Lab 5: Securing Apache Web Server

Name: Md Robiul Islam Robin

Reg: 2020831043

Introduction

HTTP protocol has no built-in security, which makes data transmission vulnerable to interception. To address this, SSL/TLS (Secure Sockets Layer/Transport Layer Security) was introduced as a security overlay in the transport layer. When combined with HTTP, it creates HTTPS (HTTP Secure), which provides:
● Confidentiality: Data is encrypted during transmission
● Authentication: Server identity is verified through certificates
● Integrity: Data cannot be modified without detection In this lab, I performed three main tasks:

1. Becoming a Certificate Authority
2. Creating certificates for virtual hosts (webserverlab.com)
3. Deploying HTTPS on Apache web server

Task1 : Own Local Certificate Authority (CA)

---

1. Create Folder & Enter It

```
mkdir myCA
cd myCA
```

2. Copy Default OpenSSL Configuration File

```
cp /usr/lib/ssl/openssl.cnf .
```

Check:

```
ls
```

You should see:

```
openssl.cnf
```

---

3. Create Folders Required by `[ CA_default ]`

Open config:

```
nano opensssl.cnf
```

From the config, these paths are needed:

```
demoCA/
demoCA/private/
demoCA/newcerts/
```

Create them:

```
mkdir demoCA
mkdir demoCA/private
mkdir demoCA/newcerts
```

---

4. Create Required Files

Empty index file:

```
touch demoCA/index.txt
```

Serial file with starting number:

```
echo 1000 > demoCA/serial
```

Check:

```
ls demoCA
```

Expected:

```
index.txt  newcerts  private  serial
```

---

5. Generate the CA’s Self-Signed Certificate

Run:

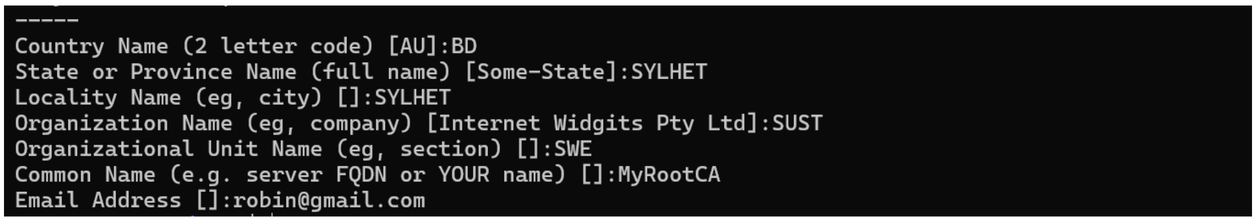
```
openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

Passphrase Example Used

```
robin
```

Certificate info used

```
Country Name: BD
State / Province: Sylhet
Locality: Sylhet
Organization Name: SUST
Organizational Unit: SWE
Common Name: MyRootCA
Email Address: robin@gmail.com
```



```
-----
Country Name (2 letter code) [AU]:BD
State or Province Name (full name) [Some-State]:SYLHET
Locality Name (eg, city) []:SYLHET
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SUST
Organizational Unit Name (eg, section) []:SWE
Common Name (e.g. server FQDN or YOUR name) []:MyRootCA
Email Address []:robin@gmail.com
```

After Completion

Two files will be created:

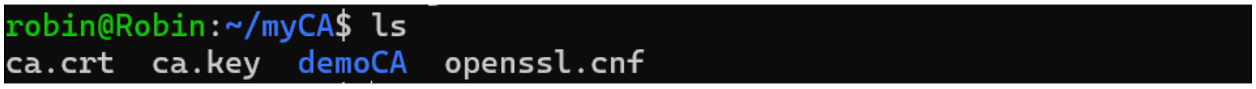
```
ca.key    ← CA private key
ca.crt   ← CA public certificate
```

Check:

```
ls
```

You should see:

```
ca.key  ca.crt  openssl.cnf  demoCA
```



```
robin@Robin:~/myCA$ ls
ca.crt  ca.key  demoCA  openssl.cnf
```

Task 2: Creating Certificates for webserverlab.com

Step 1 — Generate Public/Private Key Pair

Generated a **4096-bit RSA private key** for the server:

```
openssl genrsa -des3 -out webserverlab_server.key 4096
```

This creates a password-protected private key.

Step 2 — Generate the CSR (Certificate Signing Request)

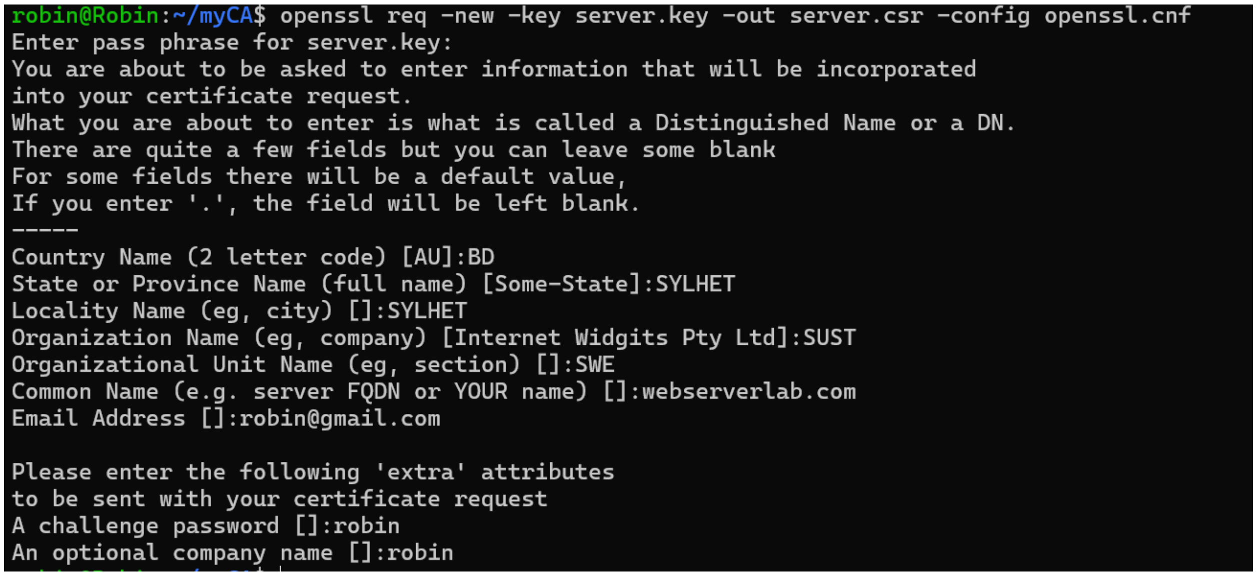
```
openssl req -new -key webserverlab_server.key -out
webserverlab_server.csr -config openssl.cnf
```

Filled in the certificate details:

```
Country Name: BD
State / Province: Sylhet
```

```
Organization: SUST CA
Organizational Unit: Web Services
Common Name: webserverlab.com
Email Address: admin@webserverlab.com
```

Common Name must match the domain → webserverlab.com



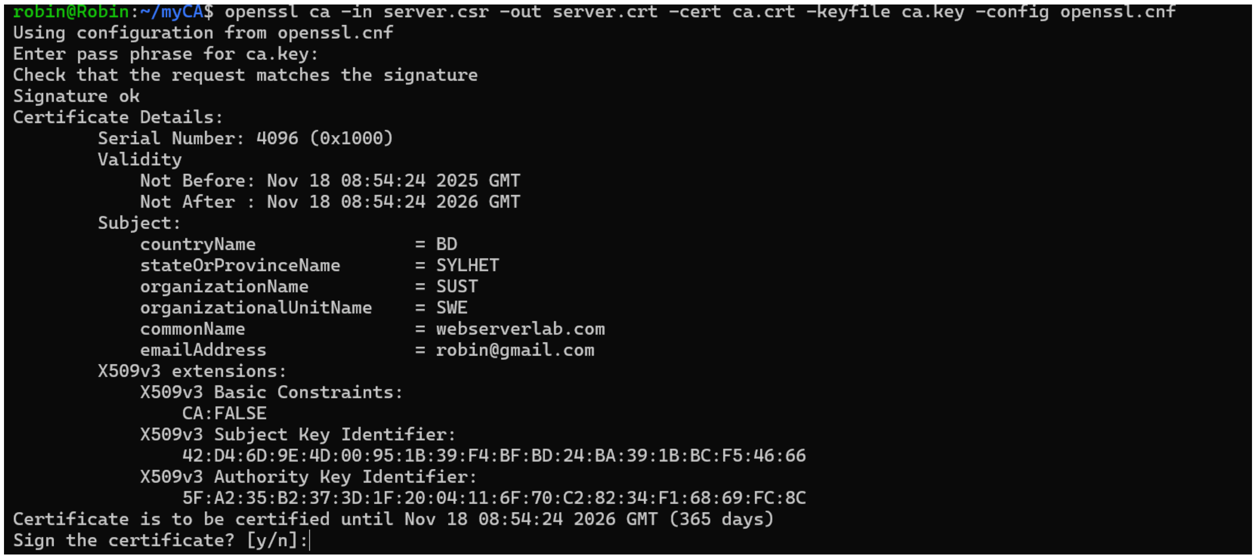
```
robin@Robin:~/myCA$ openssl req -new -key server.key -out server.csr -config openssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or 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]:BD
State or Province Name (full name) [Some-State]:SYLHET
Locality Name (eg, city) []:SYLHET
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SUST
Organizational Unit Name (eg, section) []:SWE
Common Name (e.g. server FQDN or YOUR name) []:webserverlab.com
Email Address []:robin@gmail.com

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

Step 3 — Sign the CSR Using My CA

```
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key
-config openssl.cnf
```

- Entered CA private key password
- Verified certificate details
- Approved signing by typing **y**

This created the signed server certificate:
`server.crt`



```
robin@Robin:~/myCA$ 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: Nov 18 08:54:24 2025 GMT
            Not After : Nov 18 08:54:24 2026 GMT
        Subject:
            countryName               = BD
            stateOrProvinceName       = SYLHET
            organizationName          = SUST
            organizationalUnitName    = SWE
            commonName                = webserverlab.com
            emailAddress              = robin@gmail.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                42:D4:6D:9E:4D:00:95:1B:39:F4:BF:BD:24:BA:39:1B:BC:F5:46:66
            X509v3 Authority Key Identifier:
                5F:A2:35:B2:37:3D:1F:20:04:11:6F:70:C2:82:34:F1:68:69:FC:8C
Certificate is to be certified until Nov 18 08:54:24 2026 GMT (365 days)
Sign the certificate? [y/n]:
```

Step 4 — Prepare Certificate for the OpenSSL Test Server

Option 1 — Combine server key + certificate into one PEM

```
cp webserverlab_server.key webserverlab_server.pem
cat webserverlab_server.crt >> webserverlab_server.pem
```

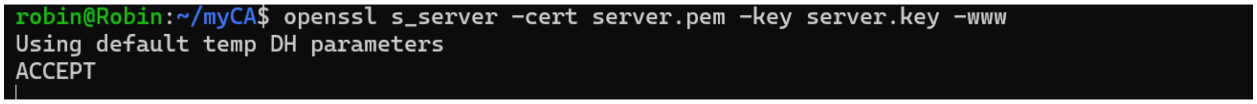
Option 2 — Create a full-chain certificate

(Server certificate + CA certificate)

```
cat server.crt ca.crt > server.pem
```

Step 5 — Start the OpenSSL Test HTTPS Server

```
openssl s_server -cert server.pem -key server.key -www -accept 4433
```



```
robin@Robin:~/myCA$ openssl s_server -cert server.pem -key server.key -www
Using default temp DH parameters
ACCEPT
```

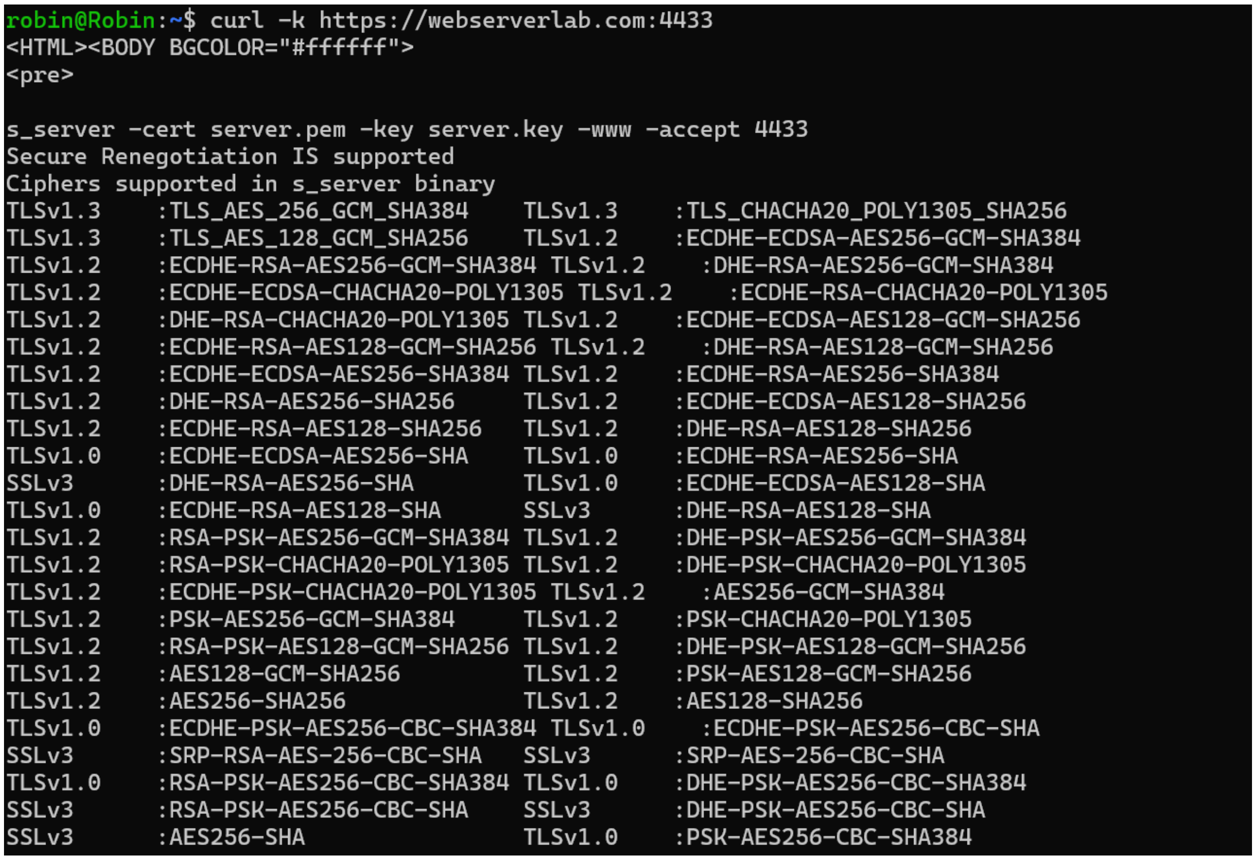
- Server listens on **port 4433**

- You can access it at:
  https://webserverlab.com:4433

Step 6 — Test the Connection

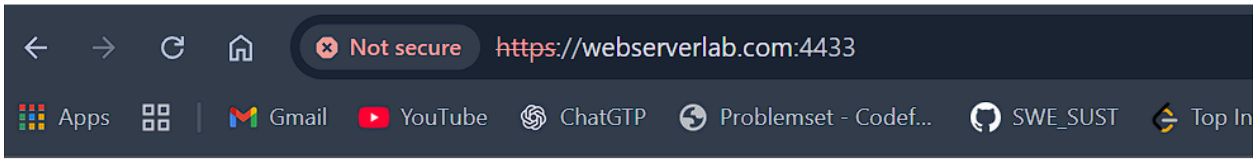
Open a new terminal:

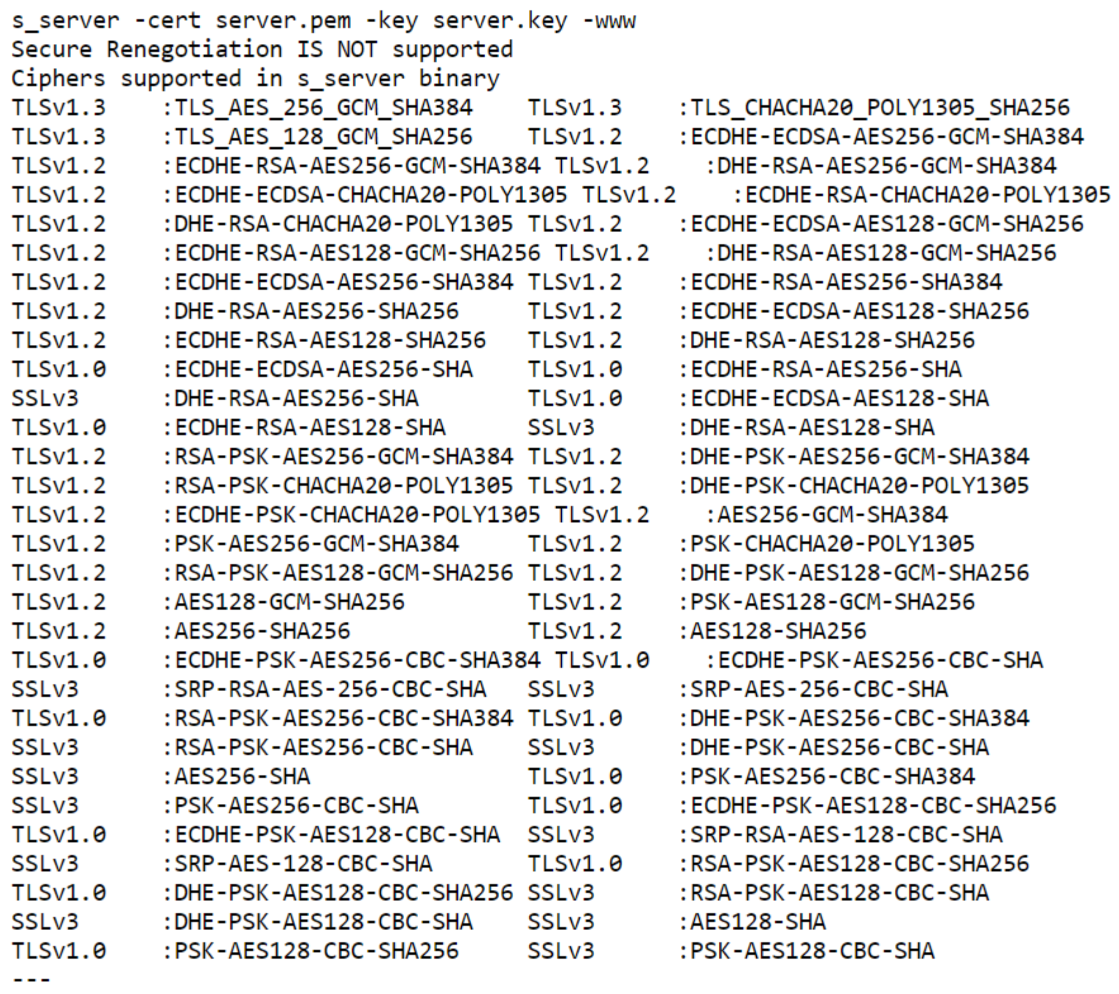
```
curl -k https://webserverlab.com:4433
```



```
robin@Robin:~$ curl -k https://webserverlab.com:4433
<HTML><BODY BGCOLOR="#ffffff">
<pre>

s_server -cert server.pem -key server.key -www -accept 4433
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
TLSv1.2    :DHE-RSA-AES256-SHA256     TLSv1.2    :ECDHE-ECDSA-AES128-SHA256
TLSv1.2    :ECDHE-RSA-AES128-SHA256   TLSv1.2    :DHE-RSA-AES128-SHA256
TLSv1.0    :ECDHE-ECDSA-AES256-SHA    TLSv1.0    :ECDHE-RSA-AES256-SHA
SSLv3      :DHE-RSA-AES256-SHA        TLSv1.0    :ECDHE-ECDSA-AES128-SHA
TLSv1.0    :ECDHE-RSA-AES128-SHA      SSLv3      :DHE-RSA-AES128-SHA
TLSv1.2    :RSA-PSK-AES256-GCM-SHA384 TLSv1.2    :DHE-PSK-AES256-GCM-SHA384
TLSv1.2    :RSA-PSK-CHACHA20-POLY1305 TLSv1.2    :DHE-PSK-CHACHA20-POLY1305
TLSv1.2    :ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2    :AES256-GCM-SHA384
TLSv1.2    :PSK-AES256-GCM-SHA384     TLSv1.2    :PSK-CHACHA20-POLY1305
TLSv1.2    :RSA-PSK-AES128-GCM-SHA256 TLSv1.2    :DHE-PSK-AES128-GCM-SHA256
TLSv1.2    :AES128-GCM-SHA256         TLSv1.2    :PSK-AES128-GCM-SHA256
TLSv1.2    :AES256-SHA256             TLSv1.2    :AES128-SHA256
TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA384 TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA
SSLv3      :SRP-RSA-AES-256-CBC-SHA   SSLv3      :SRP-AES-256-CBC-SHA
TLSv1.0    :RSA-PSK-AES256-CBC-SHA384 TLSv1.0    :DHE-PSK-AES256-CBC-SHA384
SSLv3      :RSA-PSK-AES256-CBC-SHA    SSLv3      :DHE-PSK-AES256-CBC-SHA
SSLv3      :AES256-SHA                TLSv1.0    :PSK-AES256-CBC-SHA384
```

Checking web browser





```
s_server -cert server.pem -key server.key -www
Secure Renegotiation IS NOT supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
TLSv1.2    :DHE-RSA-AES256-SHA256     TLSv1.2    :ECDHE-ECDSA-AES128-SHA256
TLSv1.2    :ECDHE-RSA-AES128-SHA256   TLSv1.2    :DHE-RSA-AES128-SHA256
TLSv1.0    :ECDHE-ECDSA-AES256-SHA    TLSv1.0    :ECDHE-RSA-AES256-SHA
SSLv3      :DHE-RSA-AES256-SHA        TLSv1.0    :ECDHE-ECDSA-AES128-SHA
TLSv1.0    :ECDHE-RSA-AES128-SHA      SSLv3      :DHE-RSA-AES128-SHA
TLSv1.2    :RSA-PSK-AES256-GCM-SHA384 TLSv1.2    :DHE-PSK-AES256-GCM-SHA384
TLSv1.2    :RSA-PSK-CHACHA20-POLY1305 TLSv1.2    :DHE-PSK-CHACHA20-POLY1305
TLSv1.2    :ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2    :AES256-GCM-SHA384
TLSv1.2    :PSK-AES256-GCM-SHA384     TLSv1.2    :PSK-CHACHA20-POLY1305
TLSv1.2    :RSA-PSK-AES128-GCM-SHA256 TLSv1.2    :DHE-PSK-AES128-GCM-SHA256
TLSv1.2    :AES128-GCM-SHA256         TLSv1.2    :PSK-AES128-GCM-SHA256
TLSv1.2    :AES256-SHA256             TLSv1.2    :AES128-SHA256
TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA384 TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA
SSLv3      :SRP-RSA-AES-256-CBC-SHA   SSLv3      :SRP-AES-256-CBC-SHA
TLSv1.0    :RSA-PSK-AES256-CBC-SHA384 TLSv1.0    :DHE-PSK-AES256-CBC-SHA384
SSLv3      :RSA-PSK-AES256-CBC-SHA    SSLv3      :DHE-PSK-AES256-CBC-SHA
SSLv3      :AES256-SHA                TLSv1.0    :PSK-AES256-CBC-SHA384
SSLv3      :PSK-AES256-CBC-SHA        TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA256
TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA  SSLv3      :SRP-RSA-AES-128-CBC-SHA
SSLv3      :SRP-AES-128-CBC-SHA       TLSv1.0    :RSA-PSK-AES128-CBC-SHA256
TLSv1.0    :DHE-PSK-AES128-CBC-SHA256 SSLv3      :RSA-PSK-AES128-CBC-SHA
SSLv3      :DHE-PSK-AES128-CBC-SHA    SSLv3      :AES128-SHA
TLSv1.0    :PSK-AES128-CBC-SHA256     SSLv3      :PSK-AES128-CBC-SHA
---
```

Importing CA Certificate into Firefox (WSL) & Accessing Server

Step 7: Importing CA Certificate into Firefox (inside WSL)

1. Installed Firefox in WSL:

```
sudo apt update
sudo apt install firefox -y
```

2. Set up the display environment for GUI apps:

```
export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print
$2}'):0
```

3. Created a new Firefox profile:

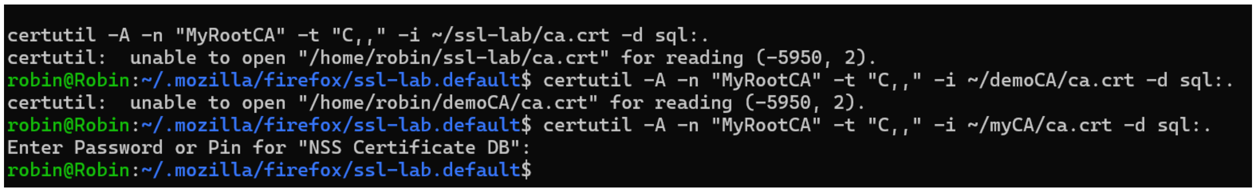
```
mkdir -p ~/.mozilla/firefox/ssl-lab.default

cd ~/.mozilla/firefox/ssl-lab.default
```

4. Initialized certificate database:

```
sudo apt install libnss3-tools -y

certutil -N -d sql:.

Password: 12345678
```

5. Imported CA certificate into Firefox:

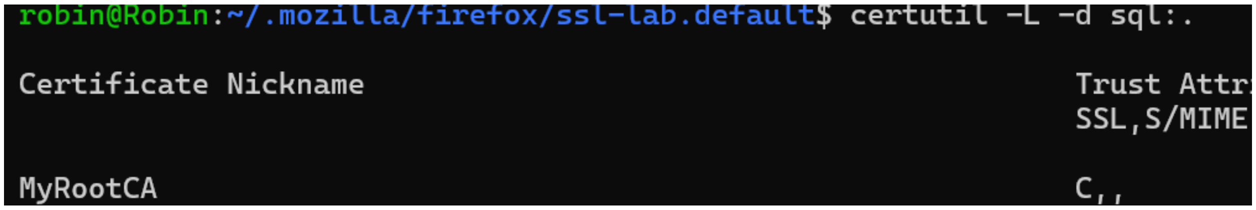
```
certutil -A -n "MyRootCA" -t "C,," -i ~/myCA/ca.crt -d sql:.
```



```
certutil -A -n "MyRootCA" -t "C,," -i ~/ssl-lab/ca.crt -d sql:.
certutil:  unable to open "/home/robin/ssl-lab/ca.crt" for reading (-5950, 2).
robin@Robin:~/.mozilla/firefox/ssl-lab.default$ certutil -A -n "MyRootCA" -t "C,," -i ~/demoCA/ca.crt -d sql:.
certutil:  unable to open "/home/robin/demoCA/ca.crt" for reading (-5950, 2).
robin@Robin:~/.mozilla/firefox/ssl-lab.default$ certutil -A -n "MyRootCA" -t "C,," -i ~/myCA/ca.crt -d sql:.
Enter Password or Pin for "NSS Certificate DB":
robin@Robin:~/.mozilla/firefox/ssl-lab.default$
```

Verify Ceritficate

certutil -L -d sql:.

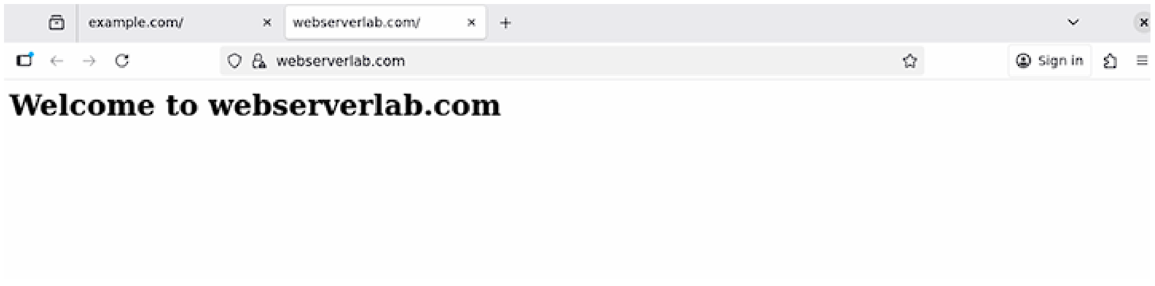


```
robin@Robin:~/.mozilla/firefox/ssl-lab.default$ certutil -L -d sql:.

Certificate Nickname                                         Trust Attr
                                                             SSL,S/MIME

MyRootCA                                                     C,,
```

Step 8: Accessing the HTTPS Server from Firefox

Launch Firefox with the custom profile and open your HTTPS server:

```
firefox -profile ~/.mozilla/firefox/ssl-lab.default
https://webserverlab.com:4433 &
```



Welcome to webserverlab.com

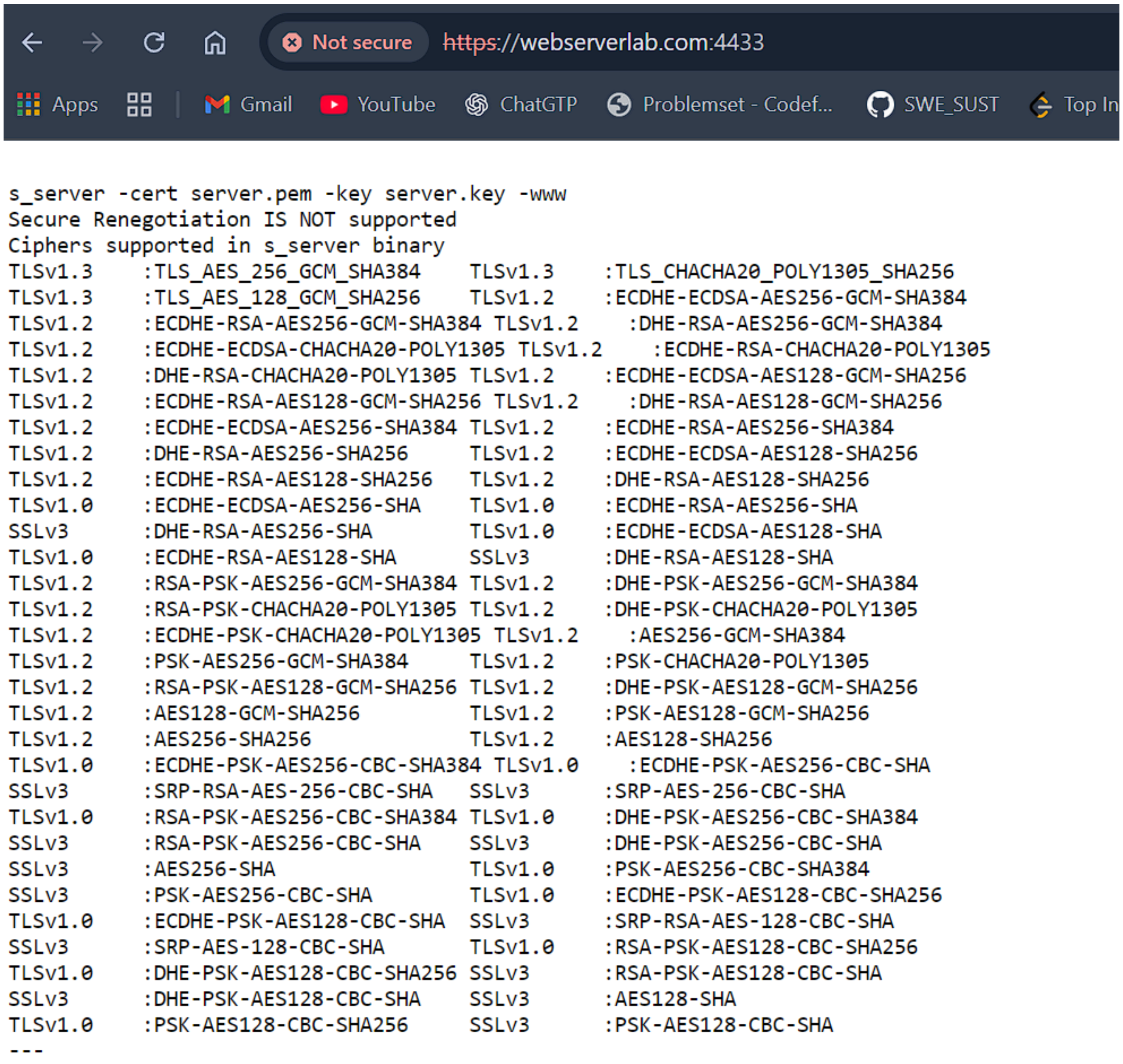
Checkpoint 1

1. OpenSSL server running on port 4433 with the certificate I created

2. Accessed via example.com:4433 showing the OpenSSL server information

page

3. Accessed via localhost:4433 showing the same result



```
s_server -cert server.pem -key server.key -www
Secure Renegotiation IS NOT supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
TLSv1.2    :DHE-RSA-AES256-SHA256     TLSv1.2    :ECDHE-ECDSA-AES128-SHA256
TLSv1.2    :ECDHE-RSA-AES128-SHA256   TLSv1.2    :DHE-RSA-AES128-SHA256
TLSv1.0    :ECDHE-ECDSA-AES256-SHA    TLSv1.0    :ECDHE-RSA-AES256-SHA
SSLv3      :DHE-RSA-AES256-SHA        TLSv1.0    :ECDHE-ECDSA-AES128-SHA
TLSv1.0    :ECDHE-RSA-AES128-SHA      SSLv3      :DHE-RSA-AES128-SHA
TLSv1.2    :RSA-PSK-AES256-GCM-SHA384 TLSv1.2    :DHE-PSK-AES256-GCM-SHA384
TLSv1.2    :RSA-PSK-CHACHA20-POLY1305 TLSv1.2    :DHE-PSK-CHACHA20-POLY1305
TLSv1.2    :ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2    :AES256-GCM-SHA384
TLSv1.2    :PSK-AES256-GCM-SHA384     TLSv1.2    :PSK-CHACHA20-POLY1305
TLSv1.2    :RSA-PSK-AES128-GCM-SHA256 TLSv1.2    :DHE-PSK-AES128-GCM-SHA256
TLSv1.2    :AES128-GCM-SHA256         TLSv1.2    :PSK-AES128-GCM-SHA256
TLSv1.2    :AES256-SHA256             TLSv1.2    :AES128-SHA256
TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA384 TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA
SSLv3      :SRP-RSA-AES-256-CBC-SHA   SSLv3      :SRP-AES-256-CBC-SHA
TLSv1.0    :RSA-PSK-AES256-CBC-SHA384 TLSv1.0    :DHE-PSK-AES256-CBC-SHA384
SSLv3      :RSA-PSK-AES256-CBC-SHA    SSLv3      :DHE-PSK-AES256-CBC-SHA
SSLv3      :AES256-SHA                TLSv1.0    :PSK-AES256-CBC-SHA384
SSLv3      :PSK-AES256-CBC-SHA        TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA256
TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA  SSLv3      :SRP-RSA-AES-128-CBC-SHA
SSLv3      :SRP-AES-128-CBC-SHA       TLSv1.0    :RSA-PSK-AES128-CBC-SHA256
TLSv1.0    :DHE-PSK-AES128-CBC-SHA256 SSLv3      :RSA-PSK-AES128-CBC-SHA
SSLv3      :DHE-PSK-AES128-CBC-SHA    SSLv3      :AES128-SHA
TLSv1.0    :PSK-AES128-CBC-SHA256     SSLv3      :PSK-AES128-CBC-SHA
---
```

Observation: Accessing the Server & Understanding SSL/TLS Handshake

Access Notes

Both:

- `https://example.com:4433`
- `https://localhost:4433`

point to the same server because in the `/etc/hosts` file:

```
127.0.0.1   example.com
```

(Localhost is mapped to example.com.)

SSL/TLS Handshake Process

When connecting to the server, the following steps occurred:

1. **Browser Request** – The browser sends a request listing supported ciphers and hash functions.
2. **Server Response** – The server selects a cipher and provides its digital certificate.
3. **Certificate Verification** – The browser verifies the server certificate using the imported CA certificate (`ca.crt`).
4. **Secure Connection Established** – A secure channel is established using **TLSv1.3 with AES-256-GCM encryption**.

Key Observations

- Without importing the CA certificate, Firefox showed a **security warning**.
- After importing `ca.crt`, the connection was **trusted**.
- The certificate successfully proved the server's identity.
- All data transmitted over this connection is **encrypted**, ensuring **confidentiality and integrity**.

Checkpoint 2

Creating Certificate for webserverlab.com

Following the same procedure as for `example.com`, I created and deployed a certificate for `webserverlab.com`.

Step 1 — Generate Private Key

```
cd ~/ssl-lab

openssl genrsa -des3 -out webserver_server.key 2048
```

- A 2048-bit RSA private key was generated.
- The key is password-protected.

Step 2 — Generate Certificate Signing Request (CSR)

```
openssl req -new -key webserver_server.key -out webserver_server.csr
-config openssl.cnf
```

- Entered `webserverlab.com` as the **Common Name (CN)**.
- The CSR contains details about the server identity.

Step 3 — Sign the Certificate Using My CA

```
openssl ca -in webserver_server.csr -out webserver_server.crt -cert
ca.crt -keyfile ca.key -config openssl.cnf
```

- The CA private key (`ca.key`) was used to sign the CSR.
- The signed certificate (`webserver_server.crt`) proves the server’s authenticity.

Step 4 — Create Fullchain Certificate

```
cat webserver_server.crt ca.crt > webserver_fullchain.pem
```

- The fullchain certificate includes both the server certificate and the CA certificate.
- This ensures clients can verify the certificate chain.

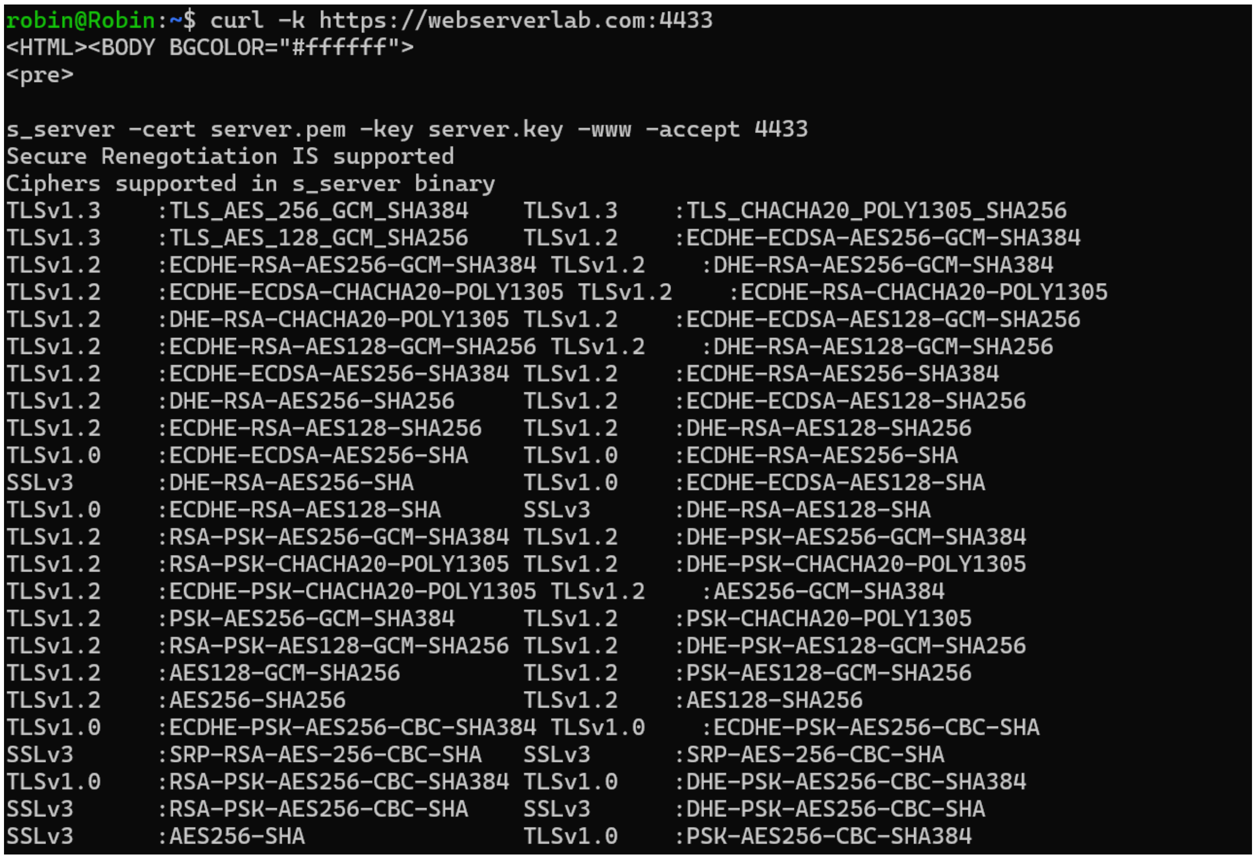
Step 5 — Start OpenSSL Test Server and Verify

```
openssl s_server -cert webserver_fullchain.pem -key
webserver_server.key -www -accept 4433
```

- The server listens on port **4433**.

Tested with `curl`:

```
curl -k https://webserverlab.com:4433
```



```
robin@Robin:~$ curl -k https://webserverlab.com:4433
<HTML><BODY BGCOLOR="#ffffff">
<pre>

s_server -cert server.pem -key server.key -www -accept 4433
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
TLSv1.2    :DHE-RSA-AES256-SHA256     TLSv1.2    :ECDHE-ECDSA-AES128-SHA256
TLSv1.2    :ECDHE-RSA-AES128-SHA256   TLSv1.2    :DHE-RSA-AES128-SHA256
TLSv1.0    :ECDHE-ECDSA-AES256-SHA    TLSv1.0    :ECDHE-RSA-AES256-SHA
SSLv3      :DHE-RSA-AES256-SHA        TLSv1.0    :ECDHE-ECDSA-AES128-SHA
TLSv1.0    :ECDHE-RSA-AES128-SHA      SSLv3      :DHE-RSA-AES128-SHA
TLSv1.2    :RSA-PSK-AES256-GCM-SHA384 TLSv1.2    :DHE-PSK-AES256-GCM-SHA384
TLSv1.2    :RSA-PSK-CHACHA20-POLY1305 TLSv1.2    :DHE-PSK-CHACHA20-POLY1305
TLSv1.2    :ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2    :AES256-GCM-SHA384
TLSv1.2    :PSK-AES256-GCM-SHA384     TLSv1.2    :PSK-CHACHA20-POLY1305
TLSv1.2    :RSA-PSK-AES128-GCM-SHA256 TLSv1.2    :DHE-PSK-AES128-GCM-SHA256
TLSv1.2    :AES128-GCM-SHA256         TLSv1.2    :PSK-AES128-GCM-SHA256
TLSv1.2    :AES256-SHA256             TLSv1.2    :AES128-SHA256
TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA384 TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA
SSLv3      :SRP-RSA-AES-256-CBC-SHA   SSLv3      :SRP-AES-256-CBC-SHA
TLSv1.0    :RSA-PSK-AES256-CBC-SHA384 TLSv1.0    :DHE-PSK-AES256-CBC-SHA384
SSLv3      :RSA-PSK-AES256-CBC-SHA    SSLv3      :DHE-PSK-AES256-CBC-SHA
SSLv3      :AES256-SHA                TLSv1.0    :PSK-AES256-CBC-SHA384
```

Checkpoint 3

I accessed the test server using Firefox with the custom profile:

```
firefox -profile ~/.mozilla/firefox/ssl-lab.default
https://example.com &
```

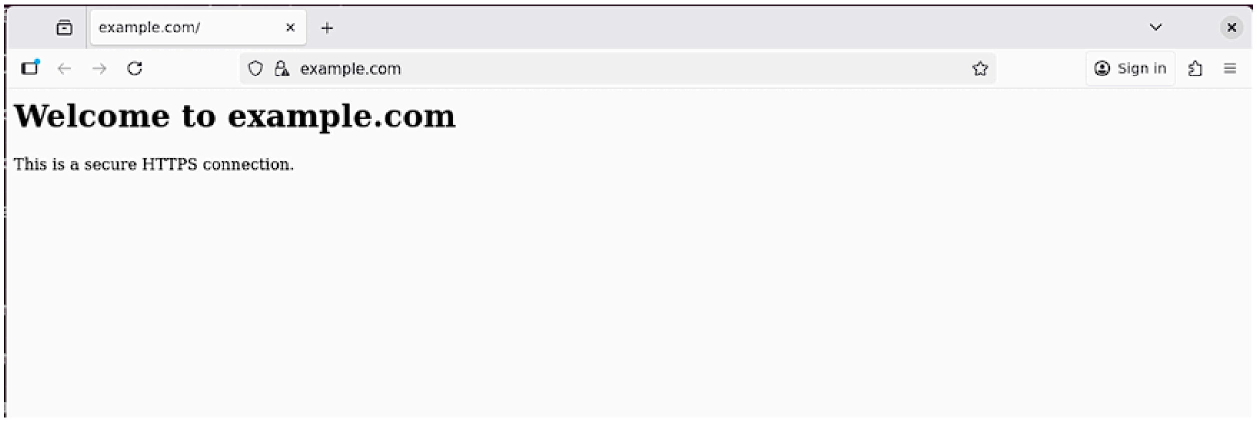
Observations

- Firefox opened the page successfully.
- Because the CA certificate (`ca.crt`) was imported into the profile, **no security warnings appeared**.

- The connection was established using **TLSv1.3**, ensuring encryption of all transmitted data.
- Both `https://example.com` and `https://localhost` point to the same server due to the `/etc/hosts` mapping:

```
127.0.0.1    example.com
```

- This confirms that the server certificate is **trusted** by Firefox and that the SSL/TLS configuration is correct.



Welcome to example.com

This is a secure HTTPS connection.