EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Course Title: Cybersecurity, Law and Ethics

Code: CSE487

Section: 02

Mini Project 1:

Securing a networked system

with Public Key Infrastructure

Submitted To:

Dr. Md. Hasanul Ferdous

Associate Professor

Department of Computer Science & Engineering

Submitted by

Name	ID
Mst. Mariam Khatun	2021-3-60-092
Lamia Binte Zaman	2021-3-60-066
Chaity Sutradhar	2020-3-60-089
Ishrat Jahan Ela	2022-1-60-379

Date of Submission : 09.08.2025

Video link: https://drive.google.com/file/d/10QY0os50WMpCy_MwyXGQ2y-JyHZHOCN1/view?usp=drivesdk

1. Introduction: Ensuring secure online communication is crucial for protecting sensitive information and verifying the authenticity of digital transactions. Public Key Infrastructure (PKI) and Transport Layer Security (TLS) are the core technologies that safeguard internet communications.

Public Key Infrastructure (PKI):

PKI is a security framework that employs cryptographic keys to encrypt and decrypt information, offering key services like authentication, encryption, and digital signatures. It uses a pair of keys—public and private—to secure data. Public keys, which are shared openly, are used for encryption, while private keys, kept confidential, are used for decryption. PKI relies on trusted Certificate Authorities (CAs) to issue digital certificates, which verify the identities of entities (e.g., servers or clients) and build a chain of trust.

- **Certificates:** Digital documents issued by a CA to validate the authenticity of public keys.
- **Certificate Authority (CA):** A reliable organization that issues and manages digital certificates.
- **Registration Authority (RA):** Supports the CA by verifying the identity of entities before certificates are granted.

PKI is vital for ensuring security, trust, and data integrity across networked systems by managing cryptographic keys and certificates. It prevents impersonation attacks by confirming the identity of users or devices and ensures that transmitted data remains unaltered. Digital signatures within PKI verify that the received data matches the original message, while non-repudiation provides accountability and legal assurance. **Transport Layer Security (TLS):**

TLS, which replaced the older Secure Sockets Layer (SSL), protects data exchanged between clients and web servers by encrypting it. This prevents eavesdropping, tampering, and impersonation,

ensuring the safe transmission of sensitive information like passwords, personal details, and financial data. TLS works alongside HTTP to form HTTPS, a secure protocol that is essential for online banking, e-commerce platforms, and services handling private information. By combining PKI for certificate management and TLS for encryption, HTTPS ensures secure connections, trust, and user privacy.

Requirements:

- Configuration of Certification Authority MariamCA with MariamRootCA as the RootCA.
- Configuration of the Web Server with Apache2 on a Linux Host.
- Certification process (Verification and Certificate Generation from CSR)
- Transferring the certificate from mariamCA to www.mariam.com
- Installation of the signed SSL certificate in the server of www.mariam.com
- Making the system trust Mariam-RootCA
- Implementation of a simple file uploading page in the server.
- Verifying the security of the connection by inspection (the padlock icon), and with Wireshark from another computer.

2.System Requirements and Setup

Necessary Elements:

- Oracle VM Virtualbox
- Ubuntu 24.0.4.2 LTS
- Firefox version 72.0(64-bit) and xamp

3. Configuration

3.1 Certification Authority Setup

1. Move to root directory using:

sudo -i

2. Create certification authority folder named ca, containing root-ca, sub-ca, server and each sub folder will contain private folder containing certs, newcerts, crl, csr folders.

mkdir -p ca/{root-ca,sub-ca,server}/{private,certs,newcerts,crl,csr}

See the tree of files inside the root:

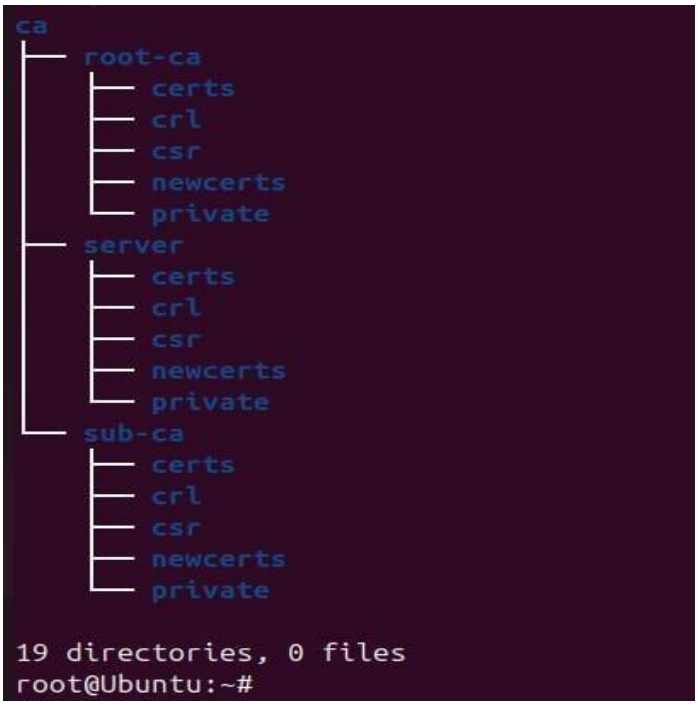
Then giving permission to edit and write in these folders. **chmod**

-v 700 ca/{root-ca,sub-ca,server}/private

```
root@rafia-VirtualBox:~# chmod -v 700 ca/{root-ca,sub-ca,server}/private
mode of 'ca/root-ca/private' changed from 0755 (rwxr-xr-x) to 0700 (rwx------)
mode of 'ca/sub-ca/private' changed from 0755 (rwxr-xr-x) to 0700 (rwx------)
mode of 'ca/server/private' changed from 0755 (rwxr-xr-x) to 0700 (rwx------)
root@rafia-VirtualBox:~# touch ca/{root-ca,sub-ca}/index
root@rafia-VirtualBox:~#
```

3.Creating file index in both root ca and sub ca to track certificate issuing and revoking.

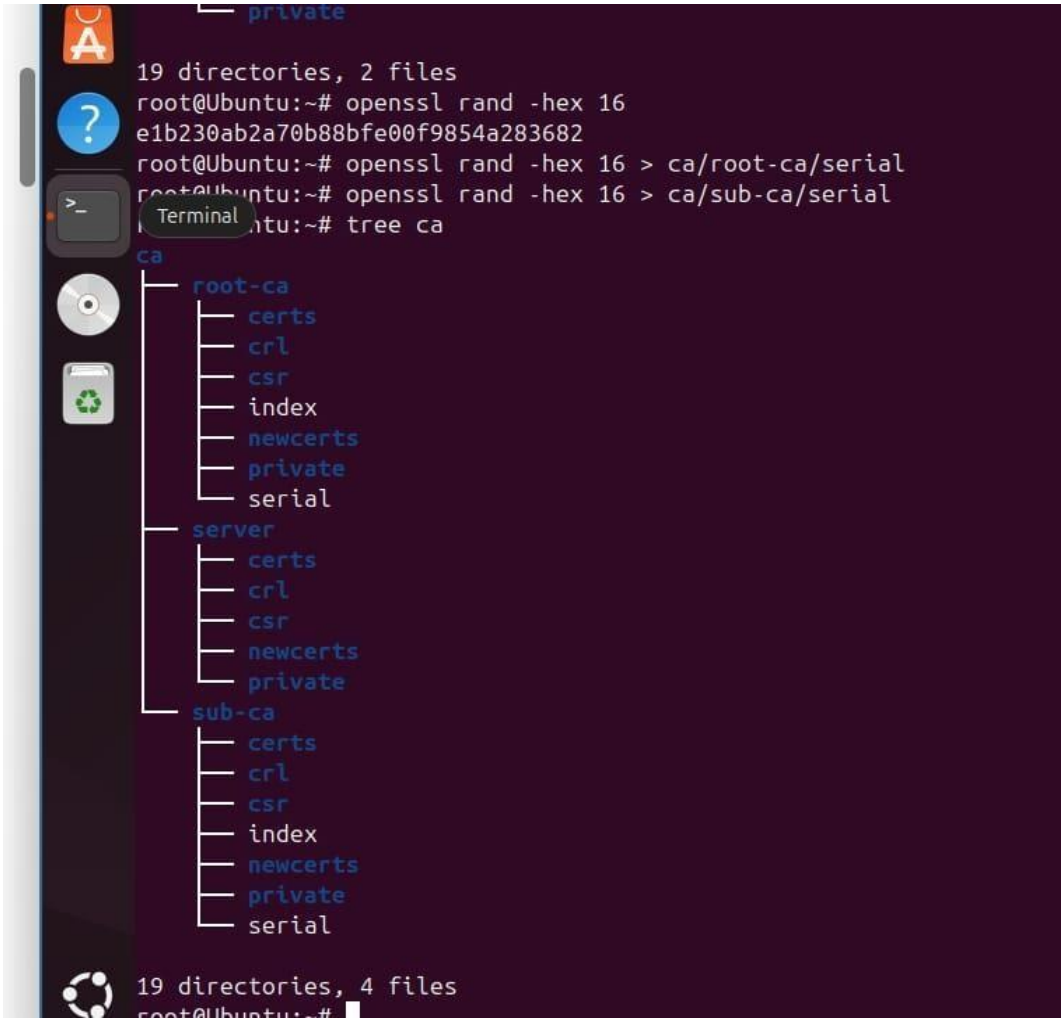
touch ca/{root-ca,sub-ca}/index



4. writing serial number of root ca and sub ca.

openssl rand -hex 16 > ca/root-ca/serial openssl

rand -hex 16 > ca/sub-ca/serial



5. Generating private key for root-ca, sub-ca and server

openssl genrsa -aes256 -out root-ca/private/ca.key 4096 openssl
genrsa -aes256 -out sub-ca/private/sub-ca.key 4096 openssl
genrsa -out server/private/server.key 2048

Public key for rootCA (key = root) and Public key for subCA (key = root), and server :

3.2 CSR Creation & Authority Signing

1. Now we will apply custom OpenSSL CA configuration file for both root-ca and sub-ca.

for root-ca:

[ca]

#/root/ca/root-ca/root-ca.conf

#see man ca default_ca

= CA_default

[CA_default] dir =

/root/ca/root-ca

certs = $dir/certs crl_dir

= $dir/crl

new_certs_dir = $dir/newcerts database

= $dir/index

serial = $dir/serial

RANDFILE = $dir/private/.rand private_key

= $dir/private/ca.key certificate =

```
$dir/certs/ca.crt crlnumber   =
$dir/crlnumber

crl    =  $dir/crl/ca.crl
crl_extensions   = crl_ext
default_crl_days      = 30
default_md   = sha256

name_opt   = ca_default
cert_opt     =  ca_default
default_days        =   365
preserve   = no policy    =
policy_strict

[  policy_strict  ]  countryName      =
supplied   stateOrProvinceName     =
supplied organizationName  = match
organizationalUnitName  =  optional
commonName   =  supplied

emailAddress          =       optional    [
policy_loose   ]   countryName       =
optional   stateOrProvinceName     =
optional  localityName     =  optional
organizationName         =     optional
organizationalUnitName   = optional
commonName              =     supplied
emailAddress   = optional

[ req ]
```

```
# Options for the req tool, man req.
default_bits   = 2048

distinguished_name  = req_distinguished_name
string_mask   = utf8only default_md   =  sha256

# Extension to add when the -x509 option is used.
x509_extensions     =  v3_ca  [  req_distinguished_name  ]
countryName                  = Country Name (2 letter code)
stateOrProvinceName           = State or Province Name
localityName                 = Locality Name 0.organizationName
= Organization Name organizationalUnitName         =
Organizational Unit Name commonName                  =
Common Name
emailAddress                 = Email Address
countryName_default  = BD

stateOrProvinceName_default = Dhaka localityName_default
= Banasree

0.organizationName_default = EWU
organizationalUnitName_default = Cyber_Security
commonName_default = Mariam emailAddress_default
= mariam@mariamroot_ca.com

[ v3_ca ]
# Extensions to apply when createing root ca # Extensions
for a typical CA, man x509v3_config subjectKeyIdentifier  =
hash     authorityKeyIdentifier         =     keyid:always,issuer
```

basicConstraints = critical, CA:true keyUsage = critical,

digitalSignature, cRLSign, keyCertSign

[v3_intermediate_ca]

Extensions to apply when creating intermediate or sub-ca

Extensions for a typical intermediate CA, same man as above

subjectKeyIdentifier = hash authorityKeyIdentifier =

keyid:always,issuer

#pathlen:0 ensures no more sub-ca can be created below an intermediate

basicConstraints = critical, CA:true, pathlen:0 keyUsage = critical,

digitalSignature, cRLSign, keyCertSign

[server_cert]

Extensions for server certificates basicConstraints =

CA:FALSE nsCertType = server nsComment = "OpenSSL

Generated Server Certificate" subjectKeyIdentifier =

hash authorityKeyIdentifier = keyid,issuer:always

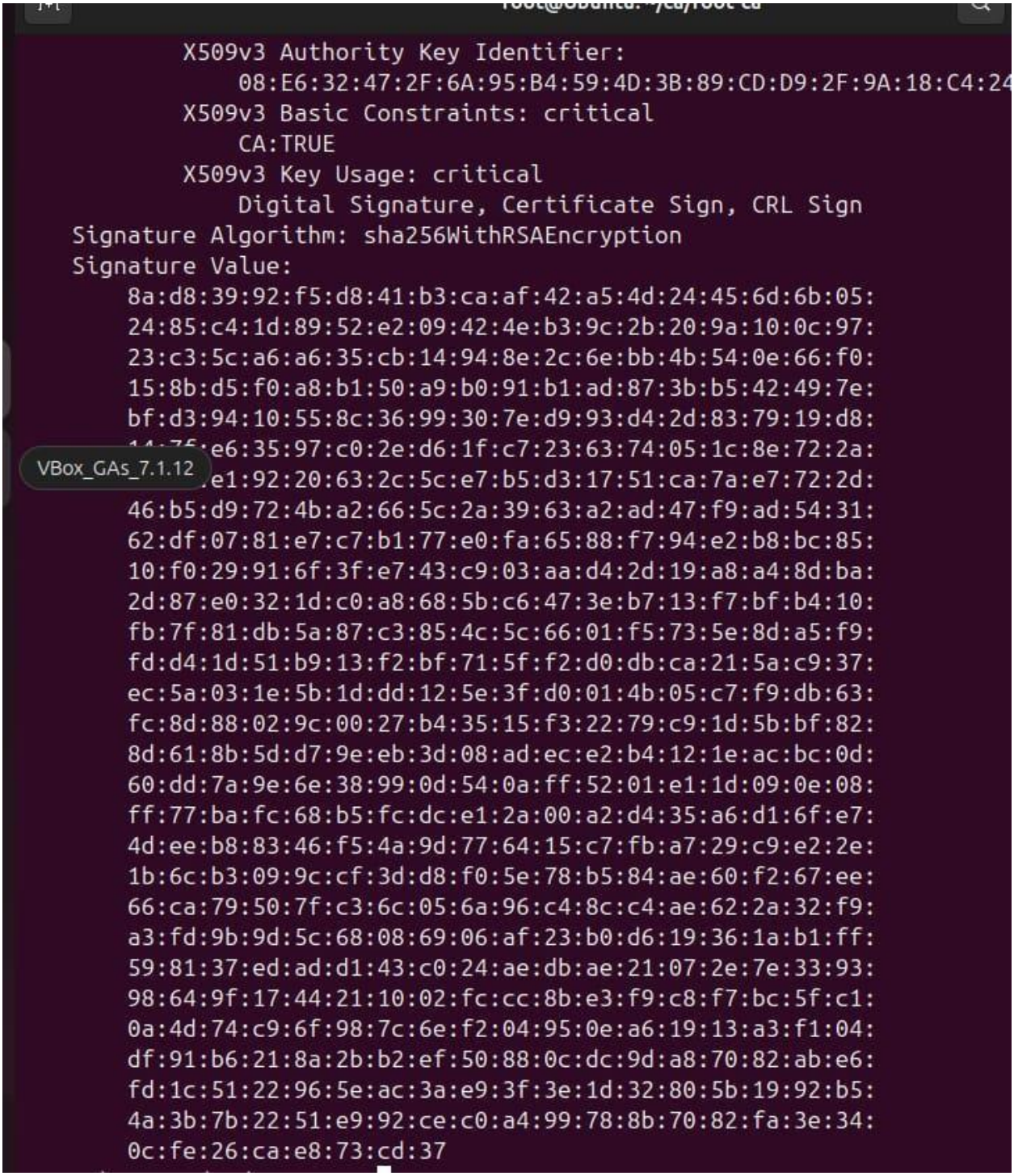
keyUsage = critical, digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth

2. Now generate root-ca certificate :

/root-ca openssl req -config root-ca.conf -key private/ca.key -new -x509 -days 7305 sha256 -extensions v3_ca -out certs/ca.crt openssl x509 -noout -in certs/ca.crt -text to check the certificate:

```
19 directories, 8 files
root@Ubuntu:~/ca# cd root-ca
root@Ubuntu:~/ca/root-ca# openssl req -config root-ca.conf -key private/ca.key -new -x5
09 -days 7305 -sha256 -extensions v3_ca -out certs/ca.crt
Enter pass phrase for private/ca.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) [BD]:
State or Province Name [Dhaka]:
Locality Name [Banasree]:
Organization Name [EWU]:
Organizational Unit Name [Cyber_Security]:
Common Name [Mariam]:
Email Address [mariam@mariamroot_ca.com]:
root@Ubuntu:~/ca/root-ca# openssl x509 -noout -in certs/ca.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5a:d3:19:6a:db:d6:fa:1d:f5:b1:12:17:a6:9c:fd:bb:36:6b:e4:2c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = BD, ST = Dhaka, L = Banasree, O = EWU, OU = Cyber_Security, CN = Ma
riam, emailAddress = mariam@mariamroot_ca.com
        Validity
            Not Before: Aug  8 23:43:28 2025 GMT
            Not After : Aug  8 23:43:28 2045 GMT
        Subject: C = BD, ST = Dhaka, L = Banasree, O = EWU, OU = Cyber_Security, CN = M
ariam, emailAddress = mariam@mariamroot_ca.com
        Subject Public Key Info:
```



[ca]

#/root/ca/sub-ca/sub-ca.conf

#see man ca default_ca

= CA_default

```
[CA_default] dir     =
/root/ca/sub-ca certs     =
$dir/certs crl_dir    =
$dir/crl
new_certs_dir   = $dir/newcerts database
= $dir/index
serial    = $dir/serial
RANDFILE   = $dir/private/.rand

private_key   = $dir/private/sub-ca.key
certificate   = $dir/certs/sub-ca.crt crlnumber
= $dir/crlnumber
crl        =     $dir/crl/ca.crl
crl_extensions   = crl_ext
default_crl_days    = 30

default_md      =  sha256
name_opt   = ca_default
cert_opt     =  ca_default
default_days        =   365
preserve   = no policy    =
policy_loose
[ policy_strict ]

countryName               =      supplied
stateOrProvinceName    =    supplied
organizationName          =      match
organizationalUnitName  =  optional
commonName   =  supplied
```

```
emailAddress   =  optional

[  policy_loose  ]  countryName      =
optional   stateOrProvinceName     =
optional  localityName     =  optional
organizationName         =     optional
organizationalUnitName   = optional
commonName   = supplied
emailAddress   = optional

[ req ]
# Options for the req tool, man req.
default_bits      =  2048  distinguished_name    =
req_distinguished_name    string_mask          =
utf8only default_md   =  sha256
# Extension to add when the -x509 option is used.
x509_extensions   = v3_ca

[ req_distinguished_name ]
countryName                  = Country Name (2 letter code) stateOrProvinceName
= State or Province Name
localityName                 = Locality Name
0.organizationName           = Organization Name
organizationalUnitName       = Organizational Unit Name
commonName                   = Common Name emailAddress
= Email Address countryName_default  = BD
stateOrProvinceName_default = Dhaka
localityName_default = Banasree
0.organizationName_default = EWU
```

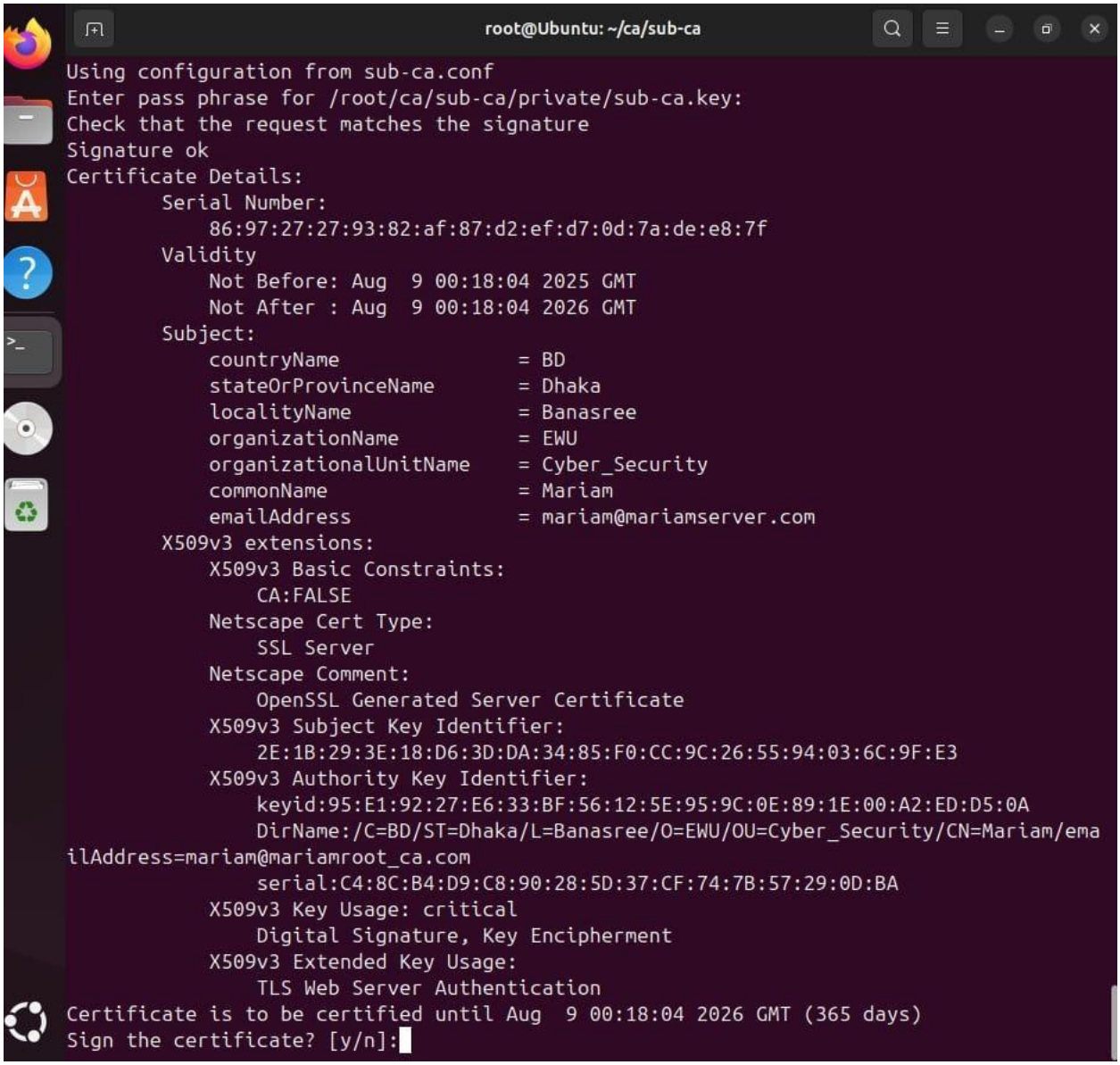
```
organizationalUnitName_default = Cyber_Security
commonName_default  = Mariam emailAddress_default =
mariam@mariamsub_ca.com
[ v3_ca ]
# Extensions to apply when createing root ca # Extensions
for a typical CA, man x509v3_config subjectKeyIdentifier  =
hash  authorityKeyIdentifier  =  keyid:always,issuer
basicConstraints  = critical, CA:true keyUsage  =  critical,
digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions to apply when creating intermediate or sub-ca #
Extensions for a typical intermediate CA, same man as above
subjectKeyIdentifier  =  hash  authorityKeyIdentifier  =
keyid:always,issuer

#pathlen:0 ensures no more sub-ca can be created below an intermediate
basicConstraints  = critical, CA:true, pathlen:0 keyUsage  = critical,
digitalSignature, cRLSign, keyCertSign
[ server_cert ]

# Extensions for server certificates
basicConstraints  =  CA:FALSE
nsCertType  = server

nsComment  =  "OpenSSL Generated Server Certificate"
subjectKeyIdentifier  =  hash  authorityKeyIdentifier  =
keyid,issuer:always  keyUsage  =  critical,
```

digitalSignature, keyEncipherment extendedKeyUsage =

serverAuth

3. Requesting for sub-ca certificate signing request. **openssl req -config sub-ca.conf -new -key private/sub-ca.key -sha256 -out csr/sub-ca.csr**

4. Signing the request of sub-ca by root ca



5. Root-ca certificate signed:

```
        df:91:b6:21:8a:2b:b2:ef:50:88:0c:dc:9d:a
        fd:1c:51:22:96:5e:ac:3a:e9:3f:3e:1d:32:8
        4a:3b:7b:22:51:e9:92:ce:c0:a4:99:78:8b:7
        0c:fe:26:ca:e8:73:cd:37
root@Ubuntu:~/ca/root-ca# cd ../sub-ca
root@Ubuntu:~/ca/sub-ca# gedit sub-ca.conf
```

6. Generating certificate signing request from server:

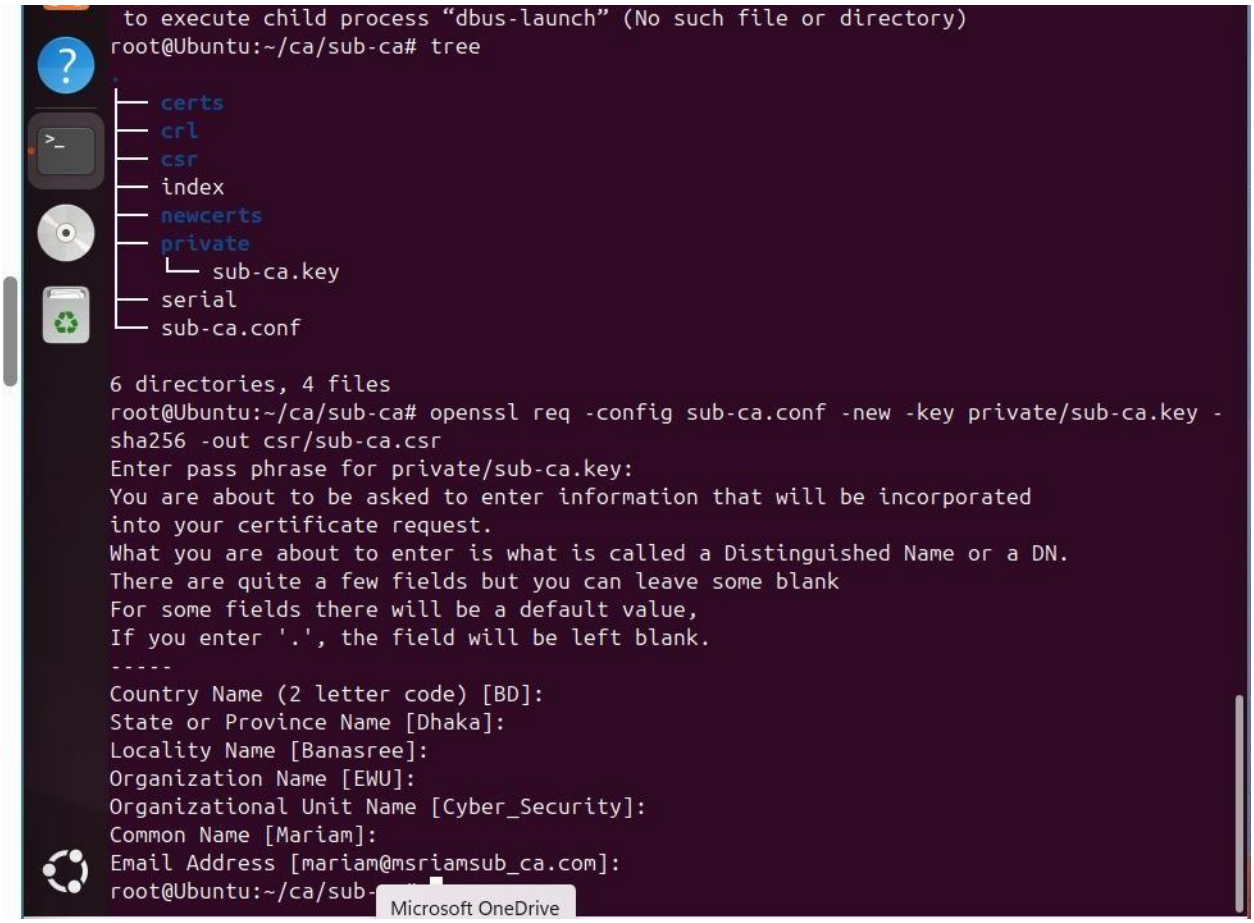
openssl req -key private/server.key -new -sha256 -out csr/server.csr

```
 to execute child process "dbus-launch" (No such file or directory)
root@Ubuntu:~/ca/sub-ca# tree
.
├── certs
├── crl
├── csr
├── index
├── newcerts
├── private
│   └── sub-ca.key
├── serial
└── sub-ca.conf

6 directories, 4 files
root@Ubuntu:~/ca/sub-ca# openssl req -config sub-ca.conf -new -key private/sub-ca.key -
sha256 -out csr/sub-ca.csr
Enter pass phrase for private/sub-ca.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) [BD]:
State or Province Name [Dhaka]:
Locality Name [Banasree]:
Organization Name [EWU]:
Organizational Unit Name [Cyber_Security]:
Common Name [Mariam]:
Email Address [mariam@msriamsub_ca.com]:
root@Ubuntu:~/ca/sub-
```

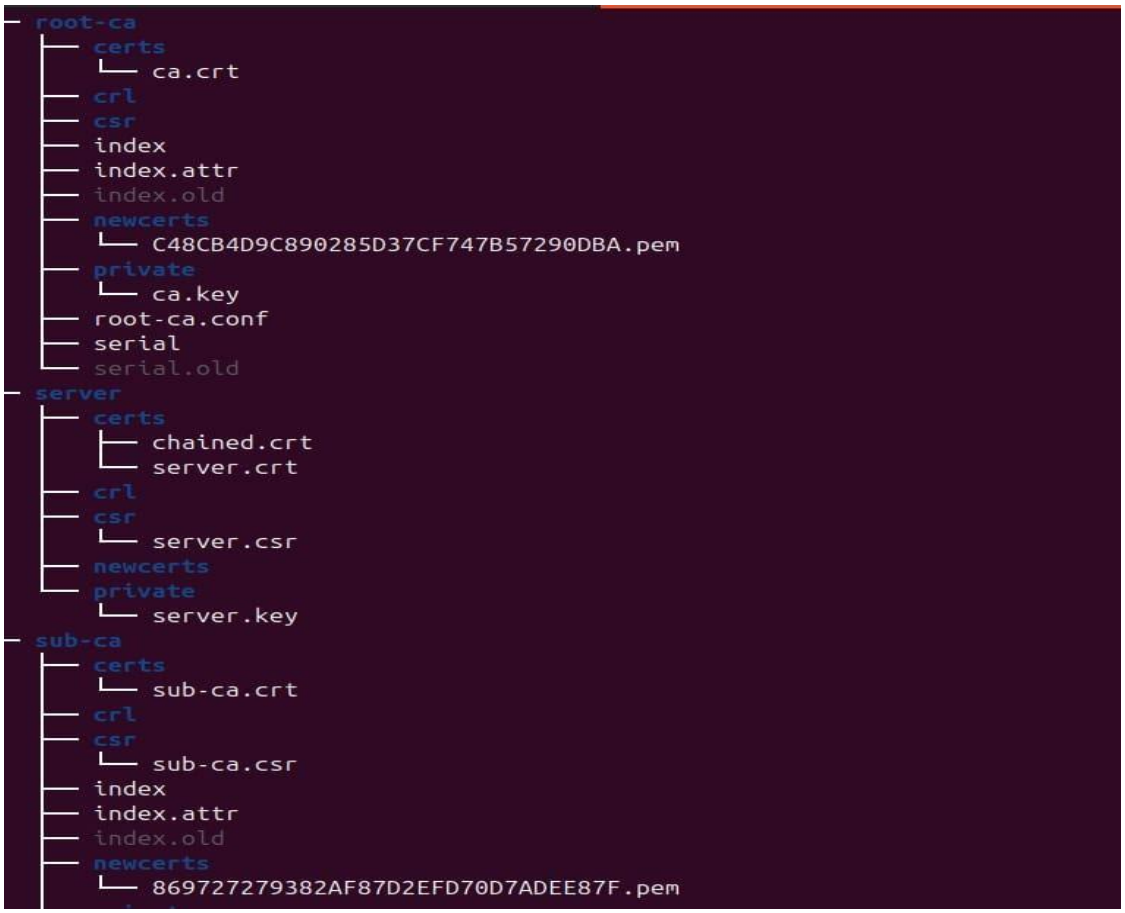
Microsoft OneDrive

7. Signing server certificate from sub-ca

openssl ca -config sub-ca.conf -extensions server_cert -days 365 -notext - in ../server/csr/server.csr -out ../server/certs/server.crt



8. Finally, the ca folder will look like this:

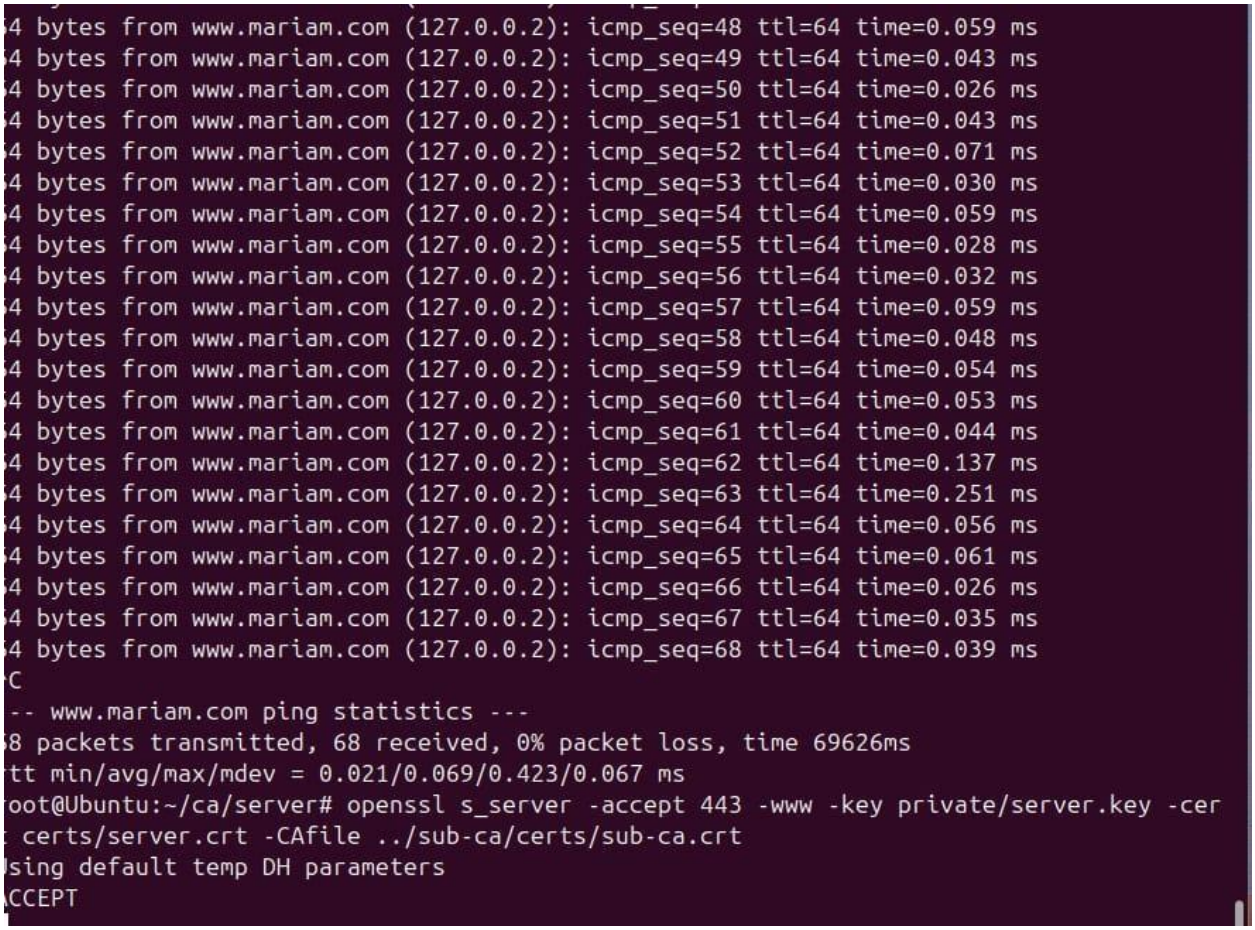


3.3 Web Server setup

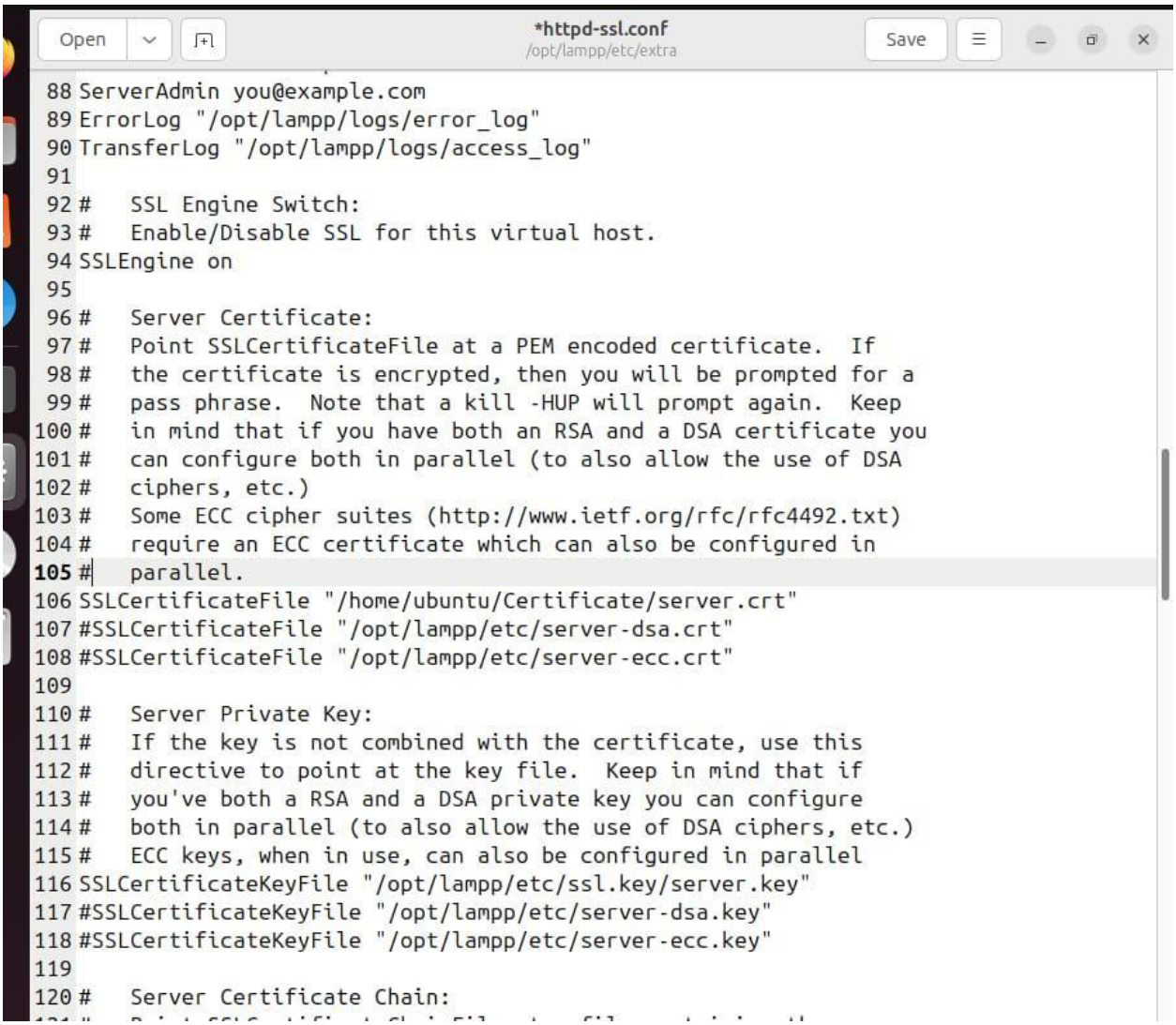
1. Configure host file to recognize local ip 127.0.0.2 as www.mariam.com.

echo "127.0.0.2 www.mariam.com" >> /etc/hosts

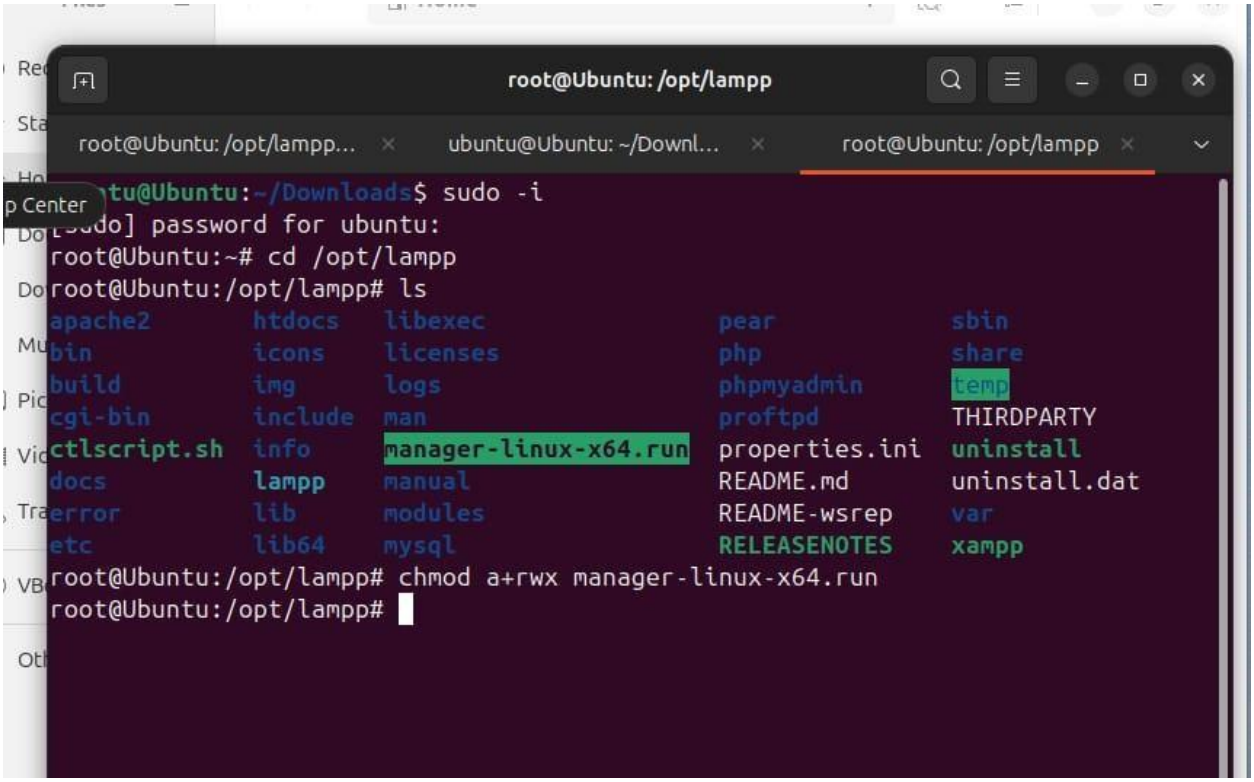
ping www.mariam.com to see if host is saved successfully

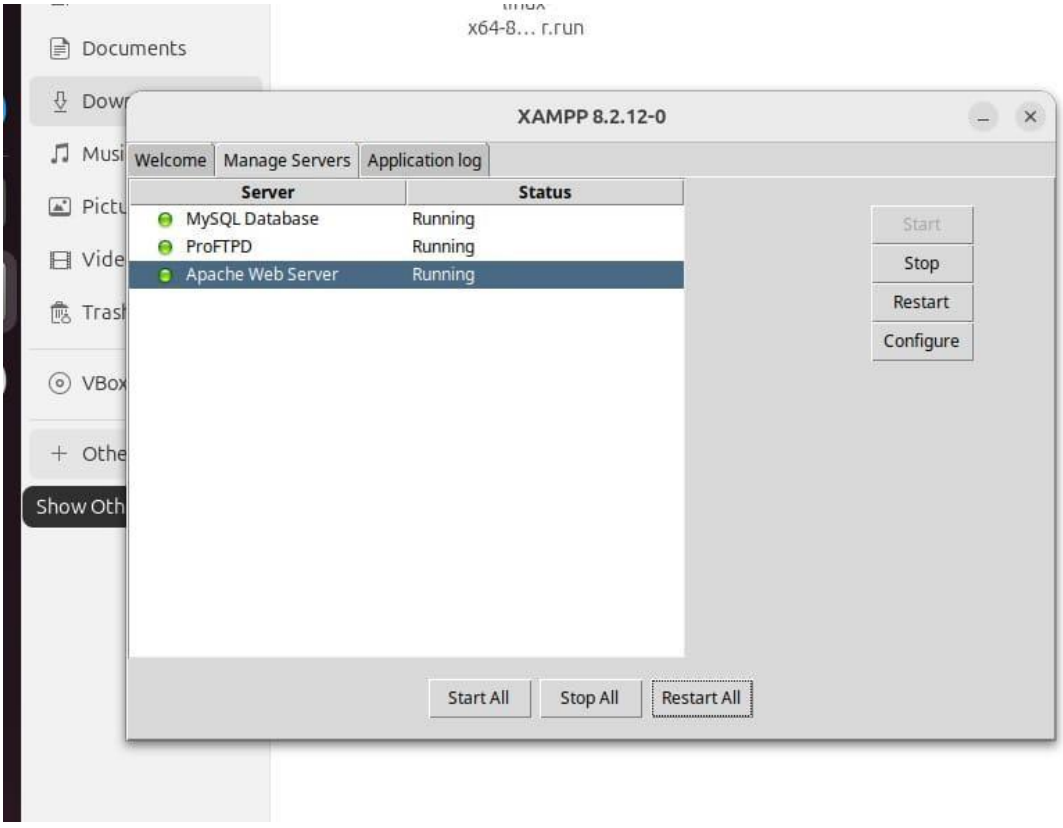


3. Now save the certificates and pem files in separate folders.
4. Install XAMPP Now we need to change the default path of XAMPP to our custom certificates and key path. Navigate to default path and put our newly created path where we copied the certificates.

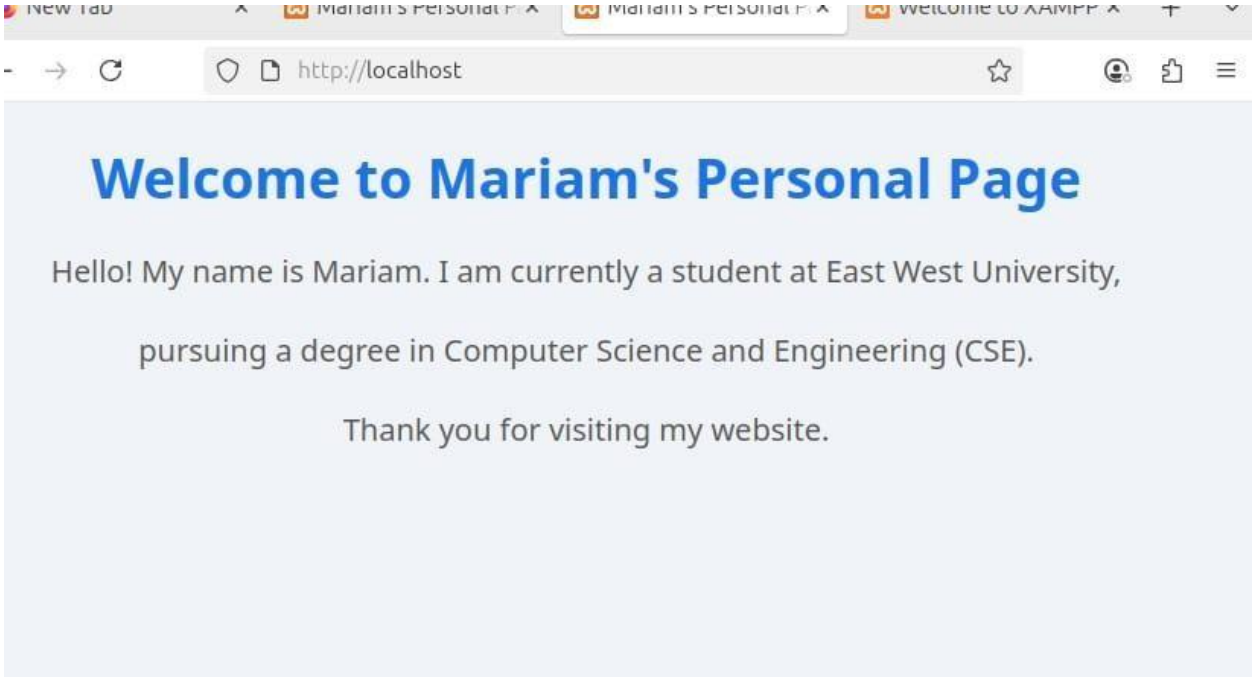


5. Open xampp. Then start server:



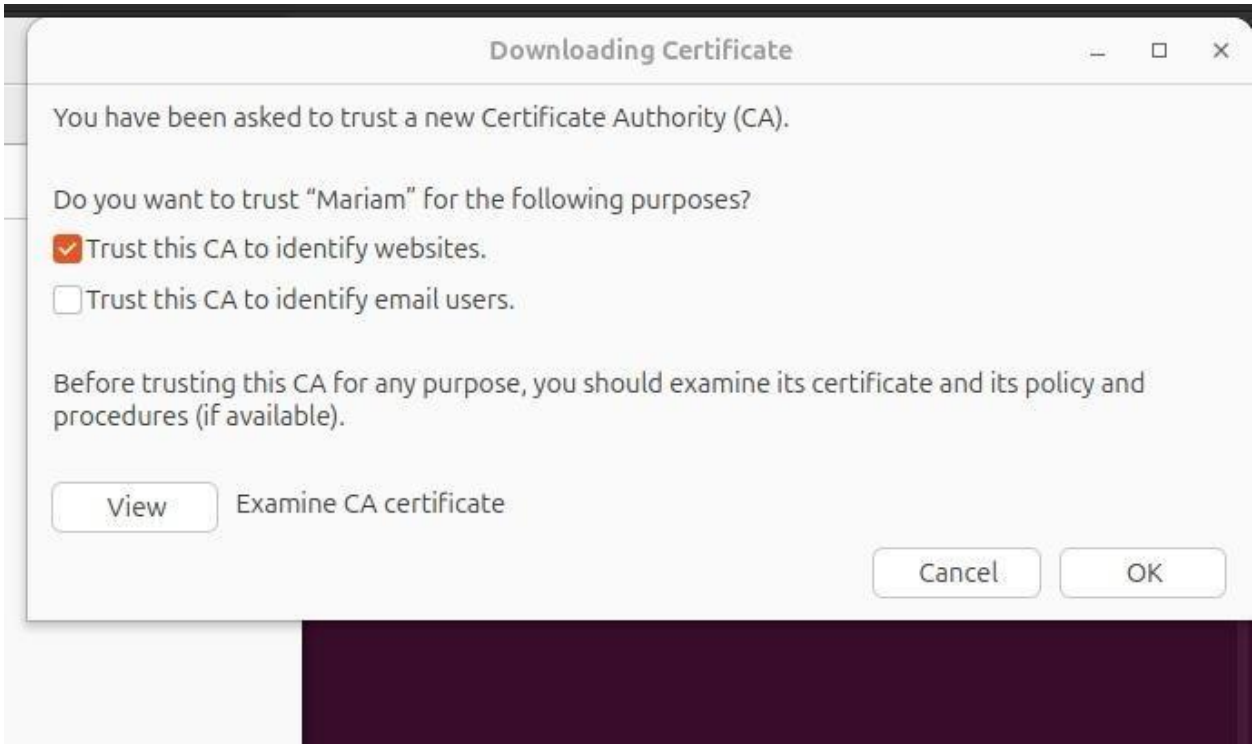


6. Change default server homepage to custom homepage (Optional).



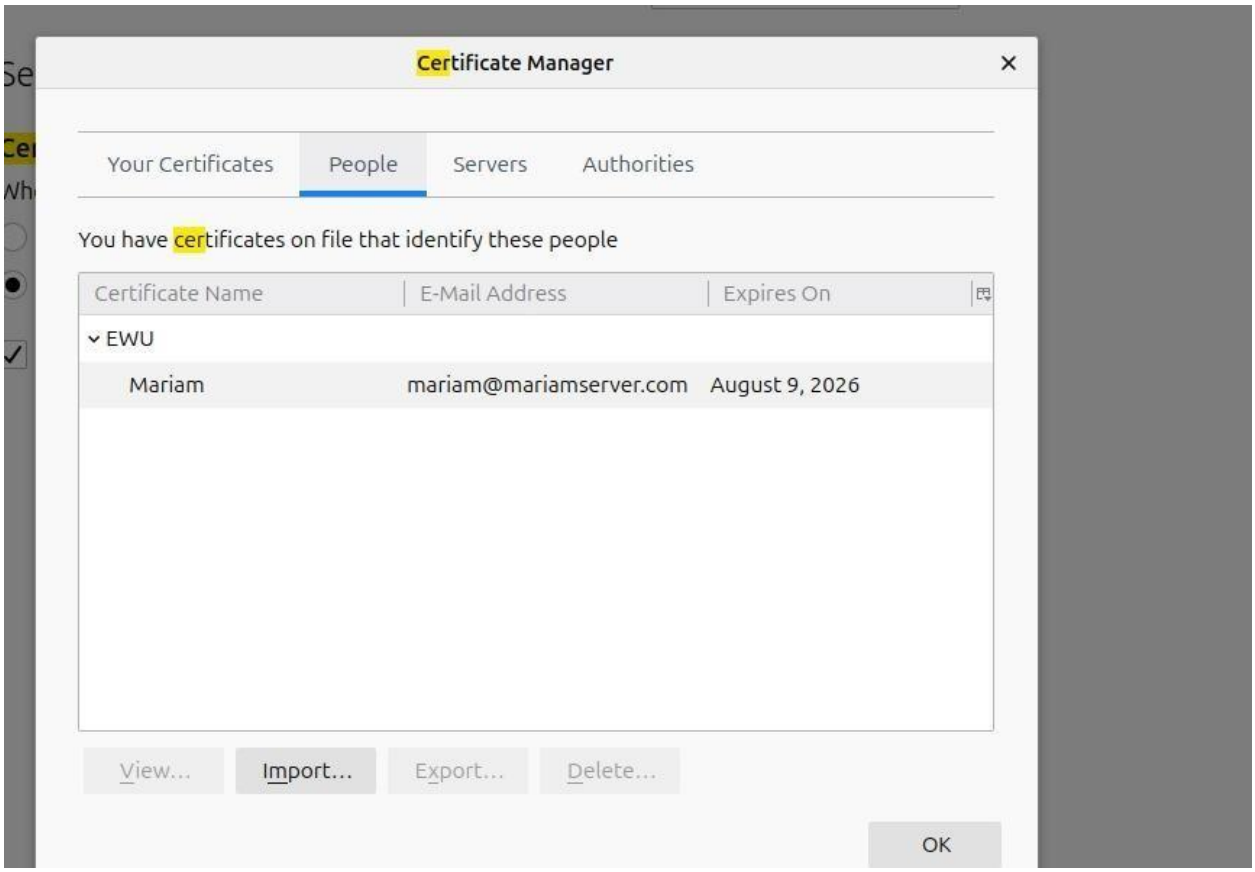
1. Import the custom certificate.

Go to **setting>security>manage certificates>Authorities>import**



Check trust this certificates:

Import the server (chained) certificate:



The server will have this certificate:

Version	3
Download	PEM (cert) PEM (chain)

Fingerprints

SHA-256	EF:AA:1B:34:AA:13:E5:C5:8F:59:7A:C8:7C:E4:FF:6F:DE:68:D1:24:50:E3:DA:70…
SHA-1	30:95:1C:85:CF:59:A2:C5:A9:EE:3D:1C:26:A7:6D:A4:5E:C4:3C:03

Basic Constraints

Certificate Authority	No

Key Usages

Purposes	Digital Signature, Key Encipherment

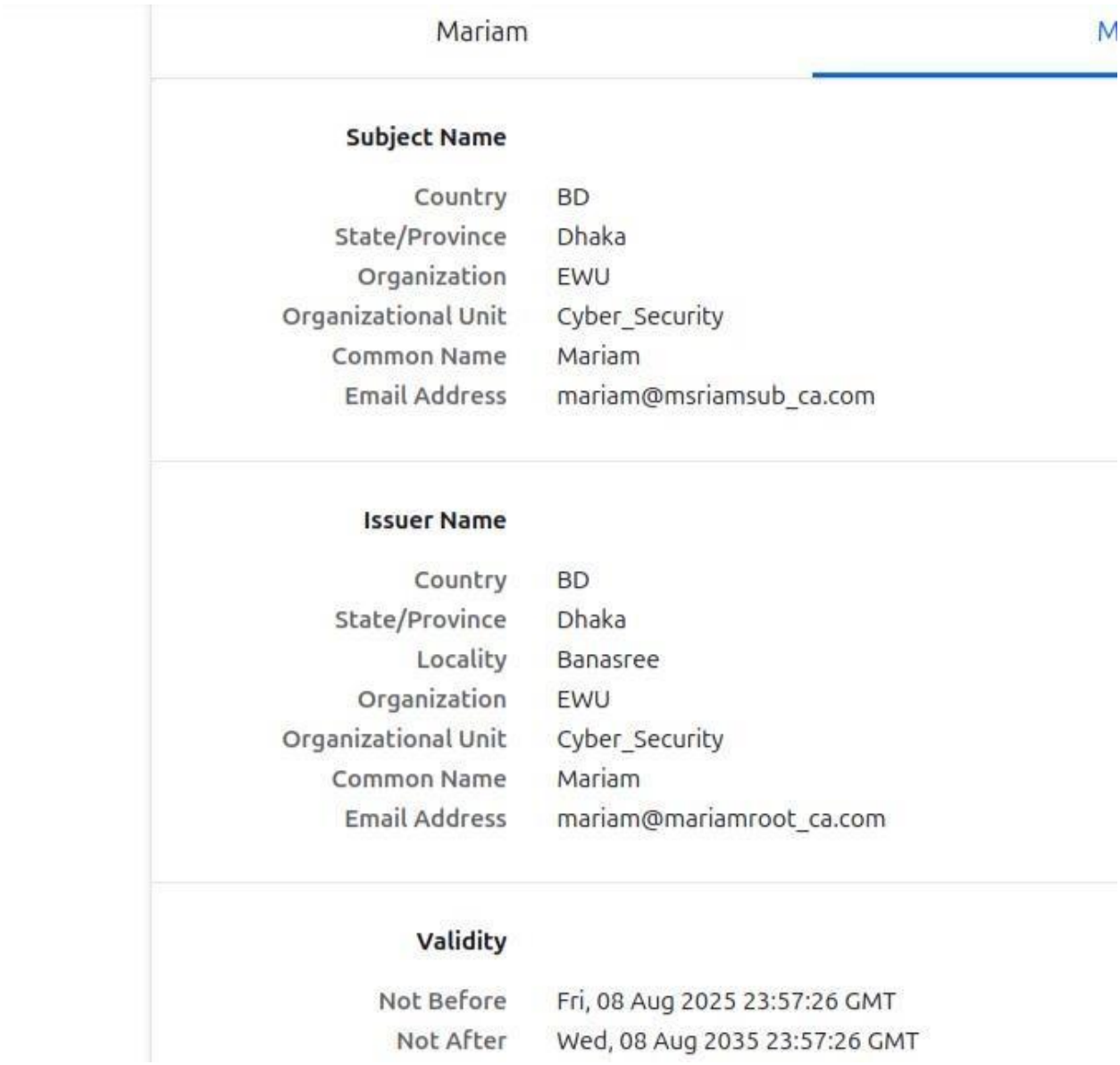
Extended Key Usages

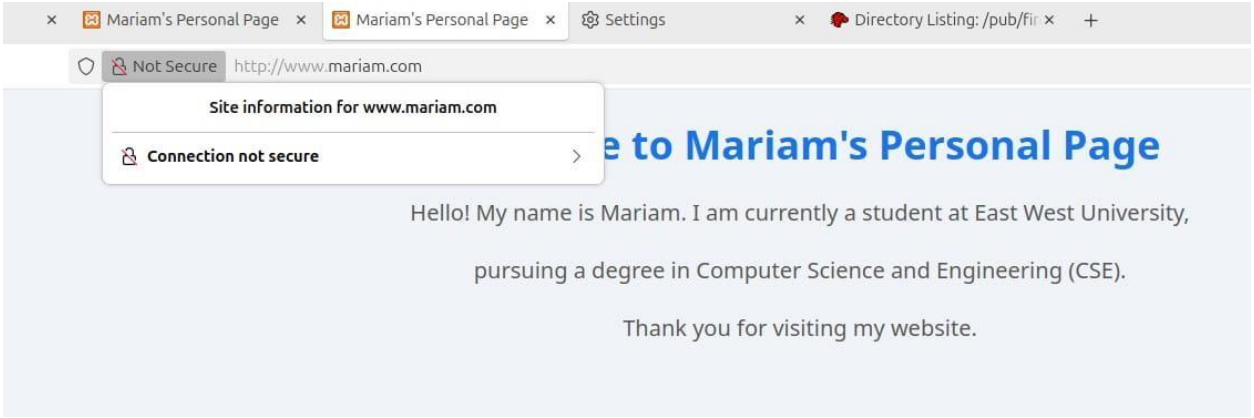
Purposes	Server Authentication

Subject Key ID

Key ID	2E:1B:29:3E:18:D6:3D:DA:34:85:F0:CC:9C:26:55:94:03:6C:9F:E3

Which was signed by acmeRoot-ca:





7. Lessons Learned

Throughout this mini project on secure network systems and cybersecurity, we gained valuable hands-on experience in building and securing a networked environment. Our first major task was creating a Certification Authority (CA), which allowed us to understand how Public Key Infrastructure (PKI) works in practice. We learned the full process of generating digital certificates, signing them, and establishing trust between clients and servers.

We then implemented Transport Layer Security (TLS) to secure HTTP traffic, ensuring that all data exchanged between the client and server was encrypted and protected from interception or tampering.

This project gave us a deeper understanding of how different components of a secure network fit together, improved our technical skills, and enhanced our problem-solving abilities in real-world cybersecurity scenarios.

8. Future Work

- While our current system operates successfully, we identified several improvements to make it more secure, efficient, and scalable:

- Automation: Use Certbot or Ansible to automate certificate issuing, renewal, and revocation.

- Cloud Deployment: Move the system to AWS or Azure for better scalability, availability, and redundancy.

- Stronger Threat Protection: Introduce more layers of IDS/IPS to detect and prevent a wider range of attacks.

- Web Application Hardening: Implement security headers, secure cookies, and rate-limiting to reduce attack surfaces.

- Advanced Monitoring: Integrate Snort logs with a Security Information and Event Management (SIEM) platform for centralized monitoring and faster incident response.

- Authentication Improvements: Add certificate-based or token-based authentication to control access more securely.

- Vulnerability Management: Conduct regular scans with OpenVAS or Nessus to find and fix security weaknesses.

- These changes would improve the system's security posture, reliability, and adaptability to evolving threats.

9. Conclusion

In this mini project, we successfully built and deployed a secure web server supported by PKI and TLS. We established our own Certification Authority, generated and signed certificates, and deployed the server with HTTPS enabled. To strengthen security, we implemented an IDS using Snort and tested the system against DoS attacks. Finally, we demonstrated certificate revocation to ensure that compromised or outdated credentials are promptly removed from the trust chain.

This project not only improved our understanding of secure communication and network defense but also reinforced the importance of ethical responsibility in safeguarding digital systems. It has given us the knowledge and confidence to design, implement, and improve secure network architectures in future projects.