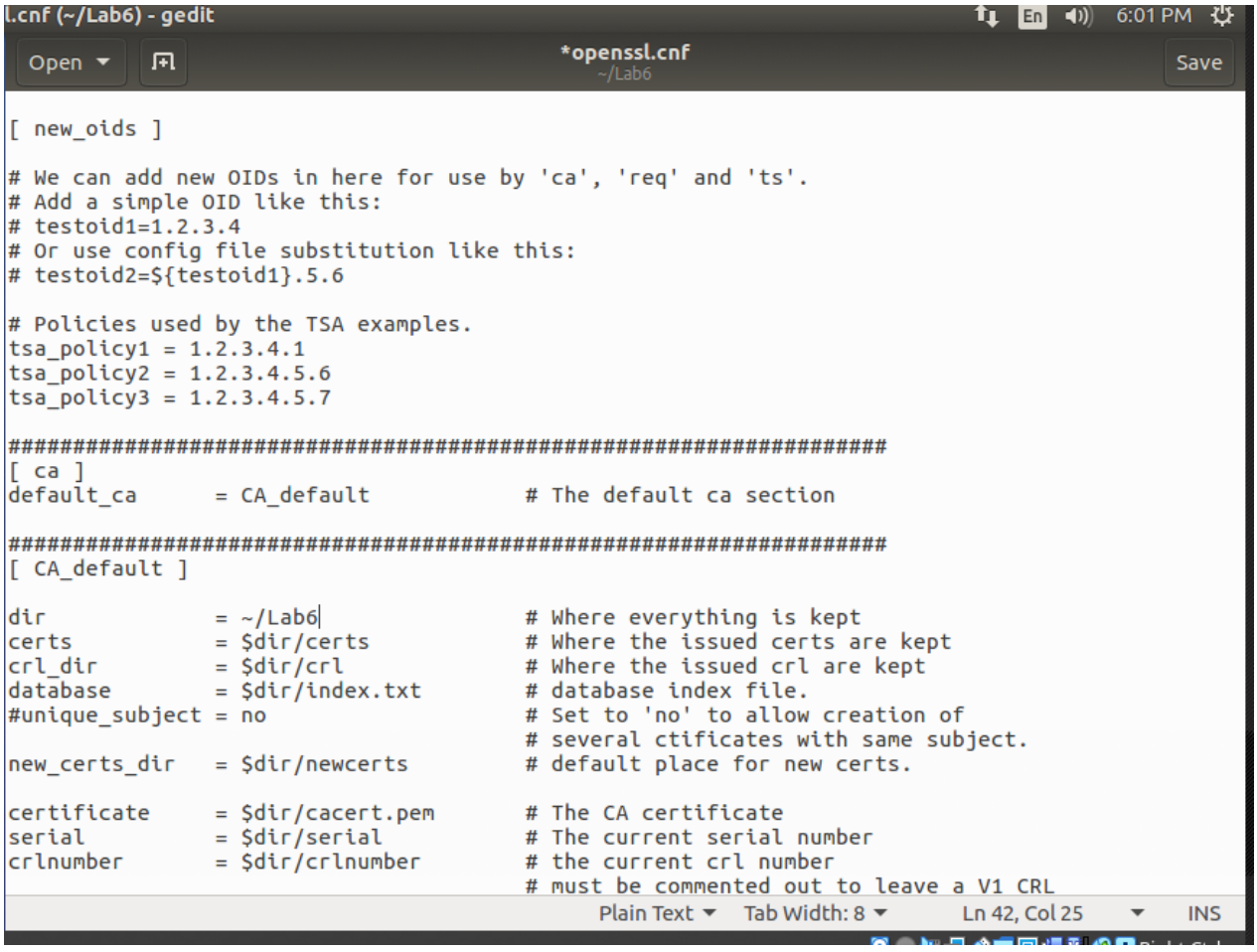
Lab 6 – Kevin Martin

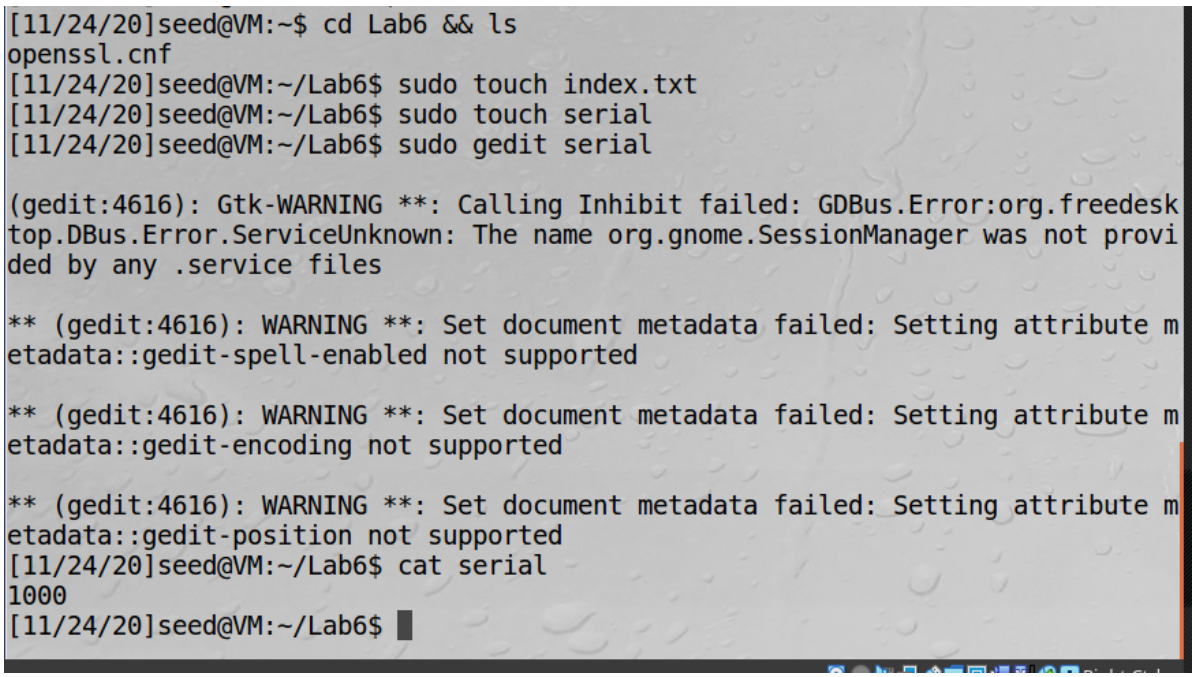
Task 1 – Becoming a Certificate Authority (CA)

To become a CA, we must first copy a the conf template file into the machine we will be using, VM A in my case. Then, edit the file to show current directory:

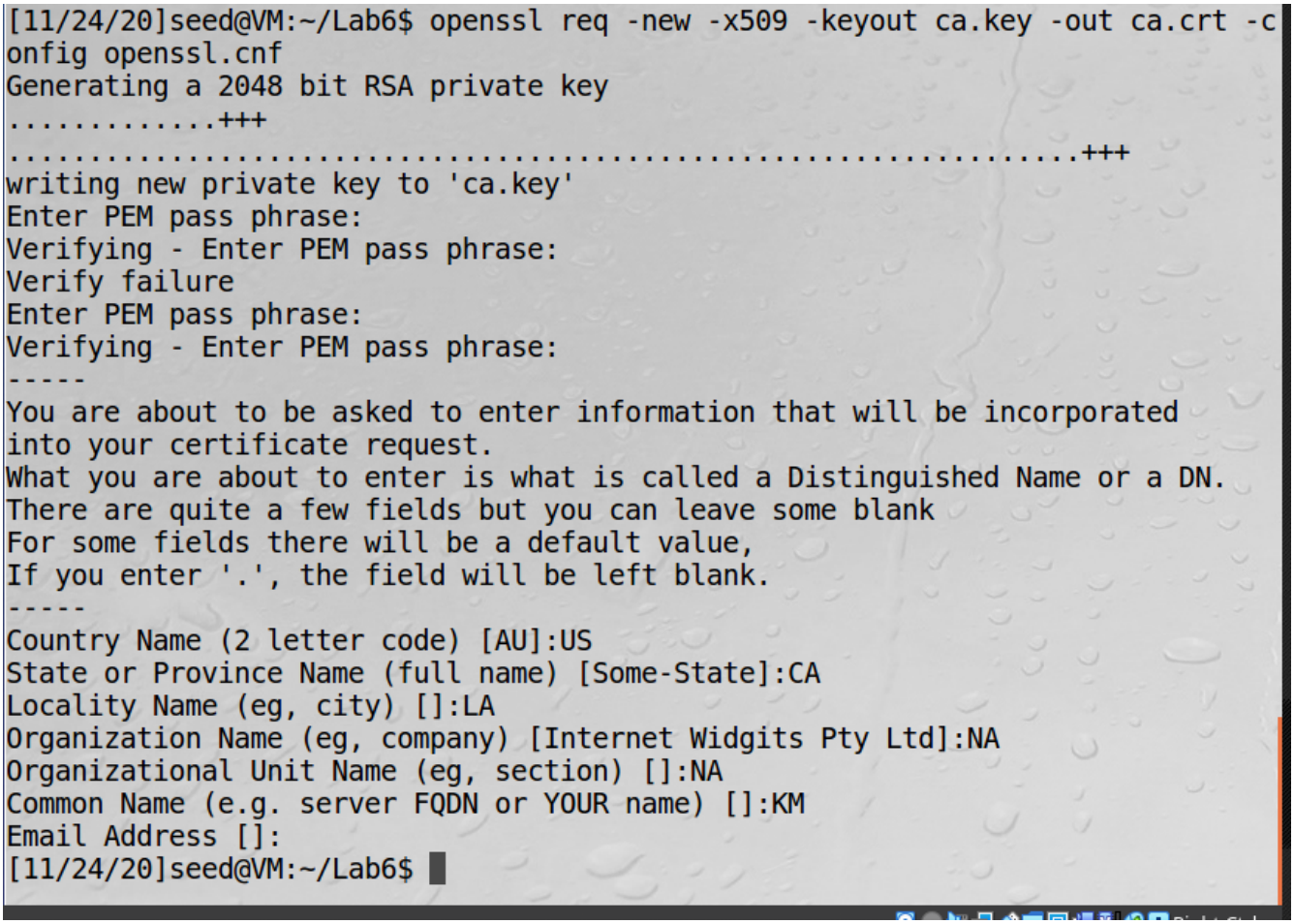
```
[11/24/20]seed@VM:.../ssl$ cp ./openssl.cnf ~/Lab6/openssl.conf
[11/24/20]seed@VM:.../ssl$
```



Next the required files, an empty **index.txt** and a **serial** file with the number 1000:



Finally, the self-signed certificate:



Observation: We used openssl to become a CA by utilizing the stock template and then modifying for our cusom configuration. The openssl command provides all the components necessary to create a new 509 certificate so long as the user enters in the all the proper identifying data.

Explanation: To utilize the PKI encryption system, a working certificate is needed. Instead of utlizing a third party, we can generate one ourselves and continue on with the encryption process.

Task 2 – Creating a Certificate for SEEDPKILab2020.com

Next we generate the RSA public and private keys. We will use the same password as the previously generated certificate:

```
[11/24/20]seed@VM:~/Lab6$ 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:
[11/24/20]seed@VM:~/Lab6$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
    00:d6:40:5d:35:dd:60:da:e5:83:f8:1f:57:06:d6:
    af:20:43:47:3a:58:44:fd:30:b8:26:10:f7:28:06:
    39:e8:92:cb:a0:6e:da:c6:e5:2a:3c:63:22:66:b6:
    8f:b7:a5:20:56:4a:f1:33:25:68:83:0d:ff:3f:6e:
    a9:24:3e:5e:9f:68:2e:fe:66:13:b3:64:c4:a3:e0:
    69:b9:de:5c:4d:3a:cb:84:9c:ea:31:7a:4a:88:47:
    7b:fb:fd:bd:95:9e:a1:8e:86:4b:12:68:06:35:2f:
    44:1d:b4:23:63:62:57:29:6a:3a:a3:c0:8e:aa:ec:
    ea:75:58:56:7b:0d:d6:f4:15
publicExponent: 65537 (0x10001)
privateExponent:
    0e:a1:28:7c:34:18:97:f9:d2:10:10:3b:c6:08:c4:
    be:84:39:89:df:5b:1e:9a:c4:08:3d:62:53:06:51:
    cf:f2:ae:1a:cf:56:fa:fb:cc:ed:80:ab:69:ae:0f:
    94:2f:9f:4c:a5:23:61:25:ad:c4:3d:21:6f:b8:63:
    ae:48:df:f7:0c:8c:85:ad:83:a9:56:51:28:53:f1:
    4a:12:e6:b0:20:a4:c3:9b:1e:d2:d4:1c:b9:cf:fe:
    34:d0:a8:31:4b:ff:41:be:55:0a:4e:14:f2:2a:fe:
    08:37:3d:73:a3:18:84:d3:e0:8a:bf:8b:dd:2b:2a:
    02:19:47:6c:20:dd:a0:a1
prime1:
    00:f0:75:6d:f1:59:b6:fc:82:25:b3:ab:7d:e4:3c:
    02:0f:32:c6:ec:47:80:ee:55:95:18:35:8b:58:02:
    94:81:99:60:16:56:cd:89:09:95:af:5d:ac:2e:6b:
    6b:ce:0c:32:52:97:40:8c:98:ee:6c:6d:38:04:ac:
    b8:ae:85:c4:8d
```

```
prime2:
    00:e4:19:50:b9:c1:6e:6b:e4:85:c7:0d:2d:b8:c0:
    a4:73:74:57:de:60:55:30:f7:6a:04:e2:48:35:c6:
    ca:a4:4b:4c:b9:22:48:b2:68:8e:cc:7d:0e:8a:07:
    34:96:66:8b:10:3c:c7:b5:c8:bb:20:99:c5:4a:6c:
    4b:c4:2f:bf:a9
exponent1:
    00:b1:fd:4a:6f:6f:88:2a:ab:2f:bb:2a:02:da:f1:
    ac:58:91:ae:7b:71:86:37:65:34:22:a2:67:cd:b2:
    c3:38:97:4e:6f:da:ba:f3:68:1b:db:9e:4b:f8:64:
    5d:7d:3c:bb:f7:34:ac:7c:26:2b:be:28:ae:61:8f:
    74:79:02:dd:75
exponent2:
    5f:fd:b7:57:34:eb:ff:43:c0:78:b0:37:19:95:18:
    9f:ed:ca:3c:af:55:aa:b1:b4:50:31:da:29:a7:6b:
    08:11:18:4c:a0:fc:ee:f7:c9:80:8c:f0:5d:6e:02:
    d7:78:77:f5:71:5e:aa:45:bb:5a:50:dc:12:21:d9:
    10:ca:f7:61
coefficient:
    7c:f6:61:2a:df:13:0d:19:9a:6b:51:30:d6:6c:c1:
    b0:8d:bb:d5:a8:6e:e6:49:49:bc:43:87:6c:c3:c9:
    2e:39:5f:1d:20:1b:b3:9f:20:06:69:bf:a1:7e:3f:
    86:ef:e3:43:f8:f9:02:f4:5a:47:ea:ab:63:d8:e5:
    a8:de:65:91
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDWQF013WDa5YP4H1cG1q8gQ0c6WET9MLgmEPcoBjnoksugbtrG
5So8YyJmto+3pSBWSvEzJWiDDf8/bqkkPl6faC7+ZhOzZMSj4Gm53lxNOsuEnOox
ekqIR3v7/b2VnqGOhksSaAY1L0QdtCNjYlcpajqjwI6q7Op1WFZ7Ddb0FQIDAQAB
AoGADqEofDQYl/nSEBA7xgjEvoQ5id9bHprECD1iUwZRz/KuGs9W+vvM7YCraa4P
lC+fTKUjYSWtxD0hb7hjrkjf9wyMha2DqVZRKFPxShLmsCCkw5se0tQcuc/+NNCo
MUv/Qb5VCk4U8ir+CDc9c6MYhNPgir+L3SsqAhlHbCDdoKECQQDwdW3xWbb8giWz
q33kPAIPMsbsR4DuVZUYNYtYApSBmWAWVs2JCZWvXawua2vODDJSl0CMmO5sbTgE
rLiuhcSNAkEA5BlQucFua+SFxw0tuMCkc3RX3mBVMPdqBOJINcbKpEtMuSJIsmiO
zH0Oigc0lmaLEDzHtci7IJnFSmxLxC+/qQJBALH9Sm9viCqrL7sqAtrxrFiRrntx
hjdlNCKiZ82ywziXTm/auvNoG9ueS/hkXX08u/c0rHwmK74ormGPdHkC3XUCQF/9
t1c06/9DwHiwNxmVGJ/tyjyvVaqxtFAx2imnawgRGEyg/O73yYCM8F1uAtd4d/Vx
XqpFu1pQ3BIh2RDK92ECQHz2YSrfEw0ZmmtRMNZswbCNu9WobuZJSbxDh2zDyS45
Xx0gG7OfIAZpv6F+P4bv40P4+QL0Wkfqq2PY5ajeZZE=
-----END RSA PRIVATE KEY-----
[11/24/20]seed@VM:~/Lab6$
```

The second step is to generate the certificate signing request (CSR) with the common name specified:

```
[11/24/20]seed@VM:~/Lab6$ openssl req -new -key server.key -out server.csr -con
fig 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]:CA
Locality Name (eg, city) []:LA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NA
Organizational Unit Name (eg, section) []:NA
Common Name (e.g. server FQDN or YOUR name) []:SEEDPKILab2020.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:dees
An optional company name []:
[11/24/20]seed@VM:~/Lab6$
```

In the last step, we can actually generate the certificate. However, I had an error with the standard settings so I modified the **openssl.cnf** document to allow for “policy = policy_anything”:

```
Open    openssl.cnf  ~/Lab6  Save
default_md      = default               # use public key default MD
preserve        = no                    # keep passed DN ordering

# 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

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

Unfortunately, I am still getting an error where the openssl program cannot access a “newcerts” directory. I created one manually and still see the error:

```
[11/24/20]seed@VM:~/Lab6$ openssl ca -in server.csr -out server.crt -cert ca.cr
t -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
I am unable to access the ~/Lab6/newcerts directory
~/Lab6/newcerts: No such file or directory
[11/24/20]seed@VM:~/Lab6$ mkdir newcerts
[11/24/20]seed@VM:~/Lab6$ openssl ca -in server.csr -out server.crt -cert ca.cr
t -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
I am unable to access the ~/Lab6/newcerts directory
~/Lab6/newcerts: No such file or directory
[11/24/20]seed@VM:~/Lab6$ ls
ca.crt  index.txt  openssl.cnf  server.csr
ca.key  newcerts   serial       server.key
[11/24/20]seed@VM:~/Lab6$
```

I finally realized that my main directory was incorrect, so after changing that I could create the certificate:

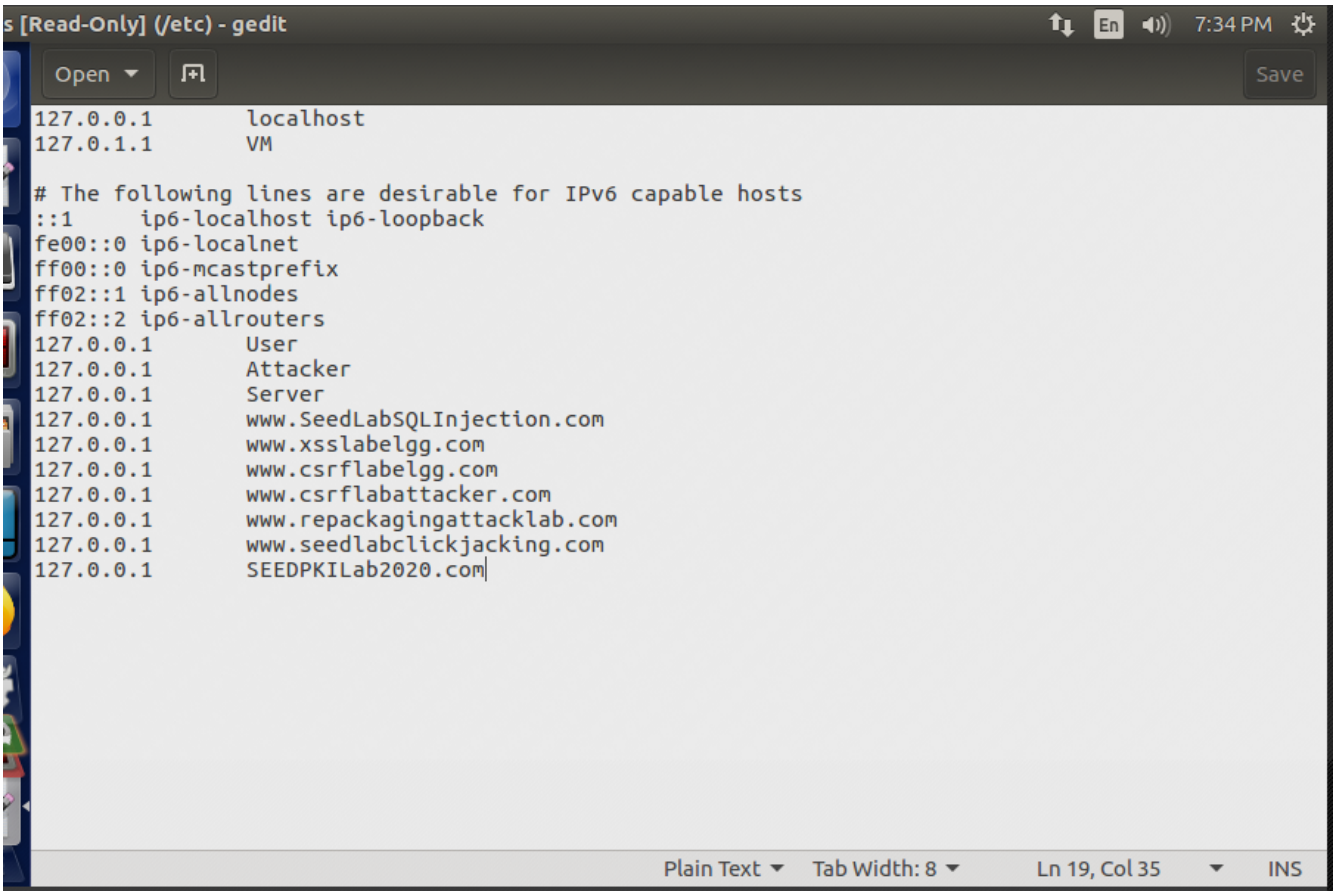
```
/bin/bash 79x25
            countryName               = US
            stateOrProvinceName       = CA
            localityName              = LA
            organizationName          = NA
            organizationalUnitName    = NA
            commonName                = SEEDPKILab2020.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                99:D8:30:D3:06:A6:0B:22:27:16:86:B4:41:0E:18:63:FE:A5:F9:C1
            X509v3 Authority Key Identifier:
                keyid:00:77:F4:21:93:05:B7:B8:6F:54:93:E4:E6:C9:85:D5:16:08:55:
39

Certificate is to be certified until Nov 25 00:18:06 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
[11/24/20]seed@VM:~/Lab6$
```

Observation: Similar to the first task, we were using openssl to create a certificate. We established ourselves as an authority first, and then recreated the events for a new “client”. This is now a valid certificate for the PKI process.

Explanation: In order to create a valid certificate, there are three requirements from the previously established CA: first, generate the public and private keys, then create a certificate signing request with the **public key**, and finally issuing the certificate.

Task 3 – Deploying Certificate in an HTTPS Web Server

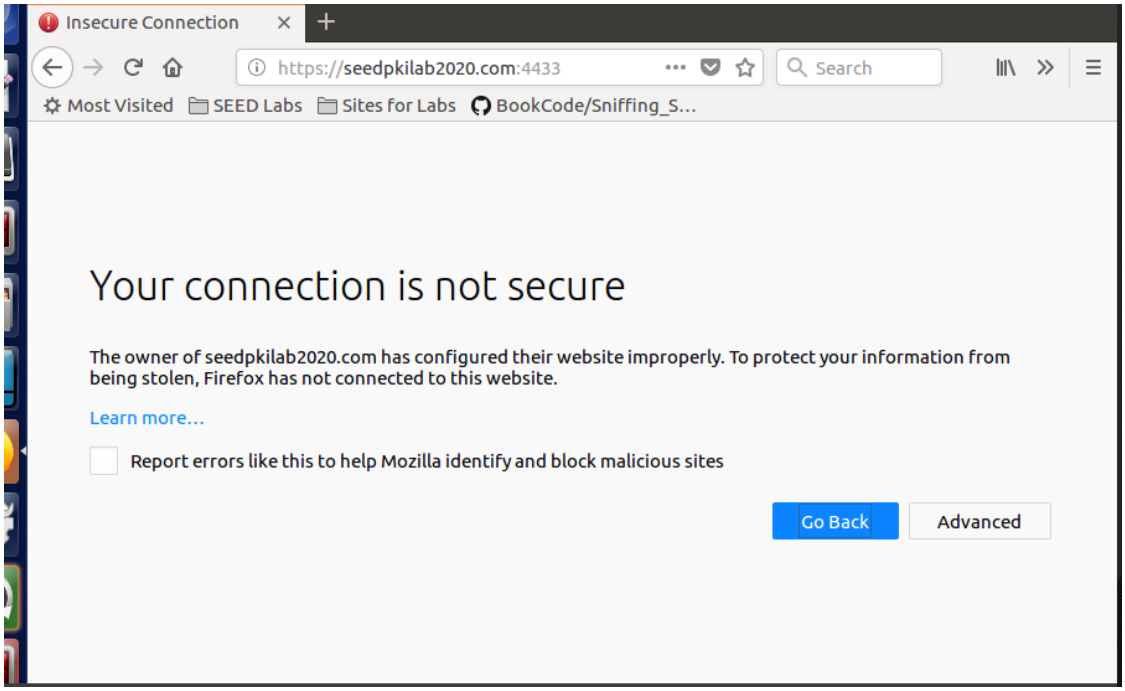
To actually use the certificate to securely browse the web, we must first add the domain to the recognized host list, which is in /etc/hosts (it shows as read only in the screenshot but I fixed it after):



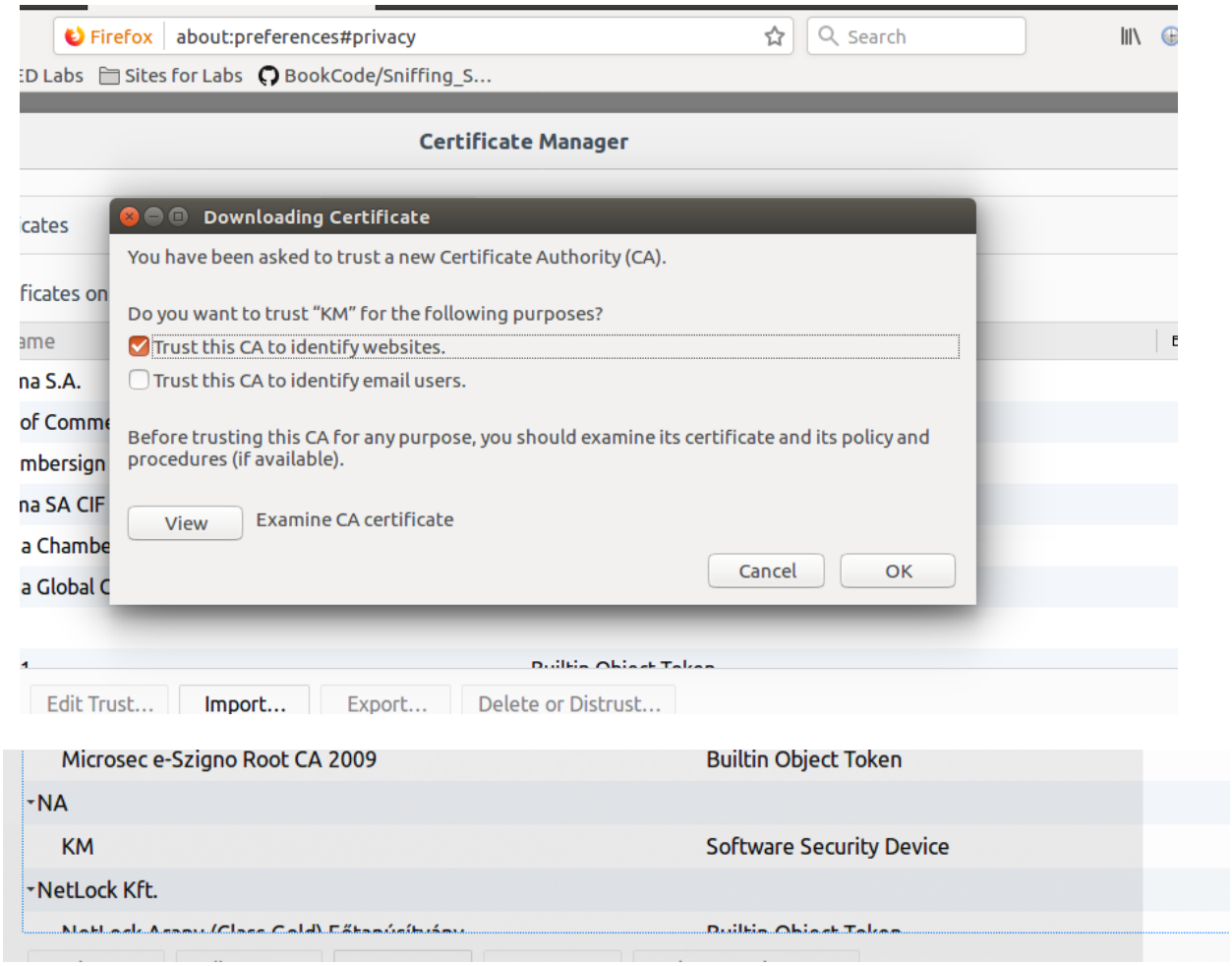
Next we will combine the secret key and certificate into one file and launch the web server:

```
[11/24/20]seed@VM:~$ cd Lab6
[11/24/20]seed@VM:~/Lab6$ cp server.key server.pem
[11/24/20]seed@VM:~/Lab6$ cat server.crt >> server.pem
[11/24/20]seed@VM:~/Lab6$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

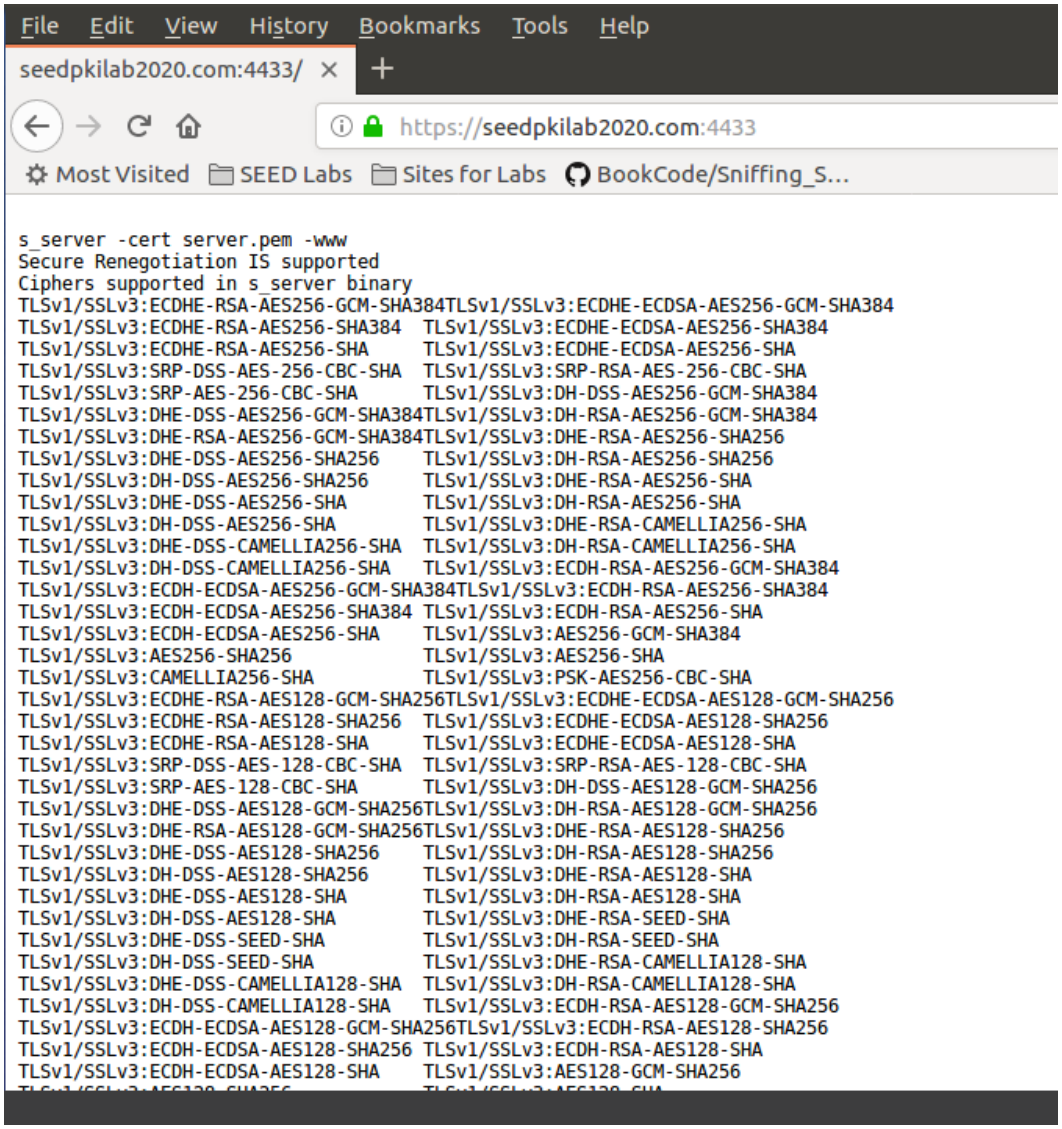
Before enabling our certificate, we get an error from the browser:



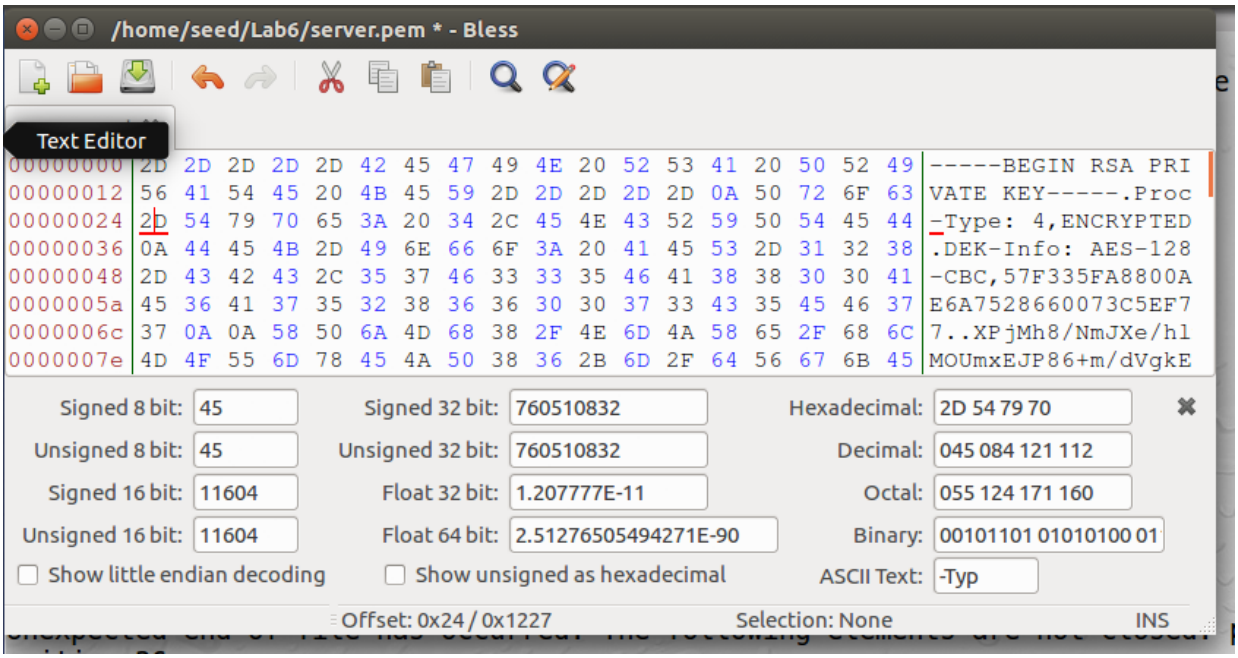
To fix, we enable **ca.crt** in Firefox:



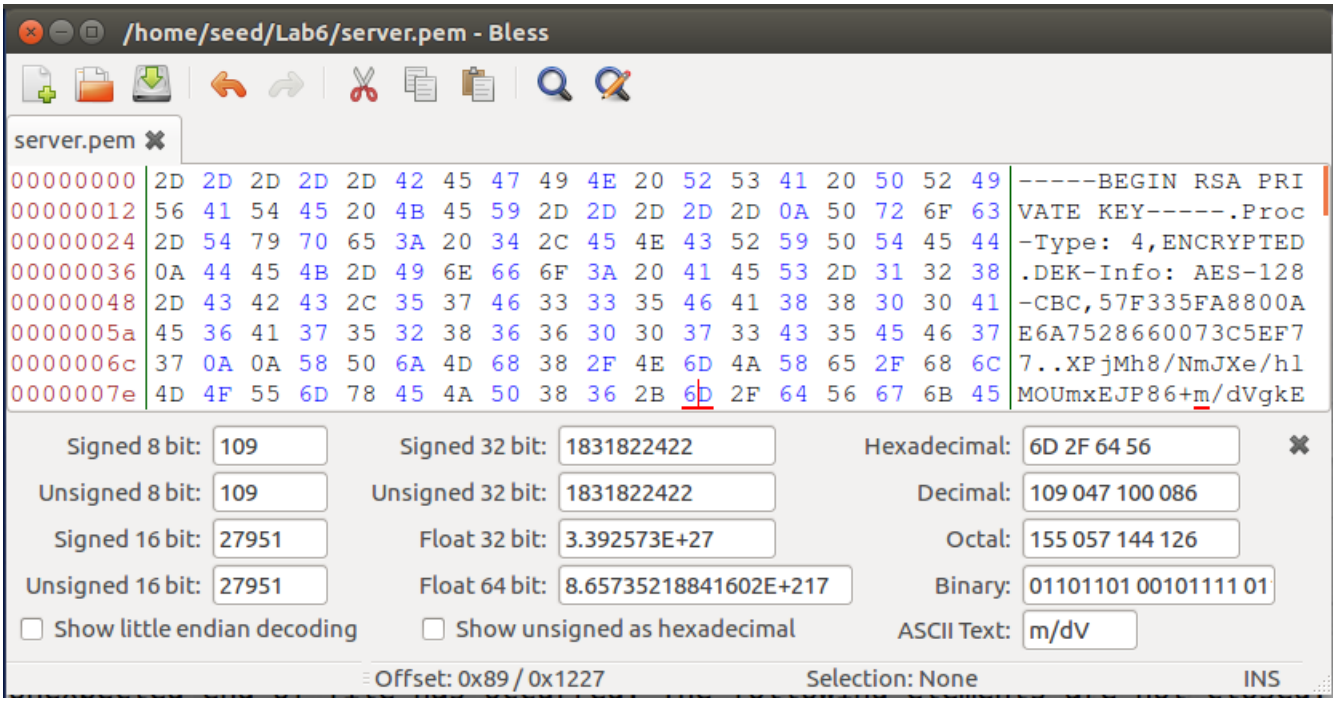
Finally, we can see the website:



To test the HTTPS website, we will modify a single byte of **server.pem**, restart the server, and reload the URL. First, the modified bit, going from “2D” to “3D” using **bless**:



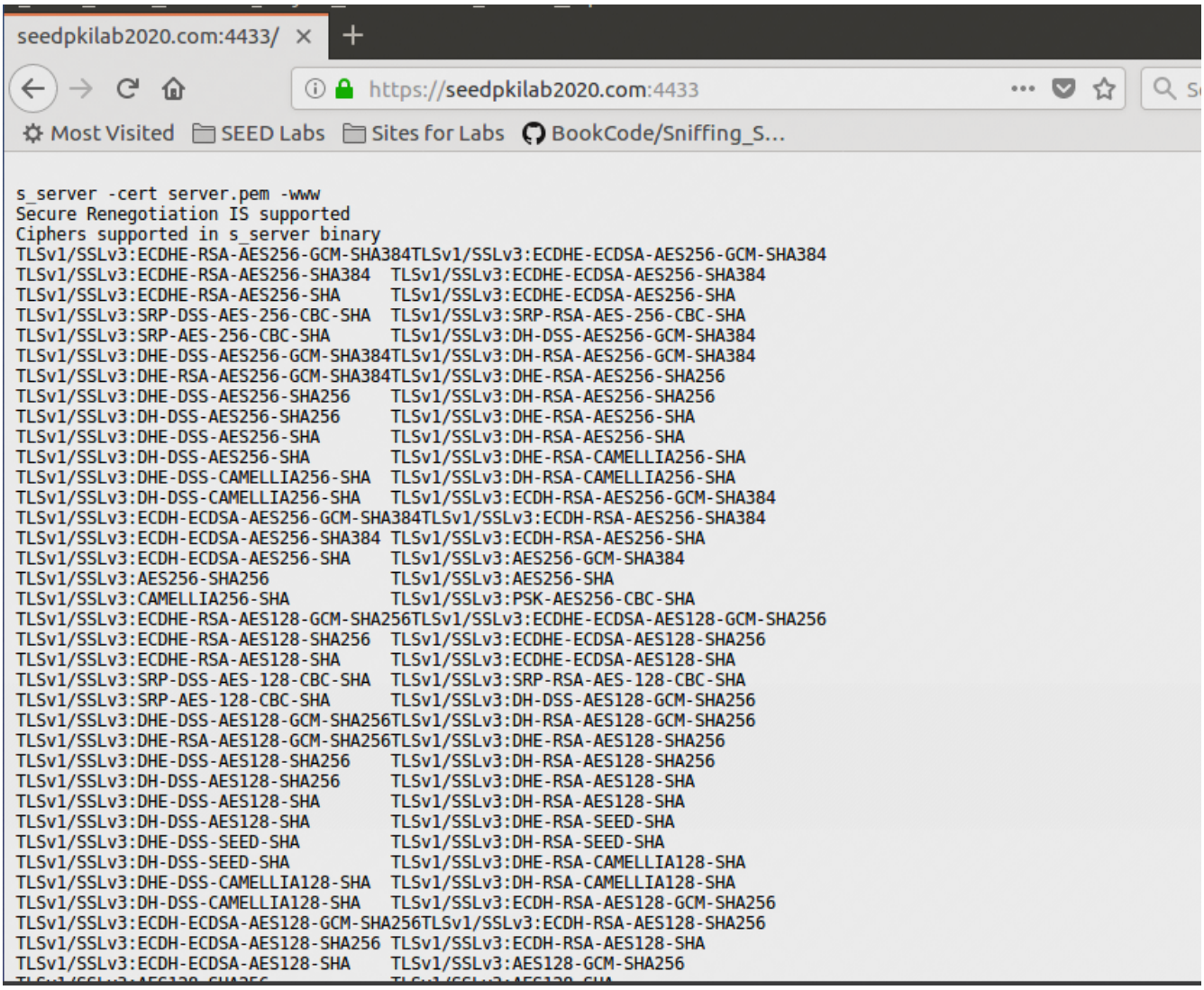
This bit caused an error, so I modifed the 6D (originally a 7D):



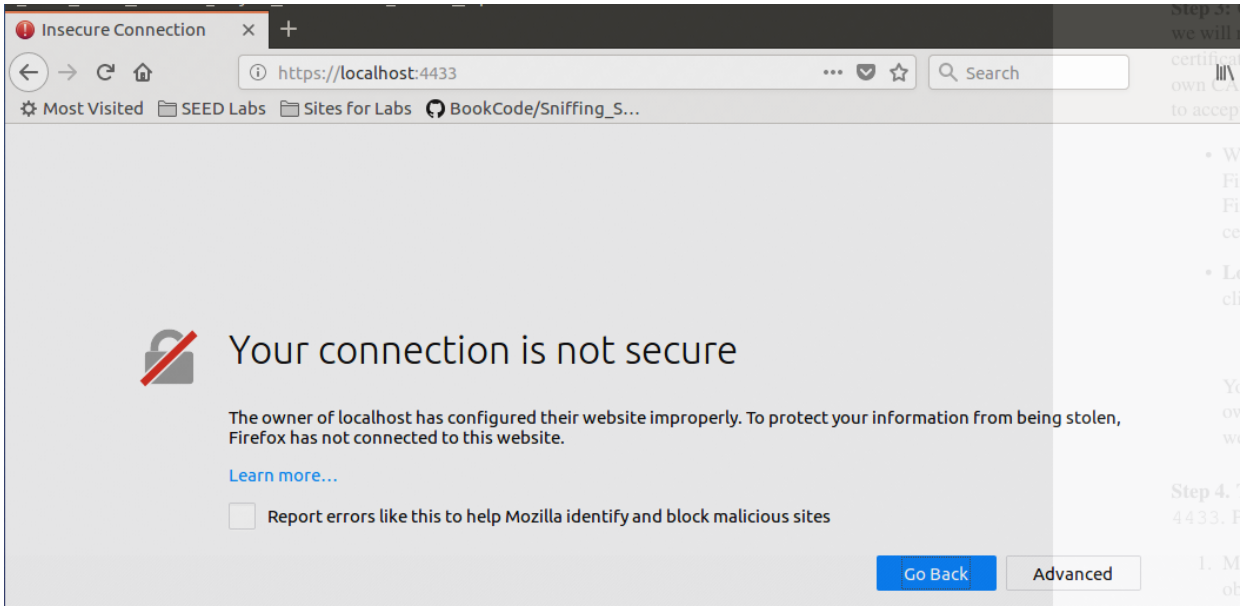
Next, restart the server:

```
[11/24/20]seed@VM:~/Lab6$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

Finally, retry the URL:



I expected to see an error, but perhaps the certificate/website was previously cached. I do however see the intended result upon trying to locate https://localhost:4433:



Observation: We were able to successfully deploy our certificate for the identified client. However, upon corrupting even a single bit, we invalidate the entire process. Additionally, to use the certificate, we needed to add our previously created CA to the list of trusted CA's in Firefox. This allowed the request to be permitted.

Explanation: By first establishing a valid CA, we can issue certificates that will be recognized via web browsers. This is all done using the openssl program, and correct entries/passwords must be used in order for the certificate to work.

Deploying Certificate in an Apache-Based HTTPS Website

To test our certificate on a real webserver, we configure the pre-installed Apache server for our new domain. The first step is to modify the **open-ssl.conf** file:

```
        #   The safe and default but still SSL/TLS standard compliant shutdown
        #   approach is that mod_ssl sends the close notify alert but doesn't wait for
        #   the close notify alert from client. When you need a different shutdown
        #   approach you can use one of the following variables:
        #   o ssl-unclean-shutdown:
        #    This forces an unclean shutdown when the connection is closed, i.e. no
        #    SSL close notify alert is send or allowed to received.  This violates
        #    the SSL/TLS standard but is needed for some brain-dead browsers. Use
        #    this when you receive I/O errors because of the standard approach where
        #    mod_ssl sends the close notify alert.
        #   o ssl-accurate-shutdown:
        #    This forces an accurate shutdown when the connection is closed, i.e. a
        #    SSL close notify alert is send and mod_ssl waits for the close notify
        #    alert of the client. This is 100% SSL/TLS standard compliant, but in
        #    practice often causes hanging connections with brain-dead browsers. Use
        #    this only for browsers where you know that their SSL implementation
        #    works correctly.
        #   Notice: Most problems of broken clients are also related to the HTTP
        #   keep-alive facility, so you usually additionally want to disable
        #   keep-alive for those clients, too. Use variable "nokeepalive" for this.
        #   Similarly, one has to force some clients to use HTTP/1.0 to workaround
        #   their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and
        #   "force-response-1.0" for this.
        # BrowserMatch "MSIE [2-6]" \
        #       nokeepalive ssl-unclean-shutdown \
        #       downgrade-1.0 force-response-1.0

    </VirtualHost>
    <VirtualHost *:44>
        ServerName              SEEDPKILab2020.com
        DocumentRoot            /var/www/SEEDPKILab2020
        SSLEngine On
        SSLCertificateFile      /home/seed/Lab6/server.pem
        SSLCertificateKeyFile   /home/seed/Lab6/server.key
    </VirtualHost>
</IfModule>

Trash  x=apache ts=4 sw=4 sts=4 sr noet
Plain Text   Tab Width: 4   Ln 137, Col 44   INS
```

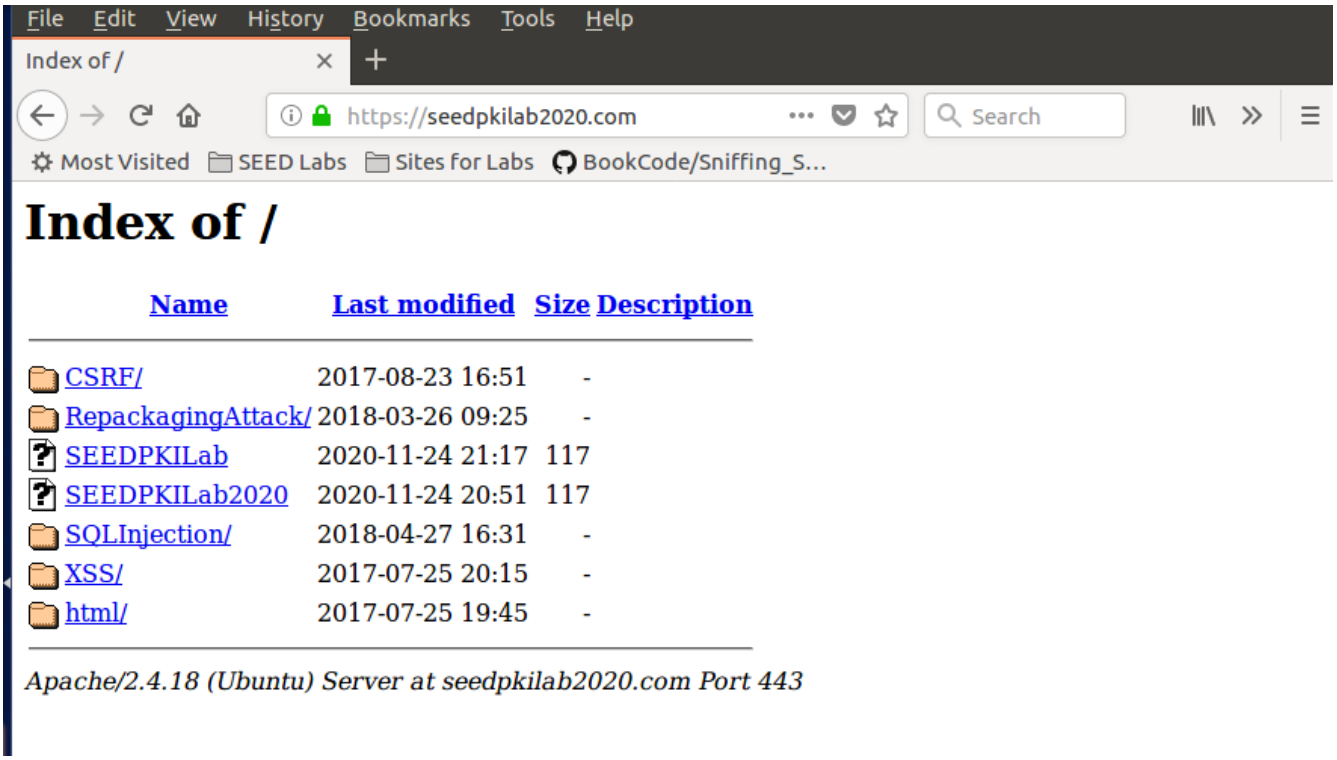
Then we have to create the website specified in the virutal host:

```
ILab2020 (/var/www) - gedit
Open   SEEDPKILab2020
       /var/www
<!DOCTYPE html>
<html>
<head>
<title>Test Site: Success</title>
</head>
<body>

<h1>Kevin 2020</h1>

</body>
</html>
```

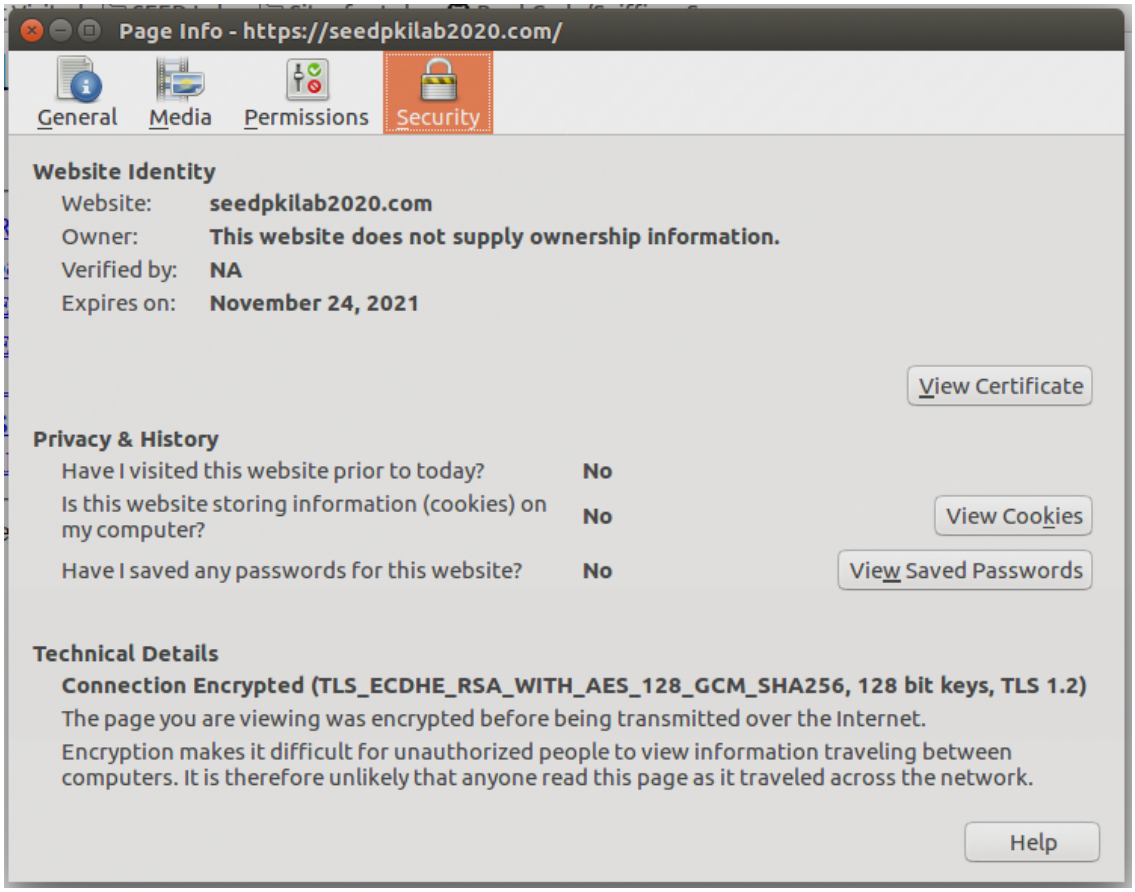
Next, we enable SSL:

```
[11/24/20]seed@VM:.../sites-available$ sudo apachectl configtest
AH00112: Warning: DocumentRoot [/var/www/seedlabclickjacking] does not exist
AH00112: Warning: DocumentRoot [/var/www/SEEDPKILab2020] does not exist
AH00558: apache2: Could not reliably determine the server's fully qualified dom
ain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress
this message
Syntax OK
[11/24/20]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
[11/24/20]seed@VM:.../sites-available$ sudo a2ensite defualt-ssl
ERROR: Site defualt-ssl does not exist!
[11/24/20]seed@VM:.../sites-available$ sudo a2ensite default-ssl
Site default-ssl already enabled
[11/24/20]seed@VM:.../sites-available$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for SEEDPKILab2020.com:443 (RSA): ********
[11/24/20]seed@VM:.../sites-available$
```

Finally, test the webiste. Initially, I was getting error saying it could not find the document root. I changed it to be just **/var/www/** which allowed me to access the following:



And we can confirmthat this site is secure per Firefox:



Observation: After setting up an actual website via Apache, we can have Firefox independently confirm that the site is secure. Note that my directory at **/var/www/** is shown, and *clearly* the missing SEEDPKILab file is there. Regardless, we have achieved a secure connection.

Explanation: By combining Apache with the previous openssl steps, we can create our own secure websites.