

Information Security

Assignment # 03

RSA and PKI Cryptography Lab in SEED's Lab

Mam Hina Binte Haq

Course Instructor

CS- 3002

SE- S

Due Date: April 02, 2023

Group Members:

Zeeshan Ali 20i-2465

Ans Zeeshan 20i-0543

Assignment # 03

Part 02

Task-1: Becoming a Certificate Authority

Step 1: The Configuration File openssl.conf

We use OpenSSL to create certificates and the following provides the configuration setup for creating and issuing certificates:

```
[04/01/23]seed@VM:~/labpart2$ cd ..
[04/01/23]seed@VM:~$ cd Desktop
[04/01/23]seed@VM:~/Desktop$ mkdir labpart2
[04/01/23]seed@VM:~/Desktop$ cd labpart2
[04/01/23]seed@VM:~/.../labpart2$ cp /usr/lib/ssl/openssl.cnf openssl.cnf
[04/01/23]seed@VM:~/.../labpart2$ ls
openssl.cnf
[04/01/23]seed@VM:~/.../labpart2$ mkdir part2
[04/01/23]seed@VM:~/.../labpart2$ cd part2/
[04/01/23]seed@VM:~/.../part2$ mkdir certs crl newcerts
[04/01/23]seed@VM:~/.../part2$ touch index.txt serial
[04/01/23]seed@VM:~/.../part2$ echo 1000 > serial
[04/01/23]seed@VM:~/.../part2$ ls
certs crl index.txt newcerts serial
[04/01/23]seed@VM:~/.../part2$ cd ..
[04/01/23]seed@VM:~/.../labpart2$ ls
openssl.cnf part2
```

Step 2: Certificate Authority (CA)

Now, we generate a self-signed certificate for our CA that will serve as the root certificate as follows:

```
[04/01/23]seed@VM:~/.../labpart2$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Generating a RSA private key
.....+
.....+
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:New-York
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Syracuse
Organizational Unit Name (eg, section) []:InternetSecurity
Common Name (e.g. server FQDN or YOUR name) []:ZeshanAli
Email Address []:zeeshanali10771@gmail.com
[04/01/23]seed@VM:~/.../labpart2$
```

The above provides a screenshot of the entered information. The output of the command is stored in two files: ca.key and ca.crt (as provided in the command). The file ca.key contains the CA's private key, while ca.crt contains the public-key certificate. The following provides the content of ca.crt:

ZeshanAli

Identity: ZeshanAli
Verified by: ZeshanAli
Expires: 05/01/2023

Details

Subject Name

C(Country): US
ST(State): New-York
L(Locality): Syracuse
O(Organization): Syracuse
OU(Organizational Unit): InternetSecurity
CN(Common Name): ZeshanAli
EMAIL(Email Address): zeshanali10771@gmail.com

Issuer Name

C(Country): US
ST(State): New-York
L(Locality): Syracuse
O(Organization): Syracuse
OU(Organizational Unit): InternetSecurity
CN(Common Name): ZeshanAli
EMAIL(Email Address): zeshanali10771@gmail.com

Issued Certificate

Version: 3
Serial Number: 55 1F 95 00 03 68 E7 71 47 9B 55 2D FA A3 62 DB 6C AD 95 FE
Not Valid Before: 2023-04-01
Not Valid After: 2023-05-01

Certificate Fingerprints

SHA1: 68 B3 ED DD F4 BD 0B 4B 04 08 46 35 9D AF 9D FF 6B DA 51 C2
MD5: 59 6B FC 55 C9 61 68 3C A3 70 09 C9 50 4B B3 5D

ca.cart

Public Key Info

Key Algorithm: RSA
Key Parameters: 05 00
Key Size: 2048
Key SHA1 Fingerprint: 97 64 B8 17 EF D2 A8 3C 4F A4 E7 21 3F F4 E8 EF F3 03 CE 2B
Public Key: 30 82 01 0A 02 82 01 01 00 BB 06 F7 02 EA DA BB CE 14 AA 60 11 72 45 96 A2 DC 2B 5F C3 AA DC C4 5E 18 60 A4 B9 4D 6C 3B 9D 90 50 7C 4F 17 7C 0C 7A 7B A1 79 10 DB B0 FC C6 09 F9 12 57 22 75 D7 C5 0E DB 88 8A 3E E9 8B B2 30 C9 E2 C4 39 0D 17 5A D5 77 C4 EA C3 C1 A1 61 B4 69 B4 E5 52 85 4B 60 E3 C6 56 95 88 16 07 0B DA 37 56 55 76 20 68 A8 9E 01 BF B2 C3 6E 2F 80 1F 03 79 43 87 D1 68 8C BD A6 52 7D 64 0E 5E 87 D6 76 DD 60 39 DC 3E 87 4A FB 92 F6 CD AB 96 9F 7A F0 4F 49 1D B3 53 FA 48 DF 10 0C EB DA AD 49 BC 10 4C 56 47 BB C0 20 55 AD 3A E8 33 11 BA DB 2A 62 08 7A 0E 04 6B 4B FA 8B F3 8C F3 E7 7B 29 6B E9 5B D8 C5 78 06 03 4F E8 CF 05 CE 4E DC BA 37 75 93 99 A8 49 46 6D 8B F0 A0 61 FB 7C 3E F0 4B 35 E0 CA A3 CF 97 1D 12 FD F1 26 15 FC 93 8B FB FC 71 05 26 AF 3C C1 A8 95 9A 05 95 34 97 02 03 01 00 01

Subject Key Identifier

Key Identifier: 01 5D DD 0F 80 09 4E 6B BB DC 53 CC 14 2E 08 78 5E E8 84 29
Critical: No

Extension

Identifier: 2.5.29.35
Value: 30 16 80 14 01 5D DD 0F 80 09 4E 6B BB DC 53 CC 14 2E 08 78 5E E8 84 29
Critical: No

Basic Constraints

Certificate Authority: Yes
Max Path Length: Unlimited
Critical: Yes

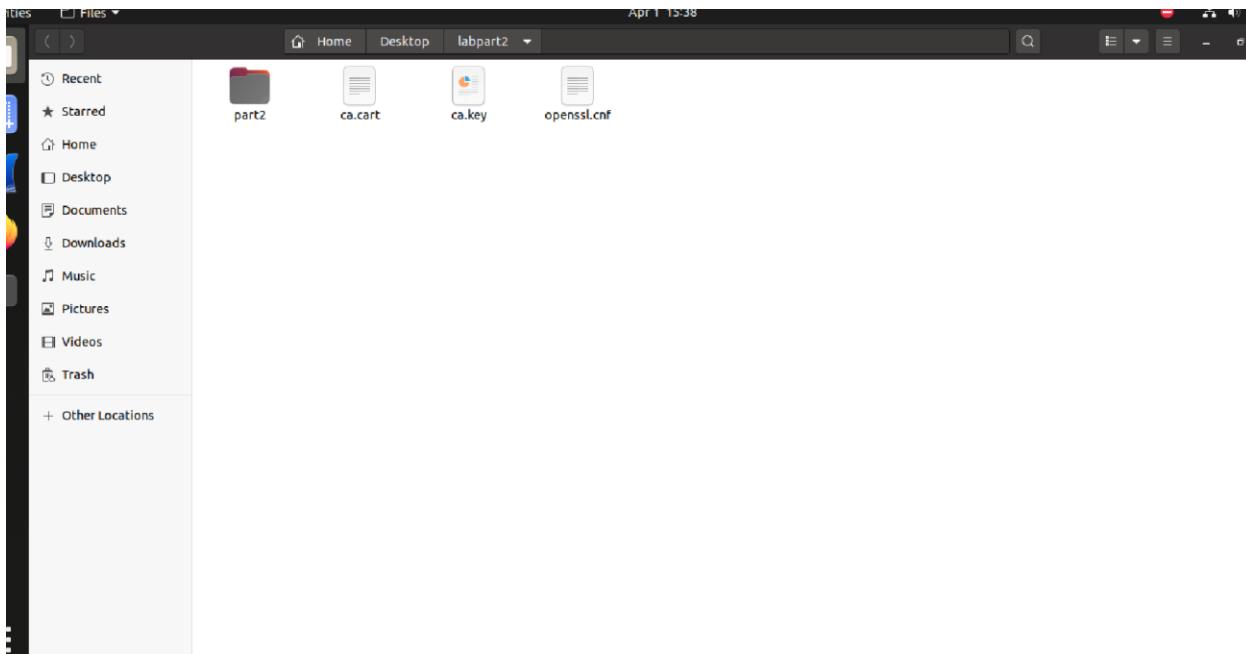
Signature

Signature Algorithm: 1.2.840.113549.1.1.11
Signature Parameters: 05 00
Signature: 39 00 35 70 60 CE A8 29 22 63 27 CF 50 3B E4 AA 49 67 CB CF CB EB A8 E8 D6 8D 6A C8 54 09 05 1C FC B3 D2 96 7A 4B 96 7E 37 19 10 D9 65 4D 00 E5 D7 26 30 8F 4D 76 37 3A CB F6 9B DD DF 30 7D A4 61 A1 37 D4 86 2B 55 03 A1 A1 34 AB B3 95 8A EE C9 AD 41 F3 C6 E8 5F F0 23 0B 4F E6 13 77 E1 12 06 AE A9 35 40 D2 F7 87 65 06 72 B8 F7 11 7B A6 5A F0 2F F6 28 D2 B8 30 09 09 49 15 A9 CC 58 51 2C 3A DD 00 18 C7 60 BB E1 60 25 FE D3 82 8D AE 1F D5 D1 9C 44 2A 23 CC 53 16 2F 0E C9 F1 AC F1 3E CF E5 2A 90 64 26 F6 DE C7 6F 53 F5 7F 98 36 39 AC 09 59 A6 1E D9 84 69 9E 7E F1 A6 55 B9 85 63 69 AC AC 53 83 C3 8D E7 88 B3 12 0A 5D 53 F9 2C C7 BE 60 DE A7 56 74 DE 16 75 14 D2 49 12 E3 0C ED B7 BA 6C AB FE BB B6 1E 9F CA 63 2D 29 9F 58 9A F3 A2 8D F8 FD 20 2F F4 C9 3F 5E BA 66 B0

ca.cart

Close **Import**

We see that the subject and issuer are the same, indicating a self-signed certificate. Also, the Certificate Authority is set to yes in Basic Constraints, indicating that this certificate can be used to issue and sign another certificate, hence becoming a certificate of a Certificate Authority (CA). This certificate will be used as the root certificate in this lab.



Task-2: Creating a Certificate for meghajak.com

Step 1: Generate public/private key pair

We first create an RSA public-private key pair using the following command:

```
[04/01/23]seed@VM:~/.../labpart2$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[04/01/23]seed@VM:~/.../labpart2$
```

We can use the following command to view the generated key: `openssl rsa -in server.key -text`

Step 2: Generate a Certificate Signing Request (CSR)

Next, we create a Certificate Signing Request (CSR) that includes the company's public key. The CSR has the following details with company's common name being `mehgajak.com` (company's domain):

```

[04/01/23]seed@VM:~/.../labpart2$ 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.
-----
sk1.C: Country Name (2 letter code) [AU]:US
sk1.S: State or Province Name (full name) [Some-State]:New-York
sk1.L: Locality Name (eg, city) []:Syracuse
sk1.O: Organization Name (eg, company) [Internet Widgits Pty Ltd]:Syracuse
sk1.U: Organizational Unit Name (eg, section) []:Labs
sk1.CN: Common Name (e.g. server FQDN or YOUR name) []:zeeshanali
sk1.E: Email Address []:zeeshanali10771@gmail.com
sk3.C: Please enter the following 'extra' attributes
sk3.C: to be sent with your certificate request
sk3.C: A challenge password []:dees
sk3.C: An optional company name []:zeeshanali
[04/01/23]seed@VM:~/.../labpart2$ 

```

The above CSR is then sent to the CA to generate a certificate for the key and company name.

Step 3: Generating Certificates

The CSR file needs to have the CA's signature to form a certificate. We change the policy to policyAnything from policyMatch to avoid any errors as seen in the following:

```

# 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          = policyAnything

```

The following command turns the certificate signing request (server.csr) into an X509 certificate (server.crt), using the CA's ca.crt and ca.key. We see the details of the CSR being signed:

```

[04/01/23]seed@VM:~/.../labpart2$ 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  1 19:57:24 2023 GMT
        Not After : Mar 31 19:57:24 2024 GMT
    Subject:
        countryName      = US
        stateOrProvinceName = New-York
        organizationName   = Syracuse
        organizationalUnitName = Labs
        commonName        = zeeshanali
        emailAddress       = zeeshanali10771@gmail.com
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:

```

"demoCA" selected (containing 5 items)

```

Subject:
    countryName          = US
    stateOrProvinceName = New-York
    organizationName   = Syracuse
    organizationalUnitName = Labs
    commonName           = zeeshanali
    emailAddress         = zeeshanali10771@gmail.com

X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
        7D:33:13:24:2A:42:B1:51:24:D8:CC:4A:F1:5F:CF:59:C9:15:37:C3
    X509v3 Authority Key Identifier:
        keyid:01:5D:DD:0F:80:09:4E:6B:BB:DC:53:CC:14:2E:08:78:5E:E8:84:29

Certificate is to be certified until Mar 31 19:57:24 2024 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
[04/01/23]seed@VM:~/.../labpart2$ 

```

This step achieves a certificate for meghajak.com from the root certificate.

Task-3: Deploying Certificate in an HTTPS Web Server

Step 1: Configuring DNS

We add the following entry to the /etc/hosts file in order for the machine to recognize zeeshanali.com on the web browser without changing the DNS:

127.0.0.1 zeeshanali.com

This entry basically maps the hostname zeeshanali.com to the localhost:

```

[04/01/23]seed@VM:~/.../labpart2$ cat /etc/hosts
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

# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

```

```
[04/02/23]seed@VM:~/.../labpart2$ cat /etc/hosts
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

# For DNS Rebinding Lab
192.168.60.80   www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80       www.seedlab-shellshock.com
127.0.0.1       zeeshanali.com
10.0.2.7        github.com
[04/02/23]seed@VM:~/.../labpart2$
```

Step 2: Configuring the web server

We combine the key and certificate into one file and then launch a web server using the openssl's s_server command. We enter the passphrase for the private key and see that the web server is now in the accept state – indicating it is running:

```
[04/01/23]seed@VM:~/.../labpart2$ cp server.key server.pem
[04/01/23]seed@VM:~/.../labpart2$ cat server.crt >> server.pem
[04/01/23]seed@VM:~/.../labpart2$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

"demoCA" selected (containing 5 i)

By default, the server listens on port 4433 and we can access the server using the URL:

<https://zeeshanali.com:4433/>

On loading the above URL, we see that the web browser displays an error message with the connection being insecure. On loading more details by clicking on Advanced, we see that it states that the issuer's certificate is unknown i.e. the root certificate is unknown to the browser. This is because, even though the root certificate is created by us, it is not known to the browser and hence it cannot validate the server's certificate i.e. zeeshanali.com's certificate.

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

In order for our root certificate to be accepted by the Firefox browser, we manually add our certificate to the browser by following the menu sequence and importing ca.crt:

Edit -> Preference -> Privacy & Security -> View Certificates

The following shows that the root CA is now in the Firefox's trusted certificates:

Certificate Name	Security Device
-Symantec Corporation	
Symantec Class 1 Public Primary Certification Authority - G6	Builtin Object Token
Symantec Class 2 Public Primary Certification Authority - G6	Builtin Object Token
Symantec Class 1 Public Primary Certification Authority - G4	Builtin Object Token
Symantec Class 2 Public Primary Certification Authority - G4	Builtin Object Token
-Syracuse	
MeghaJakhota	Software Security Device

Buttons at the bottom:

- View...**
- Edit Trust...**
- Import...**
- Export...**
- Delete or Distrust...**

Step 4. Testing our HTTPS website

Now on loading the URL, we see the web page successfully:

```

s_server -cert server.pem -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 :ECDSA-RSA-AES256-GCM-SHA384
TLSv1.2 :ECDSA-RSA-AES256-GCM-SHA384 TLSv1.2 :ECDSA-RSA-AES256-GCM-SHA384
TLSv1.2 :ECDSA-RSA-CHACHA20-POLY1305 TLSv1.2 :ECDSA-RSA-CHACHA20-POLY1305
TLSv1.2 :DHE-RSA-CHACHA20-POLY1305 TLSv1.2 :DHE-RSA-CHACHA20-POLY1305
TLSv1.2 :ECDSA-RSA-AES128-GCM-SHA256 TLSv1.2 :ECDSA-RSA-AES128-GCM-SHA256
TLSv1.2 :ECDSA-RSA-AES128-GCM-SHA256 TLSv1.2 :ECDSA-RSA-AES128-GCM-SHA256
TLSv1.2 :ECDSA-RSA-AES256-SHA384 TLSv1.2 :ECDSA-RSA-AES256-SHA384
TLSv1.2 :DHE-RSA-AES256-SHA256 TLSv1.2 :DHE-RSA-AES256-SHA256
TLSv1.2 :ECDSA-RSA-AES128-SHA256 TLSv1.2 :ECDSA-RSA-AES128-SHA256
TLSv1.2 :ECDSA-RSA-AES256-SHA256 TLSv1.2 :ECDSA-RSA-AES256-SHA256
TLSv1.0 :ECDSA-RSA-AES256-SHA TLSv1.0 :ECDSA-RSA-AES256-SHA
TLSv3 :DHE-RSA-AES256-SHA TLSv1.0 :ECDSA-RSA-AES128-SHA
TLSv1.0 :ECDSA-RSA-AES128-SHA TLSv1.0 :DHE-RSA-AES128-SHA
TLSv1.0 :RSA-PSK-AES256-GCM-SHA384 TLSv1.0 :RSA-PSK-AES256-GCM-SHA384
TLSv1.0 :RSA-PSK-CHACHA20-POLY1305 TLSv1.0 :RSA-PSK-CHACHA20-POLY1305
TLSv1.0 :ECDSA-RSA-CHACHA20-POLY1305 TLSv1.0 :AES256-GCM-SHA384
TLSv1.0 :RSA-PSK-AES256-GCM-SHA384 TLSv1.0 :PSK-CHACHA20-POLY1305
TLSv1.0 :RSA-PSK-AES128-GCM-SHA256 TLSv1.0 :DHE-RSA-AES128-GCM-SHA256
TLSv1.0 :AES128-GCM-SHA256 TLSv1.0 :PSK-AES128-GCM-SHA256
TLSv1.0 :AES256-SHA256 TLSv1.0 :AES128-SHA256
TLSv1.0 :ECDSA-RSA-AES256-CBC-SHA384 TLSv1.0 :ECDSA-RSA-AES256-CBC-SHA
SSLv3 :SRP-RSA-AES256-CBC-SHA TLSv1.0 :SRP-RSA-AES256-CBC-SHA
TLSv1.0 :RSA-PSK-AES256-CBC-SHA384 TLSv1.0 :RSA-PSK-AES256-CBC-SHA384
SSLv3 :RSA-PSK-AES256-CBC-SHA TLSv1.0 :DHE-RSA-AES256-CBC-SHA
SSLv3 :AES128-SHA TLSv1.0 :PSK-AES128-CBC-SHA
TLSv1.0 :PSK-AES256-CBC-SHA TLSv1.0 :ECDSA-RSA-AES128-GCM-SHA256
TLSv1.0 :ECDSA-RSA-AES128-CBC-SHA TLSv1.0 :SRP-RSA-AES128-CBC-SHA
SSLv3 :SRP-RSA-AES128-CBC-SHA TLSv1.0 :RSA-PSK-AES128-CBC-SHA256
TLSv1.0 :DHE-RSA-AES128-CBC-SHA256 TLSv1.0 :RSA-PSK-AES128-CBC-SHA
SSLv3 :DHE-RSA-AES128-CBC-SHA TLSv1.0 :AES128-SHA
TLSv1.0 :PSK-AES128-CBC-SHA256 TLSv1.0 :PSK-AES128-CBC-SHA
...
Ciphers common between both SSL end points:
TLS_AES_128_GCM_SHA256 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_256_GCM_SHA384
ECDSA-RSA-AES128-GCM-SHA256 ECDSA-RSA-AES128-GCM-SHA256 ECDSA-CHACHA20-POLY1305
ECDSA-RSA-CHACHA20-POLY1305 ECDSA-RSA-AES256-GCM-SHA384 ECDSA-RSA-AES256-GCM-SHA384
ECDSA-RSA-AES256-SHA ECDSA-RSA-AES128-SHA ECDSA-RSA-AES128-SHA
ECDSA-RSA-AES256-SHA AES128-GCM-SHA256 AES256-GCM-SHA384
AES128-SHA AES256-SHA
Signature Algorithms: ECDSA+SHA256;ECDSA+SHA384;ECDSA+SHA512;RSA-PSS+SHA256;RSA-PSS+SHA384;RSA-PSS+SHA512;RSA+SHA256;RSA+SHA384;RSA+SHA512;ECDSA+SHA1;RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256;ECDSA+SHA384;ECDSA+SHA512;RSA-PSS+SHA256;RSA-PSS+SHA384;RSA-PSS+SHA512;RSA+SHA256;RSA+SHA384;RSA+SHA512;ECDSA+SHA1;RSA+SHA1

```

```

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,FA85F6AA44122E677BD05A0F852D7FE0

-----END RSA PRIVATE KEY-----
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 4097 (0x1001)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, ST=New-York, L=Syracuse, O=InternetSecurity, CN=ZeshanAli/emailAddress
      address=zeeshanali10771@gmail.com
  -- INSERT --

```

Now,

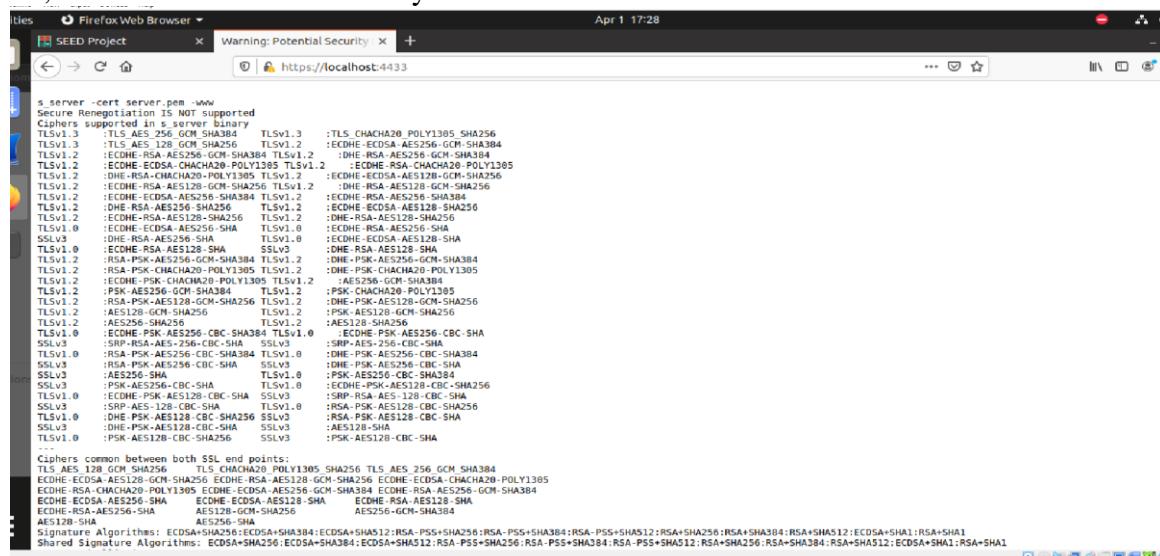
```

[04/01/23] seed@VM:~/.../labpart2$ cp server.key server.pem
[04/01/23] seed@VM:~/.../Labpart2$ cat server.crt >> server.pem
[04/01/23] seed@VM:~/.../Labpart2$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT

```

Now, on restarting the server and reloading the URL, we see that a single change in server.pem file caused the browser to throw an error of secure connection failed due to an invalid signature. This proves that the certificate's integrity is a must and any change in the certificate will break the connection.

Now, we load localhost:4433 instead of zeeshanali.com:4433. We see that it gives us an error with the connection being insecure. However, since zeeshanali.com is linked with 127.0.0.1, essentially the localhost, the website should not really be insecure.



We see that in the additional details, the certificate is invalid because it is not meant for name localhost. On clicking on Add Exception in order to see more details, we see that the certificate is for zeeshanali.com and the URL being loaded is localhost. This basically causes the error to pop up. This proves that the browser also checks for the correctness of the common name and the URL requested. The certificate is valid but for zeeshanali.com and not localhost, and even though both of them point to the same server, the names are different and hence browser does not allow this. If we add the exception, the result will be same as that of zeeshanali.com.

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

We create the pem file for the server's certificate and key to be used in the Apache website configuration:

```
[04/01/23] seed@VM:~/.../labpart2$ cp server.crt zeeshanali_cert.pem
[04/01/23] seed@VM:~/.../labpart2$ cp server.key zeeshanali_key.pem
[04/01/23] seed@VM:~/.../labpart2$ ls
ca.crt demoCA server.crt server.key zeeshanali_cert.pem
ca.key openssl.cnf server.csr server.pem
[04/01/23] seed@VM:~/.../labpart2$ 
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
```

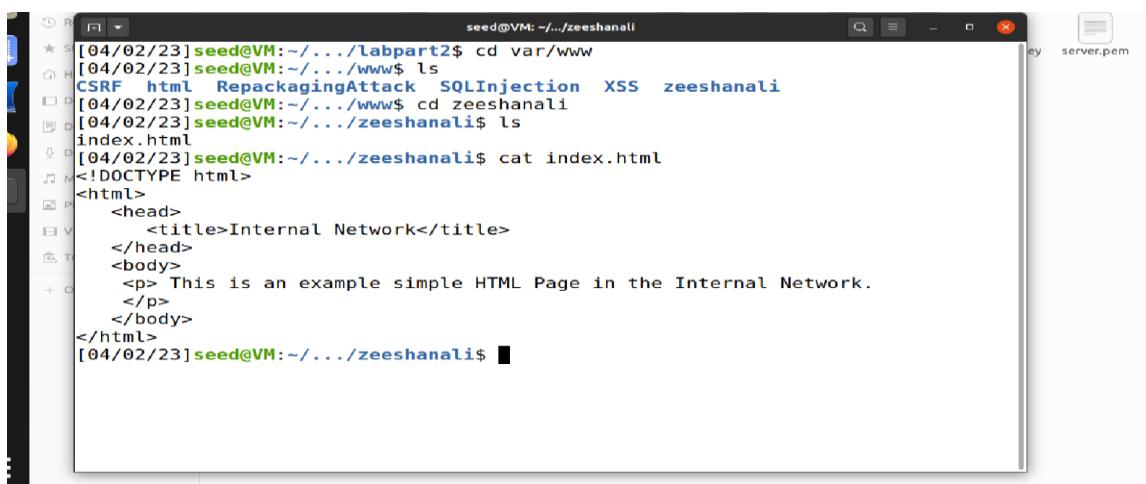
We add a VirtualHost entry to the default-ssl.conf file as follows:

```
<VirtualHost *:443>
    ServerName zeeshanali.com
    DocumentRoot /var/www/zeeshanali
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile /home/seed/Desktop/labpart2/zeeshanali_cert.pem
    SSLCertificateKeyFile /home/seed/Desktop/labpart2/zeeshanali_key.pem
</VirtualHost>
```

Plain Text ▾ Tab Width

The ServerName entry specifies the name of the website, while the DocumentRoot entry specifies where the files for the website are stored. The following show the website's files:



```
[04/02/23] seed@VM:~/.../zeeshanali$ cd var/www
[04/02/23] seed@VM:~/.../www$ ls
CSRF html RepackagingAttack SQLInjection XSS zeeshanali
[04/02/23] seed@VM:~/.../www$ cd zeeshanali
[04/02/23] seed@VM:~/.../zeeshanali$ ls
index.html
[04/02/23] seed@VM:~/.../zeeshanali$ cat index.html
<!DOCTYPE html>
<html>
    <head>
        <title>Internal Network</title>
    </head>
    <body>
        <p> This is an example simple HTML Page in the Internal Network.</p>
    </body>
</html>
[04/02/23] seed@VM:~/.../zeeshanali$
```

```
Terminal
[04/07/20]seed@VM:.../sites-available$ sudo apachectl configtest
AH00558: apache2: Could not reliably determine the server's fully qualified domain
name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this mes
sage
Syntax OK
[04/07/20]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
[04/07/20]seed@VM:.../sites-available$ sudo a2ensite default-ssl
Site default-ssl already enabled
[04/07/20]seed@VM:.../sites-available$ sudo service apache2 restart
```

Now, we open the browser and load the URL and see that we can successfully browse the HTTPS site:



There is an example simple HTML page in the Internal Network.

Task-5: Launching a Man-In-The-Middle Attack

Step 1: Setting up the malicious website

Now, we try to impersonate github.com and change the VirtualHost entry in the Apache's SSL configuration to have the ServerName as github.com, and everything else remains the same:

```
<VirtualHost *:443>
    ServerName zeeshanali.com
    DocumentRoot /var/www/zeeshanali
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile  /home/seed/Desktop/labpart2/zeeshanali_cert.pem
    SSLCertificateKeyFile /home/seed/Desktop/labpart2/zeeshanali_key.pem
</VirtualHost>
```

Plain Text ▾ Tab Width

This will launch a fake github.com website. We restart the apache web server to save the changes.

Step 2: Becoming the man in the middle

Now, to perform MITM, we change the DNS of the client machine to have an entry that will redirect the requests going to the original github.com to the fake web server set by us. We change the /etc/hosts file:

```
[root@seedVM:~]# cat /etc/hosts
[04/02/23]seed@VM:~.../labpart2$ cat /etc/hosts
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

# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

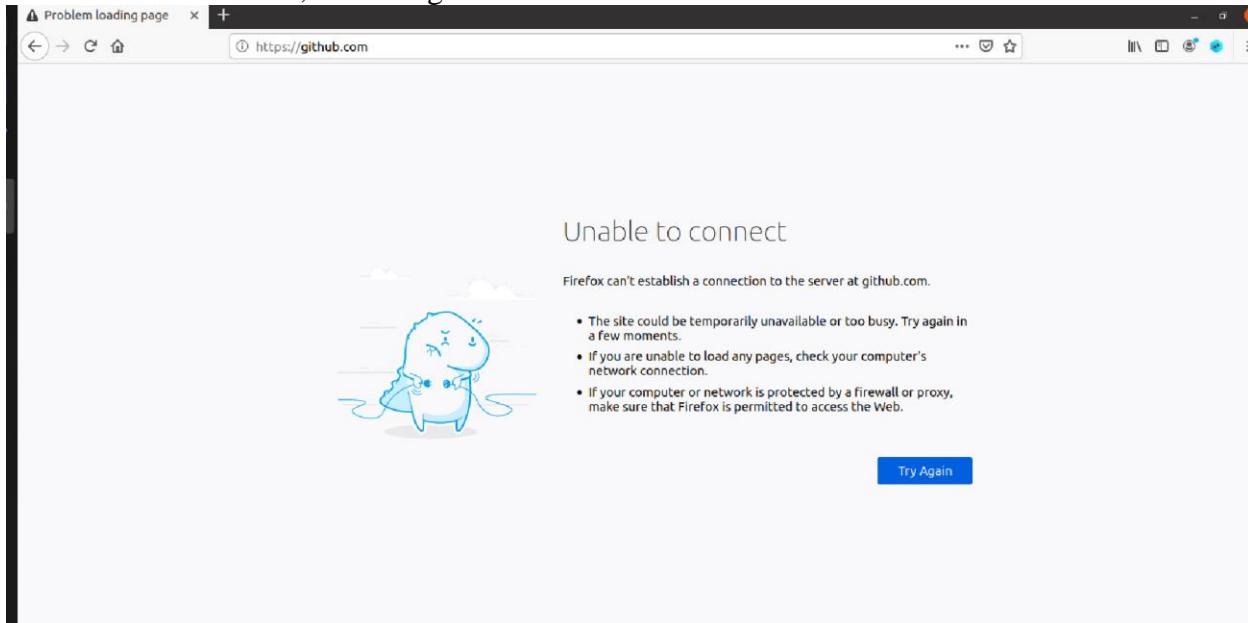
# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80       www.seedlab-shellshock.com
127.0.0.1       zeehanali.com
10.0.2.7        github.com
[04/02/23]seed@VM:~.../labpart2$
```

Step 3: Browse the target website

Now we browse the target website – github.com and see that we get an error saying that the connection is not secure. On looking at more details, we see that the common name of the requested URL and the certificate does not match, indicating the certificate is not meant for this domain.



This proves that the MITM attack is defeated in the use of public key infrastructure. If the common name matched, then the website would have loaded. However, the common name will match only if the server had a valid certificate from the CA for its domain, which it didn't, in this case.

Task-6: Launching a Man-In-The-Middle Attack with a Compromised CA

Considering the same setting as in Task 5 and the root CA's private key being compromised. We first create a private-public key for github.com, the target website. We could have used the same keys generated before but just for simplicity, we create new keys.

```
10.0.2.7      github.com
[04/02/23]seed@VM:~/.../Labpart2$ openssl genrsa -aes128 -out github_server.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
Enter pass phrase for github_server.key:
Verifying - Enter pass phrase for github_server.key:
[04/02/23]seed@VM:~/.../Labpart2$
```

Now, we create a CSR for github.com using OpenSSL in the same way as before, to be signed by a CA:

```
Terminal
[04/09/20]seed@VM:~/.../Lab10$ openssl req -new -key github_server.key -out github
server.csr -config openssl.cnf
Enter pass phrase for github_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]:NY
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Github
Organizational Unit Name (eg, section) []:Repo
Common Name (e.g. server FQDN or YOUR name) []:github.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:megha
An optional company name []:
[04/09/20]seed@VM:~/.../Lab10$
```

Now, since we have the root CA's private key and the certificate is publicly available, we can create a certificate ourselves from the CSR and do not require the CA to do it for us. We use the same command as before to create a certificate.

Now, since this certificate is signed by the root CA that is trusted by the Firefox browser, while authenticating the created certificate, the browser will confirm the validity of this certificate since a trusted root certificate vouched for it. The following shows the creation of a certificate from the CSR using the root CA's credentials:

```

[04/01/23]seed@VM:~/.../labpart2$ 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 1 19:57:24 2023 GMT
        Not After : Mar 31 19:57:24 2024 GMT
    Subject:
        countryName          = US
        stateOrProvinceName = New-York
        organizationName   = Syracuse
        organizationalUnitName = Labs
        commonName           = zeeshanali
        emailAddress         = zeeshanali10771@gmail.com
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
[04/01/23]seed@VM:~/.../labpart2$ "demoCA" selected (containing 5 items)

Subject:
    countryName          = US
    stateOrProvinceName = New-York
    organizationName   = Syracuse
    organizationalUnitName = Labs
    commonName           = zeeshanali
    emailAddress         = zeeshanali10771@gmail.com
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
        7D:33:13:24:2A:42:B1:51:24:D8:CC:4A:F1:5F:CF:59:C9:15:37:C3
    X509v3 Authority Key Identifier:
        keyid:01:5D:DD:0F:80:09:4E:6B:BB:DC:53:CC:14:2E:08:78:5E:E8:84:29

Certificate is to be certified until Mar 31 19:57:24 2024 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]
Write out database with 1 new entries
Data Base Updated
[04/01/23]seed@VM:~/.../labpart2$ 

```

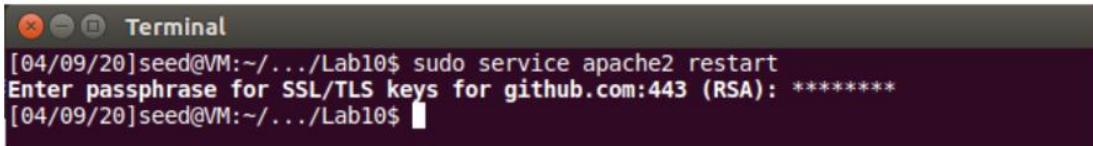
We save the created key and certificate as pem files with names used by the Apache server:

```

[04/02/23]seed@VM:~/.../labpart2$ ls
ca.crt CERT.pem github_server.crt KEY.pem    server.crt server.key task4  zeeshanali_cert.pem
ca.key demoCA  github_server.key openssl.cnf server.csr server.pem var    zeeshanali_key.pem
[04/02/23]seed@VM:~/.../labpart2$ cp github_server.crt zeeshanali_cert.pem
[04/02/23]seed@VM:~/.../labpart2$ cp github_server.key zeeshanali_key.pem
[04/02/23]seed@VM:~/.../labpart2$ ls
ca.crt CERT.pem github_server.crt KEY.pem    server.crt server.key task4  zeeshanali_cert.pem
ca.key demoCA  github_server.key openssl.cnf server.csr server.pem var    zeeshanali_key.pem
[04/02/23]seed@VM:~/.../labpart2$ 

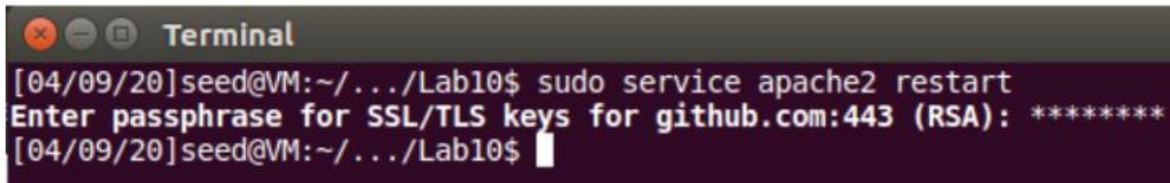
```

In order to incorporate the changes, we restart the Apache service:

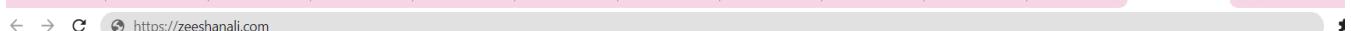


```
[04/09/20]seed@VM:~/.../Lab10$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for github.com:443 (RSA): *****
[04/09/20]seed@VM:~/.../Lab10$
```

Now, on loading the URL of the target website, we see that the browser does not throw any more errors and display the expected website.



```
[04/09/20]seed@VM:~/.../Lab10$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for github.com:443 (RSA): *****
[04/09/20]seed@VM:~/.../Lab10$
```



There is an example simple HTML page in the Internal Network.

This is because the certificate is valid due to being signed by the root CA and its common name is `github.com`. If the root CA's key is compromised, then anyone can create a certificate for themselves and impersonate any other website.

Pre Lab Setup with Docker File

```
image www [04/02/23] seed@VM:~/.../Labsetup$ dcbuild
Building web-server
Step 1/7 : FROM handsonsecurity/seed-server:apache-php
apache-php: Pulling from handsonsecurity/seed-server
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [=====]
    759B/847B Pulling fs layer
da7391352a9b: Downloading [>]
da7391352a9b: Downloading [=>]
da7391352a9b: Downloading [==>]
da7391352a9b: Downloading [===>]
da7391352a9b: Downloading [=====>]
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
d801bb9d0b6c: Pull complete
Digest: sha256:fb3b6a03575af14b6a59ada1d7a272a61bc0f2d975d0776dba98eff0948de275
Status: Downloaded newer image for handsonsecurity/seed-server:apache-php
--> 2365d0ed3ad9
Step 2/7 : ARG WWWDIR=/var/www/bank32
--> Running in d89c19f560ff
Removing intermediate container d89c19f560ff
--> 3e6a864bdc34
Step 3/7 : COPY ./index.html ./index_red.html $WWWDIR/
-----
```

```
Removing intermediate container 653b0ebc6a44
--> 607776223cd5
Step 7/7 : CMD tail -f /dev/null
--> Running in 6fbe66f26070
Removing intermediate container 6fbe66f26070
--> ac8b5ae0493c

Successfully built ac8b5ae0493c
Successfully tagged seed-image-www-pki:latest
[04/02/23] seed@VM:~/.../Labsetup$ dockerps
dockerps: command not found
[04/02/23] seed@VM:~/.../Labsetup$ dockps
[04/02/23] seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.80 ... done
Attaching to www-10.9.0.80
```

Work Division:

Name	Roll No	Work Division
Zeeshan Ali	20i-2465	RSA → pre-Task & Task 1,2, 3,4,5,6
Ans Zeeshan	20i-0543	PKI → pre-Setup & Task 1,2, 3,4,5,6