Jeffrey Cho

Professor Palmer

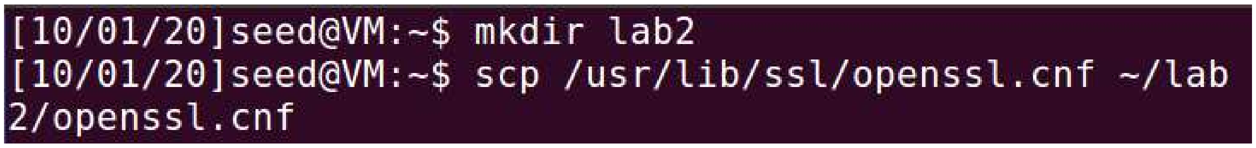
COSC 55: Security & Privacy

October 4, 2020

Lab Report 2

Task 1 Becoming a Certificate Authority (CA):

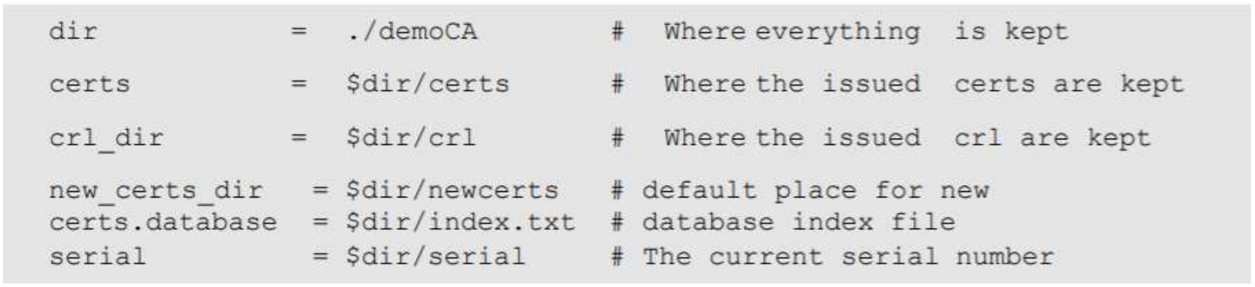
Using "**scp**," I was able to securely copy the "**openssl.cnf**" from the specified directory "*/usr/lib/ssl/openssl.cnf*" to my "**lab2**" directory using the following terminal commands in **(1.0).**



```
[10/01/20]seed@VM:~$ mkdir lab2
[10/01/20]seed@VM:~$ scp /usr/lib/ssl/openssl.cnf ~/lab
2/openssl.cnf
```

(1.0)

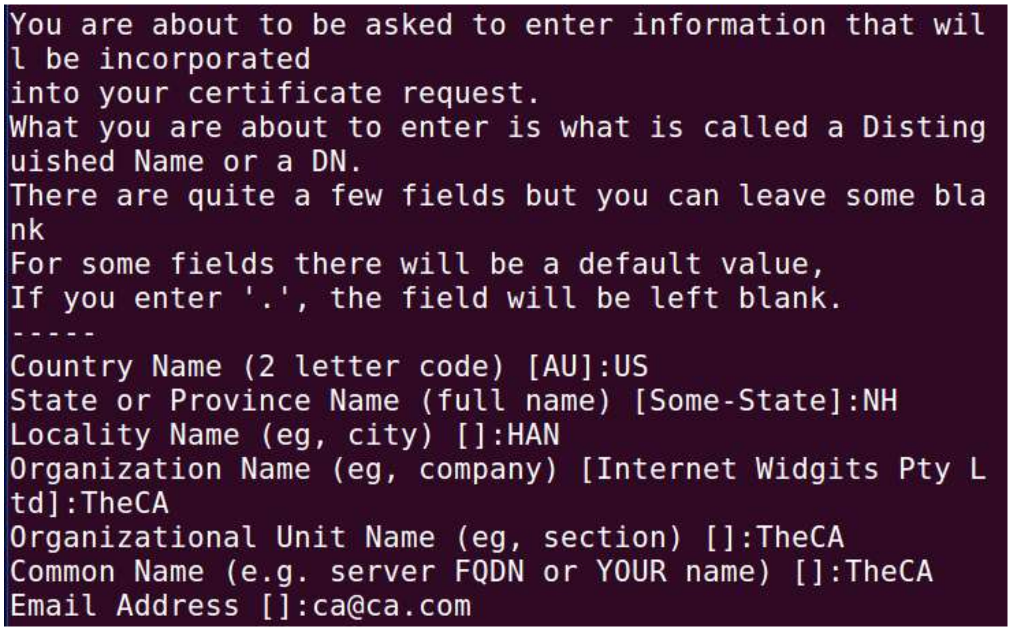
After the directory was securely copied, subdirectories were created as based on **(1.1)** as shown below.



```
dir             =  ./demoCA         #  Where everything  is kept
certs           =  $dir/certs       #  Where the issued  certs are kept
crl_dir         =  $dir/crl         #  Where the issued  crl are kept
new_certs_dir    = $dir/newcerts    # default place for new
certs.database   = $dir/index.txt   # database index file
serial           = $dir/serial      # The current serial number
```

(1.1)

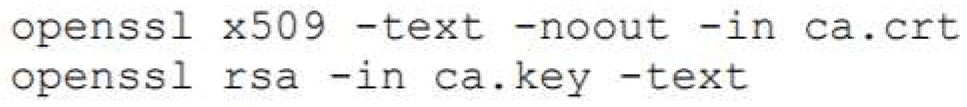
Afterwards, "**index.txt**" and "**serial**" were created, and the string "*1000*" was put into the "**serial**" file. Then the command *"$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf*" was used. The password was entered and verified (***Phidelt!22***). The "**ca.key**" and "**ca.crt**" were created after filling in the details shown in **(1.2)**.



```
You are about to be asked to enter information that wil
l be incorporated
into your certificate request.
What you are about to enter is what is called a Disting
uished Name or a DN.
There are quite a few fields but you can leave some bla
nk
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]:NH
Locality Name (eg, city) []:HAN
Organization Name (eg, company) [Internet Widgits Pty L
td]:TheCA
Organizational Unit Name (eg, section) []:TheCA
Common Name (e.g. server FQDN or YOUR name) []:TheCA
Email Address []:ca@ca.com
```

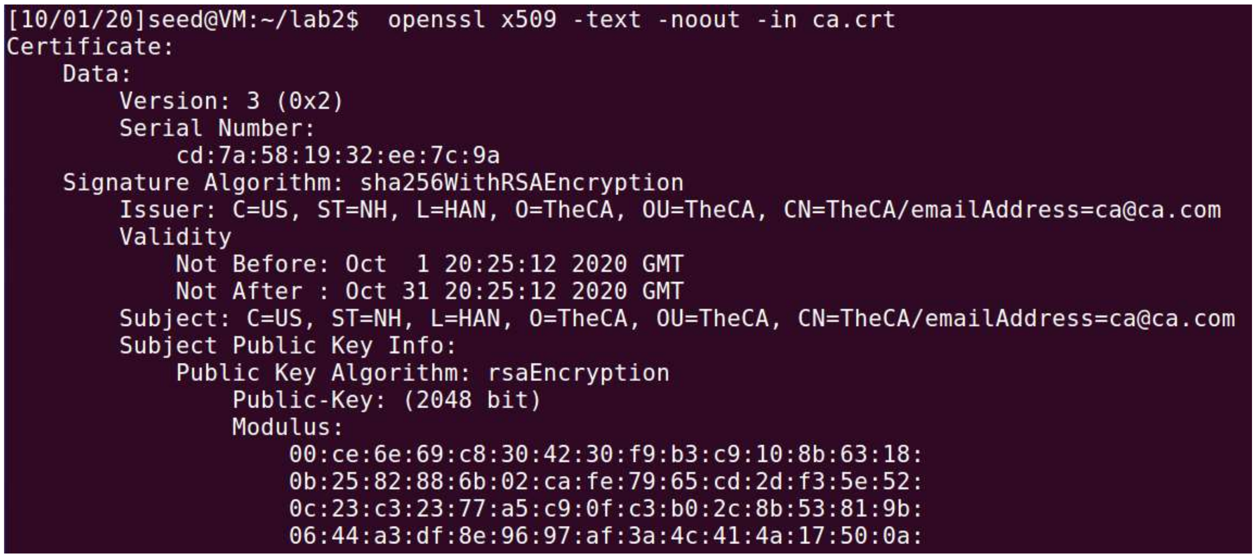
(1.2)

After running the commands shown in **(1.3)**, we can see the output in **(1.4)** & **(1.5)**.



```
openssl x509 -text -noout -in ca.crt
openssl rsa -in ca.key -text
```

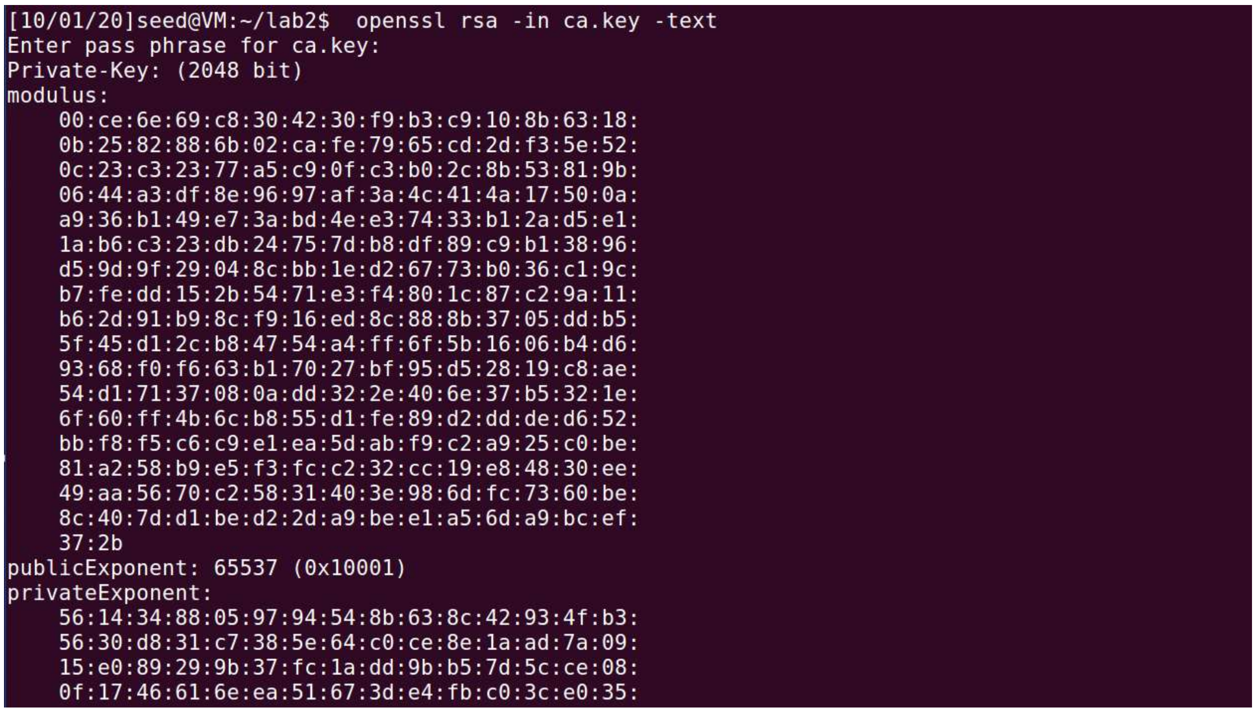
(1.3)



```
[10/01/20]seed@VM:~/lab2$  openssl x509 -text -noout -in ca.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            cd:7a:58:19:32:ee:7c:9a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=NH, L=HAN, O=TheCA, OU=TheCA, CN=TheCA/emailAddress=ca@ca.com
        Validity
            Not Before: Oct  1 20:25:12 2020 GMT
            Not After : Oct 31 20:25:12 2020 GMT
        Subject: C=US, ST=NH, L=HAN, O=TheCA, OU=TheCA, CN=TheCA/emailAddress=ca@ca.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ce:6e:69:c8:30:42:30:f9:b3:c9:10:8b:63:18:
                    0b:25:82:88:6b:02:ca:fe:79:65:cd:2d:f3:5e:52:
                    0c:23:c3:23:77:a5:c9:0f:c3:b0:2c:8b:53:81:9b:
                    06:44:a3:df:8e:96:97:af:3a:4c:41:4a:17:50:0a:
```

(1.4)

Here we can see that when running the first command in **(1.3)**, the public key is created for the certificate we created. The public-key **2048 bit** can be seen in **(1.4)**. The "**x509**" command used on the "**.cnf**" extension for the configuration file helps to generate this public-key certificate using the information we provided in **(1.2)**. We can also see the private-key output in **(1.5)**.



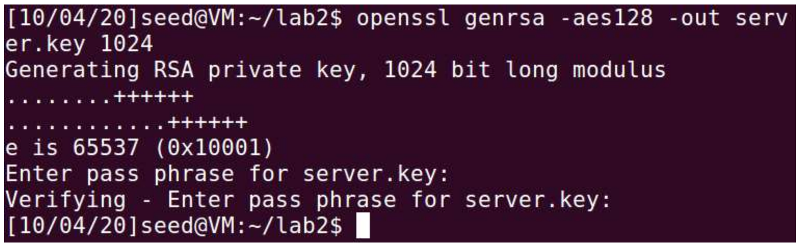
```
[10/01/20]seed@VM:~/lab2$  openssl rsa -in ca.key -text
Enter pass phrase for ca.key:
Private-Key: (2048 bit)
modulus:
    00:ce:6e:69:c8:30:42:30:f9:b3:c9:10:8b:63:18:
    0b:25:82:88:6b:02:ca:fe:79:65:cd:2d:f3:5e:52:
    0c:23:c3:23:77:a5:c9:0f:c3:b0:2c:8b:53:81:9b:
    06:44:a3:df:8e:96:97:af:3a:4c:41:4a:17:50:0a:
    a9:36:b1:49:e7:3a:bd:4e:e3:74:33:b1:2a:d5:e1:
    1a:b6:c3:23:db:24:75:7d:b8:df:89:c9:b1:38:96:
    d5:9d:9f:29:04:8c:bb:1e:d2:67:73:b0:36:c1:9c:
    b7:fe:dd:15:2b:54:71:e3:f4:80:1c:87:c2:9a:11:
    b6:2d:91:b9:8c:f9:16:ed:8c:88:8b:37:05:dd:b5:
    5f:45:d1:2c:b8:47:54:a4:ff:6f:5b:16:06:b4:d6:
    93:68:f0:f6:63:b1:70:27:bf:95:d5:28:19:c8:ae:
    54:d1:71:37:08:0a:dd:32:2e:40:6e:37:b5:32:1e:
    6f:60:ff:4b:6c:b8:55:d1:fe:89:d2:dd:de:d6:52:
    bb:f8:f5:c6:c9:e1:ea:5d:ab:f9:c2:a9:25:c0:be:
    81:a2:58:b9:e5:f3:fc:c2:32:cc:19:e8:48:30:ee:
    49:aa:56:70:c2:58:31:40:3e:98:6d:fc:73:60:be:
    8c:40:7d:d1:be:d2:2d:a9:be:e1:a5:6d:a9:bc:ef:
    37:2b
publicExponent: 65537 (0x10001)
privateExponent:
    56:14:34:88:05:97:94:54:8b:63:8c:42:93:4f:b3:
    56:30:d8:31:c7:38:5e:64:c0:ce:8e:1a:ad:7a:09:
    15:e0:89:29:9b:37:fc:1a:dd:9b:b5:7d:5c:ce:08:
    0f:17:46:61:6e:ea:51:67:3d:e4:fb:c0:3c:e0:35:
```

(1.5)

Here we can see that when running the second command in **(1.3)**, the private key is created for the certificate we created. Instead of using the "**x509**" for the configuration file extension, the "**rsa**" is used instead. Similar to how the key was shown in hex for the public key, we can also see the similar format in hex (**2048 bit**) shown in **(1.5)** for the private key certificate.

<u>Task 2 Creating a Certificate for SEEDPKILab2020.com:</u>

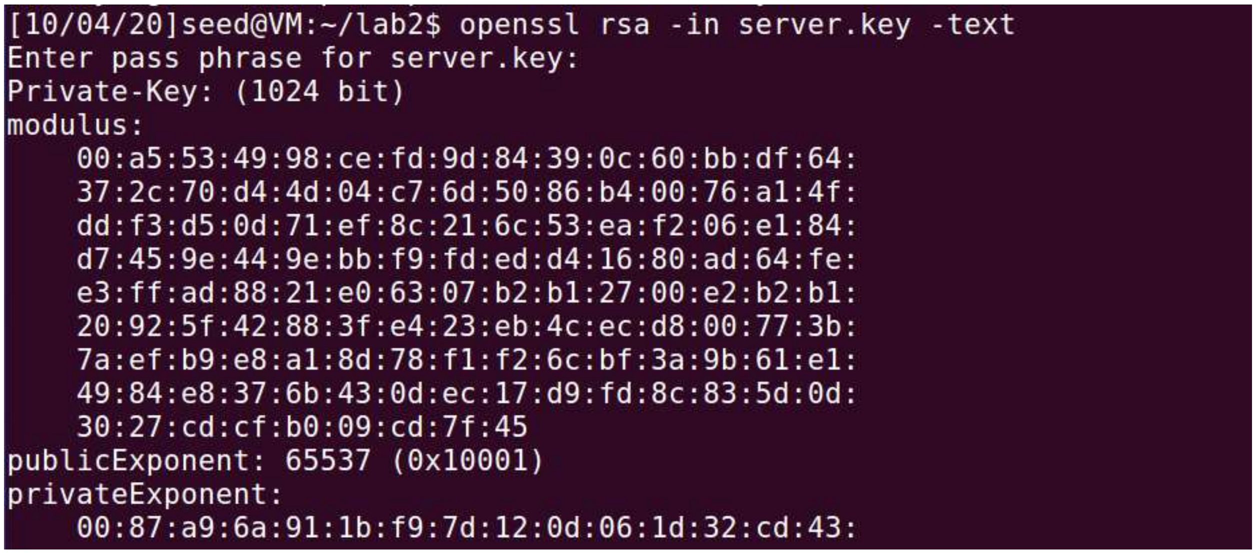
In order to get a digital certificate for ***SEEDPKILab2020.com***. and generate a public/private key for this company, the following command shown in **(2.0)** was used to do so.



```
[10/04/20]seed@VM:~/lab2$ openssl genrsa -aes128 -out serv
er.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:
[10/04/20]seed@VM:~/lab2$ 
```

(2.0)

To encrypt the private key, a password was needed, so one was generated (**Phidelt!22**). However, "**server.key**" is an encrypted text file, so in order to see the actual content (*modulus, private exponents, and etc*), the following command in **(2.1)** was used to do so.

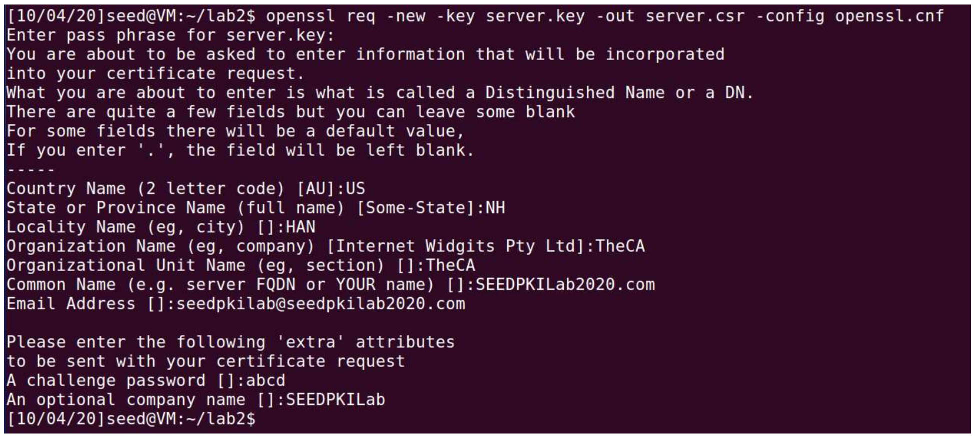


```
[10/04/20]seed@VM:~/lab2$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
    00:a5:53:49:98:ce:fd:9d:84:39:0c:60:bb:df:64:
    37:2c:70:d4:4d:04:c7:6d:50:86:b4:00:76:a1:4f:
    dd:f3:d5:0d:71:ef:8c:21:6c:53:ea:f2:06:e1:84:
    d7:45:9e:44:9e:bb:f9:fd:ed:d4:16:80:ad:64:fe:
    e3:ff:ad:88:21:e0:63:07:b2:b1:27:00:e2:b2:b1:
    20:92:5f:42:88:3f:e4:23:eb:4c:ec:d8:00:77:3b:
    7a:ef:b9:e8:a1:8d:78:f1:f2:6c:bf:3a:9b:61:e1:
    49:84:e8:37:6b:43:0d:ec:17:d9:fd:8c:83:5d:0d:
    30:27:cd:cf:b0:09:cd:7f:45
publicExponent: 65537 (0x10001)
privateExponent:
    00:87:a9:6a:91:1b:f9:7d:12:0d:06:1d:32:cd:43:
```

(2.1)

We can see the contents in the "**server.key**." They include the **private key** (**1024 bit**), **public exponent** (**65537: 0X10001**), **private exponent**, **prime1**, **prime2**, **exponent1**, **exponent2**, **coefficient**, in addition to the written-out **RSA Private Key**, which is a string of characters.

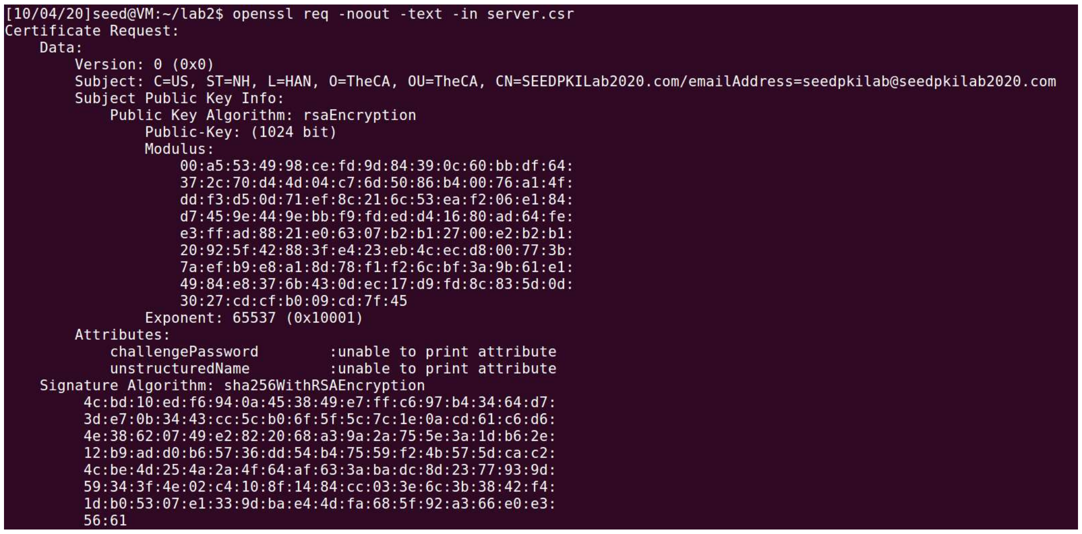
Now we need to generate a **CSR** (Certificate Signing Request) for the company. This includes the company's public key. The **CSR** is to be sent over to the **CA**, and the certificate for the key will be generated with ***SEEDPKLab2020.com*** being the common name for the **CSR**. Below **(2.2),** we can see the command to generate the **CSR**.



```
[10/04/20]seed@VM:~/lab2$ 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]:NH
Locality Name (eg, city) []:HAN
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TheCA
Organizational Unit Name (eg, section) []:TheCA
Common Name (e.g. server FQDN or YOUR name) []:SEEDPKILab2020.com
Email Address []:seedpkilab@seedpkilab2020.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:abcd
An optional company name []:SEEDPKILab
[10/04/20]seed@VM:~/lab2$
```

(2.2)

We can now see the contents of "**server.csr**" in **(2.3).**

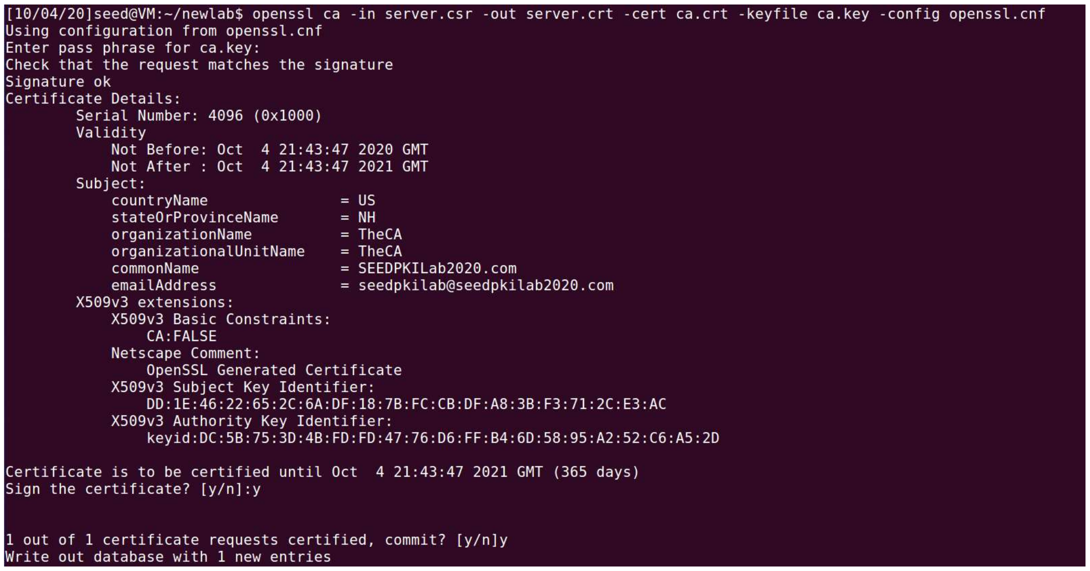


```
[10/04/20]seed@VM:~/lab2$ openssl req -noout -text -in server.csr
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=NH, L=HAN, O=TheCA, OU=TheCA, CN=SEEDPKILab2020.com/emailAddress=seedpkilab@seedpkilab2020.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:a5:53:49:98:ce:fd:9d:84:39:0c:60:bb:df:64:
                    37:2c:70:d4:4d:04:c7:6d:50:86:b4:00:76:a1:4f:
                    dd:f3:d5:0d:71:ef:8c:21:6c:53:ea:f2:06:e1:84:
                    d7:45:9e:44:9e:bb:f9:fd:ed:d4:16:80:ad:64:fe:
                    e3:ff:ad:88:21:e0:63:07:b2:b1:27:00:e2:b2:b1:
                    20:92:5f:42:88:3f:e4:23:eb:4c:ec:d8:00:77:3b:
                    7a:ef:b9:e8:a1:8d:78:f1:f2:6c:bf:3a:9b:61:e1:
                    49:84:e8:37:6b:43:0d:ec:17:d9:fd:8c:83:5d:0d:
                    30:27:cd:cf:b0:09:cd:7f:45
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :unable to print attribute
            unstructuredName         :unable to print attribute
    Signature Algorithm: sha256WithRSAEncryption
         4c:bd:10:ed:f6:94:0a:45:38:49:e7:ff:c6:97:b4:34:64:d7:
         3d:e7:0b:34:43:cc:5c:b0:6f:5f:5c:7c:1e:0a:cd:61:c6:d6:
         4e:38:62:07:49:e2:82:20:68:a3:9a:2a:75:5e:3a:1d:b6:2e:
         12:b9:ad:d0:b6:57:36:dd:54:b4:75:59:f2:4b:57:5d:ca:c2:
         4c:be:4d:25:4a:2a:4f:64:af:63:3a:ba:dc:8d:23:77:93:9d:
         59:34:3f:4e:02:c4:10:8f:14:84:cc:03:3e:6c:3b:38:42:f4:
         1d:b0:53:07:e1:33:9d:ba:e4:4d:fa:68:5f:92:a3:66:e0:e3:
         56:61
```

(2.3)

Here we can see that the contents in "**server.csr**" include the public key (**1024 bit**), which entails the modulus and the exponent. However, the **challengePassword** and the **unstructuredName** are not able to be shown when we run the command shown at the top of **(2.3).**

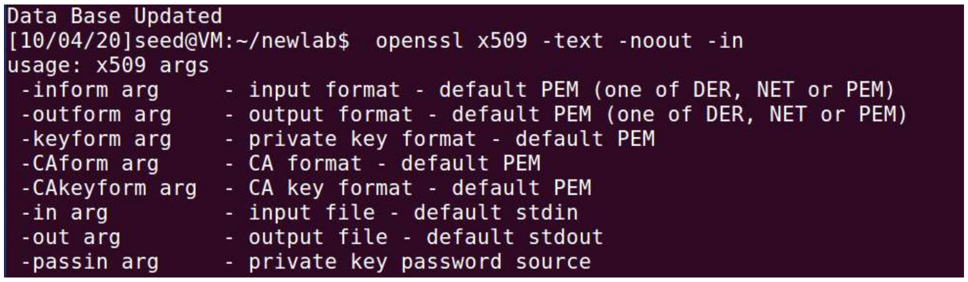
Now, the **CSR** file needs the actual **CA** signature to become a valid certificate. In order to turn the "**server.csr**" into an **X509** certificate, we can use the following command "*openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf*" and see the prompt to sign the certificate **(2.4)**.



```
[10/04/20]seed@VM:~/newlab$ openssl ca -in server.csr -out 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: 4096 (0x1000)
        Validity
            Not Before: Oct  4 21:43:47 2020 GMT
            Not After : Oct  4 21:43:47 2021 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = NH
            organizationName          = TheCA
            organizationalUnitName    = TheCA
            commonName                = SEEDPKILab2020.com
            emailAddress              = seedpkilab@seedpkilab2020.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                DD:1E:46:22:65:2C:6A:DF:18:7B:FC:CB:DF:A8:3B:F3:71:2C:E3:AC
            X509v3 Authority Key Identifier:
                keyid:DC:5B:75:3D:4B:FD:FD:47:76:D6:FF:B4:6D:58:95:A2:52:C6:A5:2D

Certificate is to be certified until Oct  4 21:43:47 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
```

(2.4)

Afterwards, we can see the certificate output below **(2.5).**



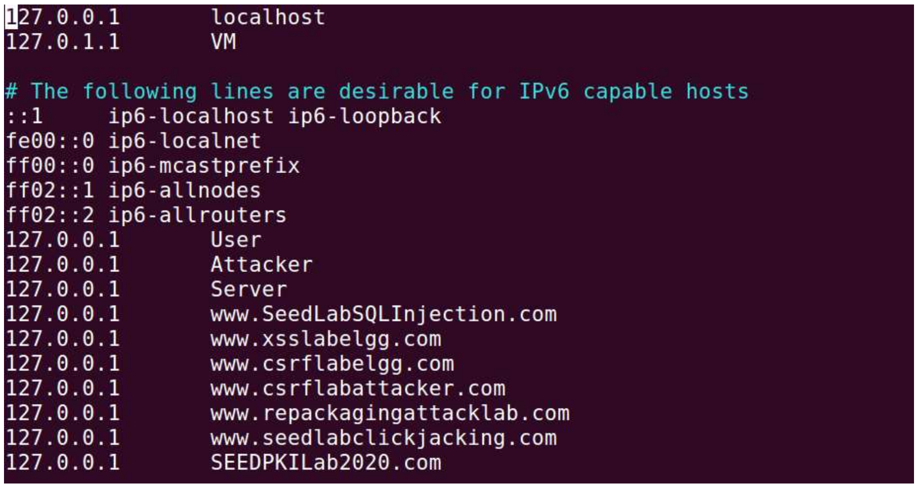
```
Data Base Updated
[10/04/20]seed@VM:~/newlab$  openssl x509 -text -noout -in
usage: x509 args
 -inform arg     - input format - default PEM (one of DER, NET or PEM)
 -outform arg    - output format - default PEM (one of DER, NET or PEM)
 -keyform arg    - private key format - default PEM
 -CAform arg     - CA format - default PEM
 -CAkeyform arg  - CA key format - default PEM
 -in arg         - input file - default stdin
 -out arg        - output file - default stdout
 -passin arg     - private key password source
```

(2.5)

Here we can see the output for generating the **CA** signature for the certificate. If we look closely, we can see the usage, which then goes into detail about the different types of arguments.

<u>Task 3 Deploying Certificate in an HTTPS Web Server</u>:

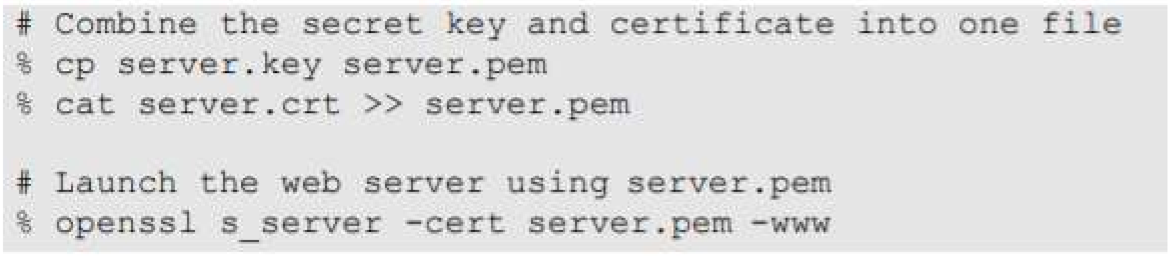
In order to choose **SEEDPKILab2020.com** as the name of the website, we can get the computer to recognize the name by using the following command "*sudo vi /etc/hosts*" and then adding it to the doc as **127.0.0.1 SEEDPKILab2020.com** as shown below **(3.0).**



```
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       SEEDPKILab2020.com
```

(3.0)

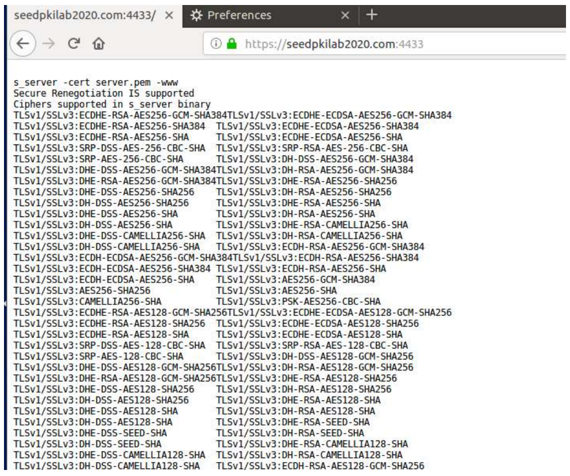
Now we can start a simple web server using the following commands below **(3.1).**



```
# Combine the secret key and certificate into one file
% cp server.key server.pem
% cat server.crt >> server.pem

# Launch the web server using server.pem
% openssl s_server -cert server.pem -www
```

(3.1)

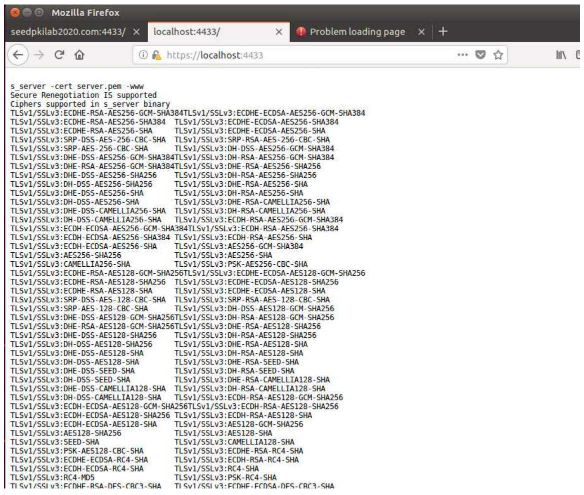
After these we can see the website when logging onto “**https://SEEDPKILab2020.com:4433/**.” We see below what the website looks like **(3.2).**



```
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
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA256  TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA256
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA     TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA
TLSv1/SSLv3:SRP-DSS-AES-128-CBC-SHA  TLSv1/SSLv3:SRP-RSA-AES-128-CBC-SHA
TLSv1/SSLv3:SRP-AES-128-CBC-SHA      TLSv1/SSLv3:DH-DSS-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-GCM-SHA256TLSv1/SSLv3:DH-RSA-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-RSA-AES128-GCM-SHA256TLSv1/SSLv3:DHE-RSA-AES128-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-SHA256    TLSv1/SSLv3:DH-RSA-AES128-SHA256
TLSv1/SSLv3:DH-DSS-AES128-SHA256     TLSv1/SSLv3:DHE-RSA-AES128-SHA
TLSv1/SSLv3:DHE-DSS-AES128-SHA       TLSv1/SSLv3:DH-RSA-AES128-SHA
TLSv1/SSLv3:DH-DSS-AES128-SHA        TLSv1/SSLv3:DHE-RSA-SEED-SHA
TLSv1/SSLv3:DHE-DSS-SEED-SHA         TLSv1/SSLv3:DH-RSA-SEED-SHA
TLSv1/SSLv3:DH-DSS-SEED-SHA          TLSv1/SSLv3:DHE-RSA-CAMELLIA128-SHA
TLSv1/SSLv3:DHE-DSS-CAMELLIA128-SHA  TLSv1/SSLv3:DH-RSA-CAMELLIA128-SHA
TLSv1/SSLv3:DH-DSS-CAMELLIA128-SHA   TLSv1/SSLv3:ECDH-RSA-AES128-GCM-SHA256
```

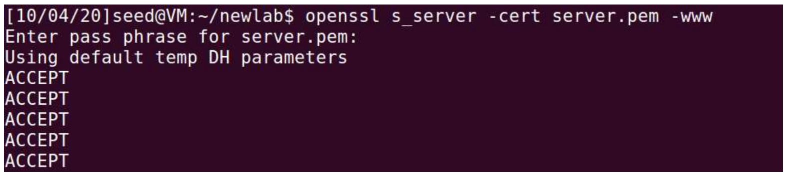
(3.2)

Here we can see that the website does exist; however, it is not quite easy to tell what exactly is on the website. In the terminal, we can see that the connection has been **ACCEPT**ed when going on to this link. We also see on the website that there is some discussion regarding Ciphers and how they are supported in **s_server** binary. As for when we use “**https://localhost:4433**” we get the same exact output as **(3.2)** shown in **(3.3).** Similarly, in the terminal, we can still see the **ACCEPT** for the connection to the server **(3.4)**. One can say that the reason the same webpage showed is because both have the same webpage since the “**https://SEEDPKILab2020.com**” points to the localhost.



```
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
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA256  TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA256
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA     TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA
TLSv1/SSLv3:SRP-DSS-AES-128-CBC-SHA  TLSv1/SSLv3:SRP-RSA-AES-128-CBC-SHA
TLSv1/SSLv3:SRP-AES-128-CBC-SHA      TLSv1/SSLv3:DH-DSS-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-GCM-SHA256TLSv1/SSLv3:DH-RSA-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-RSA-AES128-GCM-SHA256TLSv1/SSLv3:DHE-RSA-AES128-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-SHA256    TLSv1/SSLv3:DH-RSA-AES128-SHA256
TLSv1/SSLv3:DH-DSS-AES128-SHA256     TLSv1/SSLv3:DHE-RSA-AES128-SHA
TLSv1/SSLv3:DHE-DSS-AES128-SHA       TLSv1/SSLv3:DH-RSA-AES128-SHA
TLSv1/SSLv3:DH-DSS-AES128-SHA        TLSv1/SSLv3:DHE-RSA-SEED-SHA
TLSv1/SSLv3:DHE-DSS-SEED-SHA         TLSv1/SSLv3:DH-RSA-SEED-SHA
TLSv1/SSLv3:DH-DSS-SEED-SHA          TLSv1/SSLv3:DHE-RSA-CAMELLIA128-SHA
TLSv1/SSLv3:DHE-DSS-CAMELLIA128-SHA  TLSv1/SSLv3:DH-RSA-CAMELLIA128-SHA
TLSv1/SSLv3:DH-DSS-CAMELLIA128-SHA   TLSv1/SSLv3:ECDH-RSA-AES128-GCM-SHA256
TLSv1/SSLv3:ECDH-ECDSA-AES128-GCM-SHA256TLSv1/SSLv3:ECDH-RSA-AES128-SHA256
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA256 TLSv1/SSLv3:ECDH-RSA-AES128-SHA
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA    TLSv1/SSLv3:AES128-GCM-SHA256
TLSv1/SSLv3:AES128-SHA256            TLSv1/SSLv3:AES128-SHA
TLSv1/SSLv3:SEED-SHA                 TLSv1/SSLv3:CAMELLIA128-SHA
TLSv1/SSLv3:PSK-AES128-CBC-SHA       TLSv1/SSLv3:ECDHE-RSA-RC4-SHA
TLSv1/SSLv3:ECDHE-ECDSA-RC4-SHA      TLSv1/SSLv3:ECDH-RSA-RC4-SHA
TLSv1/SSLv3:ECDH-ECDSA-RC4-SHA       TLSv1/SSLv3:RC4-SHA
TLSv1/SSLv3:RC4-MD5                  TLSv1/SSLv3:PSK-RC4-SHA
TLSv1/SSLv3:ECDHE-RSA-DES-CBC3-SHA   TLSv1/SSLv3:ECDHE-ECDSA-DES-CBC3-SHA
```

(3.3)

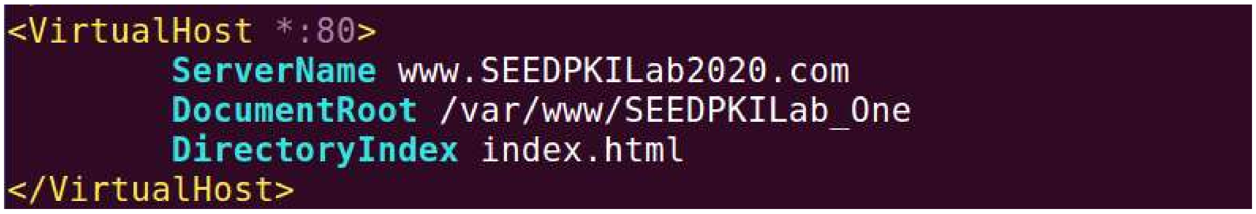


```
[10/04/20]seed@VM:~/newlab$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
ACCEPT
ACCEPT
ACCEPT
ACCEPT
```

(3.4)

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

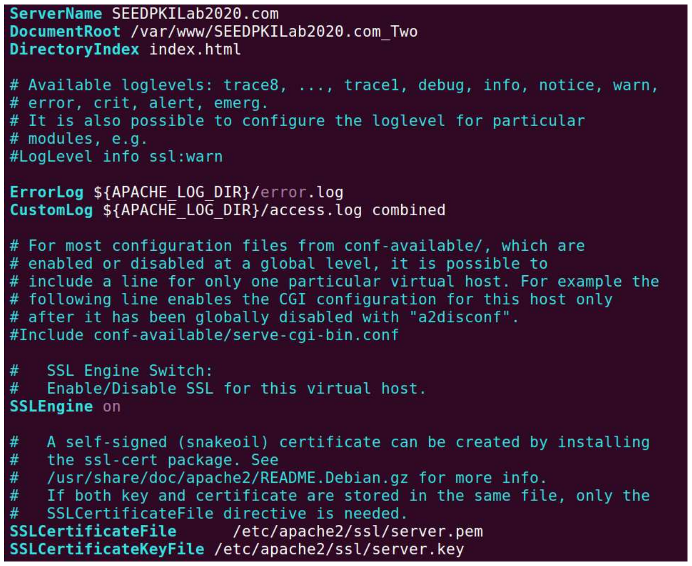
First, we must add the **HTTP** website to the **VirtualHost** file, which is located in the */etc/apache2/sites-available* directory. By using the command "**sudo vi /etc/apache2/sites-available/000-default.conf**" we can edit the file as shown below (4.0).



```
<VirtualHost *:80>
        ServerName www.SEEDPKILab2020.com
        DocumentRoot /var/www/SEEDPKILab_One
        DirectoryIndex index.html
</VirtualHost>
```

(4.0)

However, we also must add a **VirtualHost** entry to the *default-ssl.conf* file. We can see that below **(4.1).**

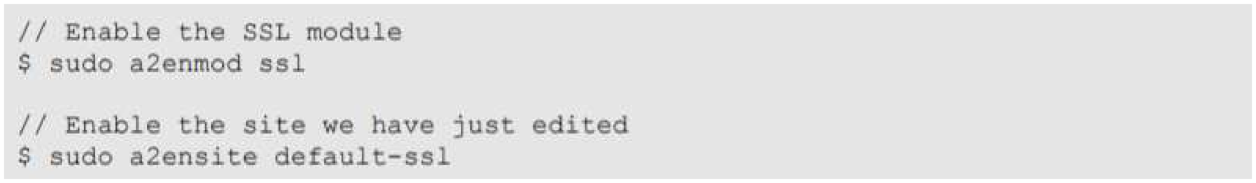


```
ServerName SEEDPKILab2020.com
DocumentRoot /var/www/SEEDPKILab2020.com_Two
DirectoryIndex index.html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf

#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   A self-signed (snakeoil) certificate can be created by installing
#   the ssl-cert package. See
#   /usr/share/doc/apache2/README.Debian.gz for more info.
#   If both key and certificate are stored in the same file, only the
#   SSLCertificateFile directive is needed.
SSLCertificateFile      /etc/apache2/ssl/server.pem
SSLCertificateKeyFile /etc/apache2/ssl/server.key
```

(4.1)

Afterwards, it was important to first use the **sudo** command to create the directory "*/etc/apache2/ssl*." Once that directory was created, "**server.pem**" and "**server.key**" needed to be added into the "*/etc/apache2/ssl/*" directory. That now allows the **SSLCertificateFile** and the **SSLCertificateKeyFile** in (4.1) to exist in the directory we had put in the "*default-ssl.conf* file."

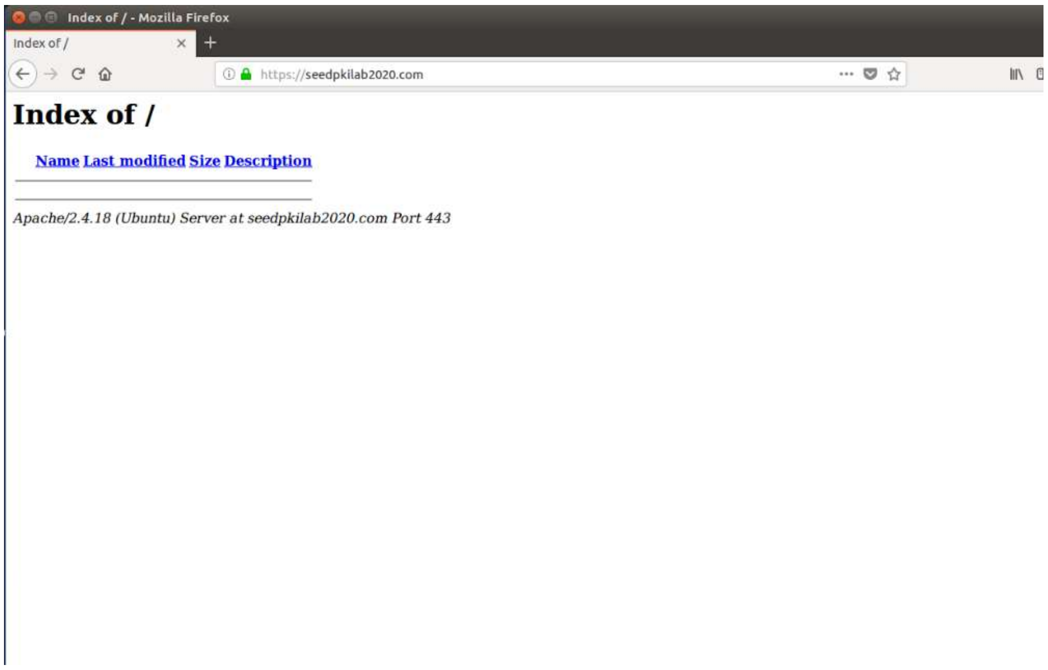
In order to not create any missing directory errors or warnings, the two directories that were mentioned in both .conf files in the "*/etc/apache2/sites-available/*" directory were created using the **sudo** command **(SEEDPKILab2020_One, SEEDPKILab2020.com_Two)**. Before testing the apache configuration file for errors, we used "**clear**" command to create a blank slate for testing and removing any previous commands in progress or that may interfere with testing. Then we tested the apache configuration file for errors. Most of the errors were relating to missing directories, and to resolve those issues the "**sudo mkdir**" command was quite useful (used carefully as well not to overuse this power). Then the remaining commands were used to help enable up the **SSL** module and the website we just edited respectively shown in **(4.2).**



```
// Enable the SSL module
$ sudo a2enmod ssl

// Enable the site we have just edited
$ sudo a2ensite default-ssl
```

(4.2)

Lastly, the Apache was restarted. Then to test the website, the website "*https://SEEDPKILab2020.com*" was entered into the search bar, and we can see the results in **(4.3).** We can see that the website is a blank .html file (**index.html**) because we have not edited or designed a specific website except for a blank one. Therefore, we see as shown in **(4.3. pictured below (next page))** an index.html for our website, and we can browse this website freely.



Index of /

Name Last modified Size Description

Apache/2.4.18 (Ubuntu) Server at seedpkilab2020.com Port 443

(4.3)

<u>Closing Thoughts:</u>

I found this lab to be very interesting especially in the aspect of adding a VirtualHost while being able to manipulate configuration files. It was very rewarding to see the website and being able to browse it freely at the end as opposed to near the beginning of the lab where we had to run a command to allow the website to even work with the addition :4433 at the end; of course that was done to make something appear on the website. I have always had an interest specifically regarding web design but seeing how exactly we can use Public Key Infrastructure to protect websites, but also how vulnerable they may exactly be is much more interesting and intellectually rewarding than coding tags.