.0.0.1      www.fake.com
127.0.0.1      www.norin.com127.0.0.1  www.norin.com1
27.0.0.1  www.norin.com127.0.0.1  www.norin.com127.0.0
.1  www.norin.com127.0.0.1  www.norin.com127.0.0.1  ww
w.norin.com127.0.0.1  www.norin.com127.0.0.1  www.nori
n.com127.0.0.1  www.norin.com127.0.0.1  www.norin.com
# The following lines are desirable for IPv6 capable h
osts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
"/etc/hosts" 36L, 1081C      5,0-1      Top

```

### Task 1: Becoming a certificate authority

#### Certificate Authority (CA).

We need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. Following command has been used to generate the self-signed certificate for the CA:

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \ -keyout ca.key -out ca.crt -config
openssl.cnf
```

Activities Terminal Oct 30 09:12 root@VM: /home/seed/norin

```
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PK
State or Province Name (full name) [Some-State]:ISB
Locality Name (eg, city) []:ISB
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FAST
Organizational Unit Name (eg, section) []:CNS
Common Name (e.g. server FQDN or YOUR name) []:PKI
Email Address []:norinsalim@gmail.com
```

---

```
.....+++++
.....+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PK
State or Province Name (full name) [Some-State]:ISB
Locality Name (eg, city) []:ISLAMABAD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FAST
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:www.salim2022.com
Email Address []:i220033@nu.edu.pk
root@VM:/home/seed/Downloads/Labsetup#
```

```

94:b9:a3:e5:77:ec:e3:e3:36:1a:d6:02:44:22:cd:66:5b:20:
d6:b1:3a:2d:32:ea:80:33:47:32:b1:43:c0:12:99:92:c1:28:
e0:4b:58:4d:d8:01:8c:cf:11:59:24:40:e5:6a:5e:29:e7:a9:
5f:4b:09:c1:4c:b5:13:e7:9a:4f:c6:f4:6c:03:be:40:75:ca:
52:dd:d2:bc:98:ce:18:85:dd:8b:9e:63:6d:04:aa:37:97:09:
8c:5b:1c:af:21:8a:0f:7a:f6:bf:fd:le:38:30:07:c1:74:71:
50:36:2f:3f:48:ec:54:cb:ef:e1:9f:c2:47:ce:0b:5a:61:f6:
36:bb:29:14:07:46:78:bf:62:9b:d7:50:2b:e1:2a:77:40:9f:
84:8f:42:52:c6:f1:45:ee:44:c9:c9:00:68:81:c1:6b:b4:2d:
61:4c:f8:63:0d:30:21:22:ee:41:9a:b5:70:85:ed:84:26:86:
1b:f9:cb:bb:3d:22:dd:7c:08:9d:0d:9d:6d:c4:e3:7c:a3:e6:
1a:b8:f2:d0:51:61:1c:cd:6d:77:ee:a0:ab:bc:4c:d3:63:90:
fb:19:7a:a2:e3:d1:5d:e4
root@VM:/home/seed/Downloads/Labsetup# openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \-key
out ca.key -out ca.crt \-subj "/CN=www.modelCA.com/O=Model CA LTD./C=US" \-passout pass:dee
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'ca.key'
-----
root@VM:/home/seed/Downloads/Labsetup# █

```

PKI - Word | Right Ctrl

## Task 2: Generating a certificate request for your web server

The command will generate a pair of public/private key, and then create a certificate signing request from the public key.

```
openssl req -newkey rsa:2048 -sha256 \-keyout server.key -out server.csr -config openssl.cnf
```

```

98:dc:ed:a2:e4:4c:7e:91:b5:cc:0f:f8:e6:14:2e:70:fb:07:
71:f6:f1:32:09:d7:b0:48:77:97:71:5d:73:a3:09:aa:2b:90:
a0:91:f7:f9:97:73:d6:96:ab:55:2b:bd:db:10:eb:ee:72:97:
e1:57:b3:2f:b8:ba:b1:bb:93:f1:76:17:79:f0:32:87:b8:24:
d5:c3:05:11:a0:66:44:4c:3b:27:cf:94:68:9c:72:ed:ab:56:
65:99:38:2a:c8:b7:7a:bc:e9:77:3f:66:6c:04:a5:38:8c:43:
1d:a3:a9:2e:c6:87:9e:df:5f:2e:26:1e:15:99:3d:b7:9c:d1:
fa:38:d1:97:9b:6a:d3:b8:6f:e4:69:8b:2d:14:44:54:b4:69:
a1:31:82:20:41:ca:c3:83:d3:32:46:17:57:00:8c:35:e1:b7:
b2:38:53:c6:7e:50:e5:56:2d:dc:35:27:b4:92:51:d9:3c:97:
e2:a0:29:2a:79:6c:89:c9
root@VM:/home/seed/Downloads/Labsetup#
root@VM:/home/seed/Downloads/Labsetup#
root@VM:/home/seed/Downloads/Labsetup# openssl req -newkey rsa:2048 \-keyout server.key
 -out server.csr \-subj "/CN=www.salim2022.com/O=salim2022 Inc./C=US" \-passout pass:dees
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'server.key'
-----
root@VM:/home/seed/Downloads/Labsetup# █

```

PKI - Word | Right Ctrl

```

root@VM: ~/Labsetup
a6:e9:87:fa:ac:aa:91:21:6d:40:f3:12:a6:21:cd:
d0:39:52:f8:3c:cf:f2:18:cf:c8:cd:84:39:58:a3:
26:62:ef:15:85:b4:b5:78:13:08:b7:3b:6f:1b:3b:
be:3a:3f:d0:9a
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDgLoSgwTYdGGzYJvRe49J+0cwfoe9hJnhDBz9J65L0SG5ZoKu
6wt/JnV6hZYykJmY/KJLaHDWXLBZMk5qAyRr9orFh97HKerZFiKEKKWXBpxH87A
HDB9ecuvQ0bb6DprA6vSBSr/cZYYeCxtgRxU1BuK8KcfIgaz2e9ldNMrGQIDAQAB
AoGBAKC0kGx1+p4v1B1dTpuci8Yl9ZE0vdDs3CyCuWxR9BFp7I5VLbLLG6PIDpFe
hXDxgTpCt4Y8WbwdENPDpMdmUrksBmExrvT8Xe+PXxmdZGRpzUbWmd2/DaM8zrp
/xXF8WV6MCCQWdnj46xlsAtH6CuSkY1E0oBPv7yxHWxVMBAkEA+9osE17KeZS2
kidajHABb0KKbS57q1j3X0LVWGRENJG9jUtCii8QsEI0kWI7XL2Vh2/33d6JE8XL
hF2Ql52reQJBAOPfr1ZyRalG9/g1BvuuxRQQrgrygAZhFbAapWDchGtTZK300WS4
daQn0RF/csixzo8W0a+VAP+d3LJF5VNT9KECQDdCy+EEL5E13El9cRTXjrkG9LC4
PlJ+lujEFWPAnI2eE0Ecnk3koXaQqaVc0kKIEEzhE0ejJ1WDS3iCdP6uKECQFXk
l9P06QEglhGB82rk+rZQfznzJFzDNYiW0qWLGYLXAUkeRbX4bad9tqmQz7RkiUaY
CxBq6uBvIaQ1AncBW2ECQQCduoa/zgjL90Y2txpL2Kbph/qsqpEhbUDzEqYhzdA5
Uvg8z/IYz8jNhDlYoyZi7xWftLV4Ewi3028b0746P9Ca
-----END RSA PRIVATE KEY-----
root@VM:~/Labsetup#

```

```

Activities Terminal Oct 30 09:22
root@VM: /home/seed/norin/PKI
ca.key openssl.cnf server.key
root@VM:/home/seed/norin/PKI# openssl ca -config openssl.cnf -policy policy_anything -md sha256 -days 3650 -in server.csr -out server.crt -batch -cert ca.crt -keyfile ca.key
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
ERROR:There is already a certificate for /C=PK/ST=ISB/L=ISB/0=FAST/OU=CNS/CN=www.norin.com/emailAddress=norinsalim@gmail.com
The matching entry has the following details
Type :Valid
Expires on :321021205355Z
Serial Number :1000
File name :unknown
Subject Name :/C=PK/ST=ISB/L=ISB/0=FAST/OU=CNS/CN=www.norin.com/emailAddress=norinsalim@gmail.com
root@VM:/home/seed/norin/PKI#

```

### Task 3: Generating a certificate for your server

```
# Combine the secret key and certificate into one file through cp server.key server.pem
```

```
% cat server.crt >> server.pem
```

```
# Launch the web server using server.pem
```

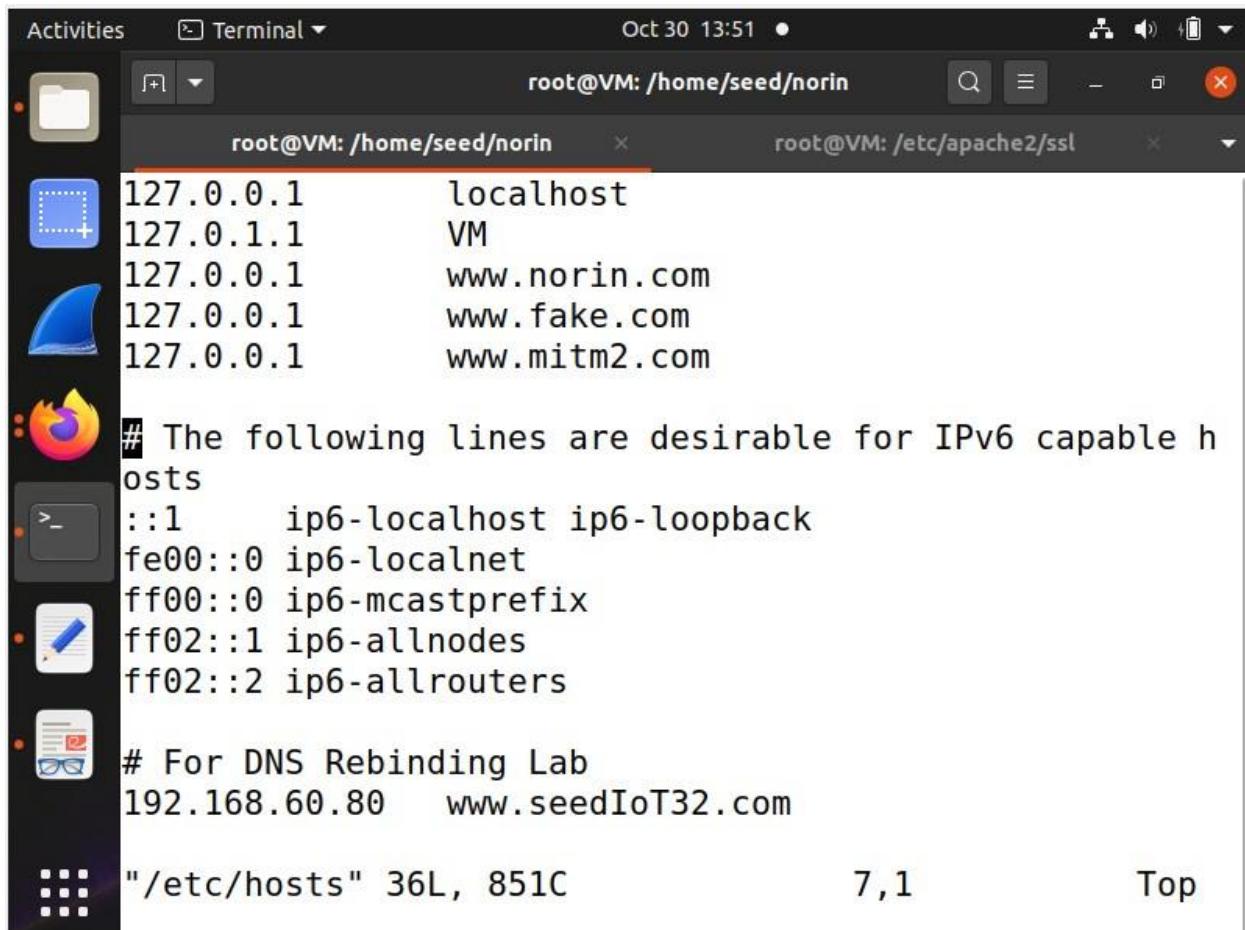
```
% openssl s_server -cert server.pem -www
```

The screenshot shows a terminal window titled "Terminal" with the command line interface. The terminal session is run as root on a VM, specifically at the path /home/seed/norin/PKI. The user is performing several commands related to certificate generation:

- ls: Lists files in the current directory, showing ca.crt, demoCA, server.csr, server.pem, ca.key, openssl.cnf, and server.key.
- cp server.crt cert.pem: An attempt to copy the server certificate to a new file named cert.pem fails because 'server.crt' does not exist.
- cd ..: Changes the directory back to the parent directory (PKI).
- cp server.key server.pem: Copies the server key to a new file named server.pem.
- cp server.crt cert.pem: Copies the server certificate to a new file named cert.pem.
- ls: Lists files again, now including cert.pem, mitm-server.key, mitm-server.csr, and mitm-server.crt.
- demoCA: A folder or command is listed, containing openssl.cnf, server.key, and server.pem.
- ls: Lists files again, including demoCA, ca.crt, ca.key, cert.pem, mitm-server.key, mitm-server.csr, and mitm-server.crt.

### Task 4: Deploying Certificate in an Apache-Based HTTPS Website

Add entry in /etc/hosts file



Activities Terminal Oct 30 13:51 root@VM: /home/seed/norin root@VM: /etc/apache2/ssl

```

root@VM: /home/seed/norin
127.0.0.1      localhost
127.0.1.1      VM
127.0.0.1      www.norin.com
127.0.0.1      www.fake.com
127.0.0.1      www.mitm2.com

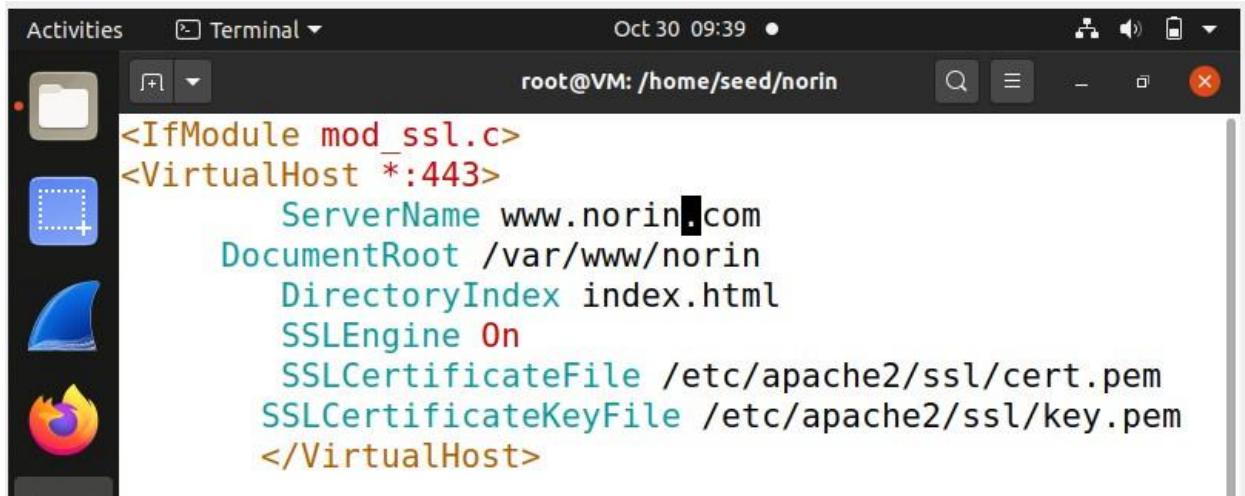
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

"/etc/hosts" 36L, 851C          7,1           Top

```

**Command: Edit Apache conf file through command vi /etc/apache2/sites-available/default-ssl.conf**

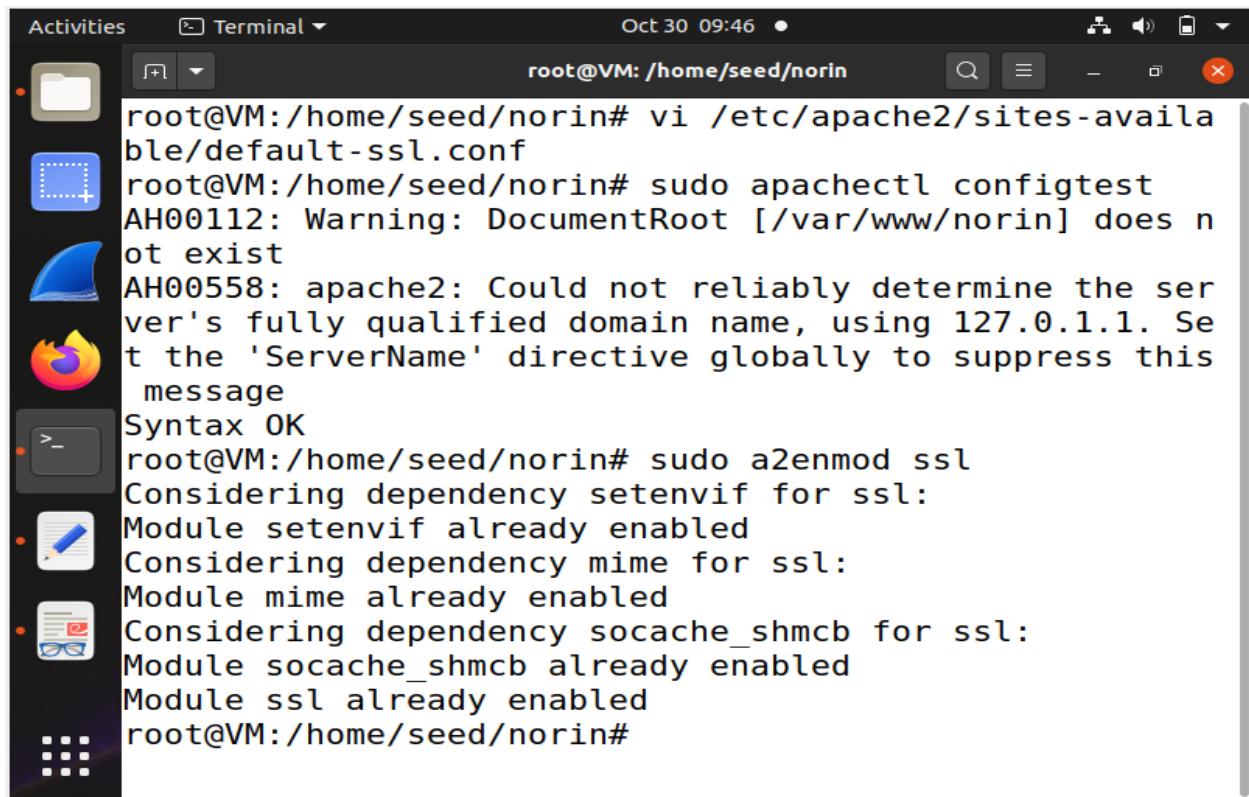


Activities Terminal Oct 30 09:39 root@VM: /home/seed/norin

```

<IfModule mod_ssl.c>
<VirtualHost *:443>
    ServerName www.norin.com
    DocumentRoot /var/www/norin
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/cert.pem
    SSLCertificateKeyFile /etc/apache2/ssl/key.pem
</VirtualHost>

```



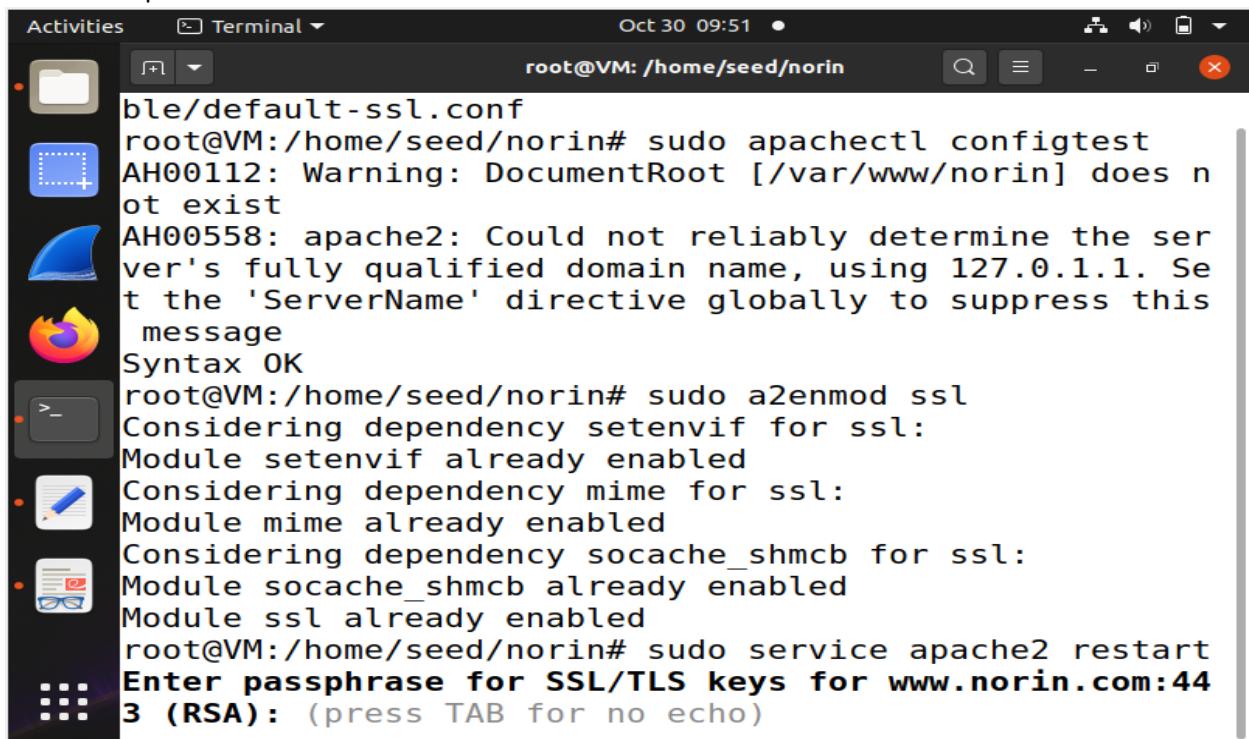
Activities Terminal Oct 30 09:46 root@VM: /home/seed/norin

```

root@VM:/home/seed/norin# vi /etc/apache2/sites-available/default-ssl.conf
root@VM:/home/seed/norin# sudo apachectl configtest
AH00112: Warning: DocumentRoot [/var/www/norin] does not exist
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Syntax OK
root@VM:/home/seed/norin# sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
root@VM:/home/seed/norin#

```

Restarted Apache:



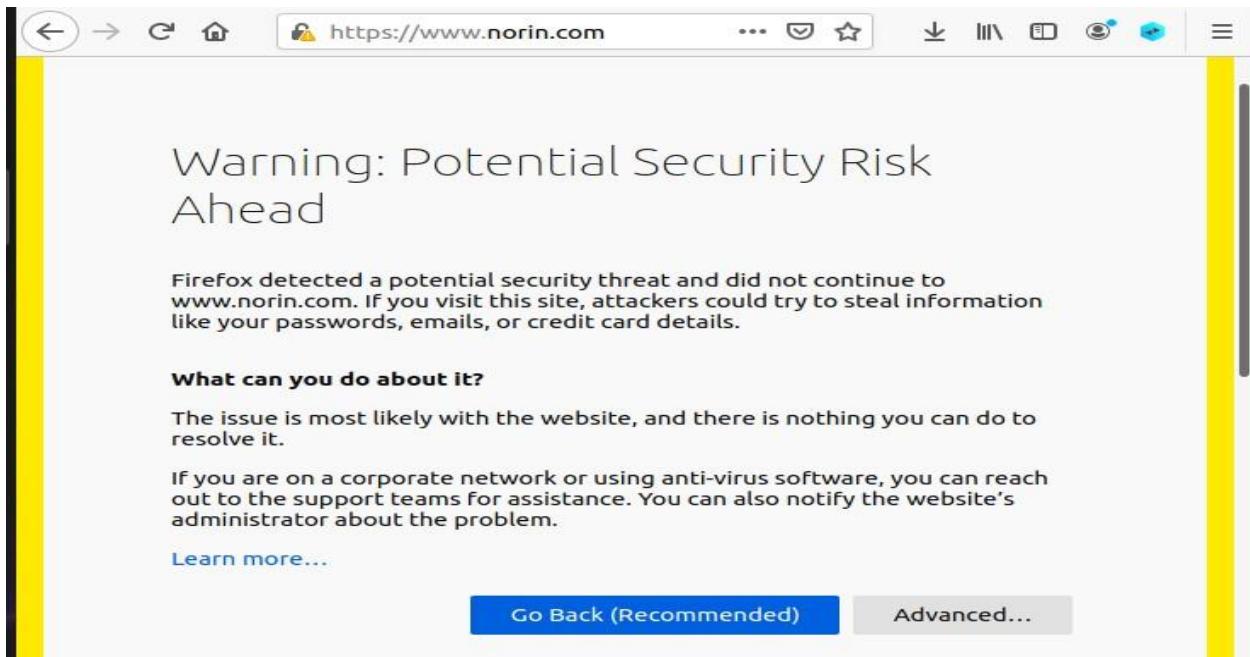
Activities Terminal Oct 30 09:51 root@VM: /home/seed/norin

```

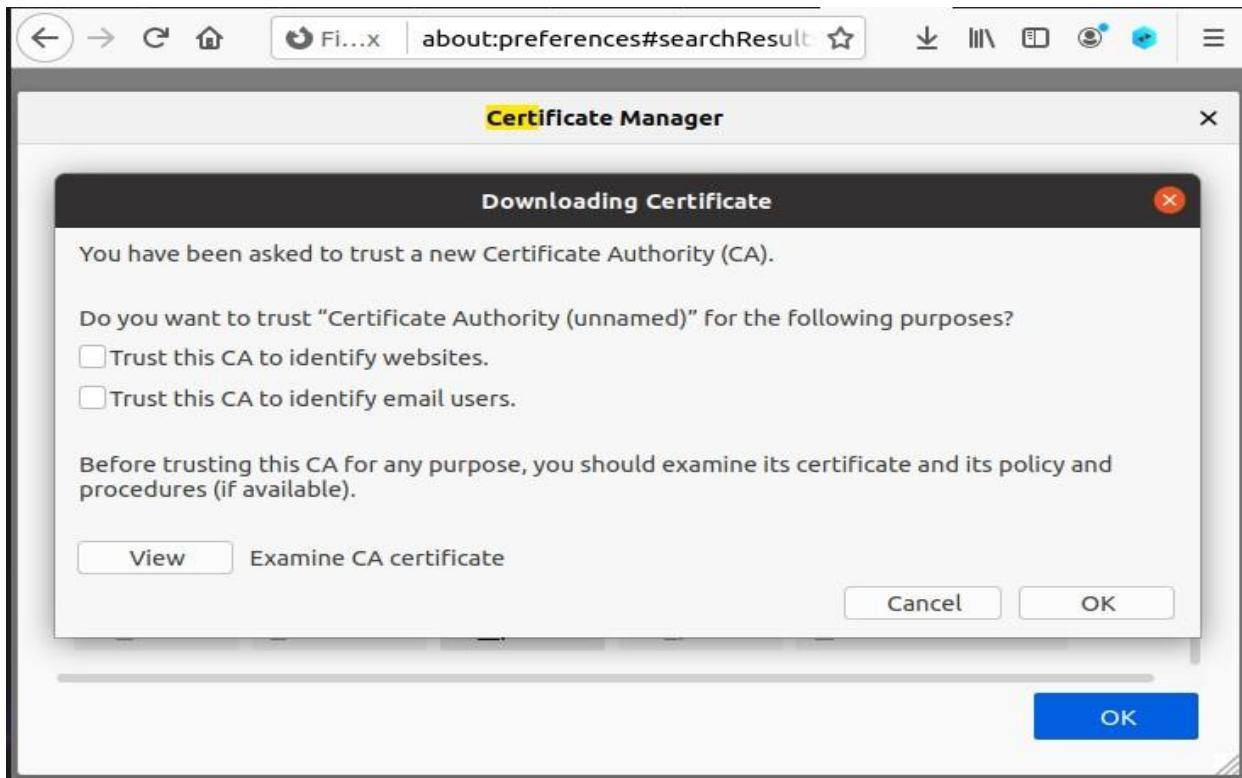
ble/default-ssl.conf
root@VM:/home/seed/norin# sudo apachectl configtest
AH00112: Warning: DocumentRoot [/var/www/norin] does not exist
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Syntax OK
root@VM:/home/seed/norin# sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
root@VM:/home/seed/norin# sudo service apache2 restart
Enter passphrase for SSL/TLS keys for www.norin.com:443 (RSA): (press TAB for no echo)

```

Website without importing the certificate:



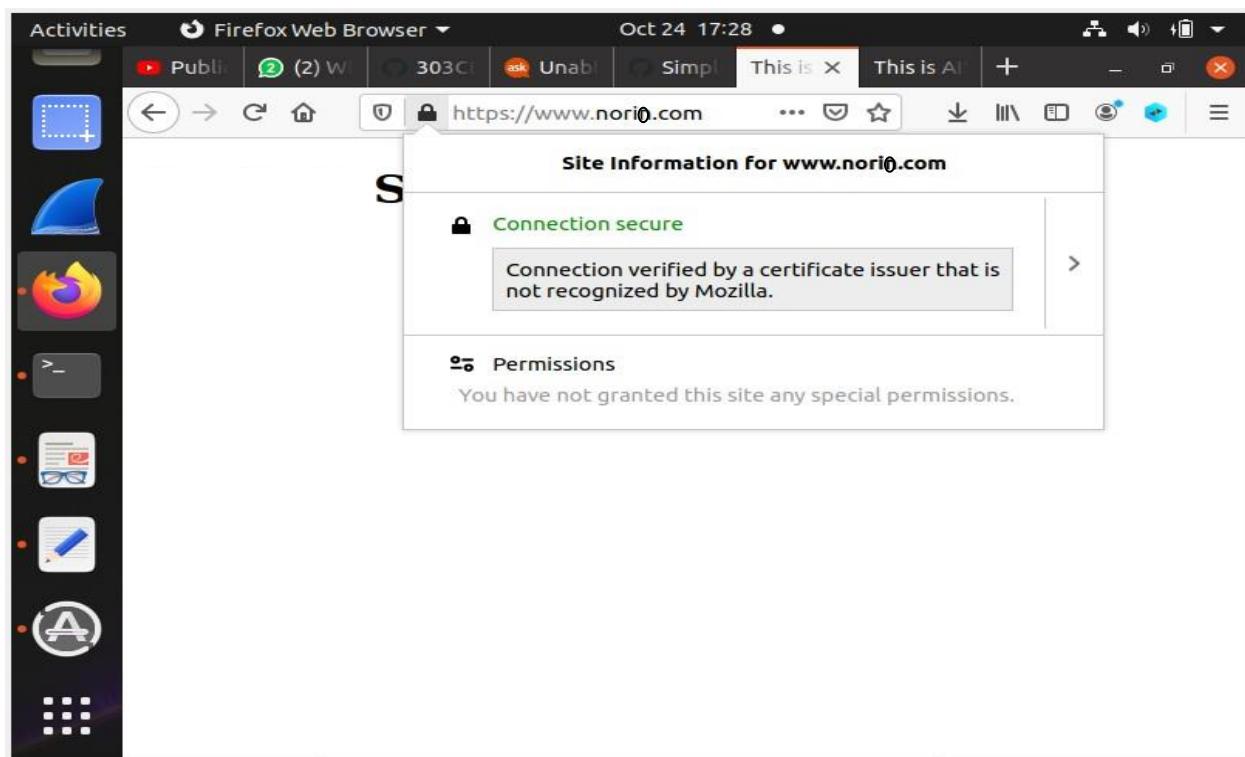
We imported the certificate on the browser here:



Here is website with valid certificate:

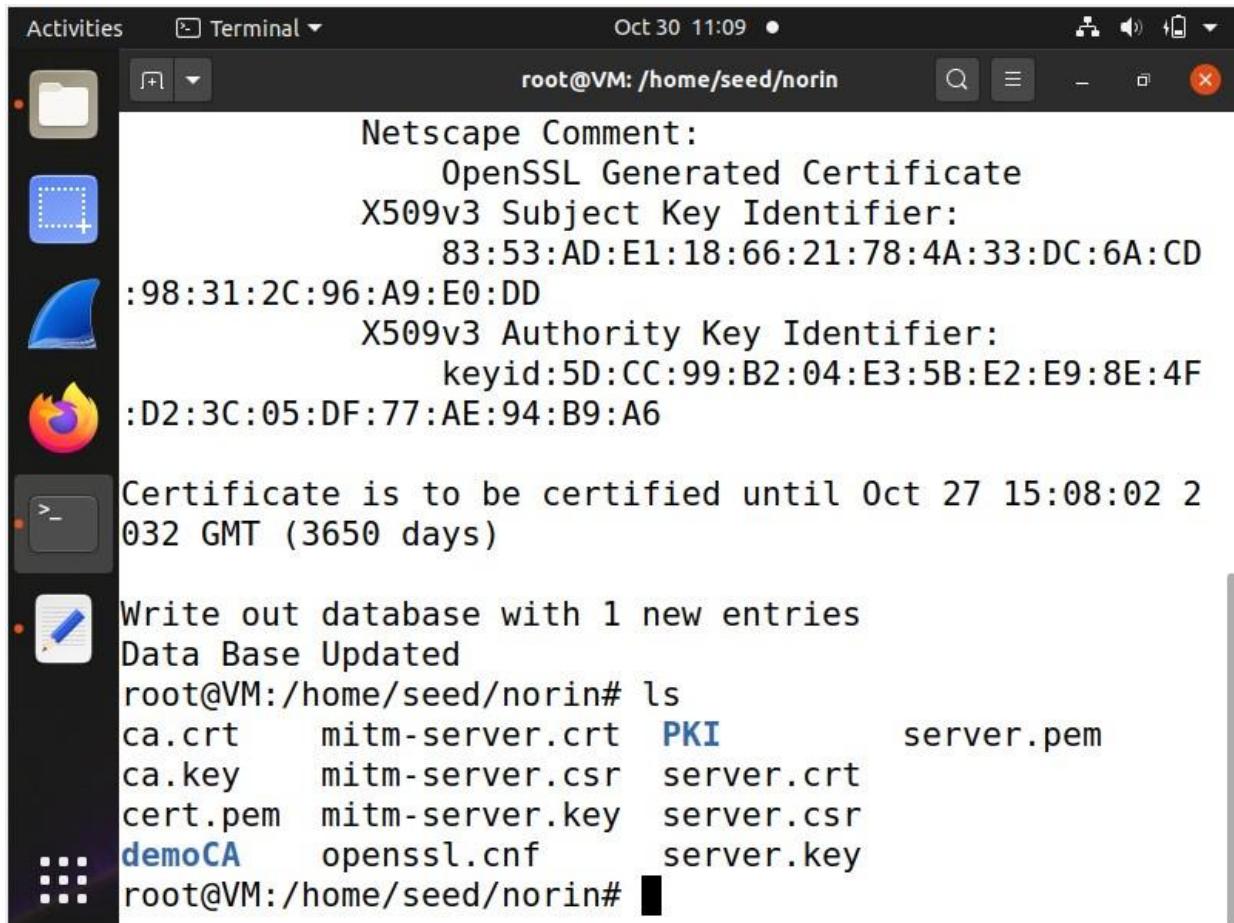


Screenshot is attached with valid certificate info



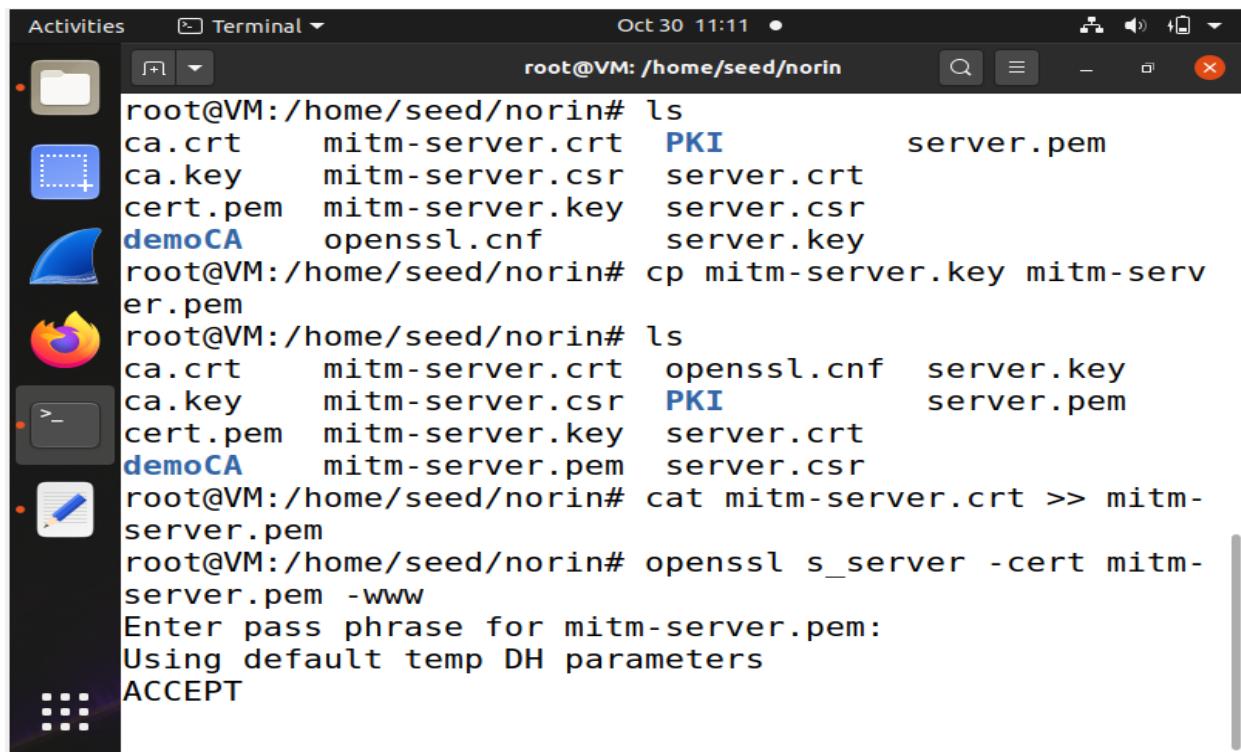
## Task 5: Launching a Man-in-the-Middle attack

### Signing the server with already generated ca.key



The screenshot shows a terminal window titled "Terminal" running as root on a VM. The terminal displays the following output:

```
Netscape Comment:  
      OpenSSL Generated Certificate  
      X509v3 Subject Key Identifier:  
          83:53:AD:E1:18:66:21:78:4A:33:DC:6A:CD  
          :98:31:2C:96:A9:E0:DD  
      X509v3 Authority Key Identifier:  
          keyid:5D:CC:99:B2:04:E3:5B:E2:E9:8E:4F  
          :D2:3C:05:DF:77:AE:94:B9:A6  
  
Certificate is to be certified until Oct 27 15:08:02 2  
032 GMT (3650 days)  
  
Write out database with 1 new entries  
Data Base Updated  
root@VM:/home/seed/norin# ls  
ca.crt      mitm-server.crt  PKI           server.pem  
ca.key       mitm-server.csr  server.crt  
cert.pem    mitm-server.key  server.csr  
demoCA      openssl.cnf     server.key  
root@VM:/home/seed/norin#
```



```

root@VM:/home/seed/norin# ls
ca.crt      mitm-server.crt  PKI          server.pem
ca.key       mitm-server.csr  server.crt
cert.pem    mitm-server.key   server.csr
demoCA      openssl.cnf     server.key
root@VM:/home/seed/norin# cp mitm-server.key mitm-server.pem
root@VM:/home/seed/norin# ls
ca.crt      mitm-server.crt  openssl.cnf  server.key
ca.key       mitm-server.csr  PKI          server.pem
cert.pem    mitm-server.key   server.crt
demoCA      mitm-server.pem  server.csr
root@VM:/home/seed/norin# cat mitm-server.crt >> mitm-server.pem
root@VM:/home/seed/norin# openssl s_server -cert mitm-server.pem -www
Enter pass phrase for mitm-server.pem:
Using default temp DH parameters
ACCEPT

```

## Step 1: Setting up the malicious website.



```

root@VM:/home/seed/norin          root@VM:/etc/apache2/ssl
127.0.0.1      localhost
127.0.1.1      VM
127.0.0.1      www.norin.com
127.0.0.1      www.fake.com
127.0.0.1      www.mitm2.com

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

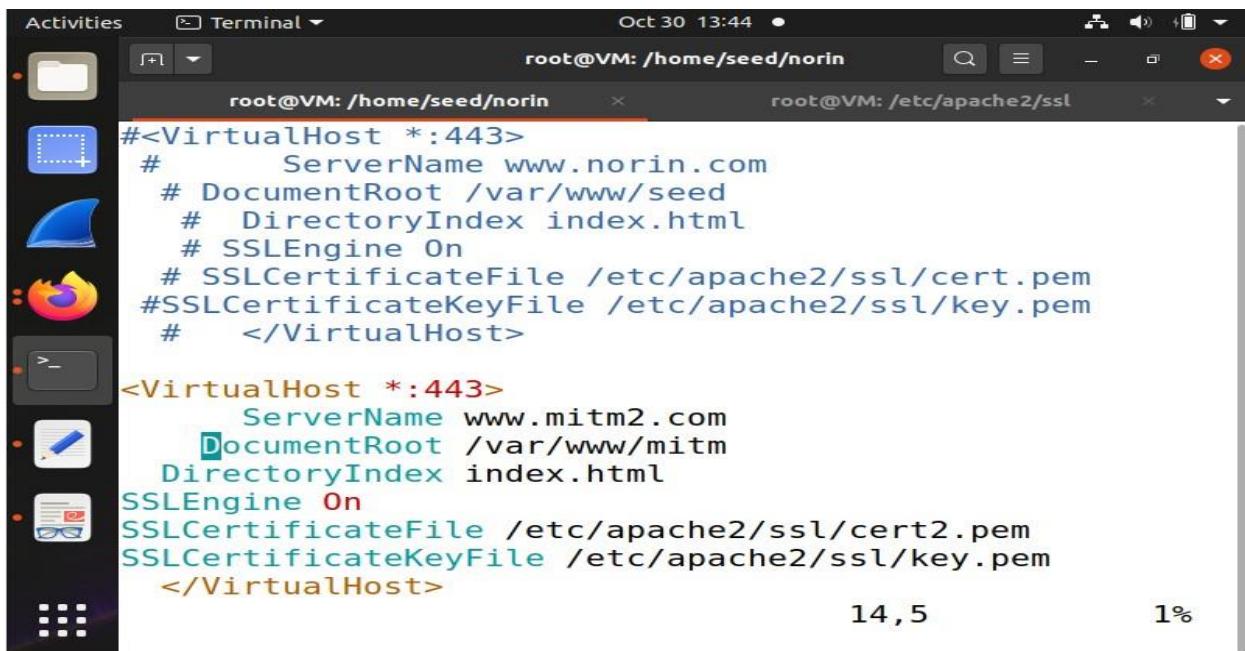
# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

"/etc/hosts" 36L, 851C           7,1           Top

```

**Step 2:** Becoming the man in the middle.

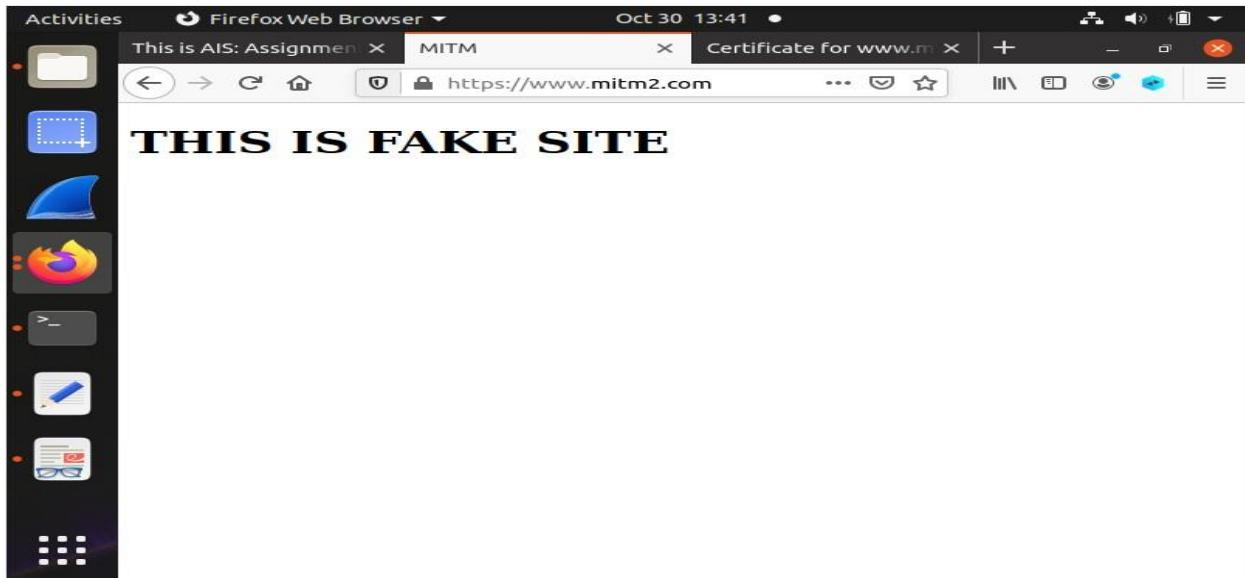
Edit Apache conf file through command vi /etc/apache2/sites-available/default-ssl.conf

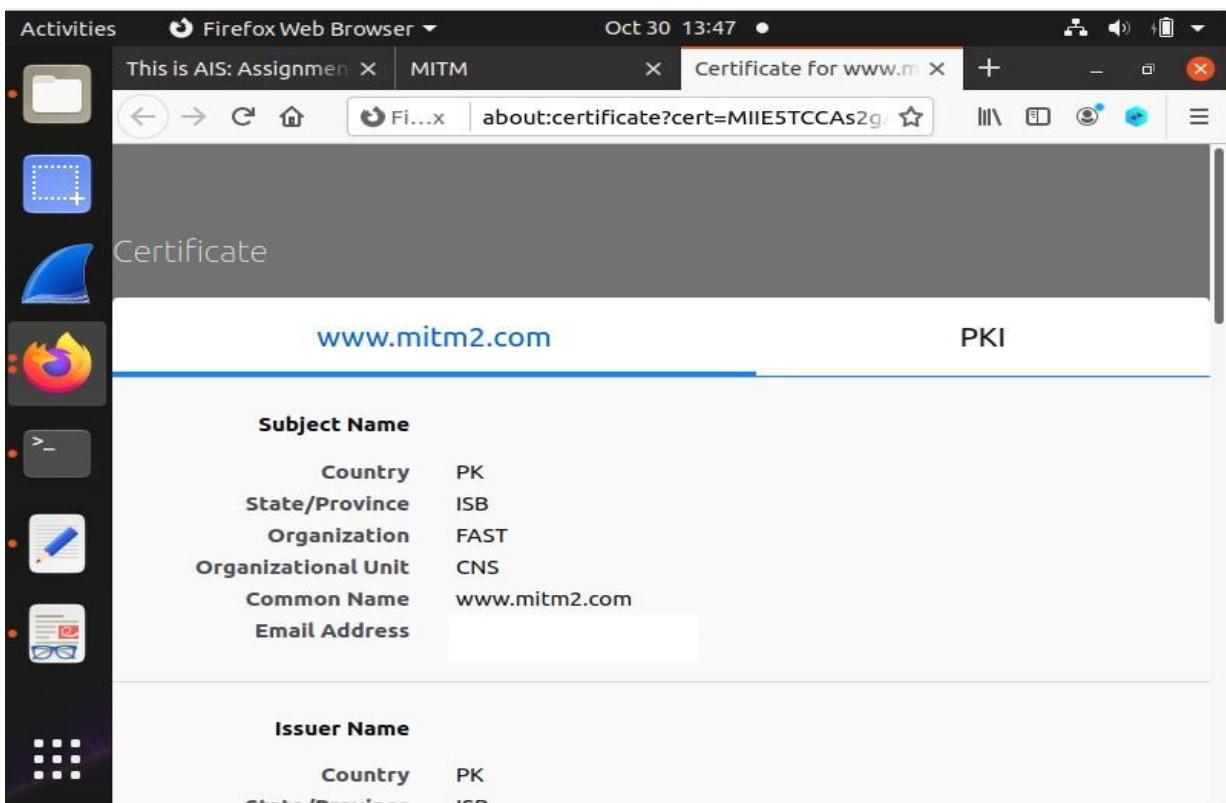
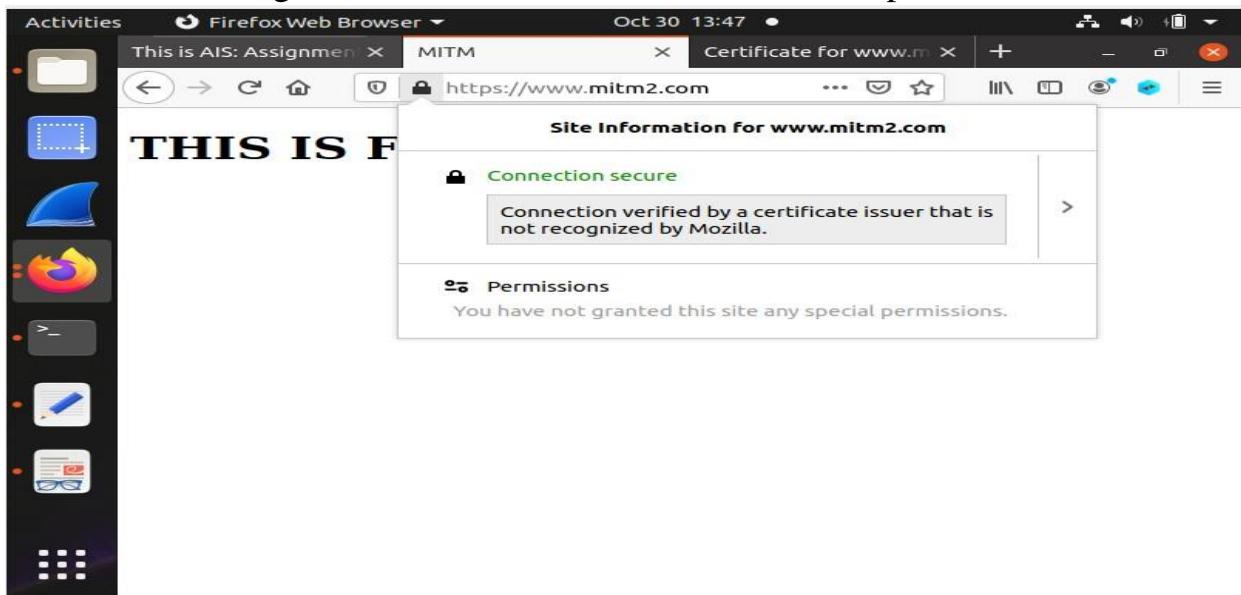


```
#<VirtualHost *:443>
#   ServerName www.norin.com
#   DocumentRoot /var/www/seed
#   DirectoryIndex index.html
#   SSLEngine On
#   SSLCertificateFile /etc/apache2/ssl/cert.pem
#SSLCertificateKeyFile /etc/apache2/ssl/key.pem
#   </VirtualHost>

<VirtualHost *:443>
    ServerName www.mitm2.com
    DocumentRoot /var/www/mitm
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/cert2.pem
    SSLCertificateKeyFile /etc/apache2/ssl/key.pem
    </VirtualHost>
```

**Step 3:** Browse the target website.



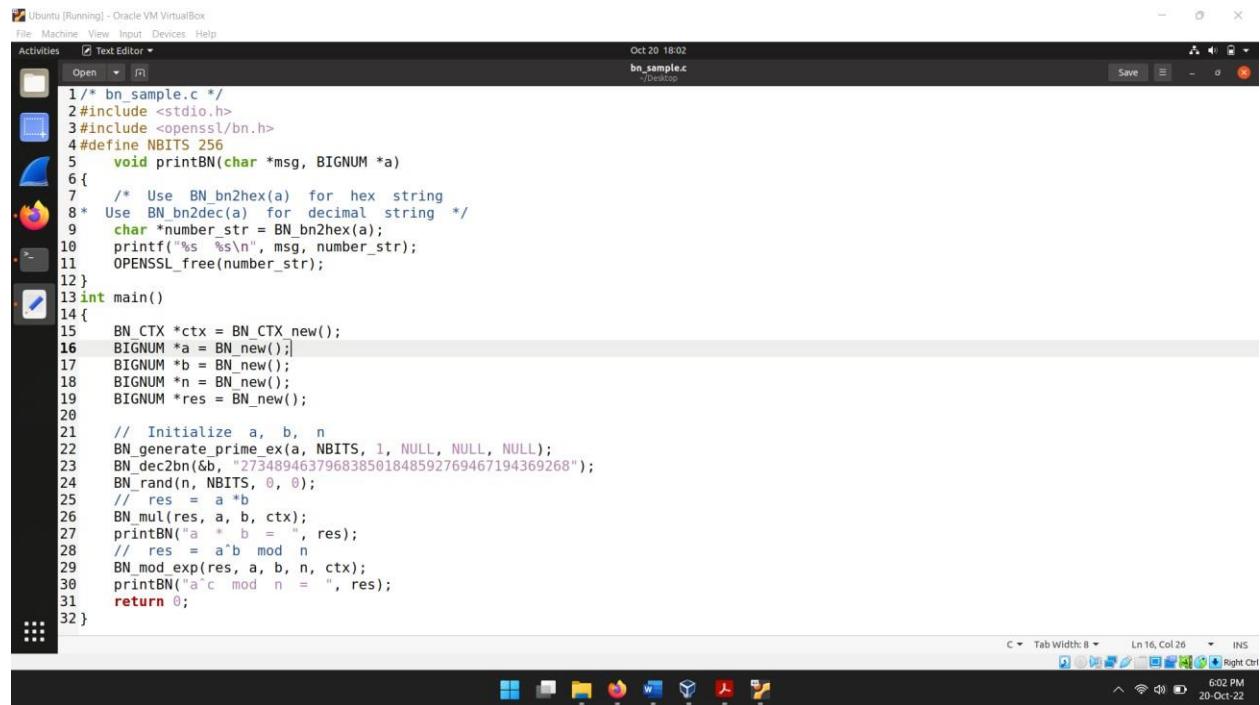
**Task 6:** Launching a Man-in-the-Middle attack with a compromised CA.

## *Section-2*

## RSA Encryption and Signature Lab

**Pre-Requisite Tasks:** Implement the BigNum example, compile and execute it.

First, we will write all our C language code in Text Editor and save it as *bn\_sample.c*. In the code we initialize a,b,n for BigNum example and used <*openssl/bn.h*> library for Big Number problem solution. a\*b and (a^b mod n) will be used for computation.



```

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor
Oct 20 18:02
bn_sample.c
-/Desktop
Save
1/* bn_sample.c */
2#include <stdio.h>
3#include <openssl/bn.h>
4#define NBITS 256
5    void printBN(char *msg, BIGNUM *a)
6{
7    /* Use BN_bn2hex(a) for hex string
8 * Use BN_bn2dec(a) for decimal string */
9    char *number_str = BN_bn2hex(a);
10   printf("%s %s\n", msg, number_str);
11   OPENSSL_free(number_str);
12 }
13int main()
14{
15    BN_CTX *ctx = BN_CTX_new();
16    BIGNUM *a = BN_new();
17    BIGNUM *b = BN_new();
18    BIGNUM *n = BN_new();
19    BIGNUM *res = BN_new();
20
21    // Initialize a, b, n
22    BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
23    BN_dec2bn(&b, "273489463796838501848592769467194369268");
24    BN_rand(n, NBITS, 0, 0);
25    // res = a * b
26    BN_mul(res, a, b, ctx);
27    printBN("a * b = ", res);
28    // res = a^b mod n
29    BN_mod_exp(res, a, b, n, ctx);
30    printBN("a^c mod n = ", res);
31    return 0;
32 }
```

Now we open Terminal and write commands:

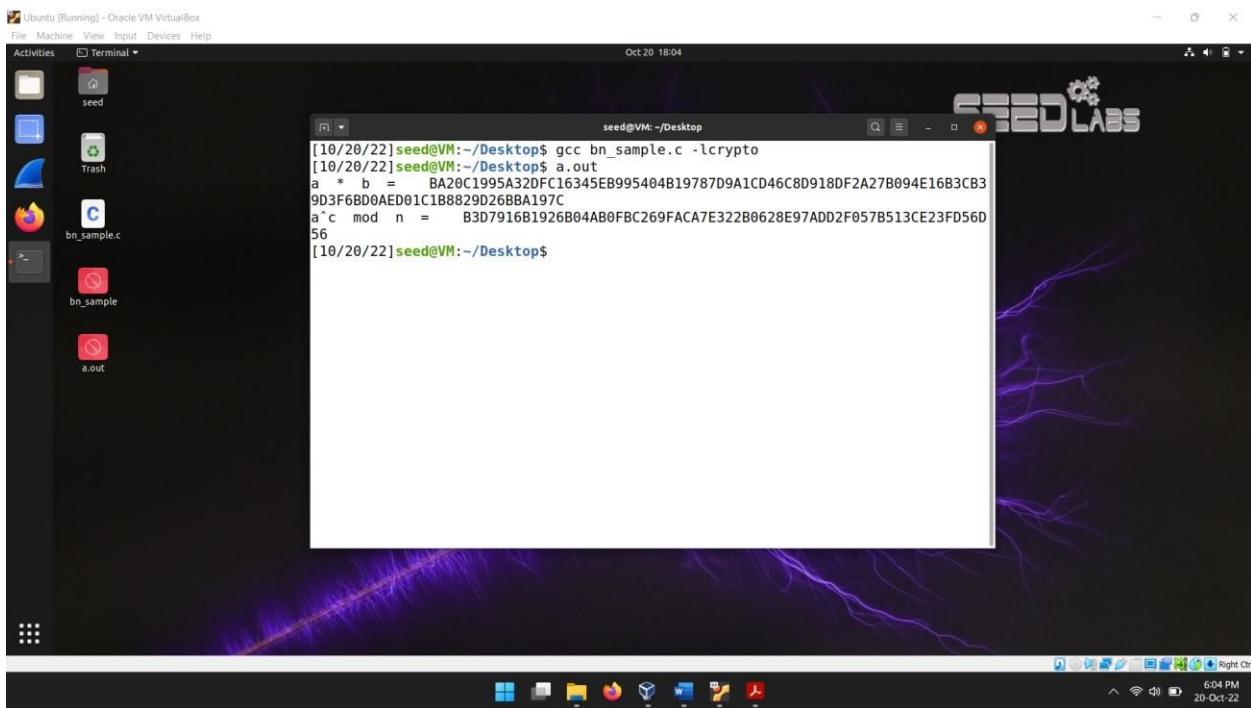
**gcc bn\_sample.c -lcrypto** to compile our C program,

**a.out** to run our C program.

After ‘–’ its small L, not the number 1(it tells compiler to use crypto library). its compiled and run successfully and we got following response:

**a \* b =**  
**BA20C1995A32DFC16345EB995404B19787D9A1CD46C8D918DF2A27B094E16B3CB39D**  
**3F6BD0AED01C1B8829D26BBA197C**

**a^c mod n =**  
**B3D7916B1926B04AB0FBC269FACA7E322B0628E97ADD2F057B513CE23FD56D56**



## Task 1: Deriving the Private Key

For our simplicity we choose 128-bit values,

Let  $p = F7E75FDC469067FFDC4E847C51F452DF$

$q = E85CED54AF57E53E092113E62F436F4F$

$e = 0D88C3$

from previous Big Number Task we get to know an idea about how can we get our private key  $d$  when  $p, q, e$  and  $n(p * q)$  are given already. We will use  $(e, n)$  as a public key. Here we have hexadecimal value of  $p, q$  and  $e$ , our  $d$  will also be in hexadecimal value.

We named our file ***task1.c*** which we created through Text Editor in Ubuntu VM, below is the code which you can see it below.

A screenshot of a Linux desktop environment (Ubuntu) running in Oracle VM VirtualBox. The desktop has a dark theme with purple lightning bolt wallpaper. In the top right corner, there's a system tray with icons for battery, signal, and date/time (20-Oct-22). A terminal window titled 'Terminal' is open, showing the command line history:

```
[10/20/22]seed@VM:~$ cd Desktop
[10/20/22]seed@VM:~/Desktop$ gcc bn_sample.c -lcrypto
[10/20/22]seed@VM:~/Desktop$ gcc task1.c -lcrypto
[10/20/22]seed@VM:~/Desktop$
```

The desktop also features a dock at the bottom with icons for various applications like a file manager, browser, and terminal.

Now we compile it using Terminal and by entering command:

**gcc task1.c -lcrypto**

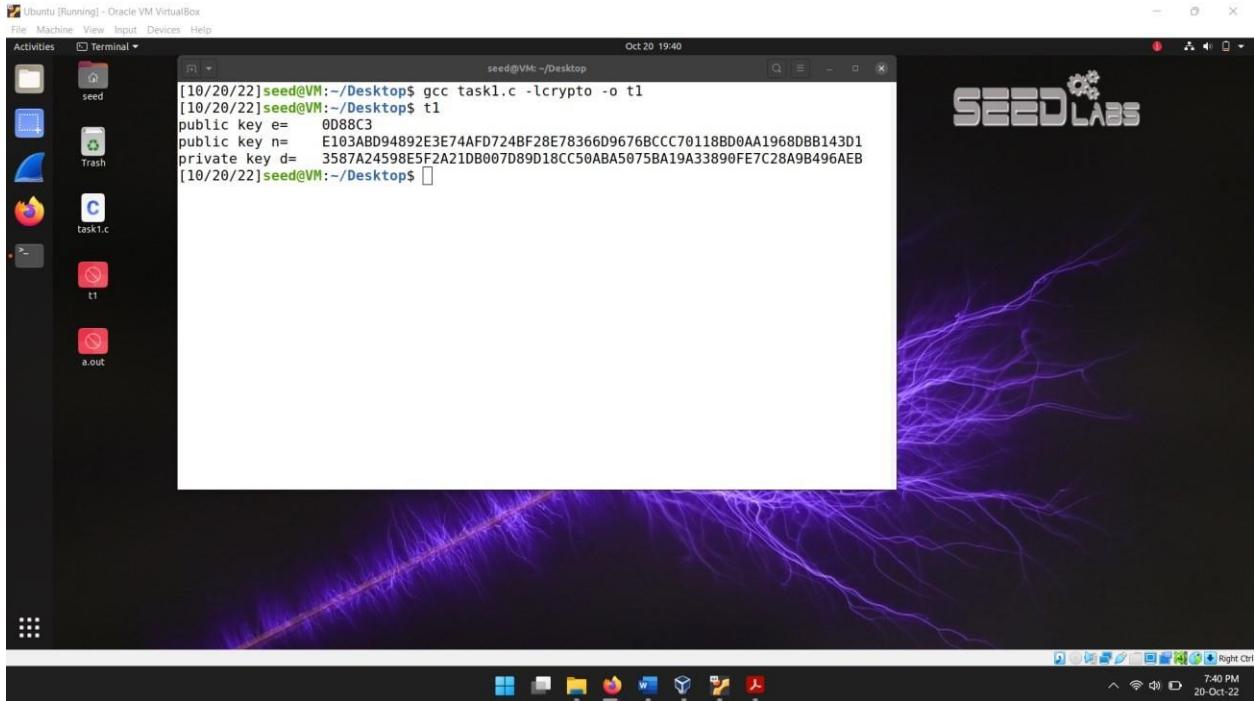
A screenshot of a Linux desktop environment (Ubuntu) running in Oracle VM VirtualBox. The desktop has a dark theme with purple lightning bolt wallpaper. A terminal window titled 'Terminal' is open, showing the command line history:

```
[10/20/22]seed@VM:~$ cd Desktop
[10/20/22]seed@VM:~/Desktop$ gcc bn_sample.c -lcrypto
[10/20/22]seed@VM:~/Desktop$ gcc task1.c -lcrypto
[10/20/22]seed@VM:~/Desktop$
```

The desktop also features a dock at the bottom with icons for various applications like a file manager, browser, and terminal.

Our file got compiled successfully, now we run it by command:

```
gcc task1.c -lcrypto -o t1
t1
```



We got the value of

**public key e= 0D88C3**

**public key n=**

E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1

**private key d=**

3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

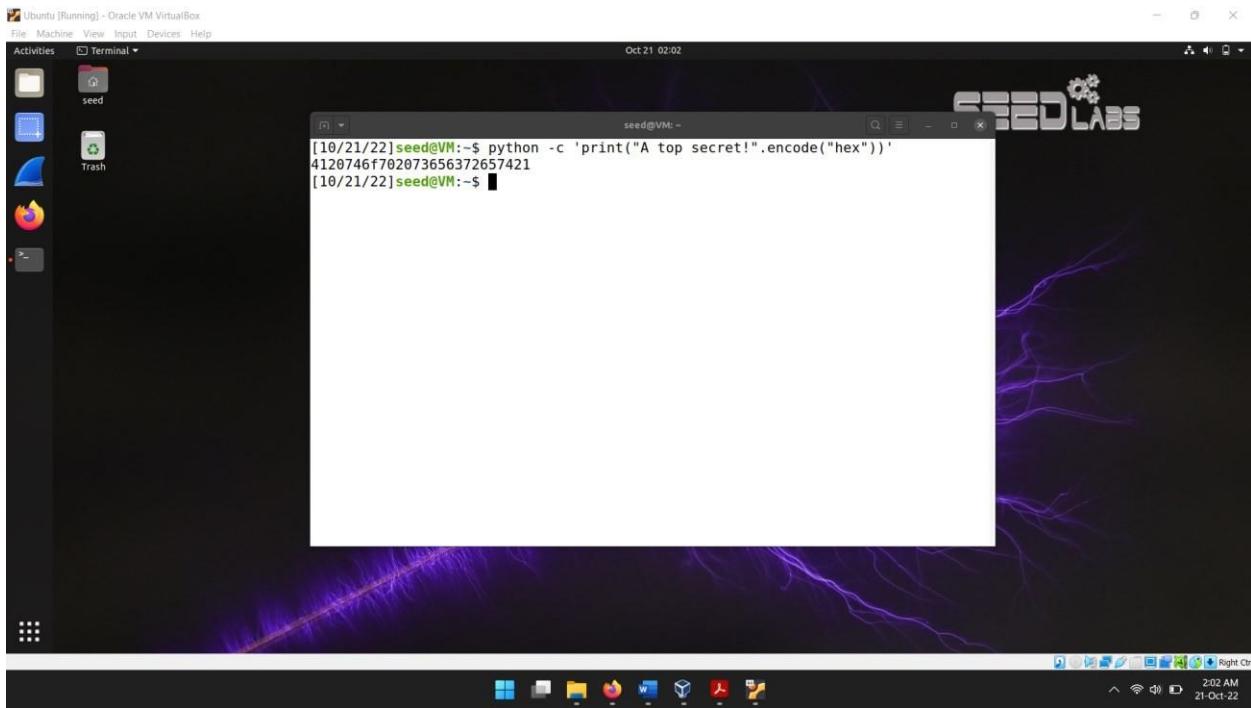
## Task 2: Encrypting a Message

Message to encrypt are “A top secret!” without quotes, first we convert ASCII in hex string by the **python** command:

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

The hex string, we got are:

4120746f702073656372657421



Now its time to convert hex string to BigNum using hex to bn Api. For that we wrote a code in C language using Text Editor and named it task2.c

```

1 #include <stdio.h>
2 #include <openssl/bn.h>
3 #define NBITS 128
4 void printBN(char *msg, BIGNUM *a)
5 { /* Use BN_bn2dec(a) for decimal string */
6     char *number_str = BN_bn2hex(a);
7     printf("%s %s\n", msg, number_str);
8     OPENSSL_free(number_str);
9 }
10
11 int main()
12 {
13     BN_CTX *ctx = BN_CTX_new();
14
15     BIGNUM *n = BN_new();
16     BIGNUM *e = BN_new();
17     BIGNUM *d = BN_new();
18     BIGNUM *m = BN_new(); //massage
19     //BIGNUM *p = BN_new(); //plaintext
20     BIGNUM *c = BN_new(); //ciphertext
21
22     BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
23     BN_hex2bn(&e, "010001");
24     BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
25     BN_hex2bn(&m, "4120746f702073656372657421");
26
27     BN_mod_exp(c, m, e, n, ctx);
28     //BN_mod_exp(p, c, d, n, ctx);
29     printBN("ciphertext:", c);
30     //printBN("plaintext:", p);
31
32     return 0;
33 }
```

To see our encrypted text, we compiled and run it by

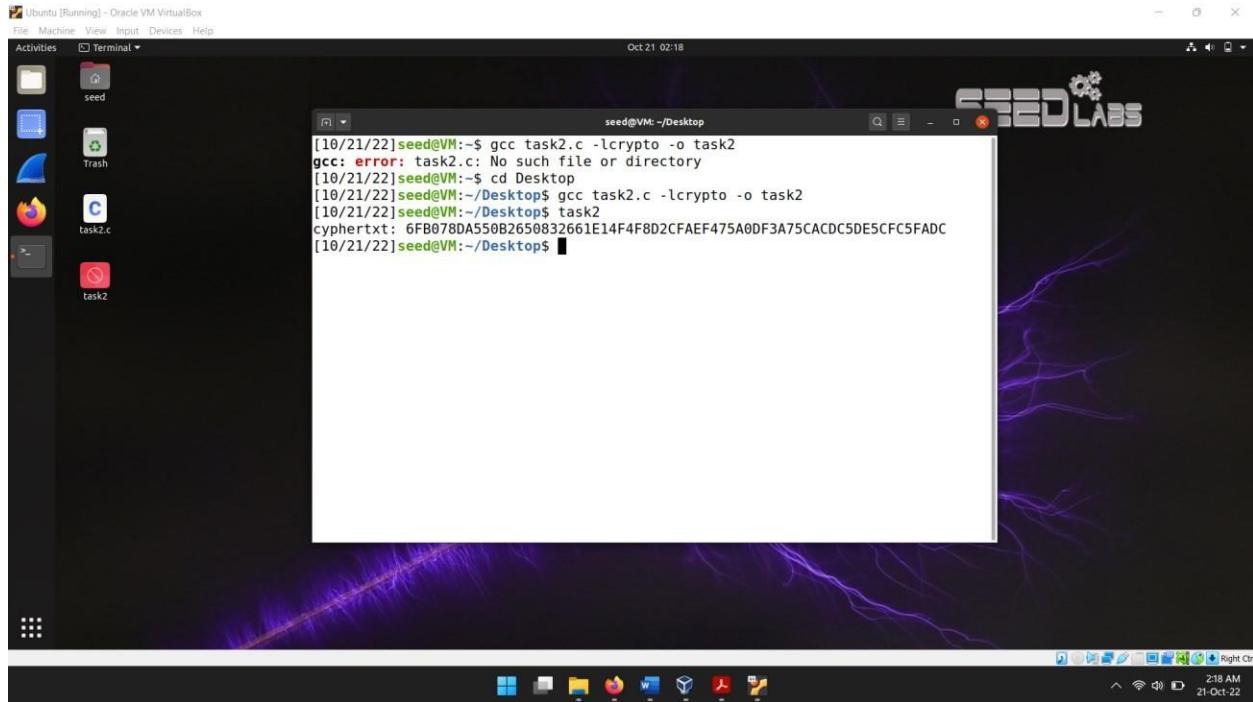
**gcc task2.c -lcrypto -o task2**

## task2

we got out encrypted text that is

ciphertext:

6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC



## Task 3: Decrypting a Message

Cypher text to decrypt is

“8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F”  
(without quotes)

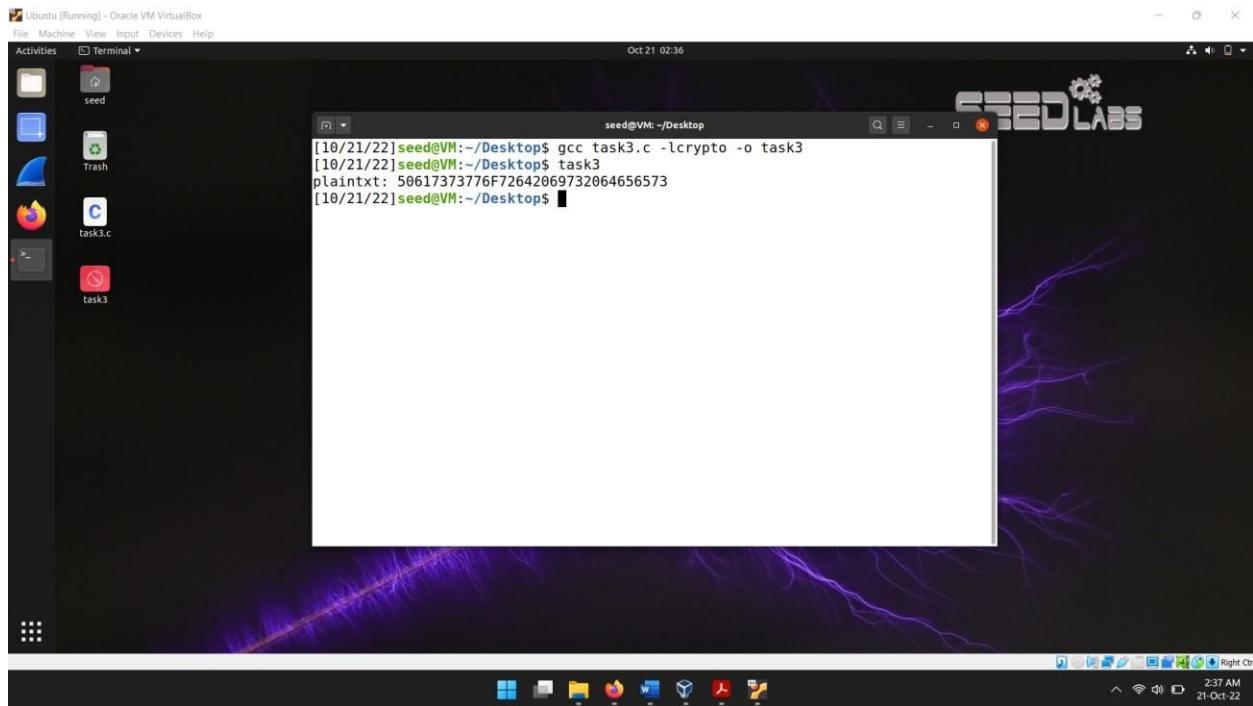
Keys are same which we used in previous task, first we code in text editor which you can see in below image and save it as **task3.c** then run by command:

**gcc task3.c -lcrypto -o task3**

## task3

In output we got following result,

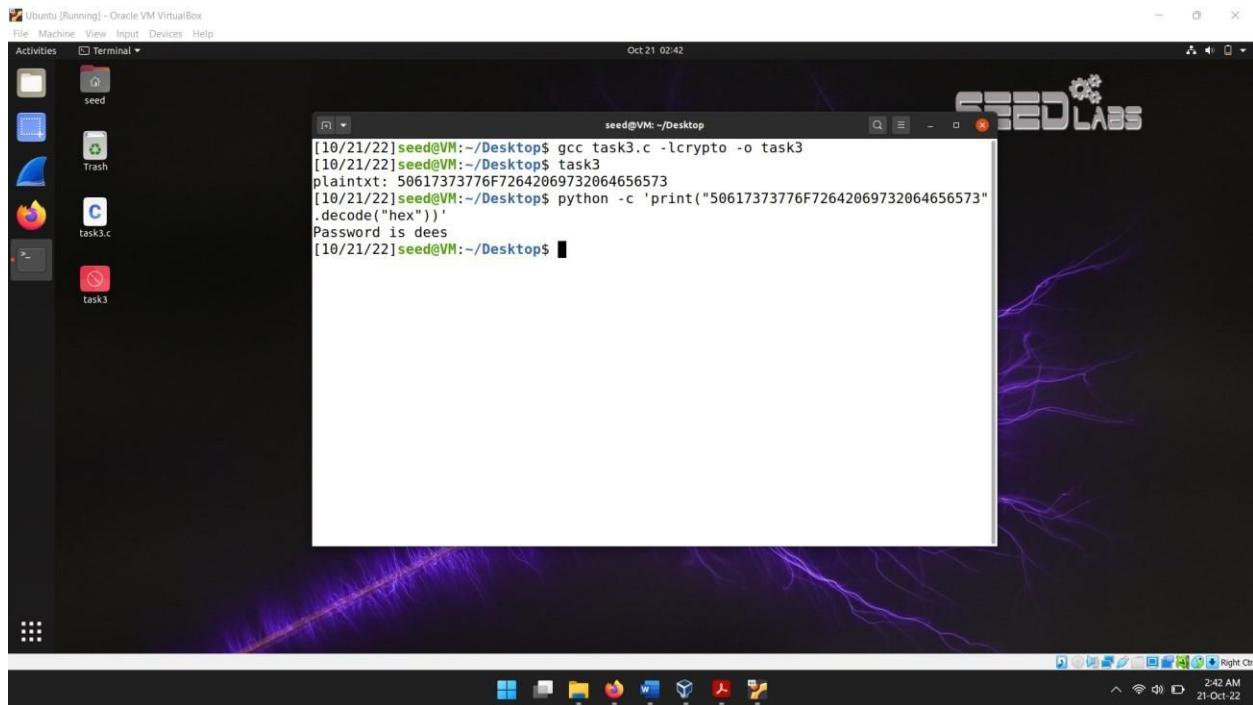
plaintxt: 50617373776F72642069732064656573



Now decrypt hex to ASCII using python script

```
python -c 'print("50617373776F72642069732064656573".decode("hex"))'
```

The result we got in ASCII is: Password is dees



## Task 4: Signing a Message

Message one: I owe you \$2000.

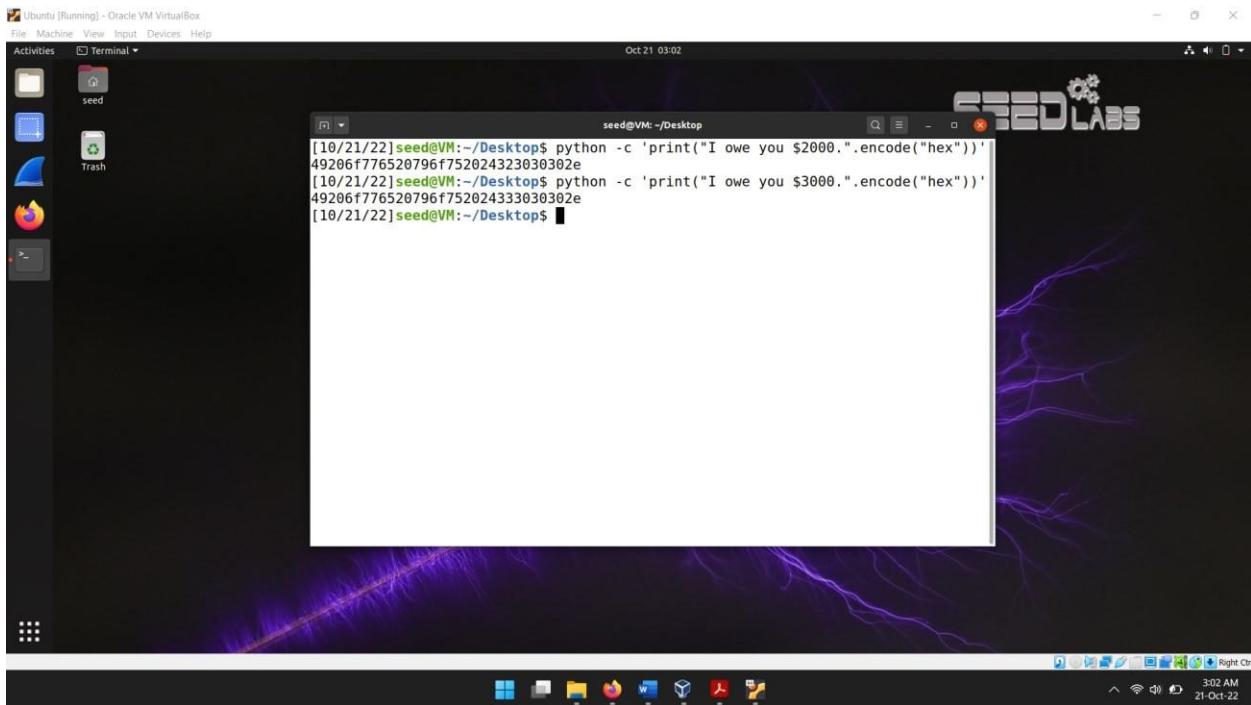
Message two: I owe you \$3000.

Private and Public keys are same used in Task 2

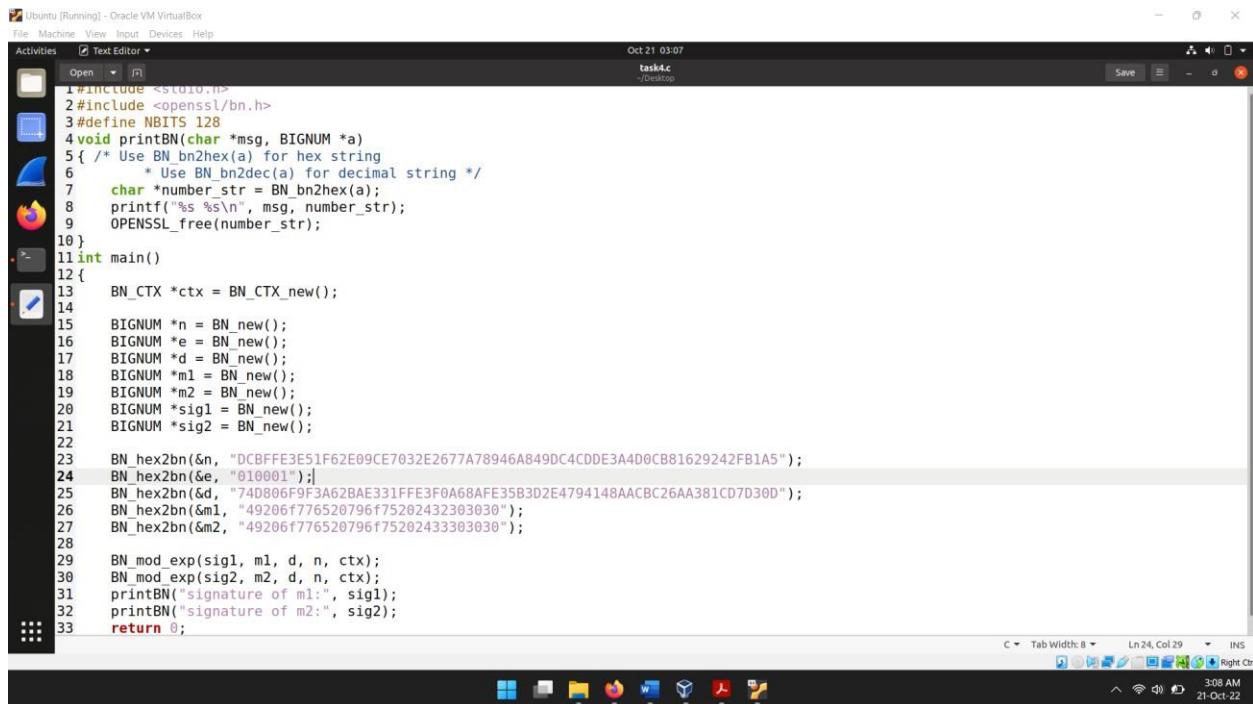
To generate signature for above messages we first of all get the hex value of the ASCII strings through python command and in return we got following value.

Hexa one: 49206f776520796f752024323030302e

Hexa two: 49206f776520796f752024333030302e



We sign direct message instead of signing the message hash for that purpose wrote C code in text editor and save it as *task4.c*.



```

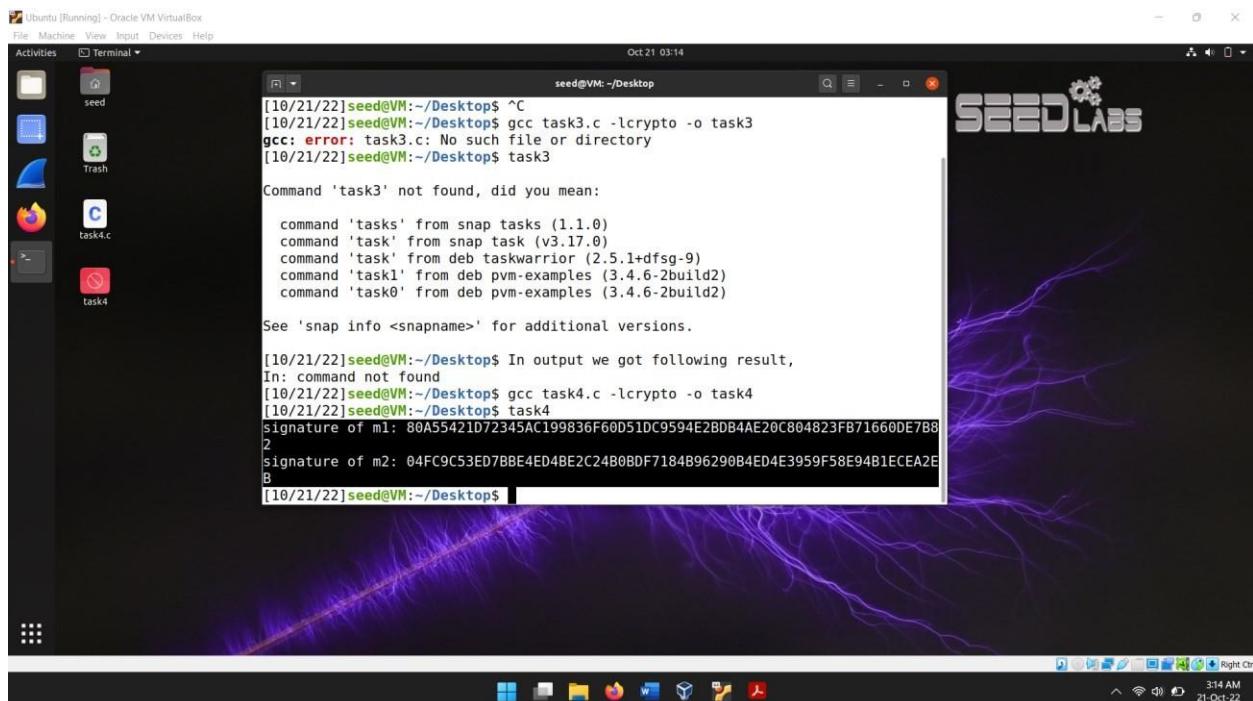
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Open Text Editor ▾
Open ... Save ...
task4.c
Oct 21 03:07
task4.c
-/Desktop
1#include <stdio.h>
2#include <openssl/bn.h>
3#define NBITS 128
4void printBN(char *msg, BIGNUM *a)
5{ /* Use BN_bn2hex(a) for hex string
   *      * Use BN_bn2dec(a) for decimal string */
7    char *number_str = BN_bn2hex(a);
8    printf("%s %s\n", msg, number_str);
9    OPENSSL_free(number_str);
10}
11int main()
12{
13    BN_CTX *ctx = BN_CTX_new();
14
15    BIGNUM *n = BN_new();
16    BIGNUM *e = BN_new();
17    BIGNUM *d = BN_new();
18    BIGNUM *m1 = BN_new();
19    BIGNUM *m2 = BN_new();
20    BIGNUM *sig1 = BN_new();
21    BIGNUM *sig2 = BN_new();
22
23    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
24    BN_hex2bn(&e, "010001");
25    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AF35B3D2E4794148AACBC26AA381CD7D300");
26    BN_hex2bn(&m1, "49206f776520796f75202432380303");
27    BN_hex2bn(&m2, "49206f776520796f75202433303030");
28
29    BN_mod_exp(sig1, m1, d, n, ctx);
30    BN_mod_exp(sig2, m2, d, n, ctx);
31    printBN("signature of m1:", sig1);
32    printBN("signature of m2:", sig2);
33    return 0;
}

```

We run command ***gcc task4.c -lcrypto -o task4***

## ***task4***

In output we got following result on terminal,



```

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal ▾
Oct 21 03:14
seed@VM:~/Desktop$ ^C
[10/21/22]seed@VM:~/Desktop$ gcc task3.c -lcrypto -o task3
gcc: error: task3.c: No such file or directory
[10/21/22]seed@VM:~/Desktop$ task3
Command 'task3' not found, did you mean:
  command 'tasks' from snap tasks (1.1.0)
  command 'task' from snap task (v3.17.0)
  command 'task' from deb taskwarrior (2.5.1+dfsg-9)
  command 'task1' from deb pvm-examples (3.4.6-2build2)
  command 'task0' from deb pvm-examples (3.4.6-2build2)
See 'snap info <snapname>' for additional versions.
[10/21/22]seed@VM:~/Desktop$ In output we got following result,
In: command not found
[10/21/22]seed@VM:~/Desktop$ gcc task4.c -lcrypto -o task4
[10/21/22]seed@VM:~/Desktop$ task4
signature of m1: 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B8
2
signature of m2: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2E
B
[10/21/22]seed@VM:~/Desktop$ 

```

Result:

signature of m1:

80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82

signature of m2:

04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB

Even we done small change but the result is totally different.

## Task 5: Verifying a Signature

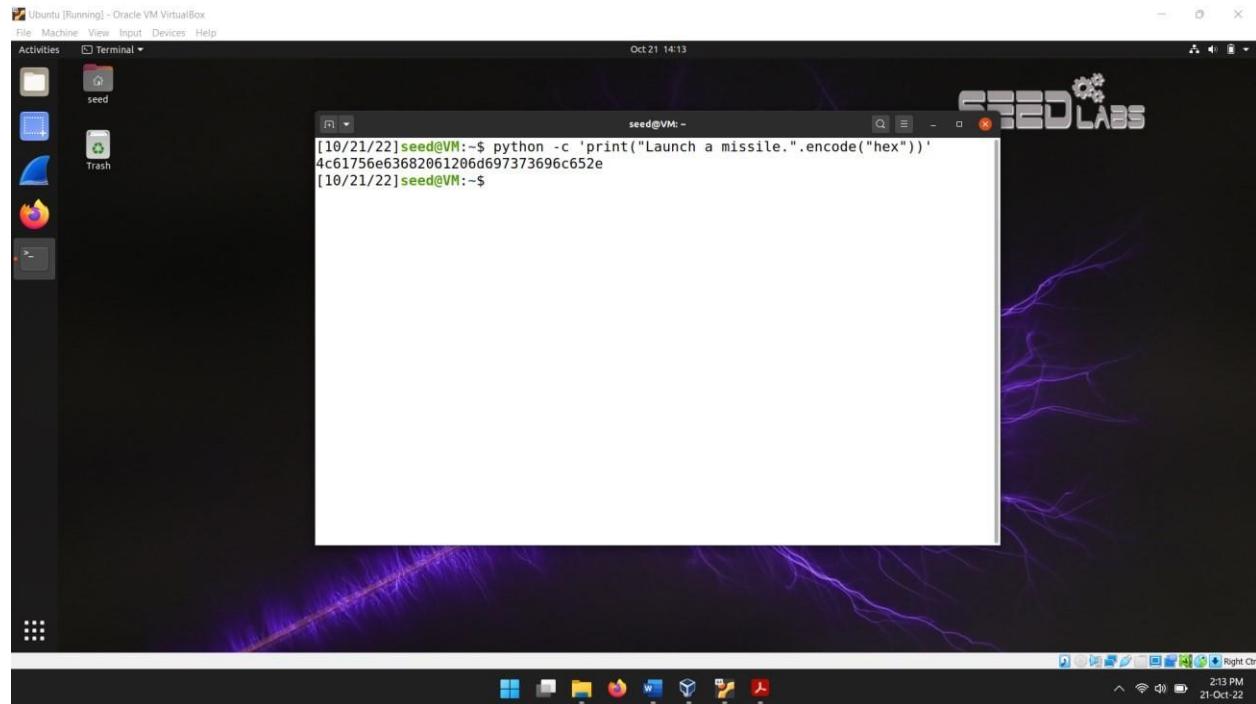
Message = M = Launch a missile.

Convert ASCII string to hexadecimal in Terminal by using command

```
python -c 'print("Launch a missile.".encode("hex"))'
```

we got following hexadecimal value:

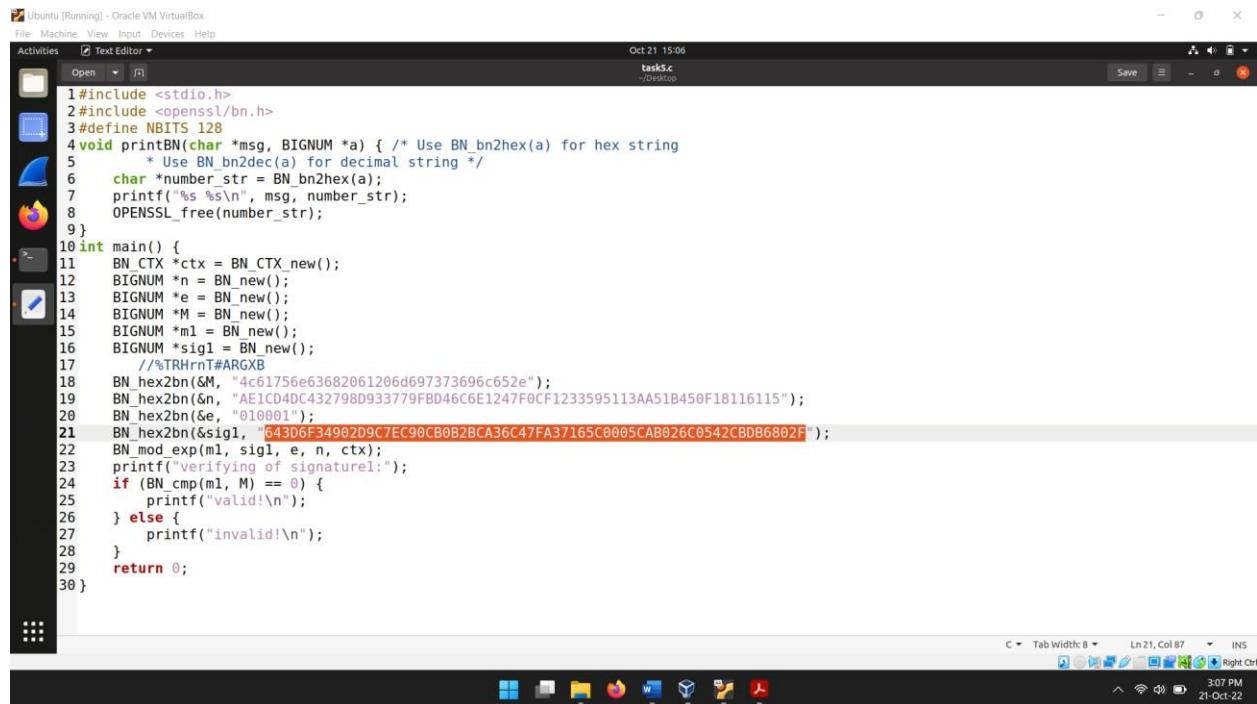
4c61756e63682061206d697373696c652e



Now we code in C language to verify the signature  $S$  that it belongs to Alice or not in text editor and saved the file with name **task5.c**, values of public key, message and signature are available to us in hexadecimal. Below is the screenshot of the code.

In below code we used signature value as,

643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F



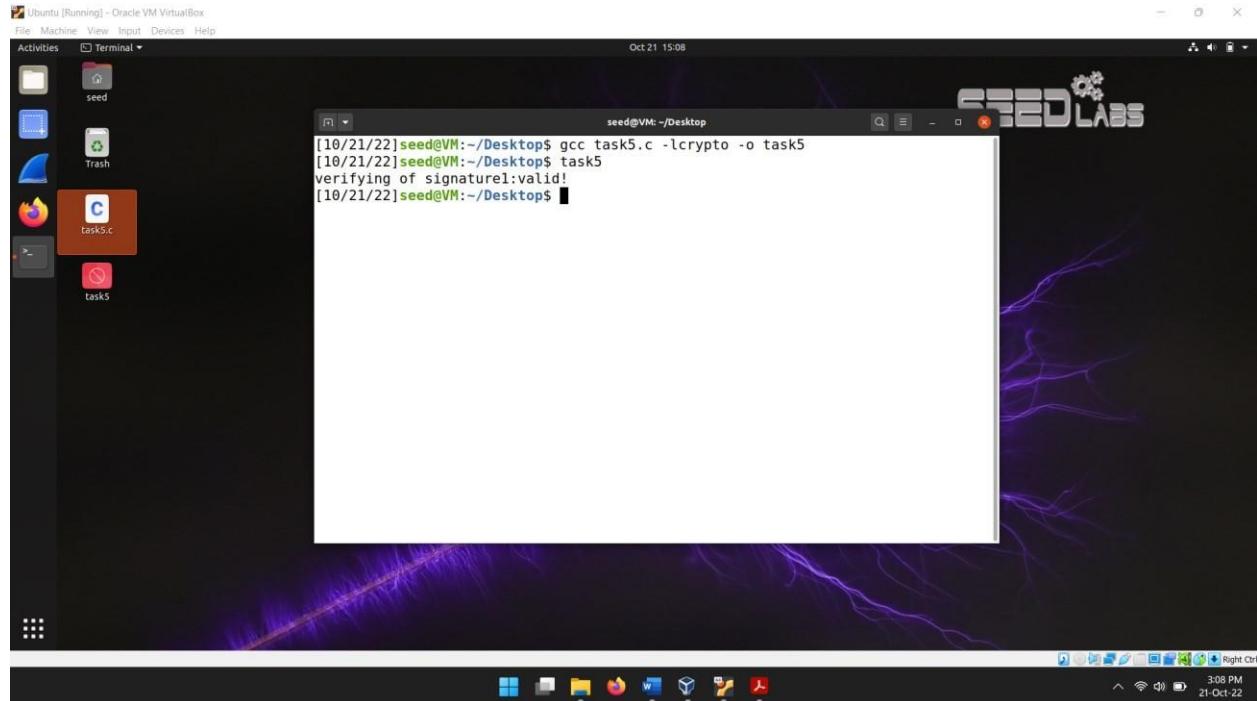
```

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor Open task5.c Oct 21 15:06
1#include <stdio.h>
2#include <openssl/bn.h>
3#define NBITS 128
4void printBN(char *msg, BIGNUM *a) { /* Use BN_bn2hex(a) for hex string
5   * Use BN_bn2dec(a) for decimal string */
6  char *number_str = BN_bn2hex(a);
7  printf("%s %s\n", msg, number_str);
8  OPENSSL_free(number_str);
9}
10int main() {
11  BN_CTX *ctx = BN_CTX_new();
12  BIGNUM *n = BN_new();
13  BIGNUM *e = BN_new();
14  BIGNUM *M = BN_new();
15  BIGNUM *m1 = BN_new();
16  BIGNUM *sig1 = BN_new();
17  //TRHnt#ARGXB
18  BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
19  BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
20  BN_hex2bn(&e, "010001");
21  BN_hex2bn(&sig1, "643D6F34902D9C7EC98CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
22  BN_mod_exp(m1, sig1, e, n, ctx);
23  printf("verifying of signature1:");
24  if (BN_cmp(m1, M) == 0) {
25    printf("valid!\n");
26  } else {
27    printf("invalid!\n");
28  }
29  return 0;
30}

```

Compile this code in terminal by using command, `gcc task5.c -lcrypto -o task5` and then run the program to check the validity of signature by command `task5`. In output we can see the result that it is valid signature because we got following output.

*verifying of signature1:valid!*



```

[10/21/22]seed@VM:~/Desktop$ gcc task5.c -lcrypto -o task5
[10/21/22]seed@VM:~/Desktop$ task5
verifying of signature1:valid!
[10/21/22]seed@VM:~/Desktop$

```

Let us assume that the last byte of signature got changed from *2F* to *3F* so the signature got corrupted by only bit. To verify the changes, we repeat all previous steps to know what we get in result.

Sig2: 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F

```

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor Oct 21 15:18
task5.c /Desktop
Save
Open R
1 #include <stdio.h>
2 #include <openssl/bn.h>
3 #define NBITS 128
4 void printBN(char *msg, BIGNUM *a) { /* Use BN_bn2hex(a) for hex string
5      * Use BN_bn2dec(a) for decimal string */
6     char *number_str = BN_bn2hex(a);
7     printf("%s %s\n", msg, number_str);
8     OPENSSL_free(number_str);
9 }
10 int main() {
11     BN_CTX *ctx = BN_CTX_new();
12     BIGNUM *n = BN_new();
13     BIGNUM *e = BN_new();
14     BIGNUM *M = BN_new();
15     BIGNUM *m2 = BN_new();
16     BIGNUM *sig2 = BN_new();
17     //TRHnT#ARGXB
18     BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
19     BN_hex2bn(&n, "AE1CD4DC43279BD933779FB046C6E1247F0CF1233595113AA51B450F18116115");
20     BN_hex2bn(&e, "010001");
21     BN_hex2bn(&sig2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
22     BN_mod_exp(m2, sig2, e, n, ctx);
23     printf("verifying of signature2:");
24     if (BN_cmp(m2, M) == 0) {
25         printf("valid!\n");
26     } else {
27         printf("invalid!\n");
28     }
29     return 0;
30 }

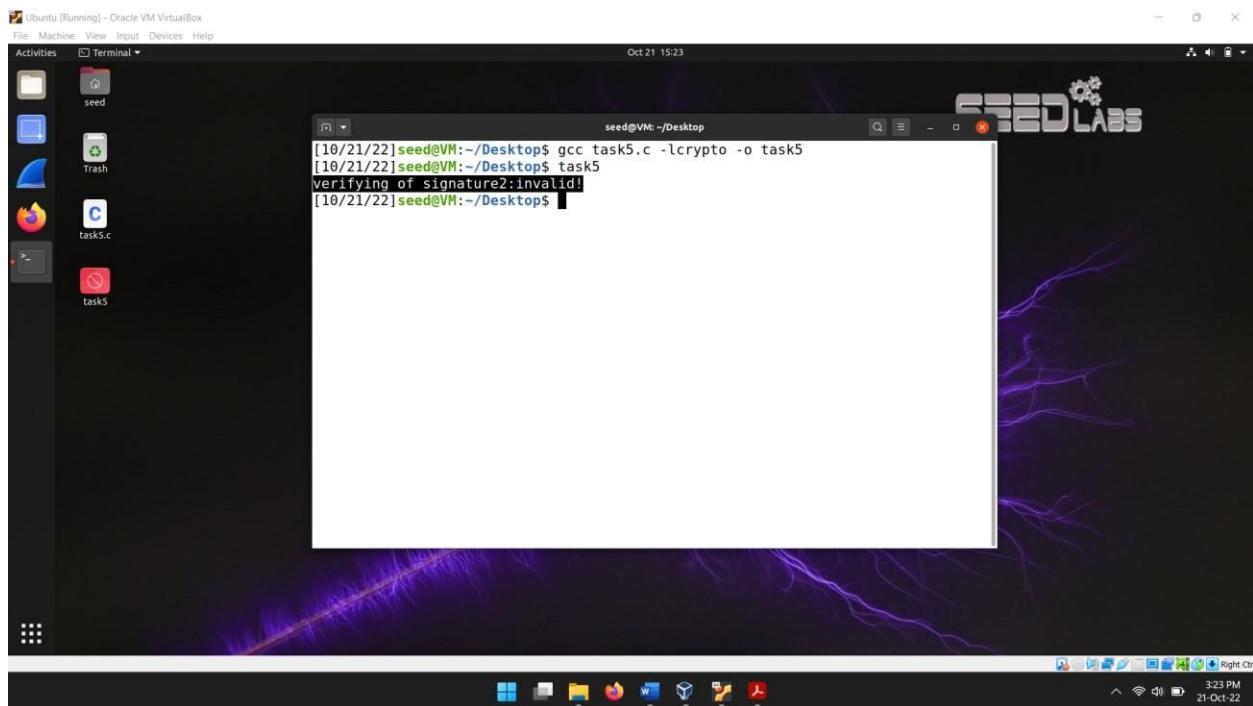
```

We recompile and run the program by commands:

**gcc task5.c -lcrypto -o task5**

**task5**

and got response as *verifying of signature2:invalid!* which can be seen below image. From this we can conclude that even if one bit is changed our whole signature will be totally changed. Due to that our signature got invalid.

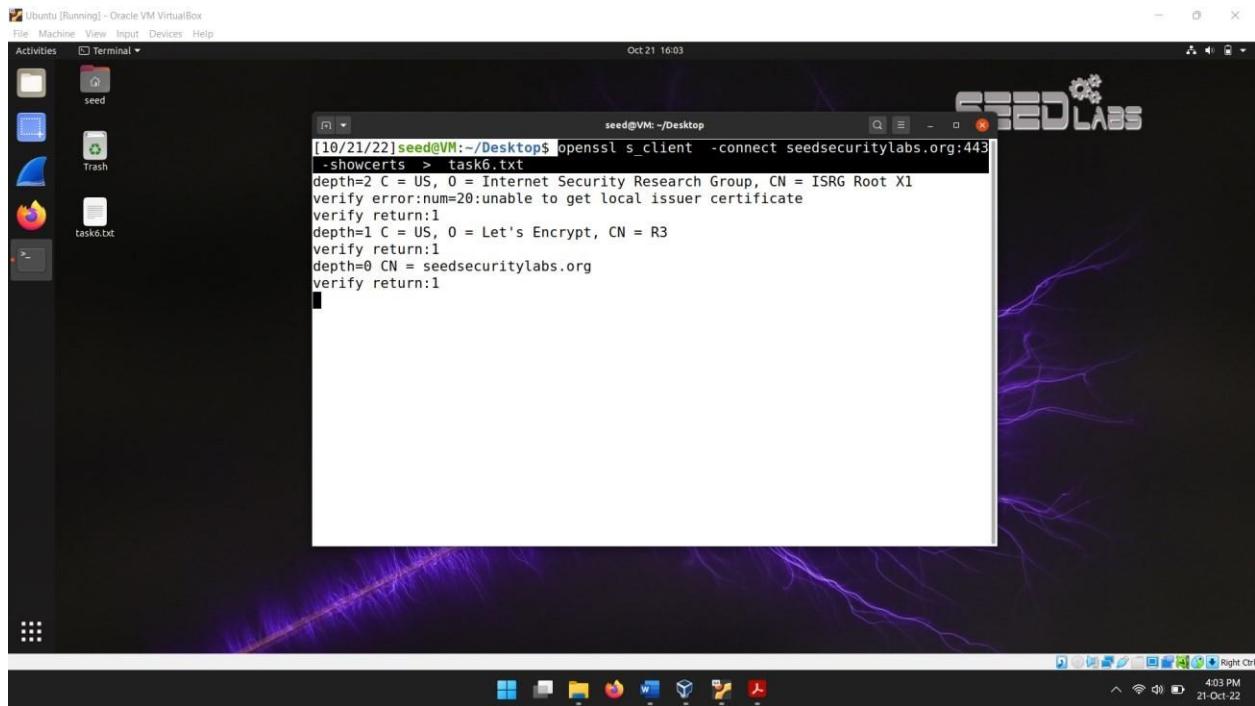


## Task 6: Manually Verifying an X.509 Certificate

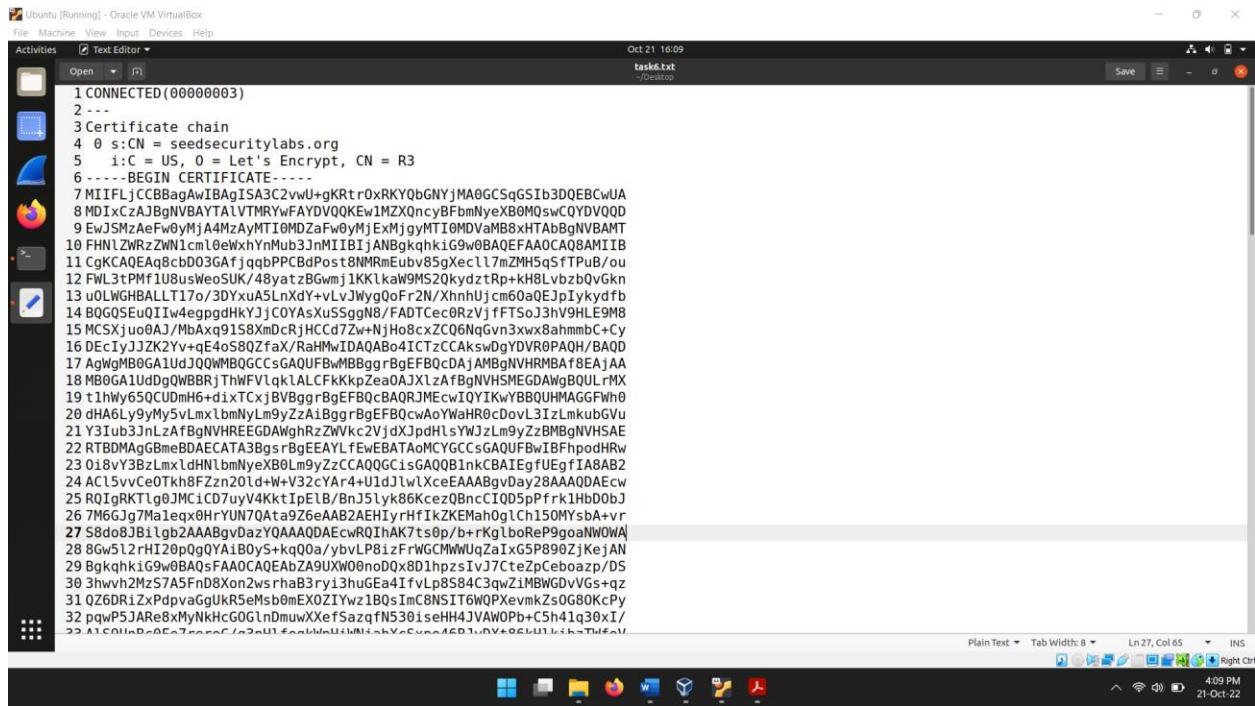
**Step 1:** Download a certificate from a real web server.

For this task we use URL <https://seedsecuritylabs.org/> as an example, we first download the certificate from web server and saved in **task6.txt** by command

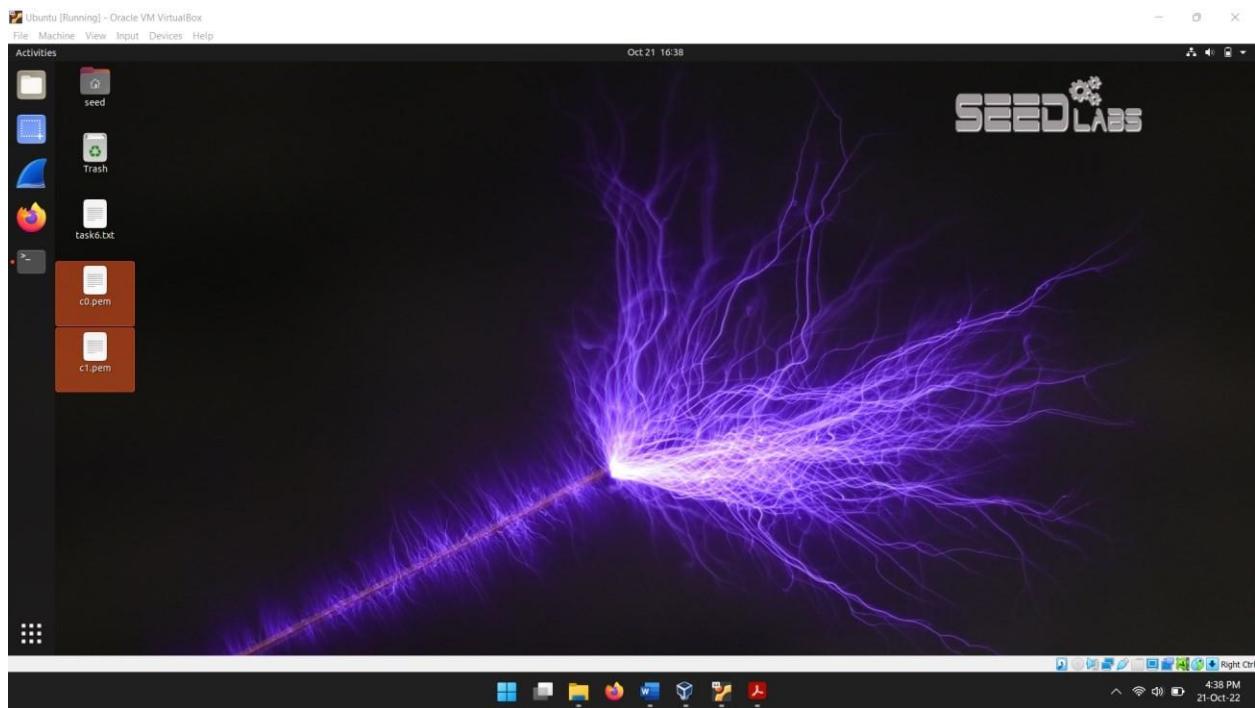
***openssl s\_client -connect seedsecuritylabs.org:443 -showcerts > task6.txt***



Certificates are founded in ***task6.txt*** file which we download and exported in ***task6.txt*** earlier



Save first certificate as c0.pem and second c1.pem



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

We extract value of  $n$  using -modulus with openssl command and that is,

```
openssl x509 -in c1.pem -noout -modulus
```

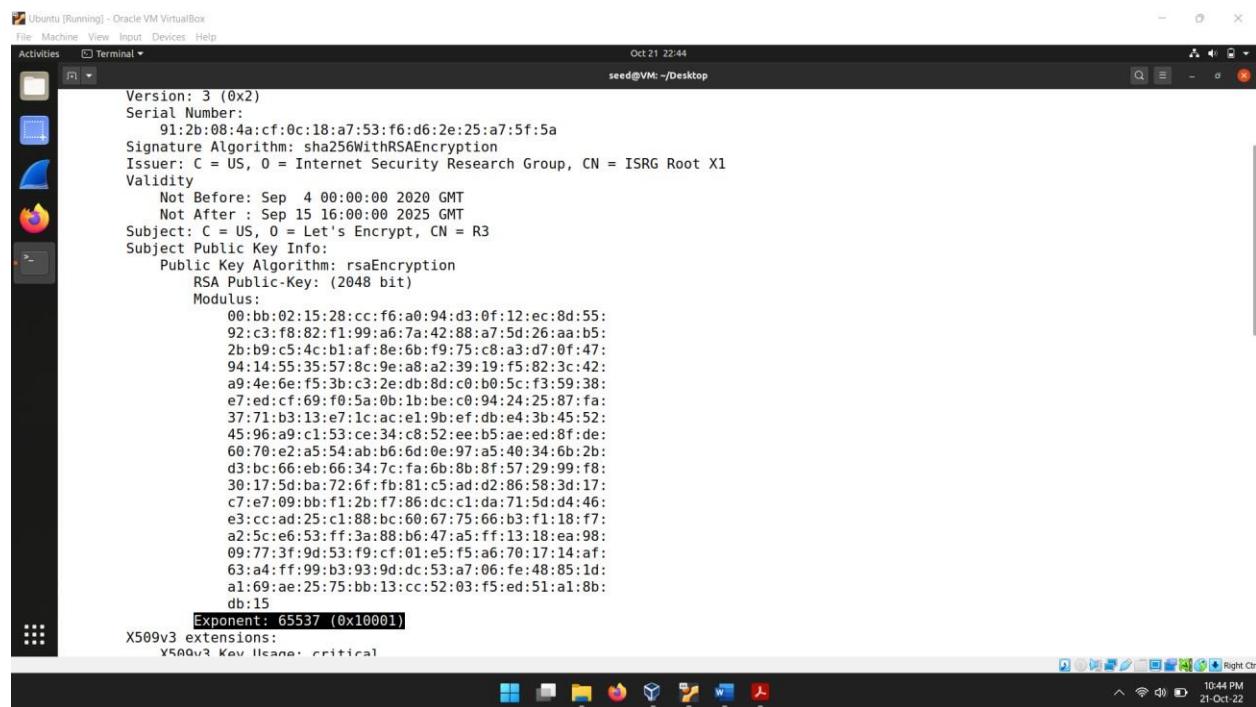
A screenshot of a Linux desktop environment, likely Ubuntu, running in Oracle VM VirtualBox. The desktop has a dark theme with a purple lightning bolt pattern as the background. At the top, there's a standard window title bar with the text "Ubuntu [Running] - Oracle VM VirtualBox". Below the title bar is a menu bar with "File", "Machine", "View", "Input", "Devices", and "Help". A docked "Activities" panel on the left lists icons for "Terminal", "seed", "Trash", "tasko.txt", "c0.pem", and "c1.pem". A "Terminal" window is open in the foreground, showing a command-line session for user "seed" at "VM". The command run is "openssl x509 -in c1.pem -noout -modulus", which outputs a long string of hex digits representing the RSA modulus. The terminal window has a dark background and light-colored text. On the right side of the screen, there's a watermark for "SEED LABS" with a gear icon. The bottom of the screen shows the system tray with various icons and the date/time "Oct 21 22:23" and "10:24 PM 21-Oct-22".

```
n =
BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1
AF8E6BF975C8A3D70F4794145535578C9EA8A23919F5823C42A94E6EF53BC32EDB8DC0
B05CF35938E7EDCF69F05A0B1BBEC094242587FA3771B313E71CACE19BEFDBE43B455
24596A9C153CE34C852EEB5AEED8FDE6070E2A554ABB66D0E97A540346B2BD3BC66E
B66347CFA6B8B8F572999F830175DBA726FFB81C5ADD286583D17C7E709BBF12BF786
DCC1DA715DD446E3CCAD25C188BC60677566B3F118F7A25CE653FF3A88B647A5FF131
8EA9809773F9D53F9CF01E5F5A6701714AF63A4FF99B3939DDC53A706FE48851DA169A
E2575BB13CC5203F5ED51A18BDB15
```

We cannot extract e but we can tell the way that can easily find  $e$ .

By using following command, we can find  $e$  (exponential)

```
openssl x509 -in c1.pem -text -noout
```



```
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Oct 21 22:44
seed@VM: ~/Desktop
Version: 3 (0x2)
Serial Number:
91:2b:08:4a:c9:0c:18:a7:53:f6:d6:2e:25:a7:5f:5a
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, O = Internet Security Research Group, CN = ISRG Root X1
Validity
Not Before: Sep 4 00:00:00 2020 GMT
Not After : Sep 15 16:00:00 2025 GMT
Subject: C = US, O = Let's Encrypt, CN = R3
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public-Key: (2048 bit)
        Modulus:
          00:bb:02:15:28:cc:f6:a0:94:d3:0f:12:ec:8d:55:
          92:c3:f8:82:f1:99:a6:7a:42:88:a7:5d:26:aa:b5:
          2b:b9:c5:4c:bl:af:8e:6b:f9:75:c8:a3:d7:0f:47:
          94:14:55:35:57:8c:9e:a8:a2:39:19:f5:82:3c:42:
          a9:4e:6e:f5:3b:c3:2e:db:8d:c0:b0:5c:f3:59:38:
          e7:ed:c1:69:f0:5a:0b:1b:be:c0:94:24:25:87:fa:
          37:71:b3:13:e7:1c:ac:1:9b:ef:db:e4:3b:45:52:
          45:96:a9:c1:53:ce:34:c8:52:ee:b5:ae:ed:8f:de:
          60:70:e2:a5:54:ab:b6:6d:0e:97:a5:40:34:6b:2b:
          d3:bc:66:eb:66:34:7c:fa:6b:8b:8f:57:29:99:f8:
          30:17:5d:ba:72:6f:fb:81:5:ad:d2:66:58:3d:17:
          c7:e7:09:bb:f1:2b:f7:86:dc:c1:da:71:5d:d4:46:
          e3:cc:ad:25:c1:88:bc:60:67:75:66:b3:f1:18:f7:
          a2:5c:e6:53:ff:3a:88:b6:47:a5:ff:13:18:ea:98:
          09:77:3f:9d:53:f9:cf:01:e5:f5:a6:70:17:14:af:
          63:a4:ff:99:b3:93:9d:dc:53:a7:06:fe:48:85:1d:
          a1:69:ae:25:75:bb:13:cc:52:03:f5:ed:51:a1:8b:
          db:15
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Key Usage: critical
```

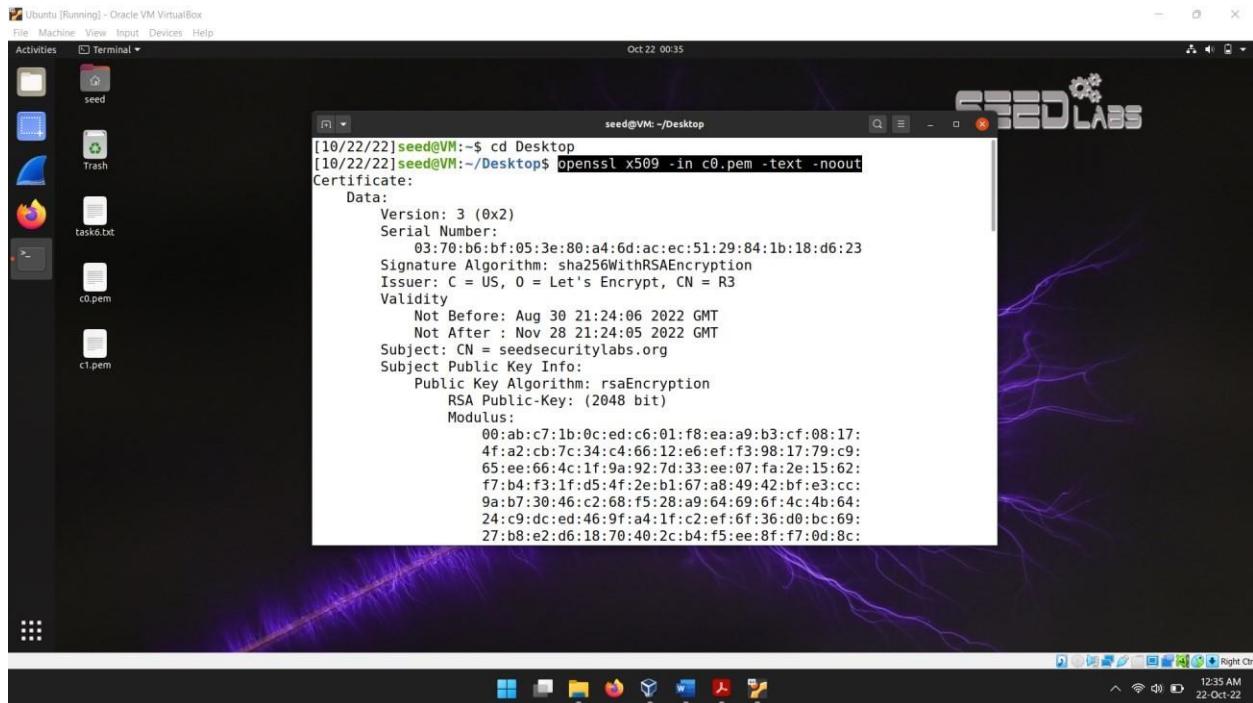
We printed all values of certificate first and then get our exponent( $e$ ) which is equal to **65537 (0x10001)**

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

To extract signature from certificate we need to print out all fields first then we copy the signature block and paste it in our file because there is no specific command to extract signature.

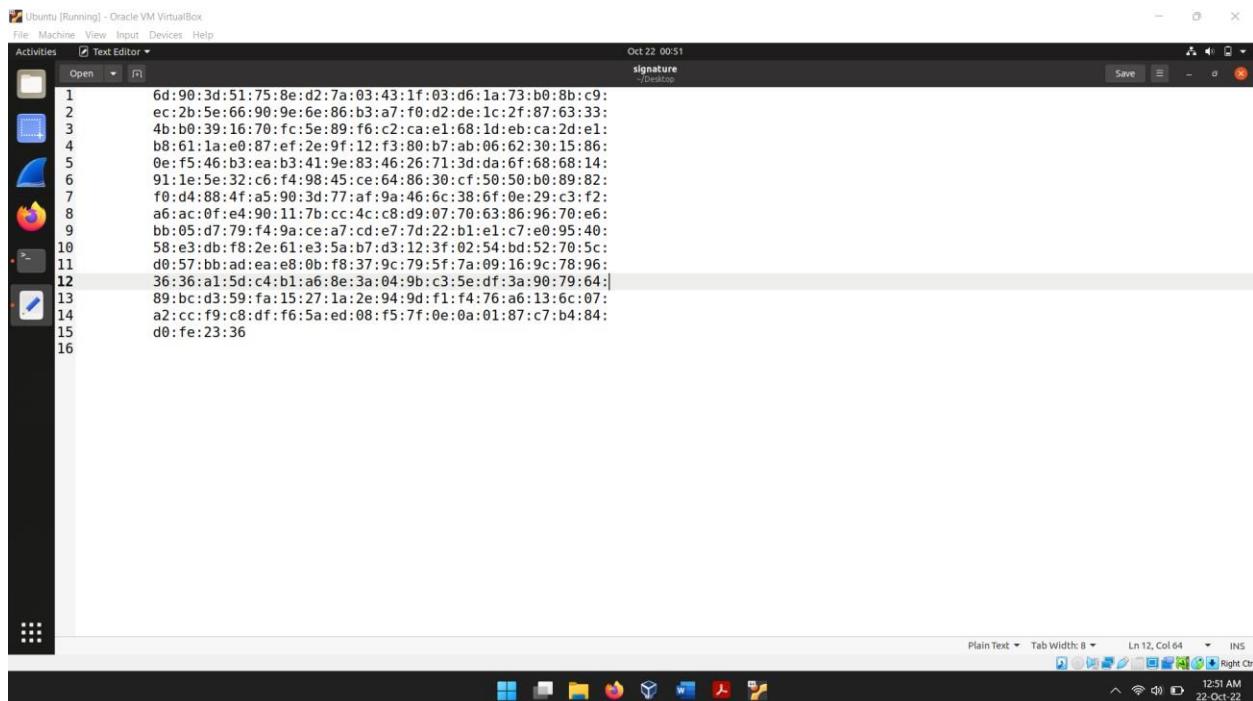
We used command in terminal to print all attributes are

**openssl x509 -in c0.pem -text -noout**



We got our data but there is one issue that it has colons and spaces to remove them there is a command in linux utility for string manipulation, *tr* and “space” and “:” can be remove by *-d* option.

For this we first save our signature body in a file called signature.



Run following commands in terminal

*vim signature*

*cat signature | tr -d '[:space:]:'*

```
[10/22/22]seed@VM:~/Desktop$ vim signature
[10/22/22]seed@VM:~/Desktop$ cat signature | tr -d '[:space:]:'
6d903d51758ed27a03431f03d61a73b08c9ec2b5e66909e6e86b3a7f0d2de1c2f8763334bb03916
70fc5e89f5c2cae1681debc2de1b8611ae087ef2e9f12f380b7ab06623015860ef546b3eab3419e
834626713ddaa6fb886814911e5e32cbf49845ce648630cf5050b08982fd4884fa5983d77af9a466c
386f0e29c3f2a6ac0fe490117bcc4cc8d9077063869670e6bb05d779f49acea7cde77d22b1e1c7e0
954058e3dbf82e61e35ab7d3123f0254bd52705cd057bbadeae88bf8379c795f7a09169c78963636
a15dc4b1a68e3a049bc35edf3a90796489bcd359fa15271a2e949df1f476a6136c07a2ccf9c8dff6
5aed08f57f0e0a0187c7b484d0fe2336[10/22/22]seed@VM:~/Desktop$
```

Here we can see we get our signature without colon and space.

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

First of all, we parse the ASN.1 structure by using *openssl* command *asn1parse* to get data from ASN.1

`openssl asn1parse -i -in c0.pem`

```
[10/22/22]seed@VM:~/Desktop$ openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1326 cons: SEQUENCE
4:d=1 hl=4 l=1046 cons: SEQUENCE
8:d=2 hl=2 l= 3 cons: cont [ 0 ]
10:d=3 hl=2 l= 1 prim: INTEGER :02
13:d=2 hl=2 l= 18 prim: INTEGER :0370B6BF053E80A46DACEC5129841B
18D623
33:d=2 hl=2 l= 13 cons: SEQUENCE
35:d=3 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
46:d=3 hl=2 l= 0 prim: NULL
48:d=2 hl=2 l= 50 cons: SEQUENCE
50:d=3 hl=2 l= 11 cons: SET
52:d=4 hl=2 l= 9 cons: SEQUENCE
54:d=5 hl=2 l= 3 prim: OBJECT :countryName
59:d=5 hl=2 l= 2 prim: PRINTABLESTRING :US
63:d=3 hl=2 l= 22 cons: SET
65:d=4 hl=2 l= 20 cons: SEQUENCE
67:d=5 hl=2 l= 3 prim: OBJECT :organizationName
```

The first highlighted field is the body of the certificate used to generate the hash,

```
[10/22/22]seed@VM:~/Desktop$ openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1326 cons: SEQUENCE
4:d=1 hl=4 l=1046 cons: SEQUENCE
8:d=2 hl=2 l= 3 cons: cont [ 0 ]
10:d=3 hl=2 l= 1 prim: INTEGER :02
13:d=2 hl=2 l= 18 prim: INTEGER :0370B6BF053E80A46DACEC5129841B
18D623
```

The second highlighted image is the signature block.

```
1054:d=1 hl=2 l= 13 cons: SEQUENCE
1056:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
1067:d=2 hl=2 l= 0 prim: NULL
1069:d=1 hl=4 l= 257 prim: BIT STRING
[10/22/22]seed@VM:~/Desktop$
```

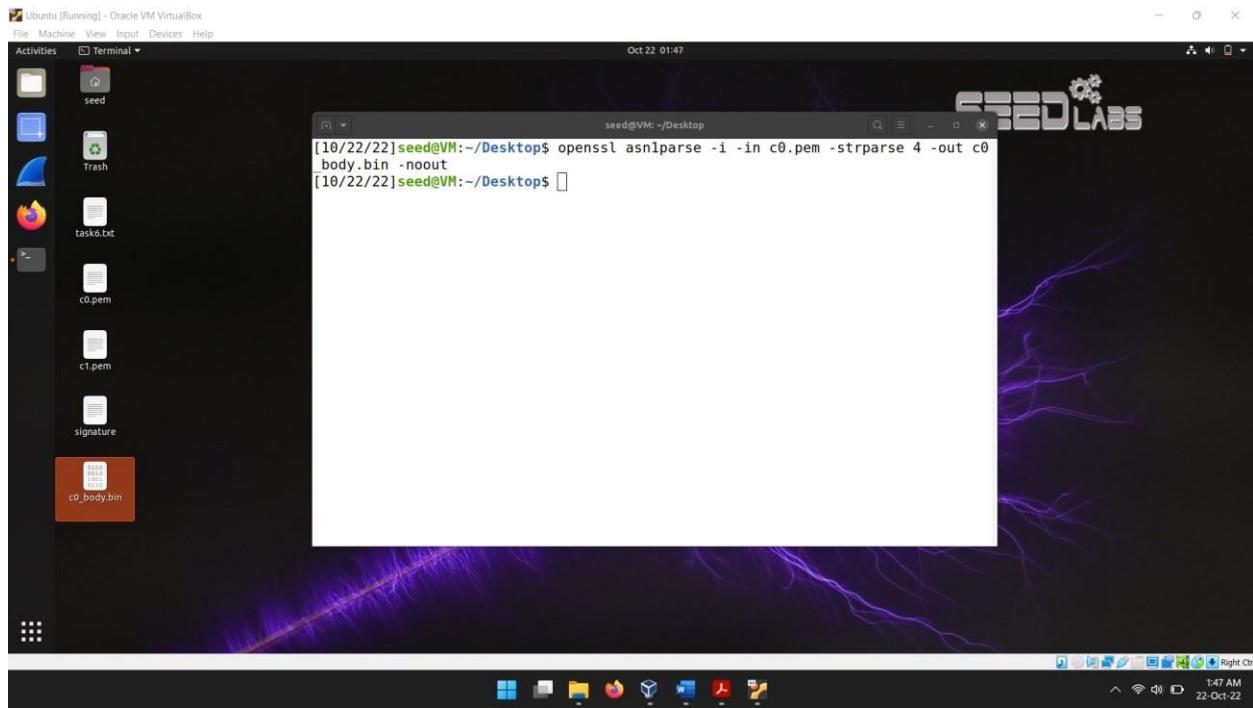
Their certificate offsets are 4 to 1053, while their signature offset is from 1054 to end of file.

In X.509 certificates starting will be same for all and that is 4 but ending will be depend on content length.

We can get body of certificate without signature block with the help of `-strparse` command,

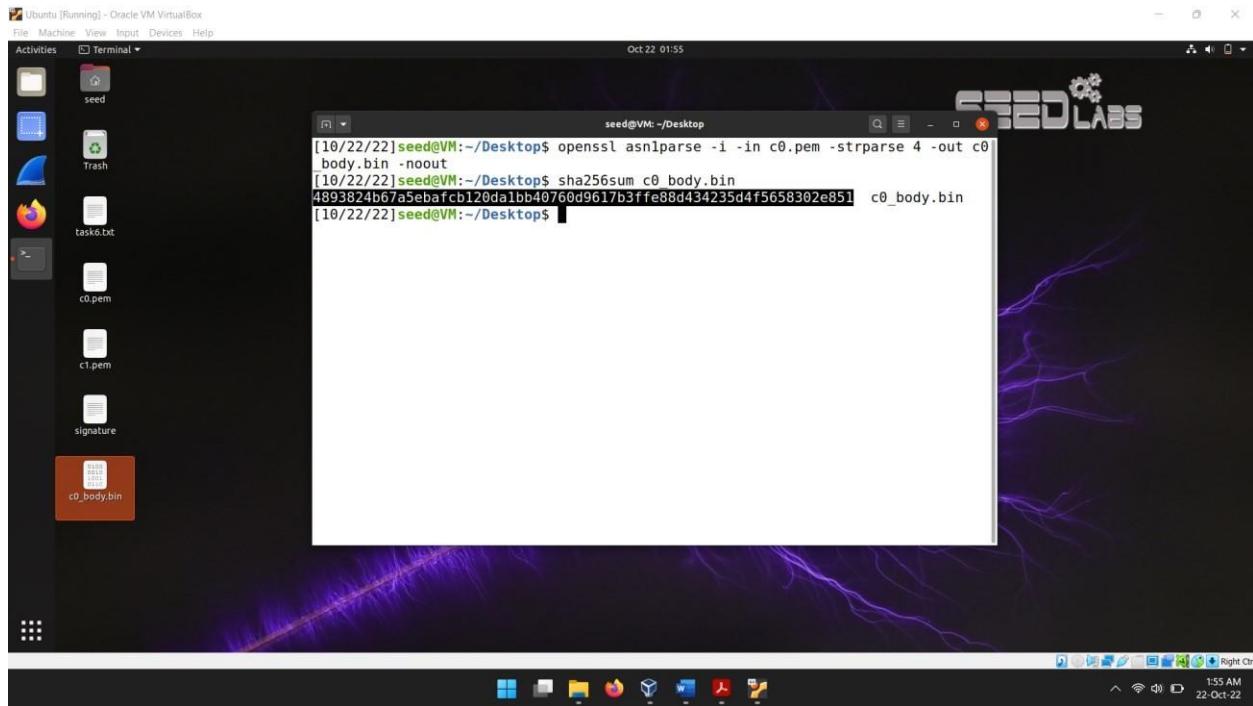
There is a file created which can be seen in below image with command:

`openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout`



Now to calculate its hash value we run command:

*sha256sum c0\_body.bin*



Its hash value is 4893824b67a5ebafcb120da1bb40760d9617b3ffe88d434235d4f5658302e851

## Step 5: Verify the signature.

To verify that our signature is valid or not we run our own program in C language.

For that we first extract our Message  $M$ ,

we have following values:

$n =$

BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1  
 AF8E6BF975C8A3D70F4794145535578C9EA8A23919F5823C42A94E6EF53BC32EDB8DC0  
 B05CF35938E7EDCF69F05A0B1BBEC094242587FA3771B313E71CACE19BEFDBE43B455  
 24596A9C153CE34C852EEB5AEED8FDE6070E2A554ABB66D0E97A540346B2BD3BC66E  
 B66347CFA6B8B8F572999F830175DBA726FFB81C5ADD286583D17C7E709BBF12BF786  
 DCC1DA715DD446E3CCAD25C188BC60677566B3F118F7A25CE653FF3A88B647A5FF131  
 8EA9809773F9D53F9CF01E5F5A6701714AF63A4FF99B3939DDC53A706FE48851DA169A  
 E2575BB13CC5203F5ED51A18BDB15

$e = 010001$

$s =$

6d903d51758ed27a03431f03d61a73b08bc9ec2b5e66909e6e86b3a7f0d2de1c2f8763334bb03916  
 70fc5e89f6c2cae1681debca2de1b8611ae087ef2e9f12f380b7ab06623015860ef546b3eab3419e83  
 4626713ddaf686814911e5e32c6f49845ce648630cf5050b08982f0d4884fa5903d77af9a466c386  
 f0e29c3f2a6ac0fe490117bcc4cc8d9077063869670e6bb05d779f49acea7cde77d22b1e1c7e09540  
 58e3dbf82e61e35ab7d3123f0254bd52705cd057bbadeae80bf8379c795f7a09169c78963636a15d  
 c4b1a68e3a049bc35edf3a90796489bcd359fa15271a2e949df1f476a6136c07a2ccf9c8dff65aed08  
 f57f0e0a0187c7b484d0fe2336

The screenshot shows a Linux desktop environment with several windows open. In the foreground, a terminal window displays a C program for RSA encryption. The code includes imports for stdio.h and openssl/bn.h, defines NBITS as 128, and implements a printBN function and a main function that generates large random numbers, performs modular exponentiation, and prints the result. A file browser window is visible in the background, showing a directory structure with files like 'task6.c' and 'task6.out'. The desktop bar at the bottom shows icons for various applications like the file manager, terminal, and browser.

```
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor
Open Save
Oct 22 02:43
task6c
-/Desktop
1#include <stdio.h>
2#include <openssl/bn.h>
3#define NBITS 128
4void printBN(char *msg, BIGNUM *a)
5{ /* Use BN_bn2hex(a) for hex string
6   * Use BN_bn2dec(a) for decimal string */
7    char *number_str = BN_bn2hex(a);
8    printf("%s %s\n", msg, number_str);
9    OPENSSL_free(number_str);
10}
11int main()
12{
13    BN_CTX *ctx = BN_CTX_new();
14
15    BIGNUM *n = BN_new();
16    BIGNUM *e = BN_new();
17    BIGNUM *sig = BN_new();
18    BIGNUM *m = BN_new();
19
20    BN_hex2bn(&n,
21    "BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F4794145535578C9EA8A23919F5823C42A94E6EF53BC32EDB8DC0E
22    BN_hex2bn(&e, "010001");
23    BN_hex2bn(&sig,
24    "6d903d51758ed27a03431f03d61a73b08bc9ec2b5e66909e6e86b3a7f0d2de1c2f8763334bb0391670fc5e89f6c2cae1681debca2de1b8611ae087ef2e9f12f380b7ab06623
25
26    BN_mod_exp(m, sig, e, n, ctx);
27
28    printBN("message:", m);
29
30    return 0;
31}
```

We save the code in text editor by naming it *task6.c*, compile and run the code in terminal by using following commands:

```
gcc task6.c -lcrypto -o task6
```

## task6

We got our Message M:

After that we edit our C code which we wrote earlier, our code looks like this now.

Now we compile and run our program using command:

```
gcc task6.c -lcrypto -o task6
```

## task6

In response we got output as

*verifying of signature:valid!*

The image below is confirming that our signature is valid.

