Task 1: Becoming a Certificate Authority (CA)

Create a workspace `~/pki/`

```
[04/19/21]seed@VM:~$ cd pki
[04/19/21]seed@VM:~$ mkdir pki
```

Get a copy of the configuration file `openssl.cnf`.

`[04/19/21]seed@VM:~/pki$ cp /usr/lib/ssl/openssl.cnf ./`

create several sub-directories as specified in the configuration file.

```
[04/19/21]seed@VM:~/pki$ mkdir demoCA
[04/19/21]seed@VM:~/pki$ cd demoCA
[04/19/21]seed@VM:~/pki/demoCA$ mkdir certs
[04/19/21]seed@VM:~/pki/demoCA$ mkdir crl newcerts
[04/19/21]seed@VM:~/pki/demoCA$ touch index.txt
[04/19/21]seed@VM:~/pki/demoCA$ touch serial
[04/19/21]seed@VM:~/pki/demoCA$ echo 1000 > serial
[04/19/21]seed@VM:~/pki/demoCA$ cat serial
1000
[04/19/21]seed@VM:~/pki/demoCA$ ls
certs  crl  index.txt  newcerts  serial
```

Now we have set up the configuration file `openssl.cnf`. Next we'll create and issue certificates.

Generate the self-signed certificate for a totally trusted CA of which certificate will serve as the root certificate.

```
[04/19/21]seed@VM:~/pki$ 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'
```

The above command does two things:

- generates CA's private key which is written to `ca.key`.
- Outputs publick key certificate to `ca.crt`.

Note that with `-x509` option it generates a self-signed certificate. While it would generate a request without it.

The pass phrase we use is `passphrase`. Other info we entered are as follows:

```
Country Name (2 letter code) [AU]:CH
State or Province Name (full name) [Some-State]:any
Locality Name (eg, city) []:any
Organization Name (eg, company) [Internet Widgits Pty Ltd]:any
Organizational Unit Name (eg, section) []:any
Common Name (e.g. server FQDN or YOUR name) []:any
Email Address []:any
```

Task 2: Creating a Certificate for `OSULabServer.com`.

Step 1: Generate public/private key pair.

```
[04/19/21]seed@VM:~/pki$ 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:
```

The pass phrase for `server.key` is `passphrase` for convenience. But for the purpose of security it is better to be different and more complicated.

The above command generate an RSA private-public key pair. The key is stored in `server.key` of which content is as follows:

```
[04/19/21]seed@VM:~/pki$ cat server.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,2F6616E5FA969672ED63087278F98502

DG+ryr+N0MU2qADXV9HhiluhsxifYQqHXyNaMXZsv09W/1N+krOu5ag6LBeKjmKd
j7dp/Yb6oGqZ53R0vNeaJxdx+U1IyT503Hdp+g6G/0VaIf2OOCWMl26WpklYbdow
UMAftbzymym/XHCZLPLVatWeQyp96kVvX0gsuEckXg5O/YYNnvt02hm3l4DwseLD
nckSE86ic4aBYvWI56H2va4HotHn/bZbRqMmM4LsGFmJWfauYnaAxPhPxrD0PoxQ
j7IOG6rEvx/RC+ujPiu3ve6SRQiyYlauAVEl1vkCqfe8NEH2peJITY7HyAk9Ss2k
MYn8mifH6/s2DFdCFSd5t0SiXXBB3l6OWF5+toHL2OJfBuMUSliA3P5dhujB+Fav
Hal0YfiSjerJVl42qLl0FhTZJ3yamA5PPIxAkSyZvO5XCFL444nWwVdrP6rIg5uI
```

```
ir2pA5uYUO0wWiA+yW6GIvKTiLuzNo9RRNjQtqP7Y55In7QUfmHuV8STHZqFAeZR
1BW3qutlPzlzNFlYjWgm5WlLWjSHYoiGvFY8wQOQz92r+A2SWXmFtJkrPiQFU1Kc
zfhtWyNeVoAJ02LhtkxJcvG1IYv7uux7bVXuujUKS5B3UsO8zA8yXSvqYyvvo3hg
Zh0yguczLAaUtuaTtOxvGuCeEuGdFE3zylkMnAyyuBqjMSYsRQ0yB9ICOpkgdrKV
16ELIgkNKk5OmIHNVD1VKF7qpfVUWTNHElCqC9pEFa8J42k+GiNGVkyRBZoLrVkN
u5+7PsoBjYweutqIhjH4CDCyQk9RosPPQHQEwr0aLu03+sWzUJEyuDkFpUC5h3tv
-----END RSA PRIVATE KEY-----
```

We can see that the content is encrypted.

It is able to display the actual content involving modulus, private exponents etc with command `openssl rsa -in server.key -text`. The command output is as follows:

```
[04/19/21]seed@VM:~/pki$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
    00:cf:08:80:26:1c:66:40:09:99:da:b1:e4:ac:c3:
    2a:ce:7d:b2:a8:17:3f:12:71:fa:e0:24:b8:76:b6:
    64:0d:bb:0e:d2:df:a9:8d:61:09:d4:63:5a:6a:b9:
    71:77:bc:06:35:1c:8c:81:fb:16:75:c6:08:08:a9:
    83:c9:b2:2a:02:90:60:53:41:28:94:0d:93:a4:82:
    22:e1:aa:df:e6:fa:fa:c6:ee:f4:60:cf:d4:e3:ba:
    ec:8f:4b:b6:45:c6:0f:1e:7e:80:e2:0c:88:c9:4b:
    b6:5c:94:6a:c4:72:75:8a:71:6b:c0:45:d1:a1:4f:
    e6:f7:4a:d2:51:1d:e6:2c:29
publicExponent: 65537 (0x10001)
privateExponent:
    7b:01:97:f2:0c:c9:8b:9e:a8:b4:d1:21:06:ac:66:
    90:8d:0a:4a:e8:94:e6:c6:a5:c1:ea:cf:56:69:03:
    85:4c:f0:7f:c5:b7:9d:72:5c:3e:be:51:08:3b:e9:
    f8:b3:d4:14:56:43:fd:2e:4b:a7:e9:e2:f8:12:8f:
    64:94:e0:73:d7:cf:98:fc:09:69:da:09:ea:ba:11:
    c3:13:e5:b3:c1:04:85:32:84:33:f0:b8:69:5d:1c:
    79:68:be:e3:7b:19:ef:32:1e:b2:8c:42:d9:de:42:
    6f:7d:26:3b:8f:ff:b1:0d:e2:17:cc:39:0c:ed:a4:
    39:7b:2d:93:01:5c:04:89
prime1:
    00:ee:0e:cc:74:35:47:d0:4b:6d:24:64:3d:af:9a:
    bb:f8:cf:de:dc:0a:42:62:e4:31:c1:e8:22:5e:0f:
    d9:e7:14:c5:14:c2:68:43:57:6b:96:95:82:7f:1d:
    25:74:27:8d:16:c1:63:ea:a9:80:f1:ab:ed:dd:d7:
    a7:f6:82:d8:bf
prime2:
    00:de:a3:19:31:28:39:44:d1:6c:14:c0:32:a0:13:
```

```
    74:19:81:66:63:fb:93:ee:dd:3f:43:56:4d:34:63:
    ec:c2:36:26:43:3e:81:13:ce:09:6d:76:b4:3e:28:
    c9:25:dc:b0:a5:22:6d:68:8c:d3:f0:e0:02:e4:ff:
    4d:1a:9d:0d:17
exponent1:
    11:59:32:bd:25:44:de:81:20:ab:0e:43:10:91:0e:
    31:cb:b4:4a:04:d5:61:af:8f:90:ba:02:07:28:d0:
    90:6e:8b:0e:40:3c:a8:eb:ae:03:83:51:c7:41:b0:
    81:0e:80:d1:af:b6:40:5b:a9:f1:f2:bd:9a:f5:24:
    60:6a:98:b1
exponent2:
    00:c2:bb:af:ee:42:3b:8d:49:0c:1b:3e:5d:49:8c:
    9a:71:30:b0:c9:65:24:68:ba:96:7a:24:83:54:fe:
    cd:f8:b5:7a:54:38:d8:97:e5:10:73:f1:6f:08:2b:
    8e:7d:12:82:63:7e:30:6a:51:3b:94:25:ac:02:76:
    98:0f:5e:2d:33
coefficient:
    00:db:14:0e:7a:18:54:d4:6d:51:b9:95:9a:33:1a:
    81:d1:1b:27:0d:33:ec:c2:97:47:dc:e4:ea:95:62:
    50:33:10:41:8a:9b:3a:21:cf:e1:06:18:d7:67:a8:
    62:b6:bf:76:a5:30:dd:d5:d4:fb:9f:b2:4a:e4:72:
    9f:c7:66:b8:c3
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDPCIAmHGZACZnaseSswyrOfbKoFz8ScfrgJLh2tmQNuw7S36mN
YQnUY1pquXF3vAY1HIyB+xZ1xggIqYPJsioCkGBTQSiUDZOkgiLhqt/m+vrG7vRg
z9TjuuyPS7ZFxg8efoDiDIjJS7ZclGrEcnWKcWvARdGhT+b3StJRHeYsKQIDAQAB
AoGAewGX8gzJi56otNEhBqxmkI0KSuiU5salwerPVmkDhUzwf8W3nXJcPr5RCDvp
+LPUFFZD/S5Lp+ni+BKPZJTgc9fPmPwJadoJ6roRwxPls8EEhTKEM/C4aV0ceWi+
43sZ7zIesoxC2d5Cb30mO4//sQ3iF8w5DO2kOXstkwFcBIkCQQDuDsx0NUfQS20k
ZD2vmrv4z97cCkJi5DHB6CJeD9nnFMUUwmhDV2uWlYJ/HSV0J40WwWPqqYDxq+3d
16f2gti/AkEA3qMZMSg5RNFsFMAyoBN0GYFmY/uT7t0/Q1ZNNGPswjYmQz6BE84J
bXa0PijJJdywpSJtaIzT8OAC5P9NGp0NFwJAEVkyvSVE3oEgqw5DEJEOMcu0SgTV
Ya+PkLoCByjQkG6LDkA8qOuuA4NRx0GwgQ6A0a+2QFup8fK9mvUkYGqYsQJBAMK7
r+5CO41JDBs+XUmMmnEwsMllJGi6lnokg1T+zfi1elQ42JflEHPxbwgrjn0SgmN+
MGpRO5QlrAJ2mA9eLTMCQQDbFA56GFTUbVG5lZozGoHRGycNM+zCl0fc5OqVYlAz
EEGKmzohz+EGGNdnqGK2v3alMN3V1PufskrkcP/HZrjD
-----END RSA PRIVATE KEY-----
```

Step 2: generate a certificate signing request(CSR)

Eith the following command, we generate a CSR which would be sent to the CA who will generate a certificate for the key.

When fullfilling the information it prompted, we use `OSULabServer.com` as the common name as requested by the lab requirement.

```
[04/19/21]seed@VM:~/pki$ 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]:CH
State or Province Name (full name) [Some-State]:any
Locality Name (eg, city) []:any
Organization Name (eg, company) [Internet Widgits Pty Ltd]:any
Organizational Unit Name (eg, section) []:any
Common Name (e.g. server FQDN or YOUR name) []:OSULabServer.com
Email Address []:any

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:challenge
An optional company name []:any
```

Step 3: Generating Certificates

We use the following command to turn the CSR (`server.csr`) into an X509 certificate (`server.crt`), using the CA's `ca.crt` and `ca.key`.

```
[04/19/21]seed@VM:~/pki$ 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: Apr 19 17:47:23 2021 GMT
            Not After : Apr 19 17:47:23 2022 GMT
        Subject:
            countryName               = CH
            stateOrProvinceName       = any
            organizationName          = any
            organizationalUnitName    = any
```

```
            commonName                = OSULabServer.com
            emailAddress              = any
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                46:38:A8:50:A5:64:8E:78:39:E6:8D:8E:1B:4C:65:93:C7:08:5C:ED
            X509v3 Authority Key Identifier:
                keyid:51:AC:DA:5C:62:6A:8C:7E:D7:3F:2E:65:7E:5F:34:DF:02:60:AB:4C

Certificate is to be certified until Apr 19 17:47:23 2022 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
```

So we can notice that the certificate request has been certified and written into data base as a new entry.

Task 3: Deploying the certificate in an HTTPS web server

Step 1: Configuring DNS

We added a new entry `127.0.0.1 OSULabServer.com` to `/etc/hosts` which will map the hostname `OSULabServer` to out localhost `127.0.0.1`.

```
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       OSULabServer.com
```

Step 2: Configuring the web server.

Start a simple web server:

- Copy server.key to server.pem

```
[04/19/21]seed@VM:~/pki$ cp server.key server.pem
```

- Append the content of server.crt to server.pem

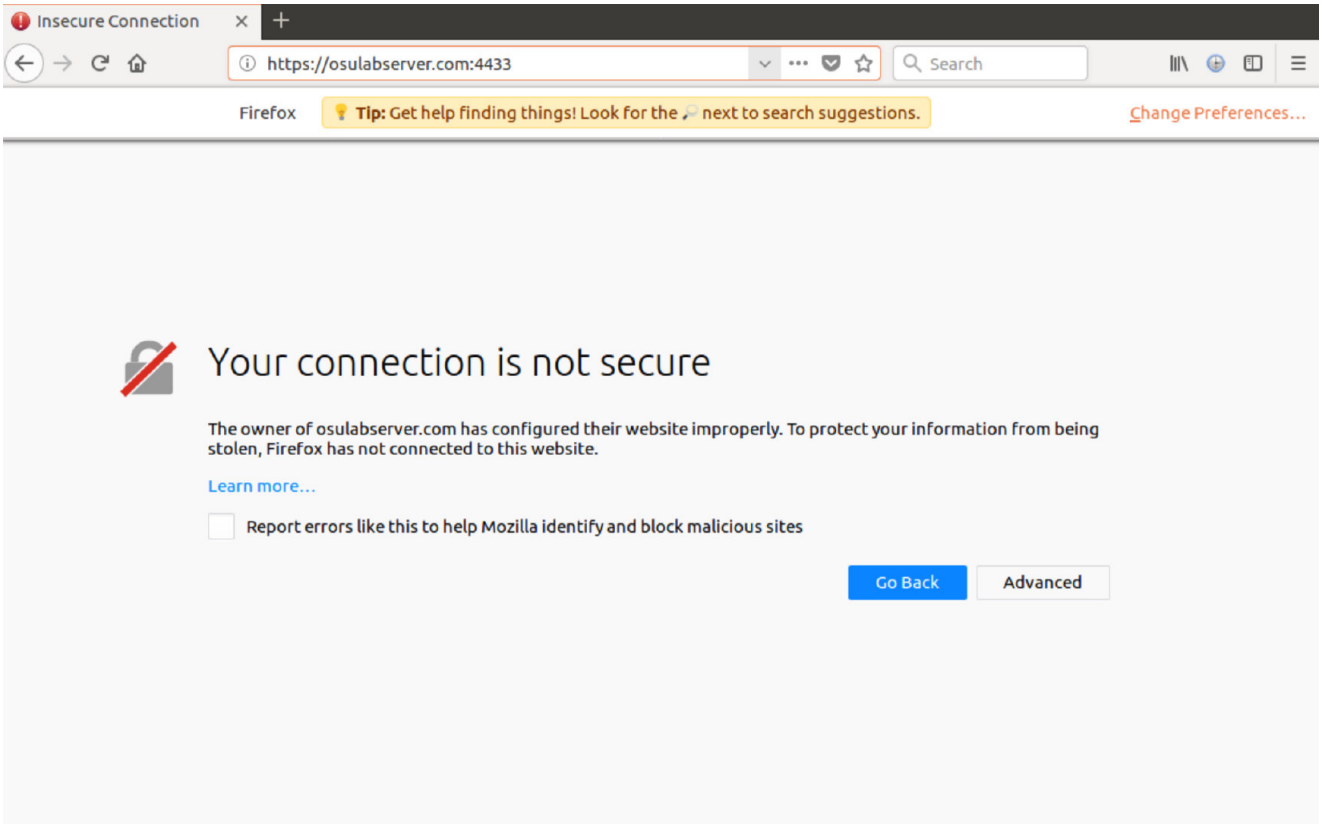
```
[04/19/21]seed@VM:~/pki$ cat server.crt >> server.pem
```

Now we have combined the secret key and certificate into one file `server.pem`.

- Launch the web server using `s_server` command with `server.pem`

```
[04/19/21]seed@VM:~/pki$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

Now visit https://OSULabServer.com:4433/ and we get an error `Your connection is not secure`. This is because browser has not accept our CA certificate yet.



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

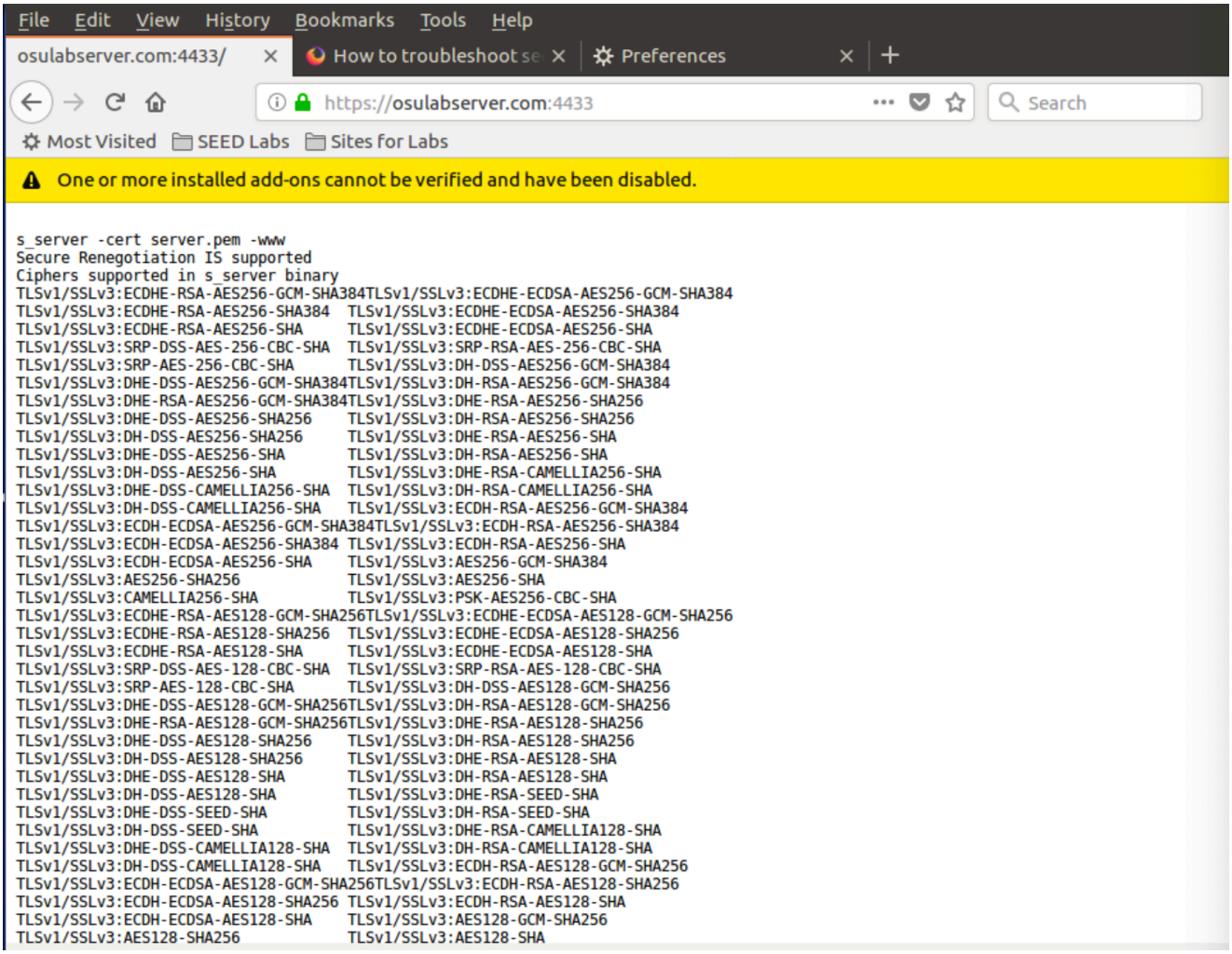
Since Firefox doesn't recognize our own CA which signed the certificate of `OSULabServer.com:4433`. We are supposed to have Firefox accept our certificate.

We choose to load `ca.crt` into Firefox.

By importing our certificate into Firefox via `Preference -> Privacy & Security -> View Certificates -> import`, we are able to vist https://OSULabServer.com:4433/ without error.

Step 4. Testing our HTTPS website.

Now point the browser again to https://OSULabServer.com:4433/ . The following is what we can see.



The complete content displayed on https://OSULabServer.com:4433/ is as follows.

```
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
TLSv1/SSLv3:SRP-DSS-3DES-EDE-CBC-SHA TLSv1/SSLv3:SRP-RSA-3DES-EDE-CBC-SHA
TLSv1/SSLv3:SRP-3DES-EDE-CBC-SHA     TLSv1/SSLv3:EDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:EDH-DSS-DES-CBC3-SHA     TLSv1/SSLv3:DH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:DH-DSS-DES-CBC3-SHA      TLSv1/SSLv3:ECDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:ECDH-ECDSA-DES-CBC3-SHA  TLSv1/SSLv3:DES-CBC3-SHA
TLSv1/SSLv3:PSK-3DES-EDE-CBC-SHA
---
Ciphers common between both SSL end points:
ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-AES256-GCM-
SHA384
ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES128-SHA       ECDHE-RSA-AES256-SHA
AES128-SHA                  AES256-SHA                 DES-CBC3-SHA
Signature Algorithms:
ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:0x04+0x08:0x05+0x08:0x06+0x08:RSA+SHA256:RS
A+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
Shared Signature Algorithms:
ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1
:RSA+SHA1
```

```
Supported Elliptic Curves: 0x001D:P-256:P-384:P-521:0x0100:0x0101
Shared Elliptic curves: P-256:P-384:P-521
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
    Session-ID-ctx: 01000000
    Master-Key:
ABAF90C3E4DB35B65DE5B37D27499481E82F6D461FBFED6B53F913BE91107545B1BA5B99A0A5454988
D529DBCABB130A
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1618856769
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
   0 items in the session cache
   0 client connects (SSL_connect())
   0 client renegotiates (SSL_connect())
   0 client connects that finished
   3 server accepts (SSL_accept())
   0 server renegotiates (SSL_accept())
   3 server accepts that finished
   0 session cache hits
   3 session cache misses
   0 session cache timeouts
   0 callback cache hits
   0 cache full overflows (128 allowed)
---
no client certificate available
```

Explanation of above observation:

The above content is the data sent to the Firefox client.

The browser is displaying a full list of ciphers as well as common ciphers shared between the endpoints when you try to access the server and details of SSL-Session.

The effect of modifying a byte in server.pem and an explanation for what you observed.

A possible option is to delete and insert a byte toward the end of the certificate, but try modifying other parts of server.pem to see what causes issues and what does not.

- If we delete a dash at the last line to make it `-----END CERTIFICATE----`, it is unable to load certificate.

```
[04/19/21]seed@VM:~/pki$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load server certificate private key file
3071280832:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad
decrypt:evp_enc.c:529:
3071280832:error:0906A065:PEM routines:PEM_do_header:bad
decrypt:pem_lib.c:482:
```

- If we change the Issuer information in Certificate from `Issuer: C=CH` to `Issuer: C=CC`, it still works.
- If we change the Public Key Algorithm name in Certificate from `Public Key Algorithm: rsaEncryption` to `Public Key Algorithm: rssEncryption`, it still works.
- It still works even when we change its Modulus info. The following is original Modulus. The '00' in its first line is modified to '01' and it still works.

```
                Modulus:
                    00:cf:08:80:26:1c:66:40:09:99:da:b1:e4:ac:c3:
                    2a:ce:7d:b2:a8:17:3f:12:71:fa:e0:24:b8:76:b6:
                    64:0d:bb:0e:d2:df:a9:8d:61:09:d4:63:5a:6a:b9:
                    71:77:bc:06:35:1c:8c:81:fb:16:75:c6:08:08:a9:
                    83:c9:b2:2a:02:90:60:53:41:28:94:0d:93:a4:82:
                    22:e1:aa:df:e6:fa:fa:c6:ee:f4:60:cf:d4:e3:ba:
                    ec:8f:4b:b6:45:c6:0f:1e:7e:80:e2:0c:88:c9:4b:
                    b6:5c:94:6a:c4:72:75:8a:71:6b:c0:45:d1:a1:4f:
                    e6:f7:4a:d2:51:1d:e6:2c:29
```

- I also tried to modify other the character on other positions. After multiple attempts, we find that the content between -----BEGIN RSA PRIVATE KEY----- and -----END RSA PRIVATE KEY-----, -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----(including these break lines) can not be modified. Otherwise, it won't work. Other part, Certificate info, as shown below, can be modified and won't influence its functionality.

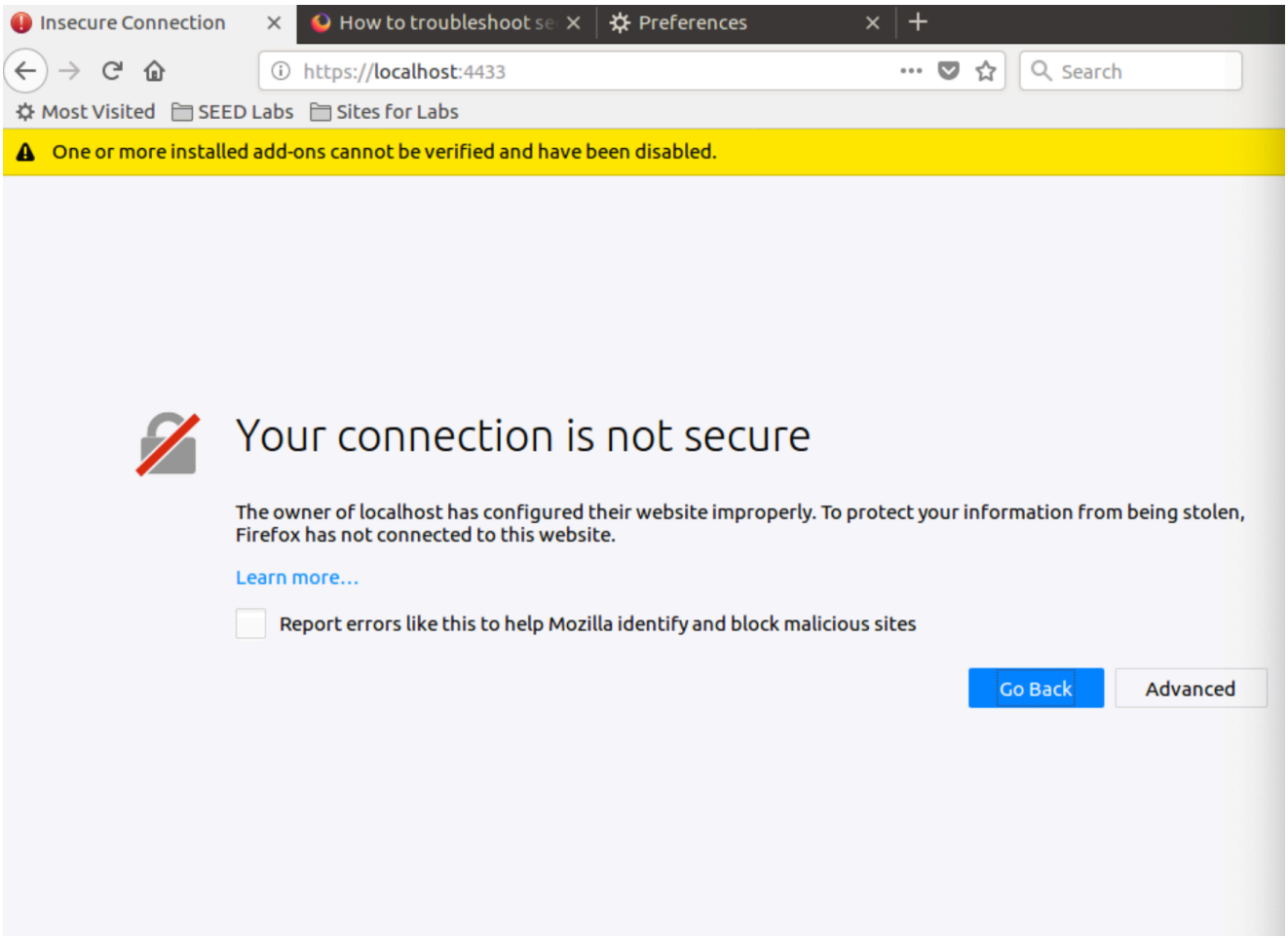
```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
```

```
        Issuer: C=CH, ST=any, L=any, O=any, OU=any, CN=any/emailAddress=any
        Validity
            Not Before: Apr 19 17:47:23 2021 GMT
            Not After : Apr 19 17:47:23 2022 GMT
        Subject: C=CH, ST=any, O=any, OU=any,
CN=OSULabServer.com/emailAddress=any
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:cf:08:80:26:1c:66:40:09:99:da:b1:e4:ac:c3:
                    2a:ce:7d:b2:a8:17:3f:12:71:fa:e0:24:b8:76:b6:
                    64:0d:bb:0e:d2:df:a9:8d:61:09:d4:63:5a:6a:b9:
                    71:77:bc:06:35:1c:8c:81:fb:16:75:c6:08:08:a9:
                    83:c9:b2:2a:02:90:60:53:41:28:94:0d:93:a4:82:
                    22:e1:aa:df:e6:fa:fa:c6:ee:f4:60:cf:d4:e3:ba:
                    ec:8f:4b:b6:45:c6:0f:1e:7e:80:e2:0c:88:c9:4b:
                    b6:5c:94:6a:c4:72:75:8a:71:6b:c0:45:d1:a1:4f:
                    e6:f7:4a:d2:51:1d:e6:2c:29
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                46:38:A8:50:A5:64:8E:78:39:E6:8D:8E:1B:4C:65:93:C7:08:5C:ED
            X509v3 Authority Key Identifier:

keyid:51:AC:DA:5C:62:6A:8C:7E:D7:3F:2E:65:7E:5F:34:DF:02:60:AB:4C

    Signature Algorithm: sha256WithRSAEncryption
         6b:c6:85:a2:f7:cf:9d:19:af:00:03:1d:f4:f2:85:90:15:8d:
         e6:eb:df:d1:1e:5c:78:4f:02:d0:21:52:5a:34:23:16:51:76:
         ec:98:b2:4f:e5:98:59:1c:55:ab:af:86:26:b6:f5:5b:77:cb:
         ac:c1:3e:b2:8c:49:48:69:c7:59:c4:18:3a:3f:73:71:bb:7d:
         ec:51:fb:7d:ee:a7:d3:0b:5d:b4:94:0b:0b:14:10:7a:c2:9d:
         bd:79:25:d0:31:a3:25:56:ec:40:d2:87:e0:eb:74:81:22:da:
         e9:37:c7:50:6a:b2:5d:82:d4:43:6f:e9:a7:0f:b3:9a:2e:88:
         24:6e:58:98:4d:cd:d8:8c:ff:c6:8a:be:27:89:77:4e:9e:ac:
         25:6f:73:4f:32:ec:87:b8:31:cc:8b:97:e5:6e:00:be:de:ef:
         b8:ec:b8:ad:5c:40:b2:09:05:b4:29:63:97:79:53:bf:04:ae:
         4c:8b:fe:24:ca:df:82:b2:d5:13:35:54:a4:53:6e:de:27:b0:
         b9:40:1b:3b:a4:7b:47:3e:65:7c:15:5a:e7:df:92:d1:fa:e2:
         25:a5:7d:e4:cd:d9:a8:b2:10:c7:a4:95:a8:10:d1:0e:25:ab:
         1d:eb:c1:4c:1d:d9:77:3b:43:6f:cd:63:5a:f8:de:03:db:34:
```

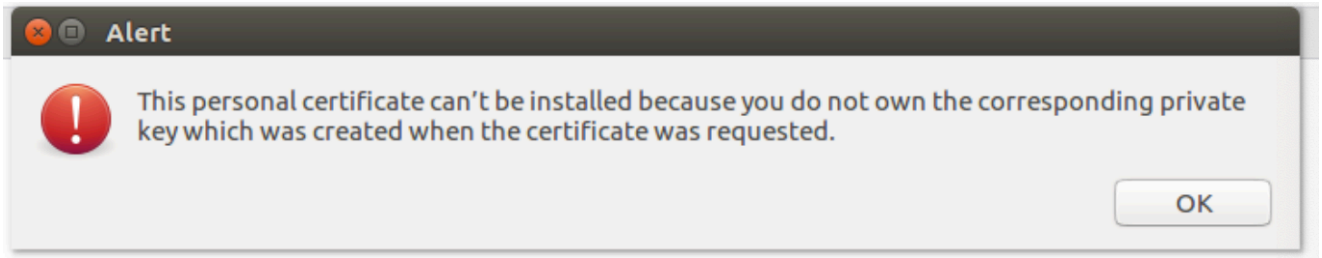
```
            99:62:4c:08
```

Try localhost

If we use https://localhost:4433 instead, we will encounter the error again.



If we try to import the same `ca.crt` file for OSULabServer.com as before, it would fail.



There are different domains mapped to 127.0.0.1 in `/etc/hosts`. If we want to access https://localhost:4433 on Firefox, we are supposed to add `localhost` to `etc/hosts` and redo the steps to generate public-private key pair, etc.

```
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       OSULabServer.com
```