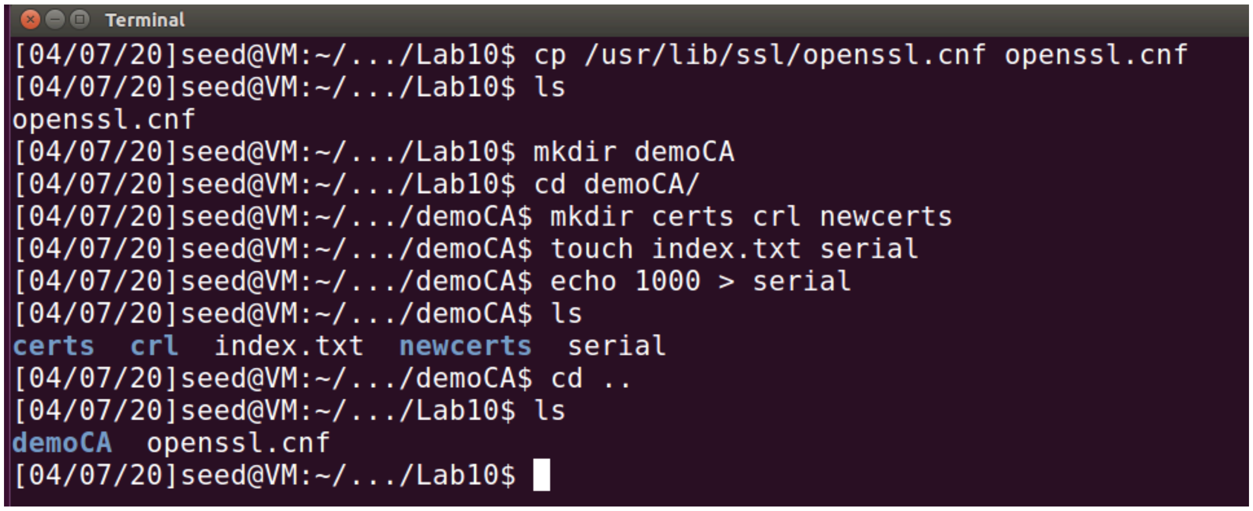
Task 1: Becoming a Certificate Authority (CA)

Step 1: The Configuration File openssl.conf

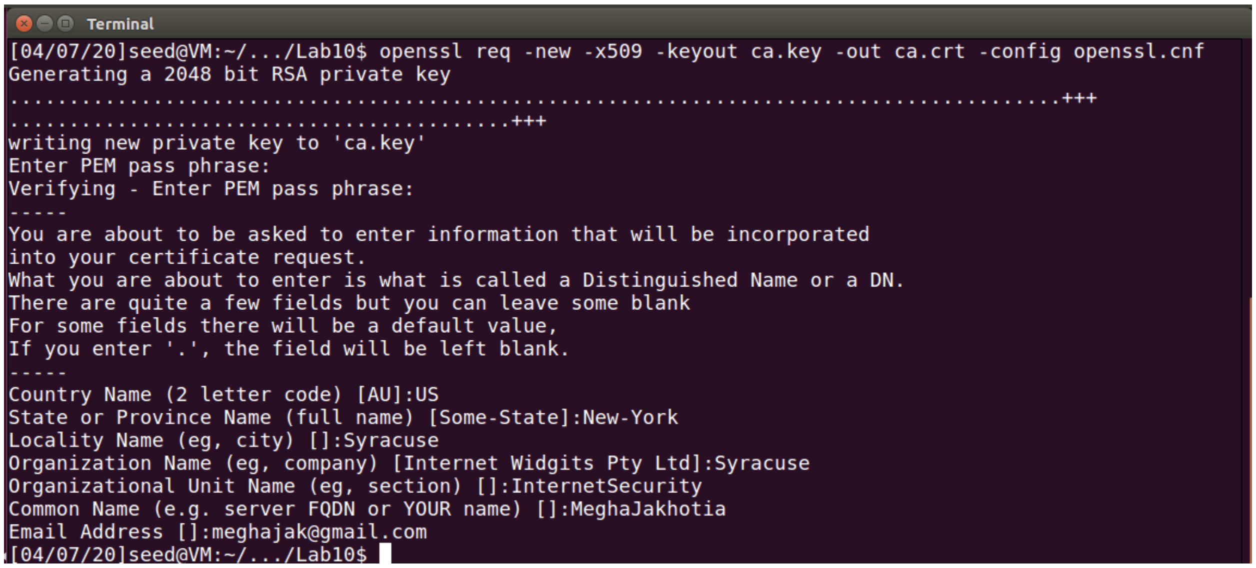
We use OpenSSL to create certificates and the following provides the configuration setup for creating and issuing certificates:



```
[04/07/20]seed@VM:~/.../Lab10$ cp /usr/lib/ssl/openssl.cnf openssl.cnf
[04/07/20]seed@VM:~/.../Lab10$ ls
openssl.cnf
[04/07/20]seed@VM:~/.../Lab10$ mkdir demoCA
[04/07/20]seed@VM:~/.../Lab10$ cd demoCA/
[04/07/20]seed@VM:~/.../demoCA$ mkdir certs crl newcerts
[04/07/20]seed@VM:~/.../demoCA$ touch index.txt serial
[04/07/20]seed@VM:~/.../demoCA$ echo 1000 > serial
[04/07/20]seed@VM:~/.../demoCA$ ls
certs  crl  index.txt  newcerts  serial
[04/07/20]seed@VM:~/.../demoCA$ cd ..
[04/07/20]seed@VM:~/.../Lab10$ ls
demoCA  openssl.cnf
[04/07/20]seed@VM:~/.../Lab10$ 
```

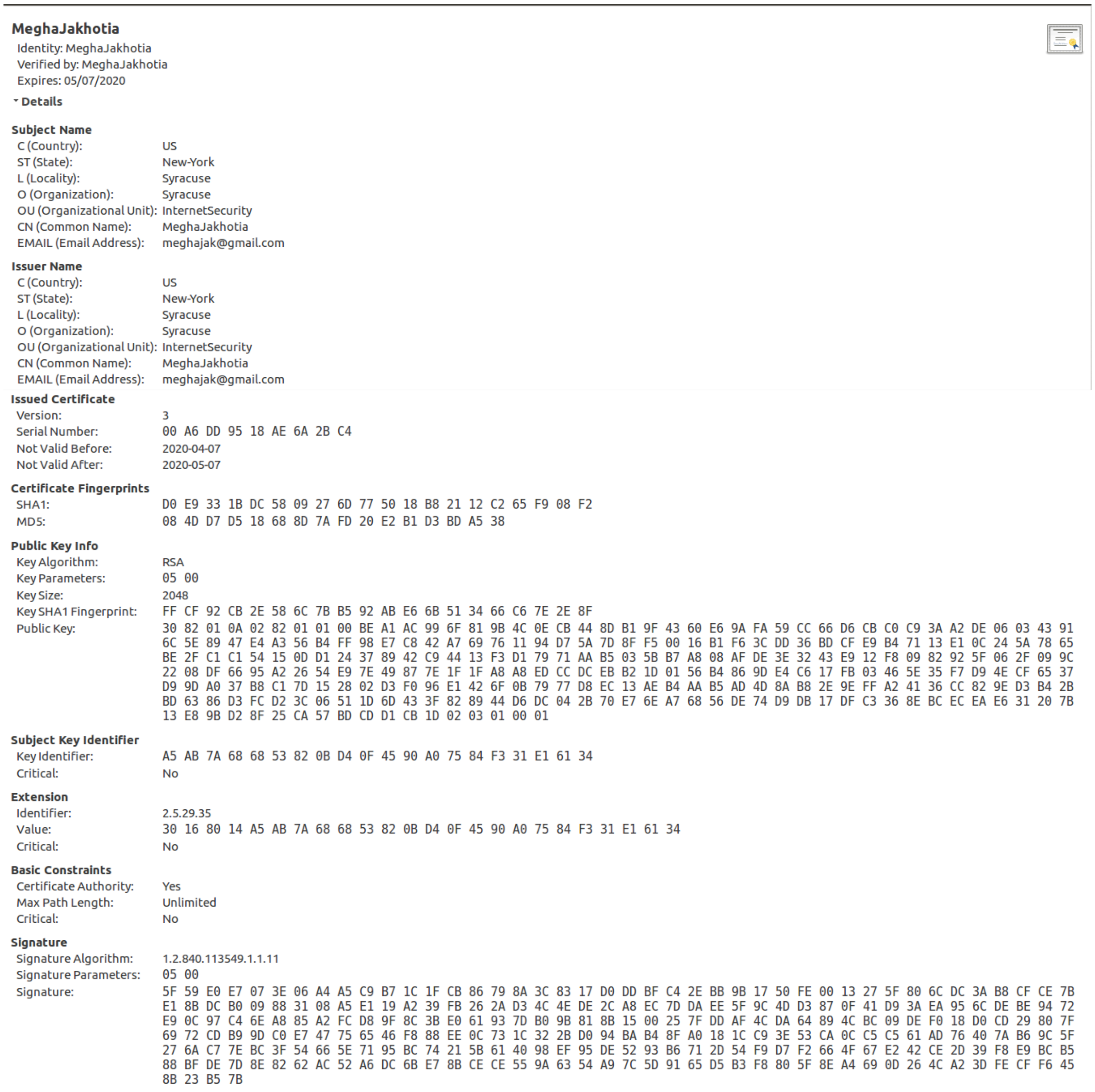
Step 2: Certificate Authority (CA)

Now, we generate a self-signed certificate for our CA that will serve as the root certificate as follows:



```
[04/07/20]seed@VM:~/.../Lab10$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Generating a 2048 bit RSA private key
..............................................................................+++
.........................................+++
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]:US
State or Province Name (full name) [Some-State]:New-York
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Syracuse
Organizational Unit Name (eg, section) []:InternetSecurity
Common Name (e.g. server FQDN or YOUR name) []:MeghaJakhotia
Email Address []:meghajak@gmail.com
[04/07/20]seed@VM:~/.../Lab10$ 
```

The above provides a screenshot of the entered information. The output of the command is stored in two files: ca.key and ca.crt (as provided in the command). The file ca.key contains the CA's private key, while ca.crt contains the public-key certificate. The following provides the content of ca.crt:



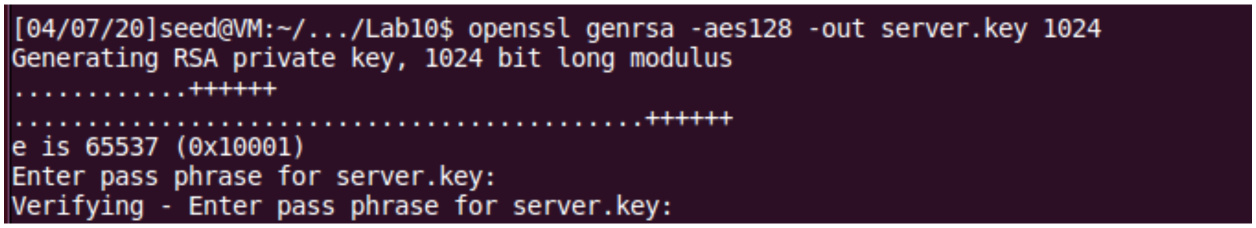
```
MeghaJakhotia
 Identity: MeghaJakhotia
 Verified by: MeghaJakhotia
 Expires: 05/07/2020
 ▾ Details

Subject Name
 C (Country):                 US
 ST (State):                  New-York
 L (Locality):                Syracuse
 O (Organization):            Syracuse
 OU (Organizational Unit):    InternetSecurity
 CN (Common Name):            MeghaJakhotia
 EMAIL (Email Address):       meghajak@gmail.com

Issuer Name
 C (Country):                 US
 ST (State):                  New-York
 L (Locality):                Syracuse
 O (Organization):            Syracuse
 OU (Organizational Unit):    InternetSecurity
 CN (Common Name):            MeghaJakhotia
 EMAIL (Email Address):       meghajak@gmail.com

Issued Certificate
 Version:                     3
 Serial Number:               00 A6 DD 95 18 AE 6A 2B C4
 Not Valid Before:            2020-04-07
 Not Valid After:             2020-05-07

Certificate Fingerprints
 SHA1:                        D0 E9 33 1B DC 58 09 27 6D 77 50 18 B8 21 12 C2 65 F9 08 F2
 MD5:                         08 4D D7 D5 18 68 8D 7A FD 20 E2 B1 D3 BD A5 38

Public Key Info
 Key Algorithm:               RSA
 Key Parameters:              05 00
 Key Size:                    2048
 Key SHA1 Fingerprint:        FF CF 92 CB 2E 58 6C 7B B5 92 AB E6 6B 51 34 66 C6 7E 2E 8F
 Public Key:                  30 82 01 0A 02 82 01 01 00 BE A1 AC 99 6F 81 9B 4C 0E CB 44 8D B1 9F 43 60 E6 9A FA 59 CC 66 D6 CB C0 C9 3A A2 DE 06 03 43 91
                              6C 5E 89 47 E4 A3 56 B4 FF 98 E7 C8 42 A7 69 76 11 94 D7 5A 7D 8F F5 00 16 B1 F6 3C DD 36 BD CF E9 B4 71 13 E1 0C 24 5A 78 65
                              BE 2F C1 C1 54 15 0D D1 24 37 89 42 C9 44 13 F3 D1 79 71 AA B5 03 5B B7 A8 08 AF DE 3E 32 43 E9 12 F8 09 82 92 5F 06 2F 09 9C
                              22 08 DF 66 95 A2 26 54 E9 7E 49 87 7E 1F 1F A8 A8 ED CC DC EB B2 1D 01 56 B4 86 9D E4 C6 17 FB 03 46 5E 35 F7 D9 4E CF 65 37
                              D9 9D A0 37 B8 C1 7D 15 28 02 D3 F0 96 E1 42 6F 0B 79 77 D8 EC 13 AE B4 AA B5 AD 4D 8A B8 2E 9E FF A2 41 36 CC 82 9E D3 B4 2B
                              BD 63 86 D3 FC D2 3C 06 51 1D 6D 43 3F 82 89 44 D6 DC 04 2B 70 E7 6E A7 68 56 DE 74 D9 DB 17 DF C3 36 8E BC EC EA E6 31 20 7B
                              13 E8 9B D2 8F 25 CA 57 BD CD D1 CB 1D 02 03 01 00 01

Subject Key Identifier
 Key Identifier:              A5 AB 7A 68 68 53 82 0B D4 0F 45 90 A0 75 84 F3 31 E1 61 34
 Critical:                    No

Extension
 Identifier:                  2.5.29.35
 Value:                       30 16 80 14 A5 AB 7A 68 68 53 82 0B D4 0F 45 90 A0 75 84 F3 31 E1 61 34
 Critical:                    No

Basic Constraints
 Certificate Authority:       Yes
 Max Path Length:             Unlimited
 Critical:                    No

Signature
 Signature Algorithm:         1.2.840.113549.1.1.11
 Signature Parameters:        05 00
 Signature:                   5F 59 E0 E7 07 3E 06 A4 A5 C9 B7 1C 1F CB 86 79 8A 3C 83 17 D0 DD BF C4 2E BB 9B 17 50 FE 00 13 27 5F 80 6C DC 3A B8 CF CE 7B
                              E1 8B DC B0 09 88 31 08 A5 E1 19 A2 39 FB 26 2A D3 4C 4E DE 2C A8 EC 7D DA EE 5F 9C 4D D3 87 0F 41 D9 3A EA 95 6C DE BE 94 72
                              E9 0C 97 C4 6E A8 85 A2 FC D8 9F 8C 3B E0 61 93 7D B0 9B 81 8B 15 00 25 7F DD AF 4C DA 64 89 4C BC 09 DE F0 18 D0 CD 29 80 7F
                              69 72 CD B9 9D C0 E7 47 75 65 46 F8 88 EE 0C 73 1C 32 2B D0 94 BA B4 8F A0 18 1C C9 3E 53 CA 0C C5 C5 61 AD 76 40 7A B6 9C 5F
                              27 6A C7 7E BC 3F 54 66 5E 71 95 BC 74 21 5B 61 40 98 EF 95 DE 52 93 B6 71 2D 54 F9 D7 F2 66 4F 67 E2 42 CE 2D 39 F8 E9 BC B5
                              88 BF DE 7D 8E 82 62 AC 52 A6 DC 6B E7 8B CE CE 55 9A 63 54 A9 7C 5D 91 65 D5 B3 F8 80 5F 8E A4 69 0D 26 4C A2 3D FE CF F6 45
                              8B 23 B5 7B
```

We see that the subject and issuer are the same, indicating a self-signed certificate. Also, the Certificate Authority is set to Yes in Basic Constraints, indicating that this certificate can be used to issue and sign another certificate, hence becoming a certificate of a Certificate Authority (CA). This certificate will be used as the root certificate in this lab.

Task 2: Creating a Certificate for meghajak.com

Step 1: Generate public/private key pair

We first create an RSA public-private key pair using the following command:

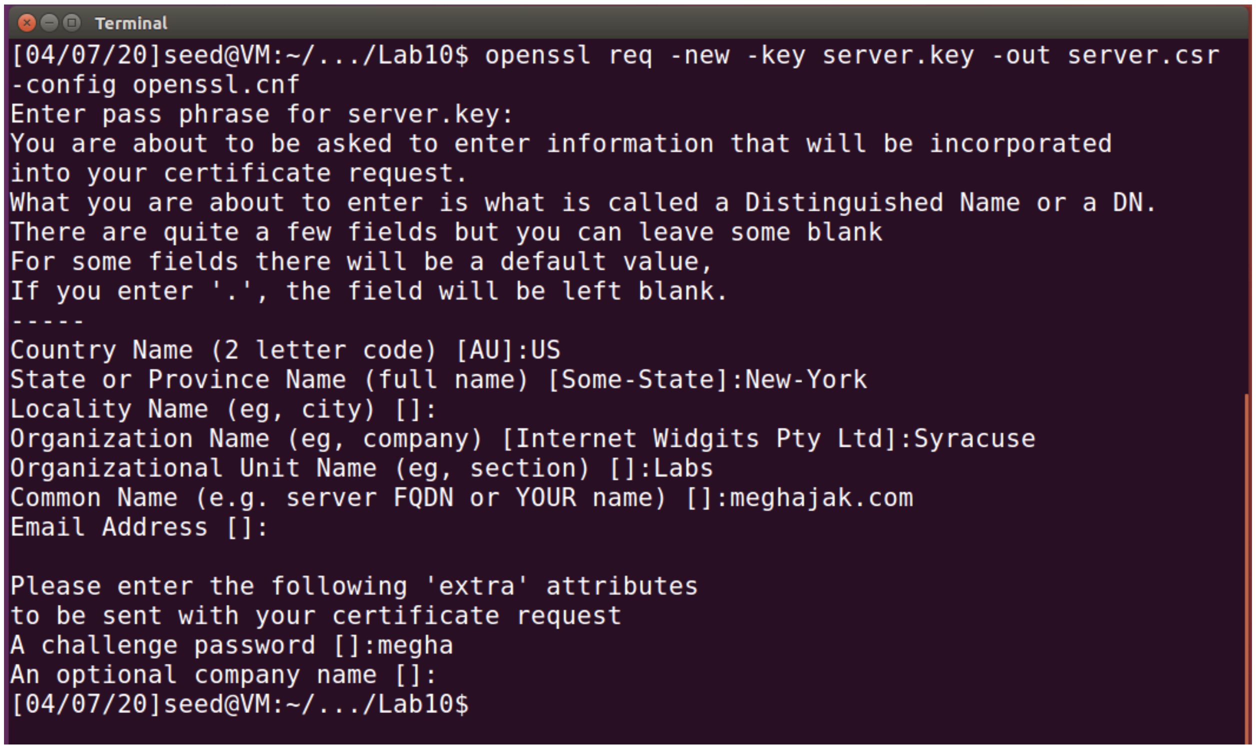


```
[04/07/20]seed@VM:~/.../Lab10$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
............++++++
...........................................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

We can use the following command to view the generated key: `openssl rsa -in server.key -text`

Step 2: Generate a Certificate Signing Request (CSR)

Next, we create a Certificate Signing Request (CSR) that includes the company's public key. The CSR has the following details with company's common name being meghajak.com (company's domain):

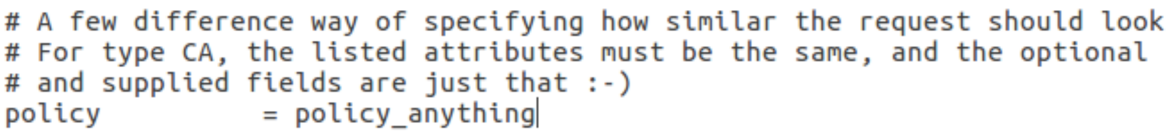


```
[04/07/20]seed@VM:~/.../Lab10$ openssl req -new -key server.key -out server.csr
-config openssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished 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]:US
State or Province Name (full name) [Some-State]:New-York
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Syracuse
Organizational Unit Name (eg, section) []:Labs
Common Name (e.g. server FQDN or YOUR name) []:meghajak.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:megha
An optional company name []:
[04/07/20]seed@VM:~/.../Lab10$
```

The above CSR is then sent to the CA to generate a certificate for the key and company name.

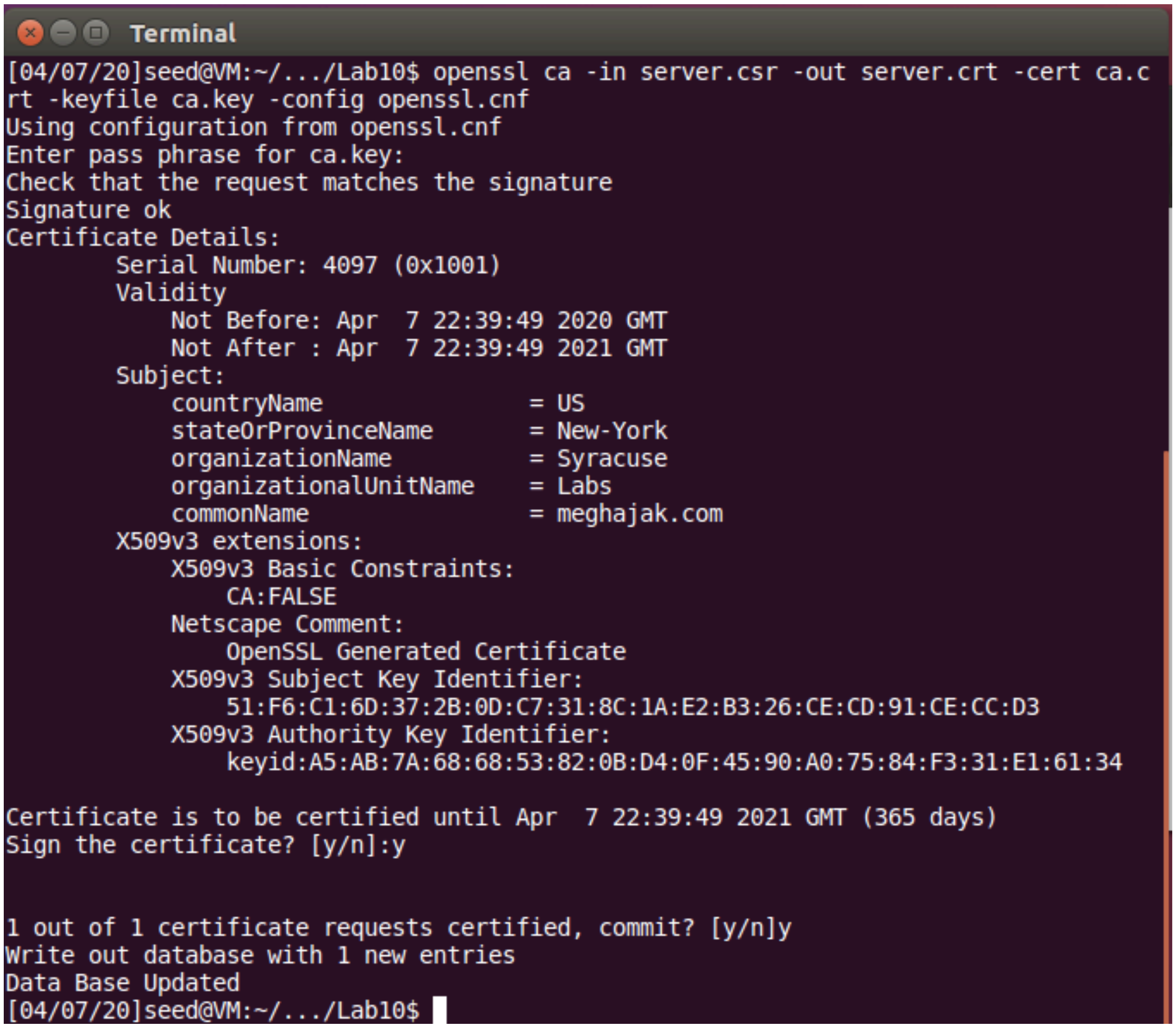
Step 3: Generating Certificates

The CSR file needs to have the CA's signature to form a certificate. We change the policy to policy_anything from policy_match to avoid any errors as seen in the following:



```
# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_anything
```

The following command turns the certificate signing request (server.csr) into an X509 certificate (server.crt), using the CA's ca.crt and ca.key. We see the details of the CSR being signed:



```
[04/07/20]seed@VM:~/.../Lab10$ openssl ca -in server.csr -out server.crt -cert ca.c
rt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Apr  7 22:39:49 2020 GMT
            Not After : Apr  7 22:39:49 2021 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = New-York
            organizationName          = Syracuse
            organizationalUnitName    = Labs
            commonName                = meghajak.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                51:F6:C1:6D:37:2B:0D:C7:31:8C:1A:E2:B3:26:CE:CD:91:CE:CC:D3
            X509v3 Authority Key Identifier:
                keyid:A5:AB:7A:68:68:53:82:0B:D4:0F:45:90:A0:75:84:F3:31:E1:61:34

Certificate is to be certified until Apr  7 22:39:49 2021 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[04/07/20]seed@VM:~/.../Lab10$
```

This step achieves a certificate for meghajak.com from the root certificate.

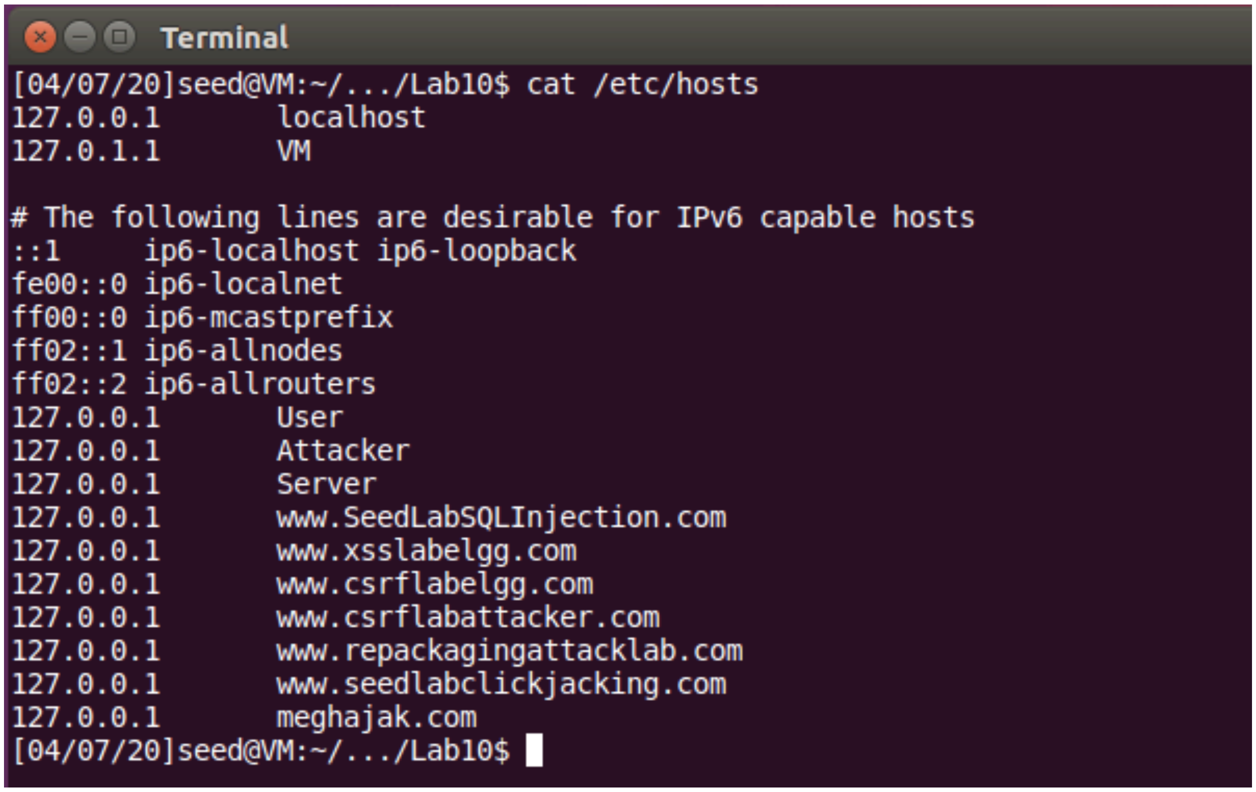
Task 3: Deploying Certificate in an HTTPS Web Server

Step 1: Configuring DNS

We add the following entry to the /etc/hosts file in order for the machine to recognize meghajak.com on the web browser without changing the DNS:

127.0.0.1 meghajak.com

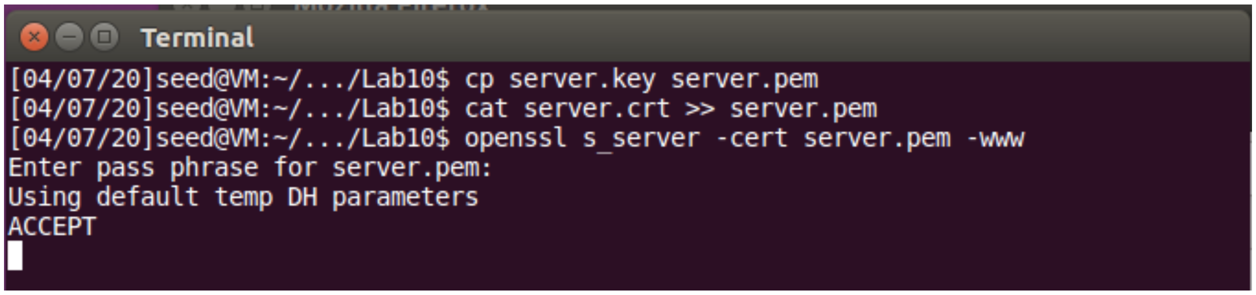
This entry basically maps the hostname meghajak.com to the localhost:



```
[04/07/20]seed@VM:~/.../Lab10$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       VM

# 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
127.0.0.1       User
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
127.0.0.1       meghajak.com
[04/07/20]seed@VM:~/.../Lab10$
```

Step 2: Configuring the web server

We combine the key and certificate into one file and then launch a web server using the openssl's s_server command. We enter the passphrase for the private key and see that the web server is now in the accept state – indicating it is running:

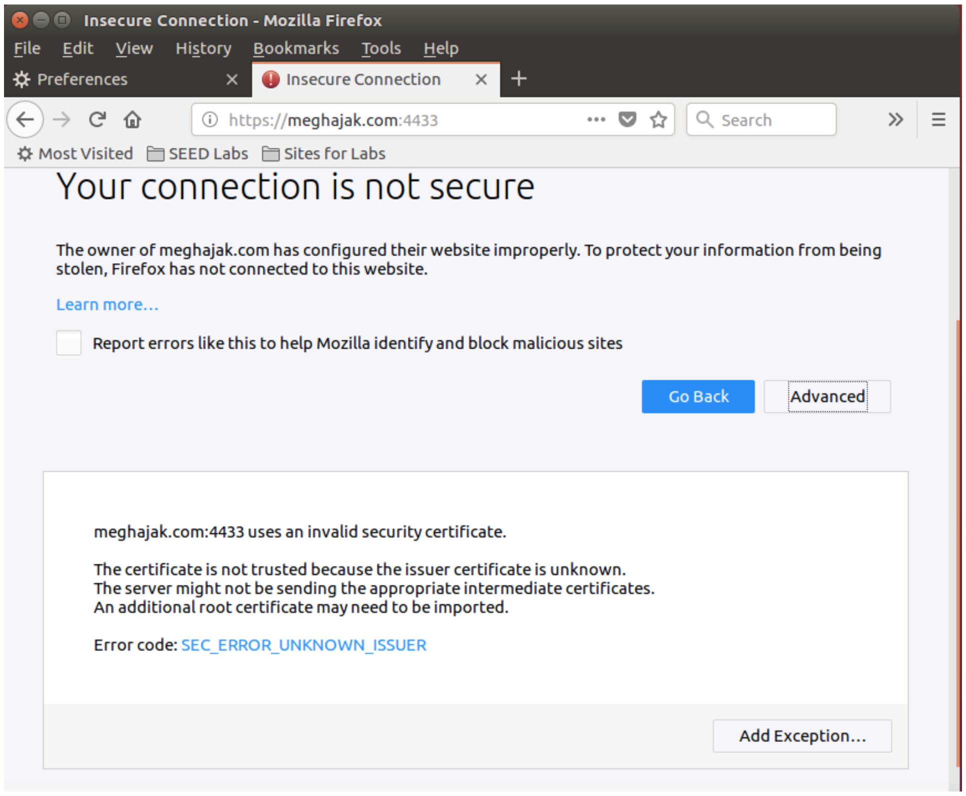


```
[04/07/20]seed@VM:~/.../Lab10$ cp server.key server.pem
[04/07/20]seed@VM:~/.../Lab10$ cat server.crt >> server.pem
[04/07/20]seed@VM:~/.../Lab10$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

By default, the server listens on port 4433 and we can access the server using the URL:

https://meghajak.com:4433/

On loading the above URL, we see that the web browser displays an error message with the connection being insecure. On loading more details by clicking on Advanced, we see that it states that the issuer's certificate is unknown i.e. the root certificate is unknown to the browser. This is because, even though the root certificate is created by us, it is not known to the browser and hence it cannot validate the server's certificate i.e. meghajak.com's certificate.


Insecure Connection - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Preferences
Insecure Connection
https://meghajak.com:4433
Search
Most Visited SEED Labs Sites for Labs

Your connection is not secure

The owner of meghajak.com has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more…

Report errors like this to help Mozilla identify and block malicious sites

meghajak.com:4433 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.
The server might not be sending the appropriate intermediate certificates.
An additional root certificate may need to be imported.

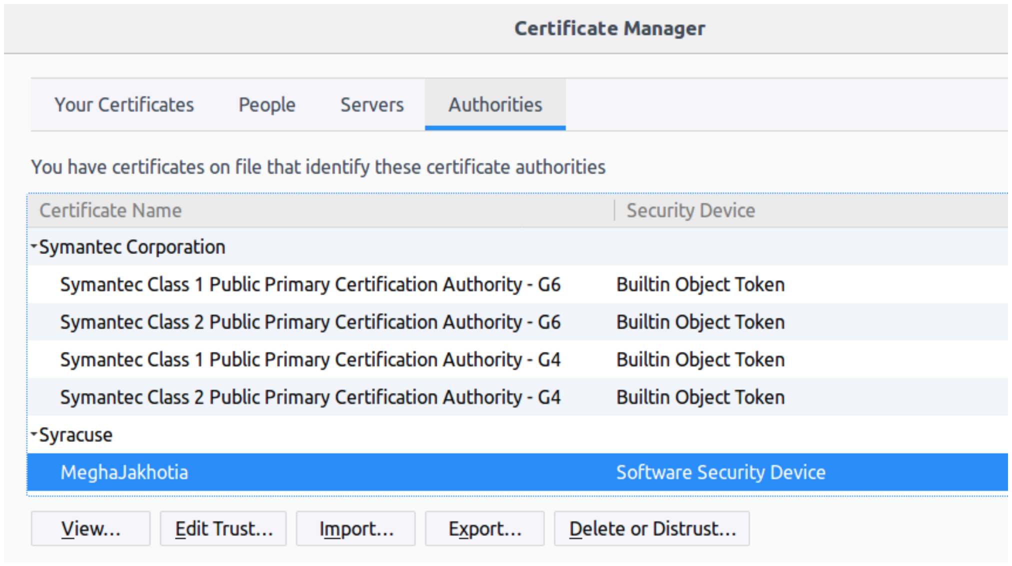
Error code: SEC_ERROR_UNKNOWN_ISSUER

Step 3: Getting the browser to accept our CA certificate.

In order for our root certificate to be accepted by the Firefox browser, we manually add our certificate to the browser by following the menu sequence and importing ca.crt:

```
Edit -> Preference -> Privacy & Security -> View Certificates
```

The following shows that the root CA is now in the Firefox's trusted certificates:

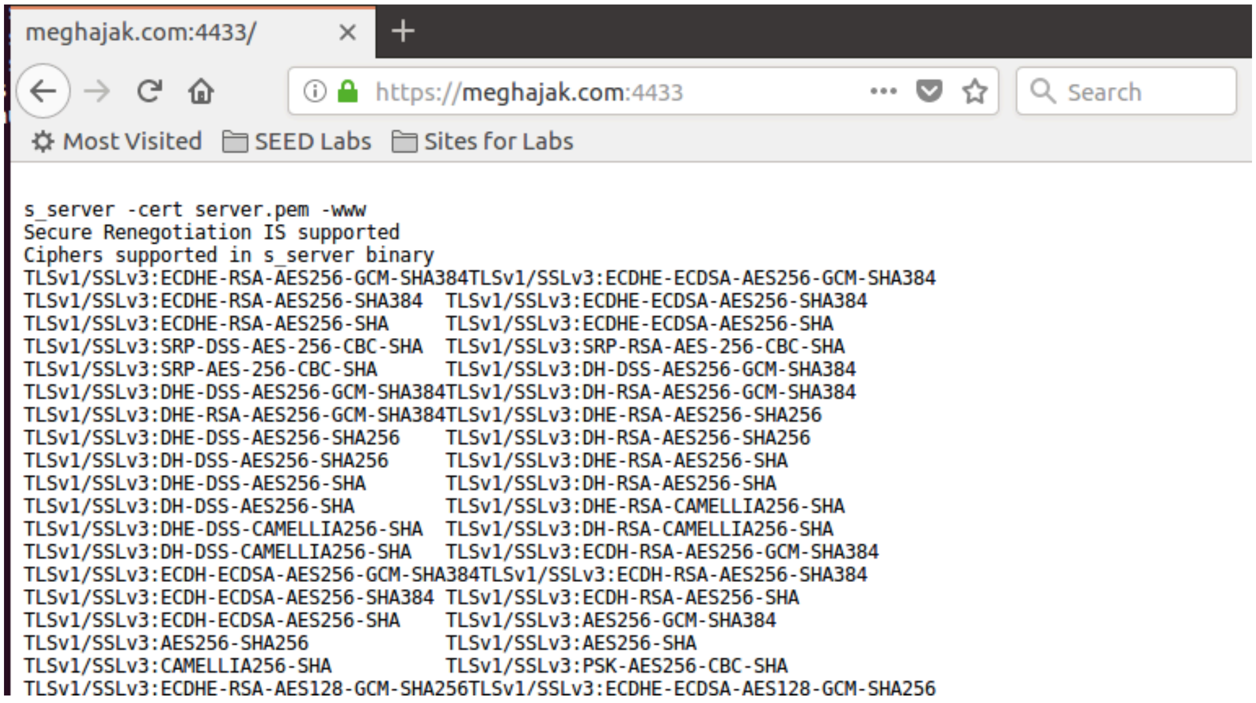


You have certificates on file that identify these certificate authorities

Certificate Name	Security Device
Symantec Corporation	
Symantec Class 1 Public Primary Certification Authority - G6	Builtin Object Token
Symantec Class 2 Public Primary Certification Authority - G6	Builtin Object Token
Symantec Class 1 Public Primary Certification Authority - G4	Builtin Object Token
Symantec Class 2 Public Primary Certification Authority - G4	Builtin Object Token
Syracuse	
MeghaJakhotia	Software Security Device

Step 4. Testing our HTTPS website.

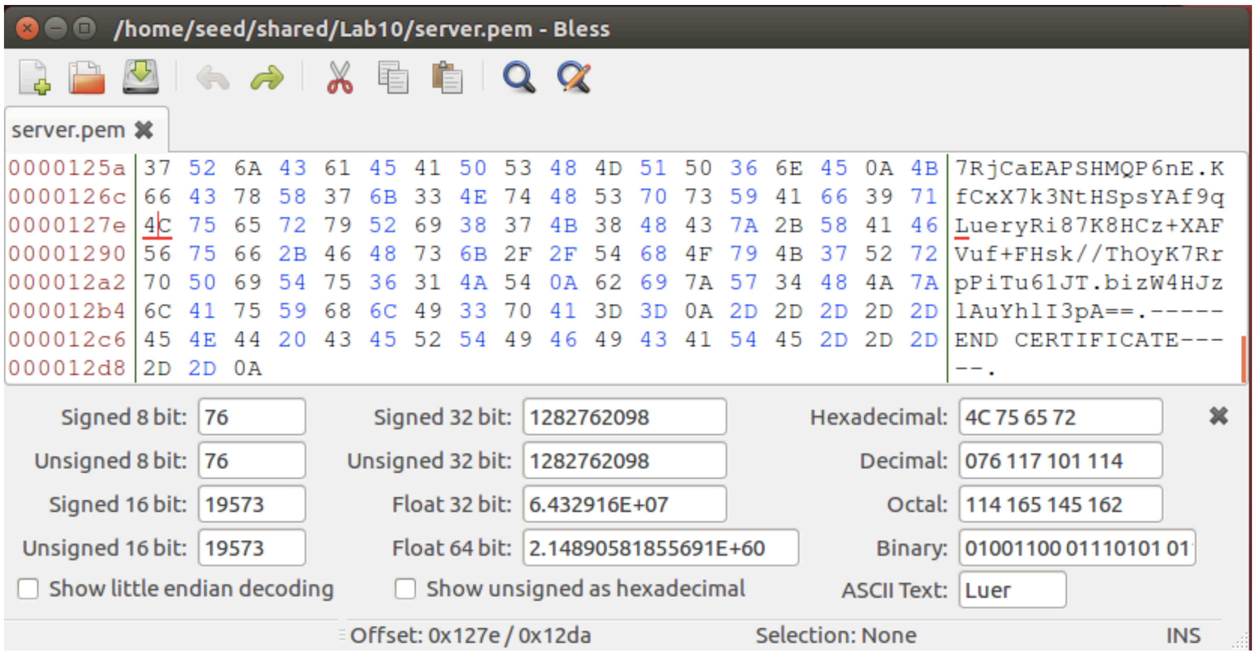
Now on loading the URL, we see the web page successfully:



```
s_server -cert server.pem -www
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:ECDHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA384  TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA     TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA
TLSv1/SSLv3:SRP-DSS-AES-256-CBC-SHA  TLSv1/SSLv3:SRP-RSA-AES-256-CBC-SHA
TLSv1/SSLv3:SRP-AES-256-CBC-SHA      TLSv1/SSLv3:DH-DSS-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-DSS-AES256-GCM-SHA384TLSv1/SSLv3:DH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:DHE-RSA-AES256-SHA256
TLSv1/SSLv3:DHE-DSS-AES256-SHA256    TLSv1/SSLv3:DH-RSA-AES256-SHA256
TLSv1/SSLv3:DH-DSS-AES256-SHA256     TLSv1/SSLv3:DHE-RSA-AES256-SHA
TLSv1/SSLv3:DHE-DSS-AES256-SHA       TLSv1/SSLv3:DH-RSA-AES256-SHA
TLSv1/SSLv3:DH-DSS-AES256-SHA        TLSv1/SSLv3:DHE-RSA-CAMELLIA256-SHA
TLSv1/SSLv3:DHE-DSS-CAMELLIA256-SHA  TLSv1/SSLv3:DH-RSA-CAMELLIA256-SHA
TLSv1/SSLv3:DH-DSS-CAMELLIA256-SHA   TLSv1/SSLv3:ECDH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3:ECDH-ECDSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDH-RSA-AES256-SHA384
TLSv1/SSLv3:ECDH-ECDSA-AES256-SHA384 TLSv1/SSLv3:ECDH-RSA-AES256-SHA
TLSv1/SSLv3:ECDH-ECDSA-AES256-SHA    TLSv1/SSLv3:AES256-GCM-SHA384
TLSv1/SSLv3:AES256-SHA256            TLSv1/SSLv3:AES256-SHA
TLSv1/SSLv3:CAMELLIA256-SHA          TLSv1/SSLv3:PSK-AES256-CBC-SHA
TLSv1/SSLv3:ECDHE-RSA-AES128-GCM-SHA256TLSv1/SSLv3:ECDHE-ECDSA-AES128-GCM-SHA256
```

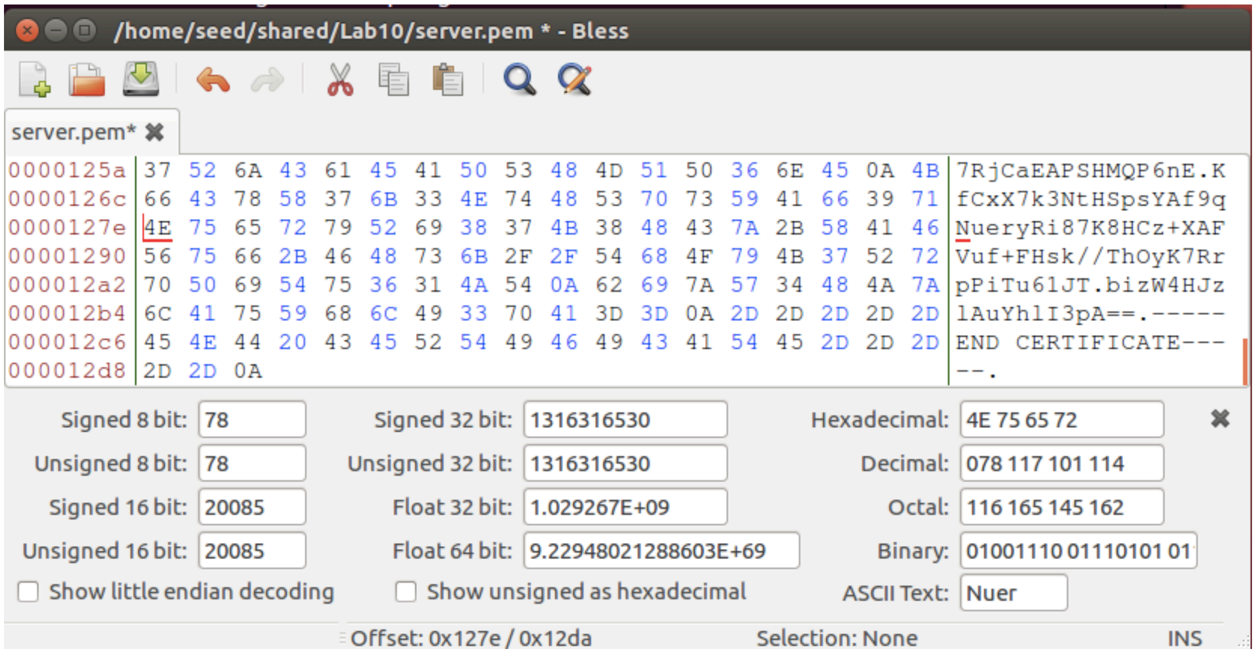
I. Modify a single byte of server.pem

We modify a single byte in the server.pem file from 4C to 4E as follows:



```
0000125a 37 52 6A 43 61 45 41 50 53 48 4D 51 50 36 6E 45 0A 4B 7RjCaEAPSHMQP6nE.K
0000126c 66 43 78 58 37 6B 33 4E 74 48 53 70 73 59 41 66 39 71 fCxX7k3NtHSpsYAf9q
0000127e 4C 75 65 72 79 52 69 38 37 4B 38 48 43 7A 2B 58 41 46 LueryRi87K8HCz+XAF
00001290 56 75 66 2B 46 48 73 6B 2F 2F 54 68 4F 79 4B 37 52 72 Vuf+FHsk//ThOyK7Rr
000012a2 70 50 69 54 75 36 31 4A 54 0A 62 69 7A 57 34 48 4A 7A pPiTu61JT.bizW4HJz
000012b4 6C 41 75 59 68 6C 49 33 70 41 3D 3D 0A 2D 2D 2D 2D 2D lAuYhlI3pA==.-----
000012c6 45 4E 44 20 43 45 52 54 49 46 49 43 41 54 45 2D 2D 2D END CERTIFICATE---
000012d8 2D 2D 0A                                              --.
```

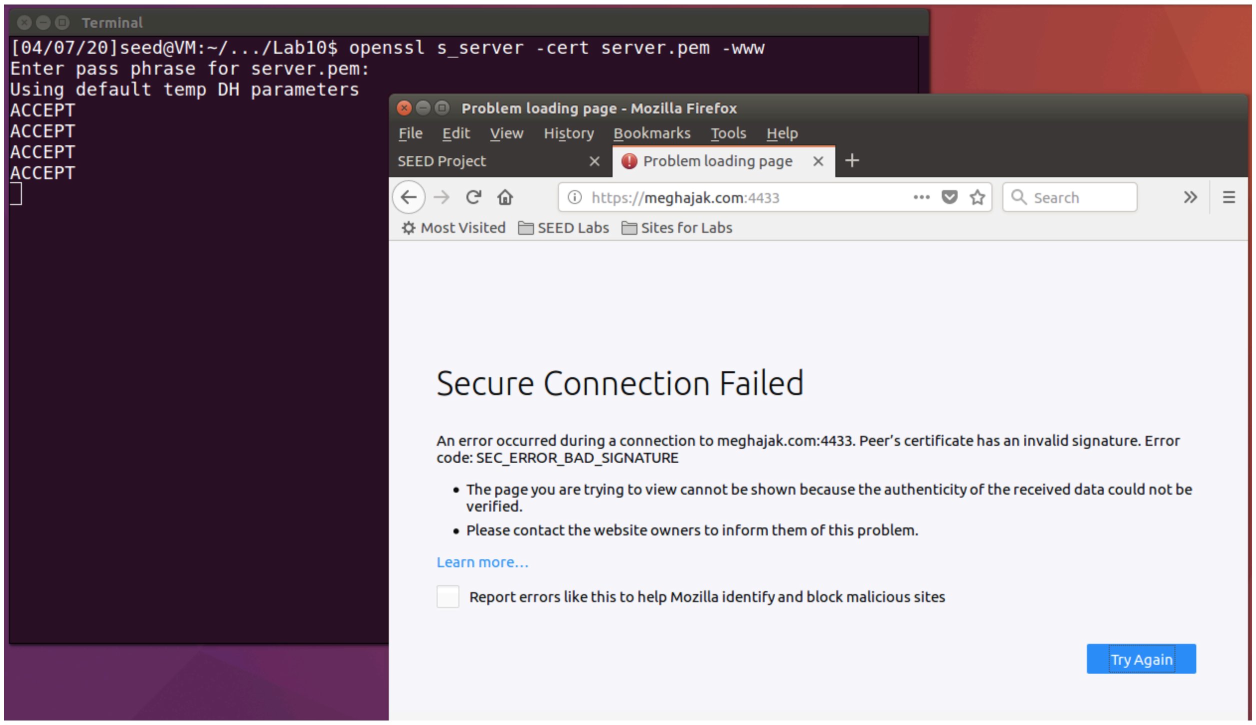
```
Signed 8 bit: 76                Signed 32 bit: 1282762098          Hexadecimal: 4C 75 65 72
Unsigned 8 bit: 76              Unsigned 32 bit: 1282762098        Decimal: 076 117 101 114
Signed 16 bit: 19573            Float 32 bit: 6.432916E+07         Octal: 114 165 145 162
Unsigned 16 bit: 19573          Float 64 bit: 2.14890581855691E+60 Binary: 01001100 01110101 01
☐ Show little endian decoding   ☐ Show unsigned as hexadecimal     ASCII Text: Luer
Offset: 0x127e / 0x12da         Selection: None                    INS
```



```
0000125a 37 52 6A 43 61 45 41 50 53 48 4D 51 50 36 6E 45 0A 4B 7RjCaEAPSHMQP6nE.K
0000126c 66 43 78 58 37 6B 33 4E 74 48 53 70 73 59 41 66 39 71 fCxX7k3NtHSpsYAf9q
0000127e 4E 75 65 72 79 52 69 38 37 4B 38 48 43 7A 2B 58 41 46 NueryRi87K8HCz+XAF
00001290 56 75 66 2B 46 48 73 6B 2F 2F 54 68 4F 79 4B 37 52 72 Vuf+FHsk//ThOyK7Rr
000012a2 70 50 69 54 75 36 31 4A 54 0A 62 69 7A 57 34 48 4A 7A pPiTu61JT.bizW4HJz
000012b4 6C 41 75 59 68 6C 49 33 70 41 3D 3D 0A 2D 2D 2D 2D 2D lAuYhlI3pA==.-----
000012c6 45 4E 44 20 43 45 52 54 49 46 49 43 41 54 45 2D 2D 2D END CERTIFICATE---
000012d8 2D 2D 0A                                              --.
```

```
Signed 8 bit: 78          Signed 32 bit: 1316316530         Hexadecimal: 4E 75 65 72
Unsigned 8 bit: 78        Unsigned 32 bit: 1316316530       Decimal: 078 117 101 114
Signed 16 bit: 20085      Float 32 bit: 1.029267E+09        Octal: 116 165 145 162
Unsigned 16 bit: 20085    Float 64 bit: 9.22948021288603E+69 Binary: 01001110 01110101 01
☐ Show little endian decoding  ☐ Show unsigned as hexadecimal  ASCII Text: Nuer
Offset: 0x127e / 0x12da     Selection: None     INS
```

Now, on restarting the server and reloading the URL, we see that a single change in server.pem file caused the browser to throw an error of secure connection failed due to an invalid signature:


Terminal
Problem loading page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SEED Project
Problem loading page
https://meghajak.com:4433
Search
Most Visited
SEED Labs
Sites for Labs
Try Again

```
[04/07/20]seed@VM:~/.../Lab10$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
ACCEPT
ACCEPT
ACCEPT
```

Secure Connection Failed

An error occurred during a connection to meghajak.com:4433. Peer's certificate has an invalid signature. Error code: SEC_ERROR_BAD_SIGNATURE

- The page you are trying to view cannot be shown because the authenticity of the received data could not be verified.
- Please contact the website owners to inform them of this problem.

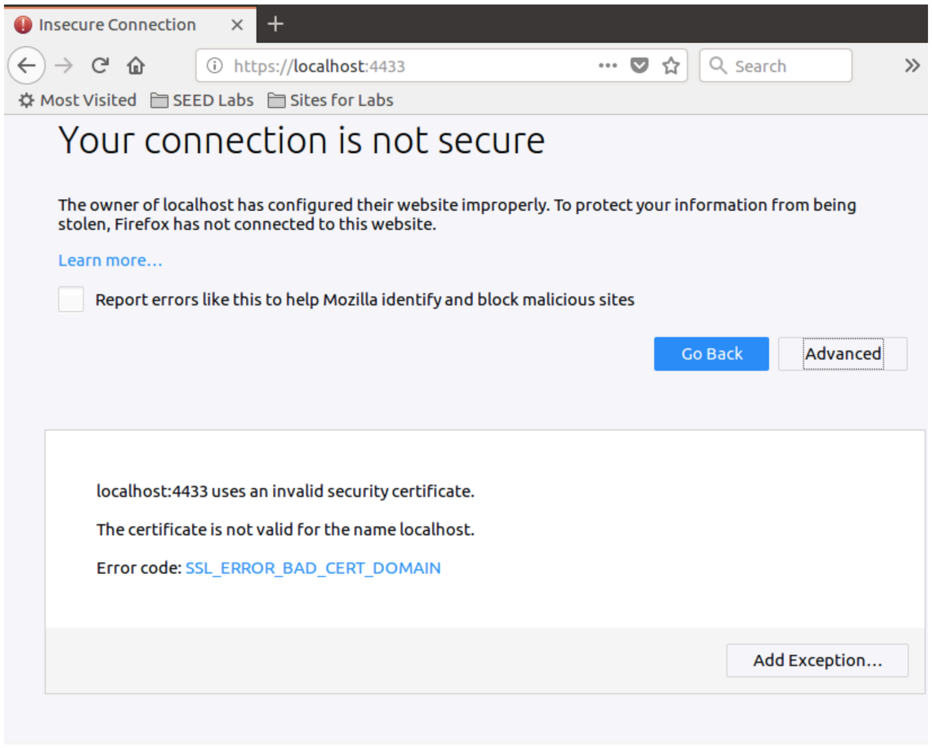
Learn more…

☐ Report errors like this to help Mozilla identify and block malicious sites

This proves that the certificate's integrity is a must and any change in the certificate will break the connection.

II. Accessing localhost instead of meghajak.com

Now, we load localhost:4433 instead of meghajak.com:4433. We see that it gives us an error with the connection being insecure. However, since meghajak.com is linked with 127.0.0.1, essentially the localhost, the website should not really be insecure.



Your connection is not secure

The owner of localhost has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more…

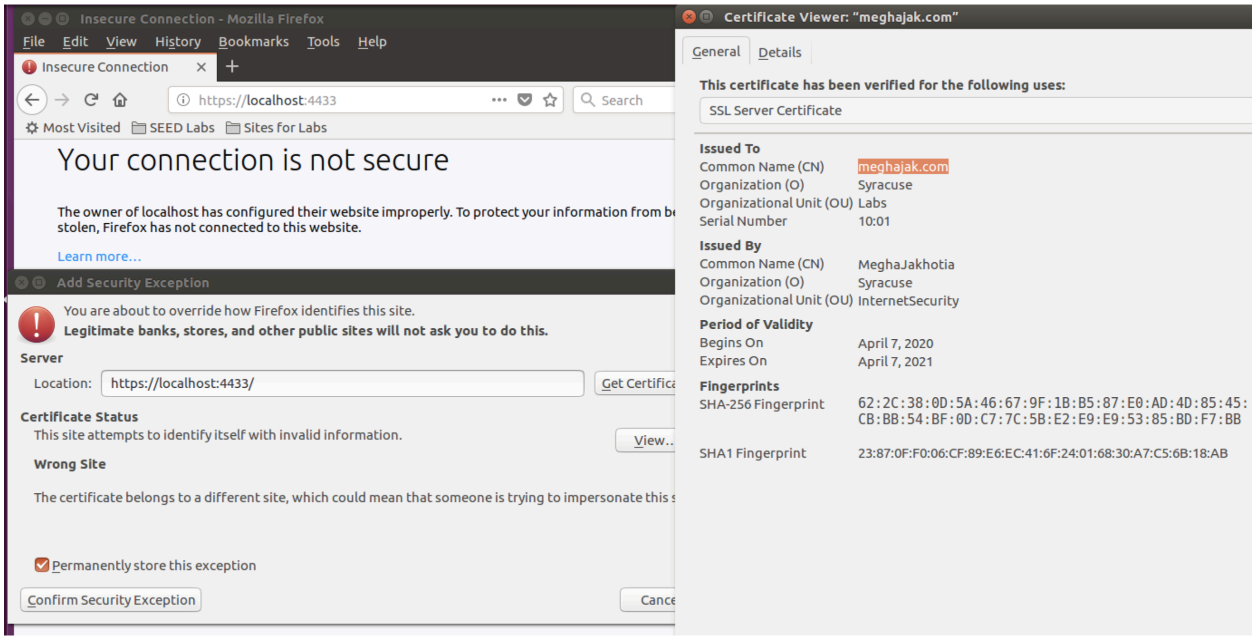
Report errors like this to help Mozilla identify and block malicious sites

localhost:4433 uses an invalid security certificate.

The certificate is not valid for the name localhost.

Error code: SSL_ERROR_BAD_CERT_DOMAIN

We see that in the additional details, the certificate is invalid because it is not meant for name localhost. On clicking on Add Exception in order to see more details, we see that the certificate is for meghajak.com and the URL being loaded is localhost. This basically causes the error to pop up.



Your connection is not secure

The owner of localhost has configured their website improperly. To protect your information from be stolen, Firefox has not connected to this website.

Learn more…

You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://localhost:4433/

Certificate Status

This site attempts to identify itself with invalid information.

Wrong Site

The certificate belongs to a different site, which could mean that someone is trying to impersonate this s

☑ Permanently store this exception

This certificate has been verified for the following uses:

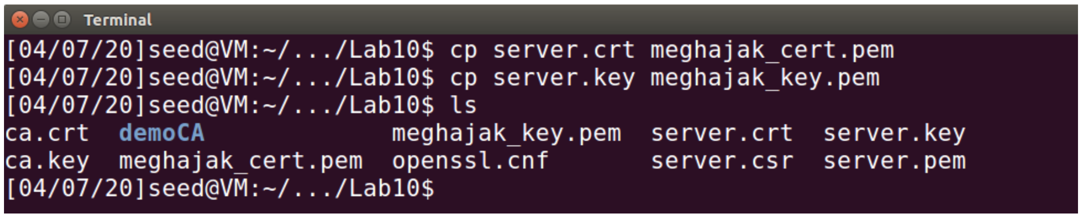
SSL Server Certificate

Issued To	
Common Name (CN)	meghajak.com
Organization (O)	Syracuse
Organizational Unit (OU)	Labs
Serial Number	10:01
Issued By	
Common Name (CN)	MeghaJakhotia
Organization (O)	Syracuse
Organizational Unit (OU)	InternetSecurity
Period of Validity	
Begins On	April 7, 2020
Expires On	April 7, 2021
Fingerprints	
SHA-256 Fingerprint	62:2C:38:0D:5A:46:67:9F:1B:B5:87:E0:AD:4D:85:45: CB:BB:54:BF:0D:C7:7C:5B:E2:E9:E9:53:85:BD:F7:BB
SHA1 Fingerprint	23:87:0F:F0:06:CF:89:E6:EC:41:6F:24:01:68:30:A7:C5:6B:18:AB

This proves that the browser also checks for the correctness of the common name and the url requested. The certificate is valid but for meghajak.com and not localhost, and even though both of them point to the same server, the names are different and hence browser does not allow this. If we add the exception, the result will be same as that of meghajak.com.

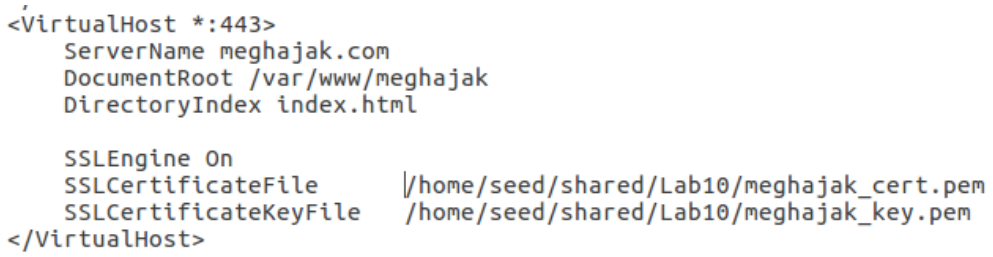
Task 4: Deploying Certificate in an Apache-Based HTTPSWebsite

We create the pem file for the server's certificate and key to be used in the Apache website configuration:



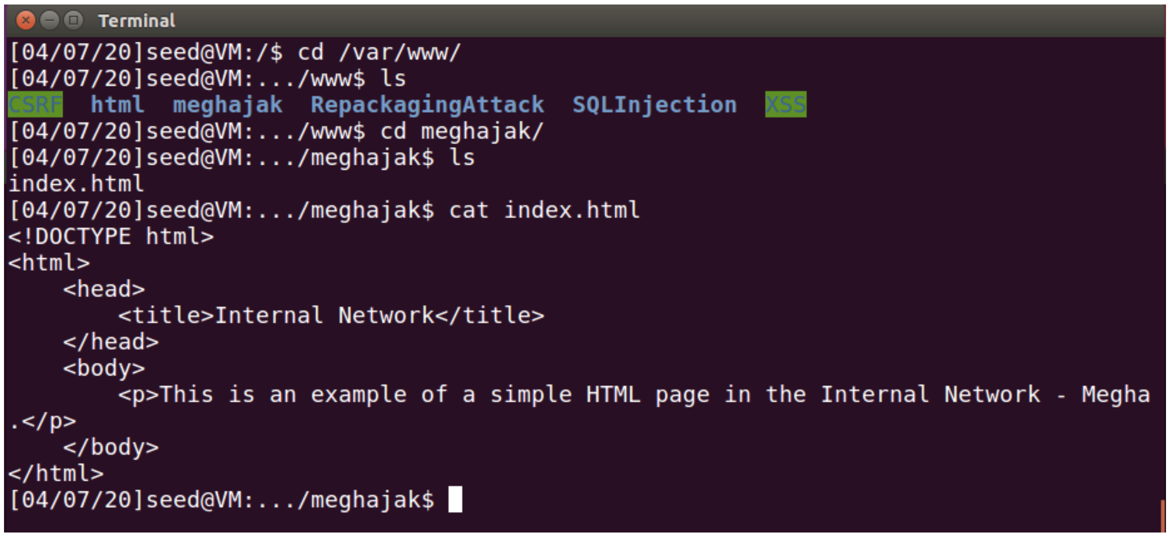
```
[04/07/20]seed@VM:~/.../Lab10$ cp server.crt meghajak_cert.pem
[04/07/20]seed@VM:~/.../Lab10$ cp server.key meghajak_key.pem
[04/07/20]seed@VM:~/.../Lab10$ ls
ca.crt  democA           meghajak_key.pem  server.crt  server.key
ca.key  meghajak_cert.pem  openssl.cnf      server.csr  server.pem
[04/07/20]seed@VM:~/.../Lab10$
```

We add a VirtualHost entry to the default-ssl.conf file as follows:



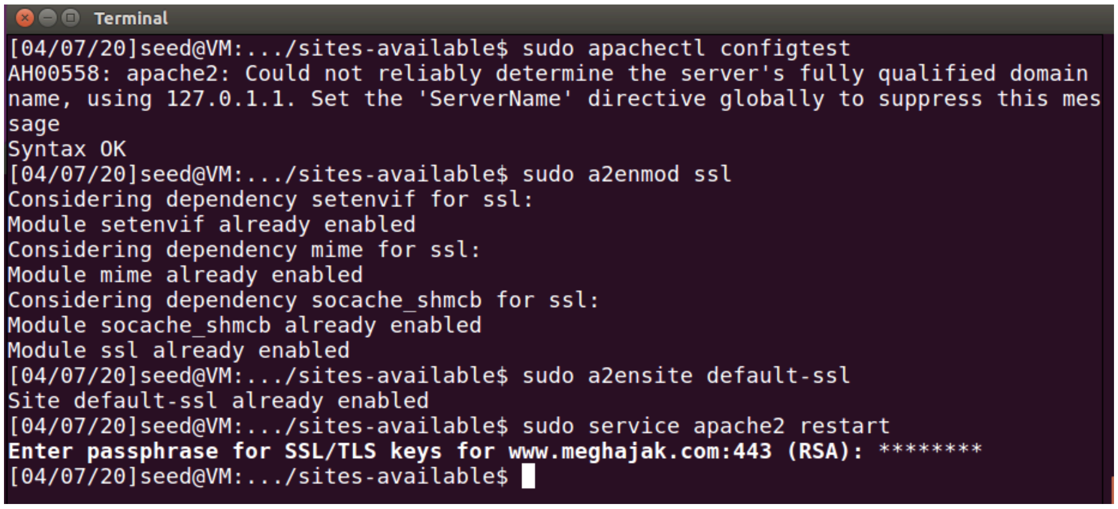
```
<VirtualHost *:443>
    ServerName meghajak.com
    DocumentRoot /var/www/meghajak
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile      /home/seed/shared/Lab10/meghajak_cert.pem
    SSLCertificateKeyFile   /home/seed/shared/Lab10/meghajak_key.pem
</VirtualHost>
```

The ServerName entry specifies the name of the website, while the DocumentRoot entry specifies where the files for the website are stored. The following show the website's files:



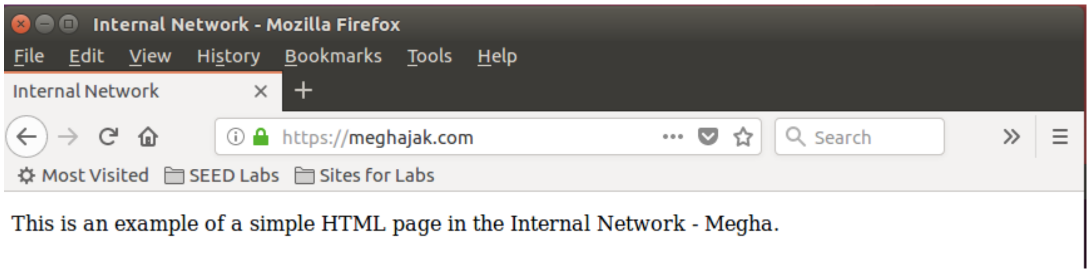
```
[04/07/20]seed@VM:/$ cd /var/www/
[04/07/20]seed@VM:.../www$ ls
CSRF  html  meghajak  RepackagingAttack  SQLInjection  XSS
[04/07/20]seed@VM:.../www$ cd meghajak/
[04/07/20]seed@VM:.../meghajak$ ls
index.html
[04/07/20]seed@VM:.../meghajak$ cat index.html
<!DOCTYPE html>
<html>
    <head>
        <title>Internal Network</title>
    </head>
    <body>
        <p>This is an example of a simple HTML page in the Internal Network - Megha
.</p>
    </body>
</html>
[04/07/20]seed@VM:.../meghajak$
```

After the default-ssl.conf file is modified, we run a series of commands to enable SSL and start the Apache server as seen in the following screenshot:



```
[04/07/20]seed@VM:.../sites-available$ sudo apachectl configtest
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 mes
sage
Syntax OK
[04/07/20]seed@VM:.../sites-available$ 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
[04/07/20]seed@VM:.../sites-available$ sudo a2ensite default-ssl
Site default-ssl already enabled
[04/07/20]seed@VM:.../sites-available$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for www.meghajak.com:443 (RSA): ********
[04/07/20]seed@VM:.../sites-available$
```

Now, we open the browser and load the URL and see that we can successfully browse the HTTPS site:


Internal Network - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Internal Network
https://meghajak.com
Search
Most Visited
SEED Labs
Sites for Labs

This is an example of a simple HTML page in the Internal Network - Megha.

Task 5: Launching a Man-In-The-Middle Attack

Step 1: Setting up the malicious website.

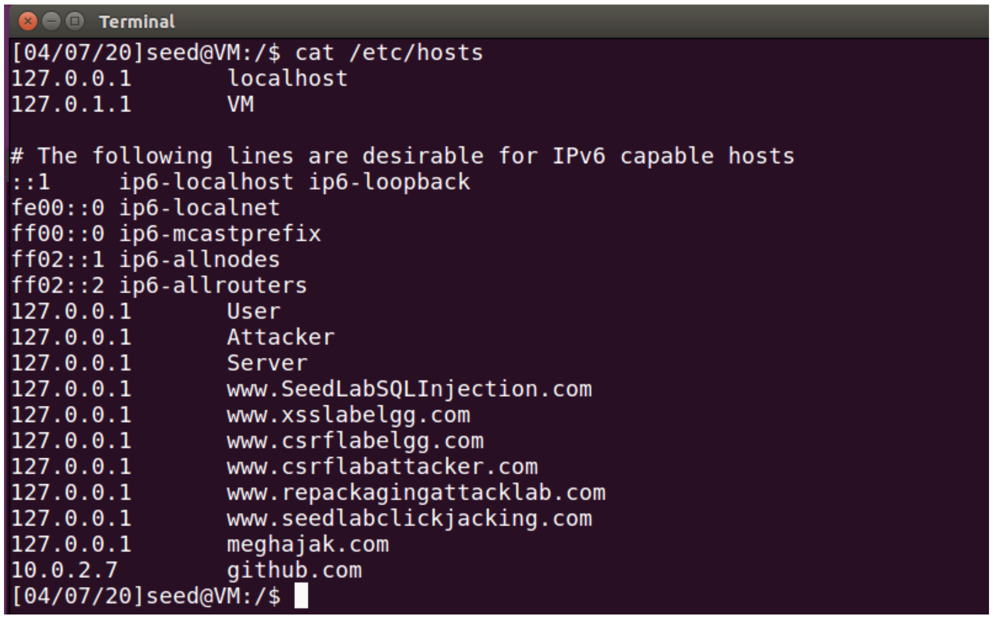
Now, we try to impersonate github.com and change the VirtualHost entry in the Apache's SSL configuration to have the ServerName as github.com, and everything else remains the same:

```
</VirtualHost>
<VirtualHost *:443>
    ServerName github.com
    DocumentRoot /var/www/meghajak
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile      /home/seed/shared/Lab10/meghajak_cert.pem
    SSLCertificateKeyFile   /home/seed/shared/Lab10/meghajak_key.pem
</VirtualHost>
```

This will launch a fake github.com website. We restart the apache web server to save the changes.

Step 2: Becoming the man in the middle

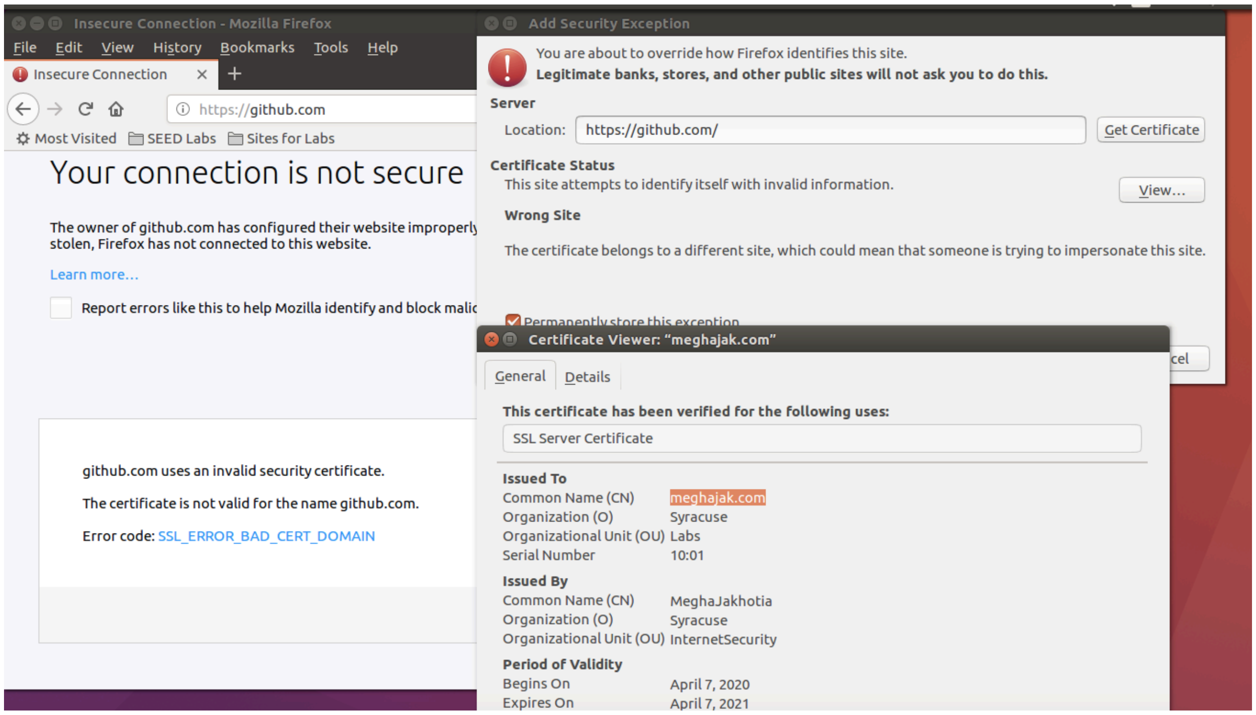
Now, to perform MITM, we change the DNS of the client machine to have an entry that will redirect the requests going to the original github.com to the fake web server set by us. We change the /etc/hosts file:



```
[04/07/20]seed@VM:/$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       VM

# 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
127.0.0.1       User
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
127.0.0.1       meghajak.com
10.0.2.7        github.com
[04/07/20]seed@VM:/$
```

Step 3: Browse the target website.

Now we browse the target website – github.com and see that we get an error saying that the connection is not secure. On looking at more details, we see that the common name of the requested URL and the certificate does not match, indicating the certificate is not meant for this domain.



Your connection is not secure

The owner of github.com has configured their website improperly stolen, Firefox has not connected to this website.

Learn more…

Report errors like this to help Mozilla identify and block malic

github.com uses an invalid security certificate.

The certificate is not valid for the name github.com.

Error code: SSL_ERROR_BAD_CERT_DOMAIN

Add Security Exception

You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.

Server
Location: https://github.com/

Certificate Status
This site attempts to identify itself with invalid information.

Wrong Site

The certificate belongs to a different site, which could mean that someone is trying to impersonate this site.

Permanently store this exception

Certificate Viewer: "meghajak.com"

This certificate has been verified for the following uses:
SSL Server Certificate

Issued To
Common Name (CN) meghajak.com
Organization (O) Syracuse
Organizational Unit (OU) Labs
Serial Number 10:01

Issued By
Common Name (CN) MeghaJakhotia
Organization (O) Syracuse
Organizational Unit (OU) InternetSecurity

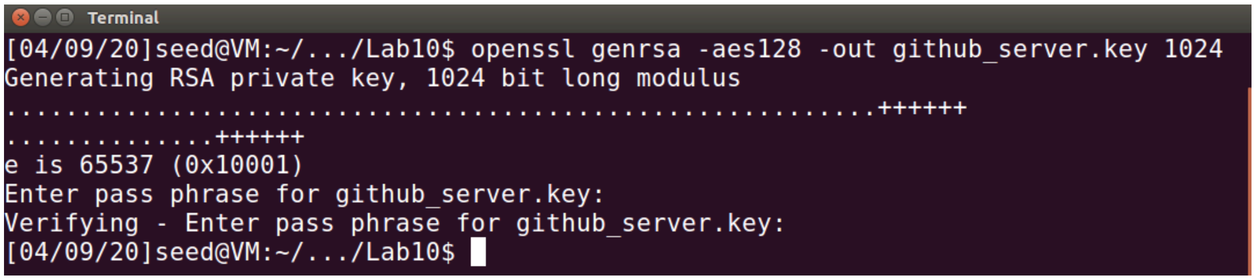
Period of Validity
Begins On April 7, 2020
Expires On April 7, 2021

Insecure Connection - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Insecure Connection
https://github.com
Most Visited SEED Labs Sites for Labs
Get Certificate
View…
General Details

This proves that the MITM attack is defeated in the use of public key infrastructure. If the common name matched, then the website would have loaded. However, the common name will match only if the server had a valid certificate from the CA for its domain, which it didn't, in this case.

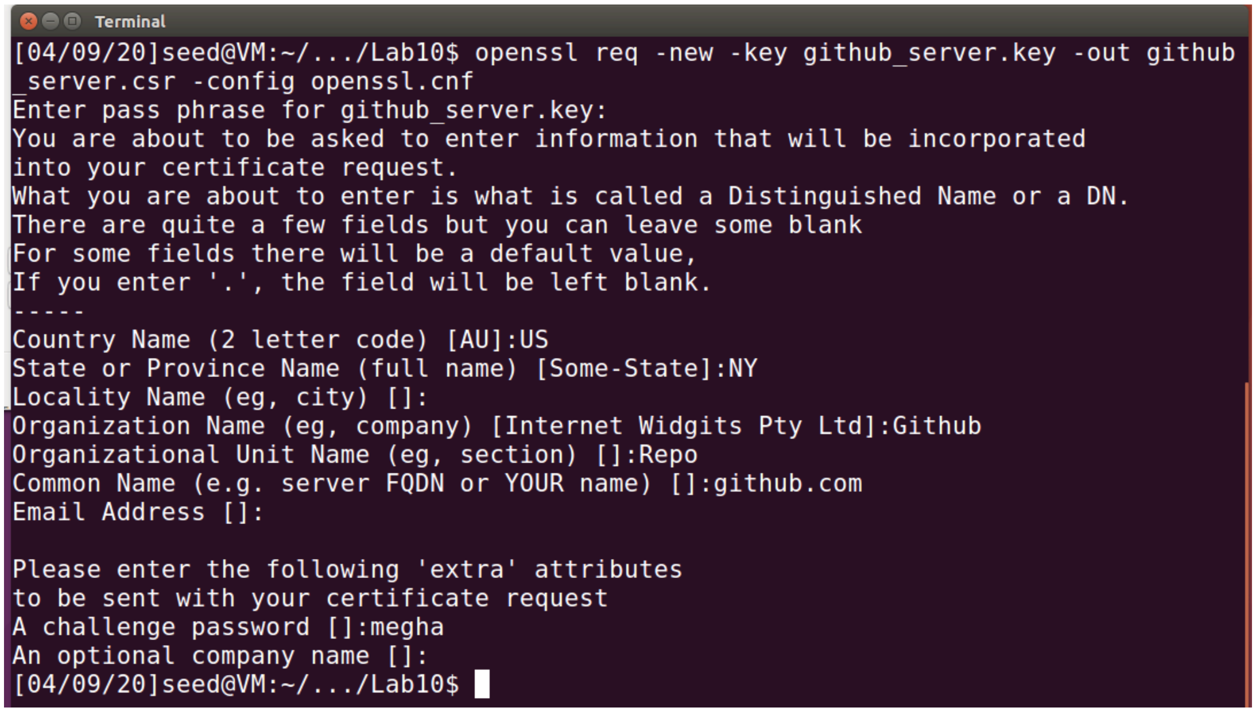
Task 6: Launching a Man-In-The-Middle Attack with a Compromised CA

Considering the same setting as in Task 5 and the root CA's private key being compromised. We first create a private-public key for github.com, the target website. We could have used the same keys generated before but just for simplicity, we create new keys.



```
[04/09/20]seed@VM:~/.../Lab10$ openssl genrsa -aes128 -out github_server.key 1024
Generating RSA private key, 1024 bit long modulus
.........................................................++++++
..............++++++
e is 65537 (0x10001)
Enter pass phrase for github_server.key:
Verifying - Enter pass phrase for github_server.key:
[04/09/20]seed@VM:~/.../Lab10$
```

Now, we create a CSR for github.com using OpenSSL in the same way as before, to be signed by a CA:

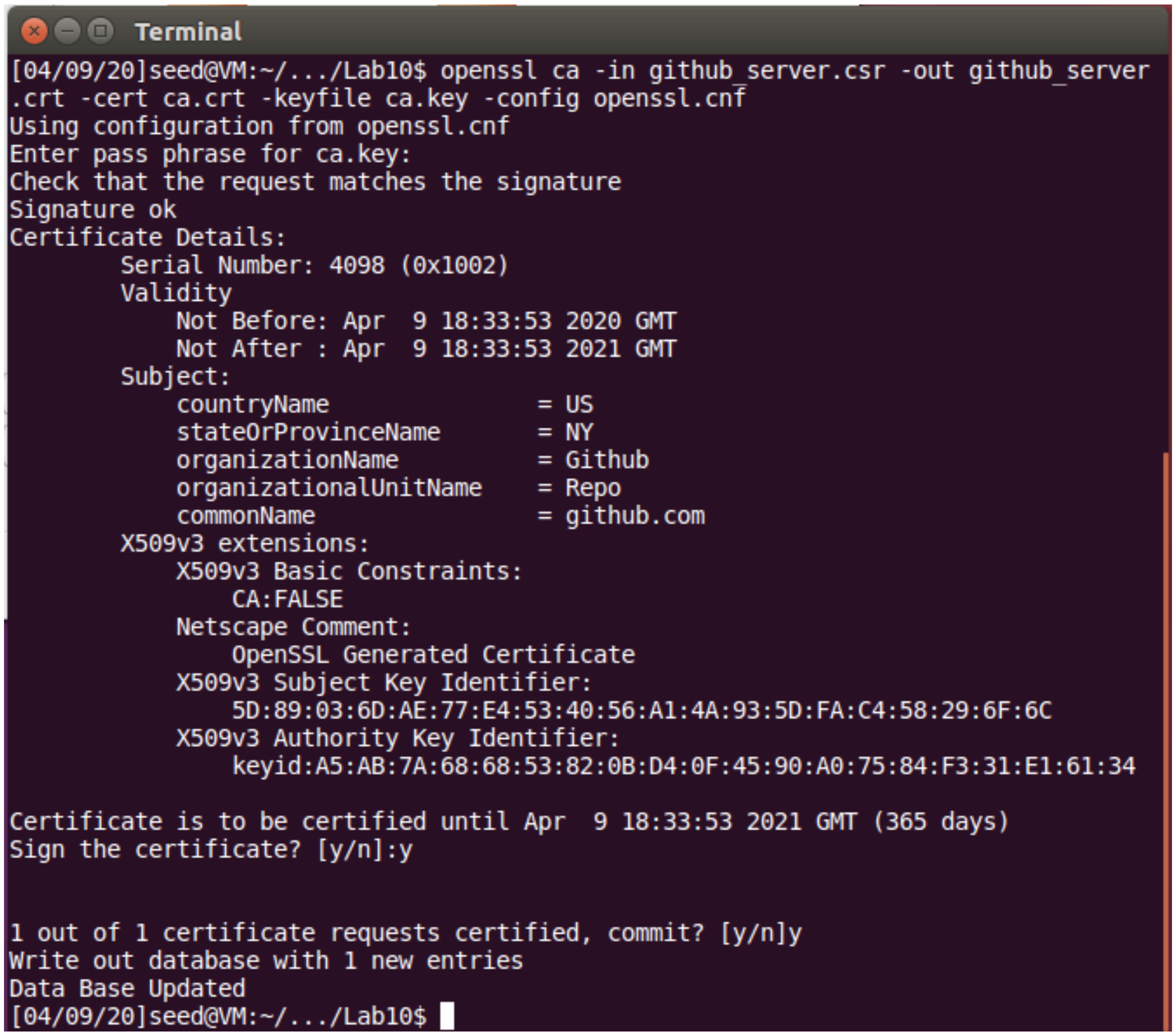


```
[04/09/20]seed@VM:~/.../Lab10$ openssl req -new -key github_server.key -out github
_server.csr -config openssl.cnf
Enter pass phrase for github_server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is 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]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Github
Organizational Unit Name (eg, section) []:Repo
Common Name (e.g. server FQDN or YOUR name) []:github.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:megha
An optional company name []:
[04/09/20]seed@VM:~/.../Lab10$
```

Now, since we have the root CA's private key and the certificate is publicly available, we can create a certificate ourselves from the CSR and do not require the CA to do it for us. We use the same command as before to create a certificate.

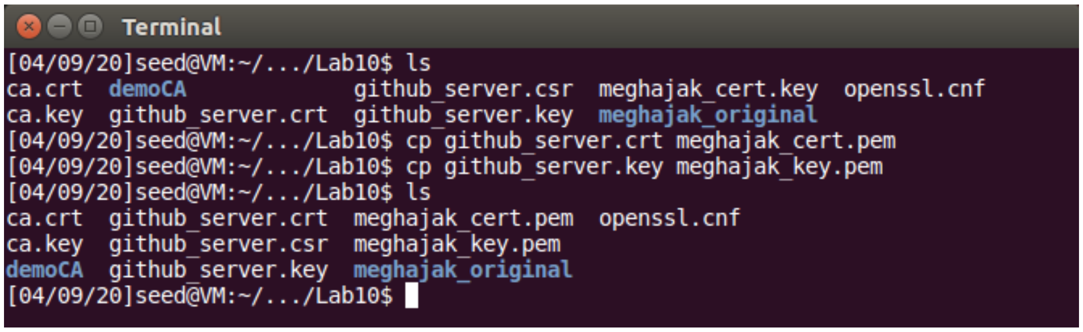
Now, since this certificate is signed by the root CA that is trusted by the Firefox browser, while authenticating the created certificate, the browser will confirm the validity of this certificate since a trusted root certificate vouched for it.

The following shows the creation of a certificate from the CSR using the root CA's credentials:



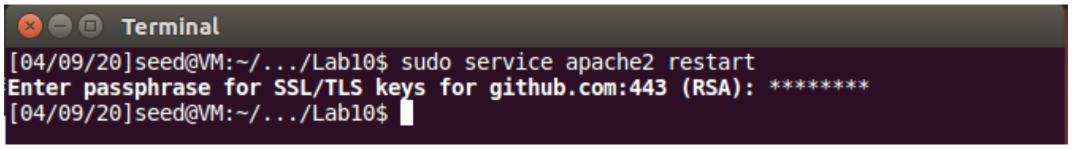
```
[04/09/20]seed@VM:~/.../Lab10$ openssl ca -in github_server.csr -out github_server
.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4098 (0x1002)
        Validity
            Not Before: Apr  9 18:33:53 2020 GMT
            Not After : Apr  9 18:33:53 2021 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = NY
            organizationName          = Github
            organizationalUnitName    = Repo
            commonName                = github.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                5D:89:03:6D:AE:77:E4:53:40:56:A1:4A:93:5D:FA:C4:58:29:6F:6C
            X509v3 Authority Key Identifier:
                keyid:A5:AB:7A:68:68:53:82:0B:D4:0F:45:90:A0:75:84:F3:31:E1:61:34

Certificate is to be certified until Apr  9 18:33:53 2021 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[04/09/20]seed@VM:~/.../Lab10$
```

We save the created key and certificate as pem files with names used by the Apache server:



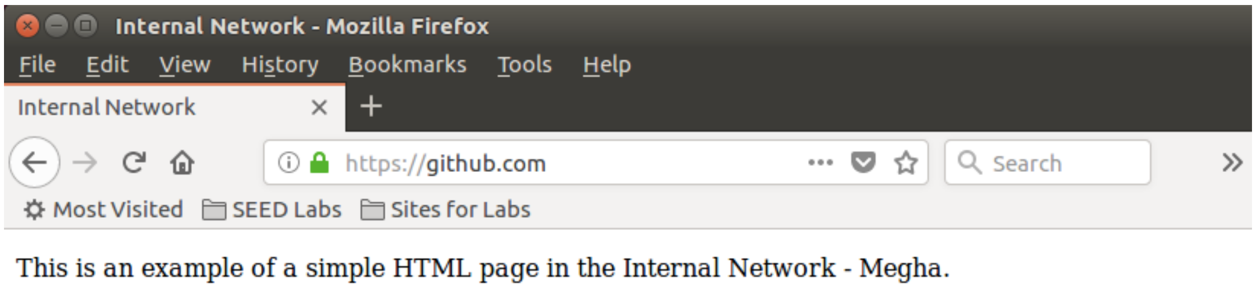
```
[04/09/20]seed@VM:~/.../Lab10$ ls
ca.crt  demoCA              github_server.csr  meghajak_cert.key  openssl.cnf
ca.key  github_server.crt   github_server.key  meghajak_original
[04/09/20]seed@VM:~/.../Lab10$ cp github_server.crt meghajak_cert.pem
[04/09/20]seed@VM:~/.../Lab10$ cp github_server.key meghajak_key.pem
[04/09/20]seed@VM:~/.../Lab10$ ls
ca.crt  github_server.crt  meghajak_cert.pem  openssl.cnf
ca.key  github_server.csr  meghajak_key.pem
demoCA  github_server.key  meghajak_original
[04/09/20]seed@VM:~/.../Lab10$
```

In order to incorporate the changes, we restart the Apache service:



```
[04/09/20]seed@VM:~/.../Lab10$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for github.com:443 (RSA): ********
[04/09/20]seed@VM:~/.../Lab10$
```

Now, on loading the url of the target website, we see that the browser does not throw any more errors and display the expected website.


Internal Network - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Internal Network
https://github.com
Search
Most Visited
SEED Labs
Sites for Labs

This is an example of a simple HTML page in the Internal Network - Megha.

This is because the certificate is valid due to being signed by the root CA and its common name is github.com. If the root CA's key is compromised, then anyone can create a certificate for themselves and impersonate any other website.