CPS 472/572 Fall 2024

Prof: Zhongmei Yao

Lab 3 Report: Public Key Infrastructure Lab

1. Dinesh Kumar Kala (kalad1@udayton.edu) Student ID:101745354
2. Akhil Reddy Kotha (kothaa6@udayton.edu) Student ID:101790043

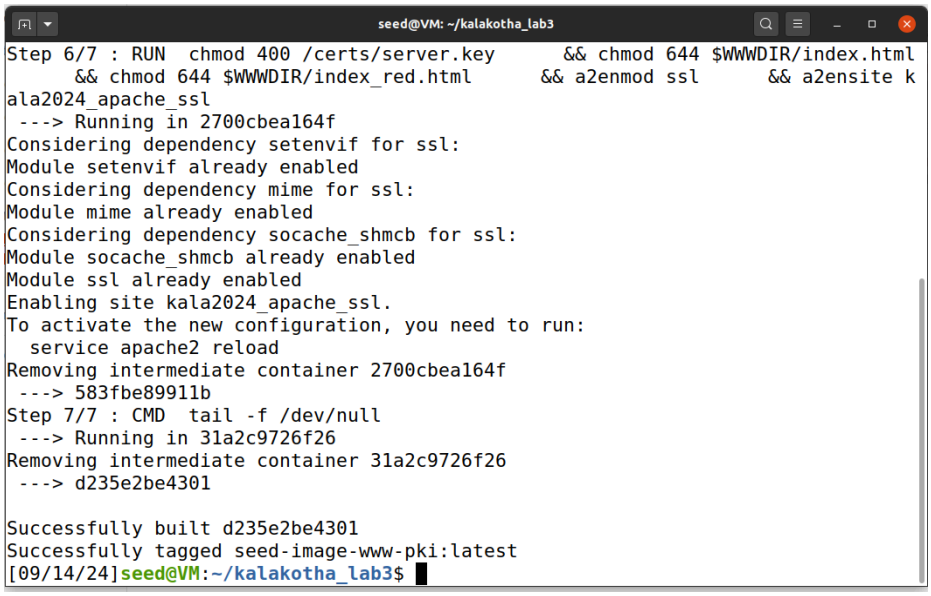
1. Overview:

In this lab, we set up the environment and became as a Certificate Authority (CA). we first created a Certificate Signing Request (CSR) and then generated a certificate from it. After that, we secured an Apache website with HTTPS using the certificate. Lastly, we launched a man-in-the-middle attack to understand how PKI helps prevent such attacks by confirming the authenticity of public keys and certificates. This lab gave me a clearer understanding of how PKI works to protect web communications.

Takeaways: In the lab, we explored how Public Key Infrastructure (PKI) keeps communications secure. we set up a Certificate Authority (CA), created Certificate Signing Requests (CSRs), and issued certificates, which we then used to enable HTTPS on an Apache server. we also performed a man-in-the-middle attack to see how PKI defends against such threats by verifying the security of communications. This experience underscored the importance of trust in PKI and the potential issues if this trust is compromised.

2. Lab Environment:

We downloaded the Labsetup zip file provided by the seed labs and unzipped it. Using “dcbuild” and “dcup” we built the docker and started it.



Next after Building, using “ sudo nano /etc/hosts “ command we mapped the following websites to the IP addresses.

www.bank32.com — 10.9.0.80
www.kala2024.com — 19.9.0.80

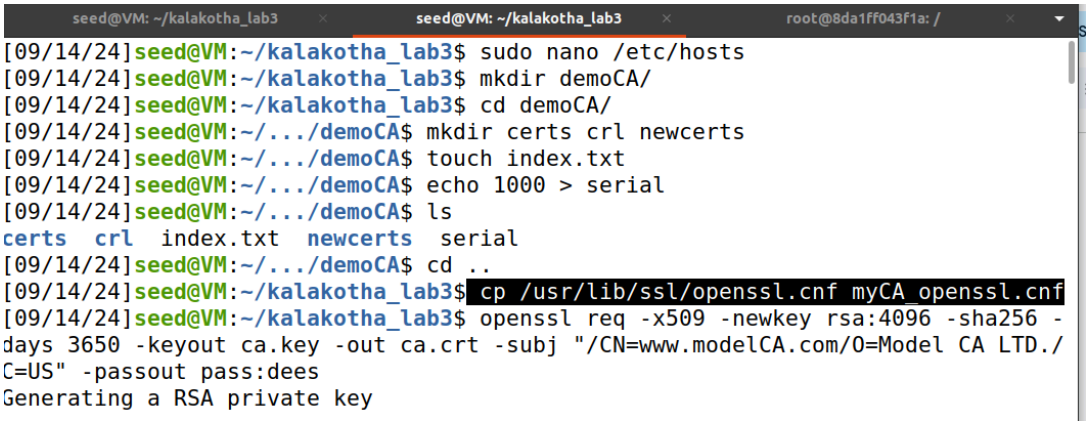
After successfully mapping, we are all set to do the Tasks.

Takeaways: we set up public-key certificates to secure a web server running in containers. We used simple Docker Compose commands to manage the containers and accessed them with `dockps` and `docksh`. For HTTPS, we edited the `/etc/hosts` file to connect a custom domain (www.kala2024.com) to the container's IP address.

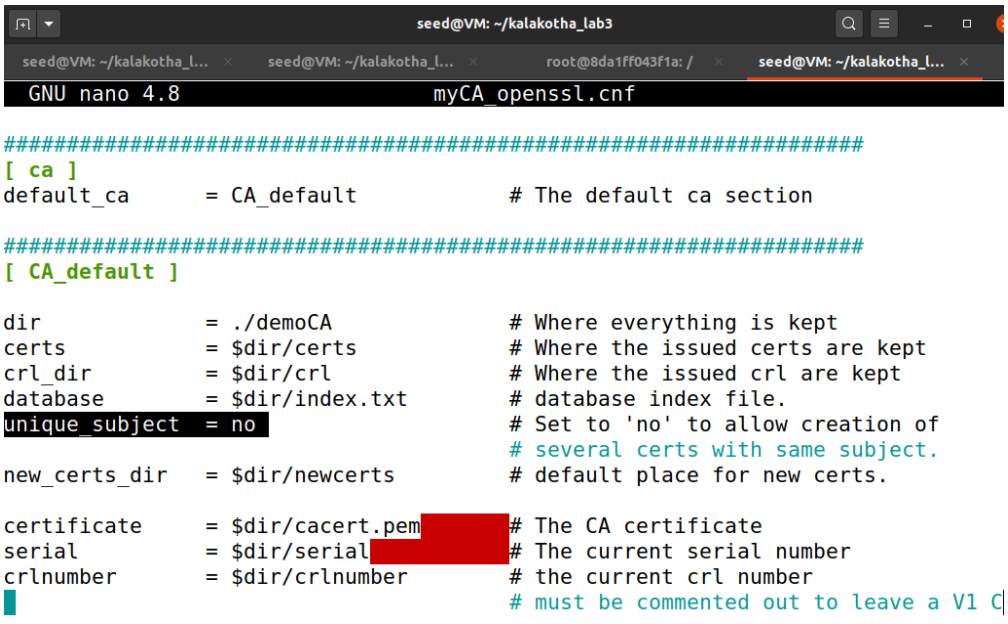
Task 3:

3.1 Task 1: Becoming a Certificate Authority (CA)

We started doing task by making some directories “demoCA” and sub-directories “certs, newcerts, crl. We also created the serial → 1000 and index.txt file.



Then we copied the openssl.cnf file and added to my directory with name “myCA_openssl.cnf”.



After that we made some changes in the myCA_openssl.cnf file
Changes made:

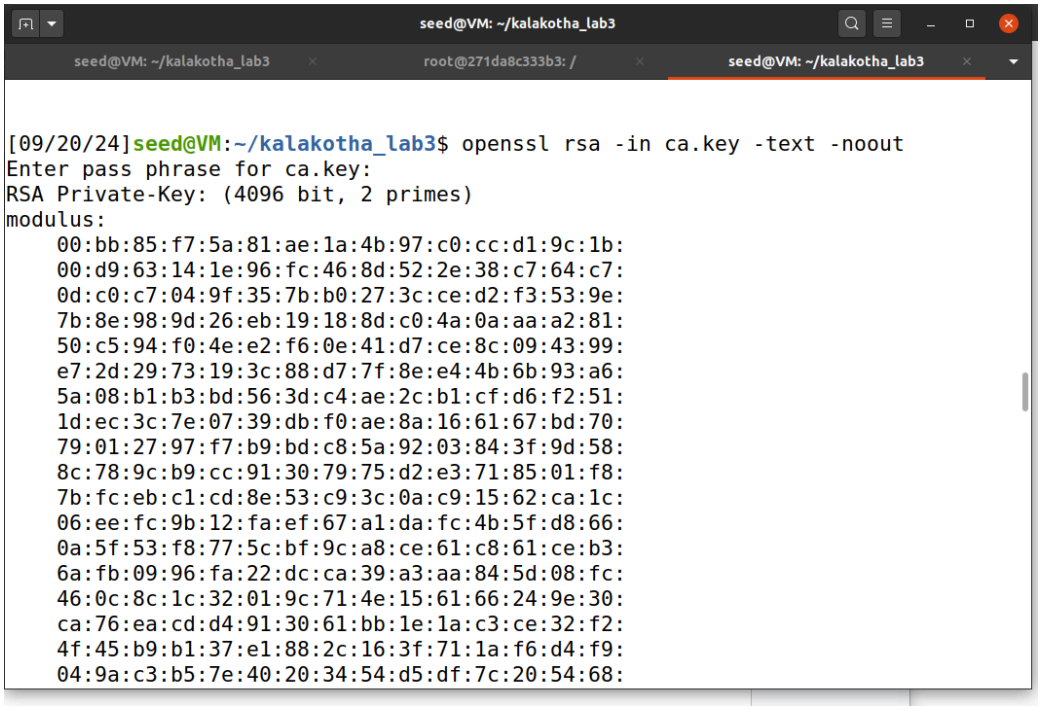
1. Uncommented the unique_subject

```
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl req -x509 -newkey rsa:4096 -sha256 -
days 3650 -keyout ca.key -out ca.crt -subj "/CN=www.modelCA.com/O=Model CA LTD./
C=US" -passout pass:dees
Generating a RSA private key
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................+++
.....................................................................................
.............l.........................................+++
writing new private key to 'ca.key'
-----
```

Now .cnf file is ready to use. Initially, we became a CA (Certificate Authority) by creating a new keys using RSA Algorithm (generates both public key and private key) along with giving the CA website. After running this command, we generated the CA certificate which can used to sign and generate certificates for other websites.

```
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            48:4a:dc:f9:8d:49:42:39:1f:46:e6:39:43:42:e5:cf:da:c8:b5:39
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Sep 14 05:12:47 2024 GMT
            Not After : Sep 12 05:12:47 2034 GMT
        Subject: CN = www.modelCA.com, O = Model CA LTD., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:ca:71:0c:80:c8:bc:08:5a:98:68:d4:5d:42:02:
                    a7:fc:5b:58:6c:9f:cd:63:13:97:f1:15:b6:4e:2c:
                    55:8f:cd:c7:e6:f6:81:ef:28:c9:e7:42:81:31:6c:
```

This is the CA Certificate and below is the CA private key



Q1. What part of the certificate indicates this is a CA's certificate?

Answer:

```
[09/20/24]seed@VM:~/kalakotha_lab3$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5a:ed:0f:8d:34:19:0b:6e:49:59:07:ab:d8:7a:7a:f4:09:39:48:4c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Sep 16 01:19:57 2024 GMT
            Not After : Sep 14 01:19:57 2034 GMT
        Subject: CN = www.modelCA.com, O = Model CA LTD., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
```

```
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                85:EB:8A:4D:3A:84:9F:CA:9F:23:EA:E9:85:B0:2C:A8:09:BA:CB:CD
            X509v3 Authority Key Identifier:
                keyid:85:EB:8A:4D:3A:84:9F:CA:9F:23:EA:E9:85:B0:2C:A8:09:BA:C
D

            X509v3 Basic Constraints: critical
                CA:TRUE
```

By seeing the Issuer and CA : TRUE which means it is a CA certificate.

Q2. What part of the certificate indicates this is a self-signed certificate?

Answer:

```
[09/20/24]seed@VM:~/kalakotha_lab3$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5a:ed:0f:8d:34:19:0b:6e:49:59:07:ab:d8:7a:7a:f4:09:39:48:4c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Sep 16 01:19:57 2024 GMT
            Not After : Sep 14 01:19:57 2034 GMT
        Subject: CN = www.modelCA.com, O = Model CA LTD., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
```

This is a Surely self signed certificate because subject and Issurer are Same.

Q3. In the RSAalgorithm, we have a public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq. Please identify the values for these elements in your certificate and key files.
Answer:
We checked ca.key and ca.crt but we only got the “publicExponent: 65537 (0x10001)” remaining all elements are encrypted using RSA algorithm

Takeaways: In this task, we made a self-signed Certificate Authority (CA) using OpenSSL. After setting up the needed folders and settings, we created the CA's certificate, resulting in a private key (ca.key) and a public certificate (ca.crt). The note CA:TRUE showed it was a CA certificate, and matching issuer and subject fields indicated it was self-signed. We found the public exponent (e) and modulus (n) in the certificate, and the private exponent (d) in the key file, establishing a trusted root CA.

3.2 Task 2: Generating a Certificate Request for Your Web Server

Now it is time to create keys and .csr file to our own website
Note: we need a .csr file (contains our details) which is provided to the CA (Certificate Authority) to verify and validate the websites and to authenticate with the browser.

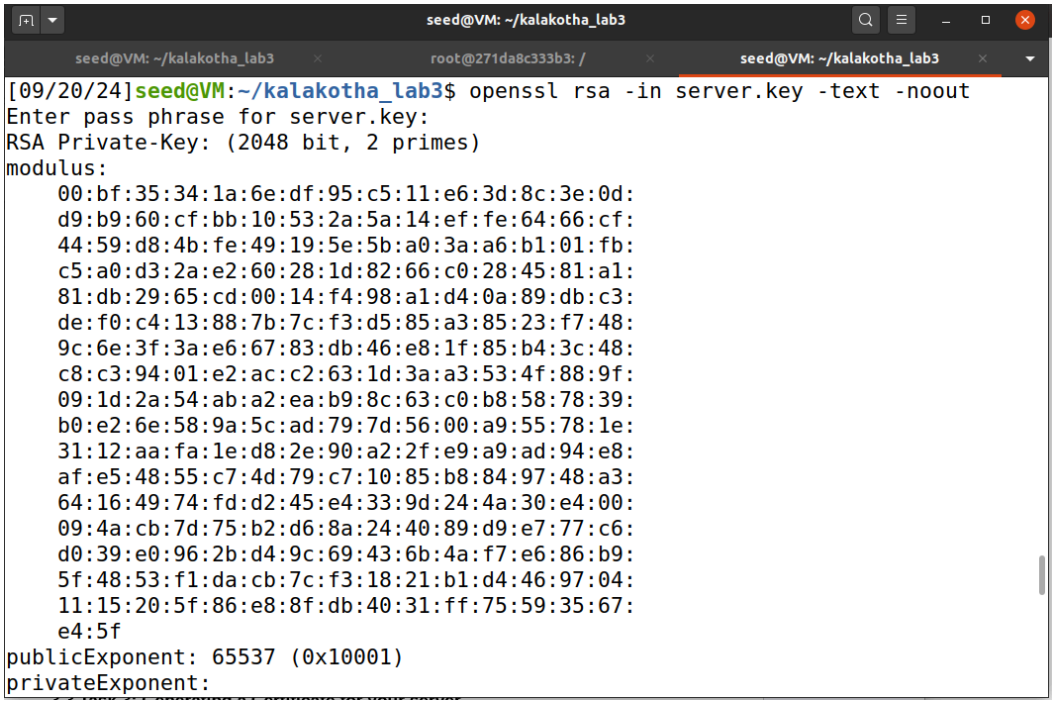
```
-----
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl req -newkey rsa:2048 -sha256 -keyout
 server.key -out server.csr -subj "/CN=www.kala2024.com/O=kala2024 Inc./C=US" -p
assout pass:dees -addext "subjectAltName=DNS:www.kala2024A.com,DNS:www.kala2024B
.com,DNS:www.kala2024.com"
Generating a RSA private key
.......+++++
.............+++++
writing new private key to 'server.key'
-----
[09/14/24]seed@VM:~/kalakotha_lab3$ ls
```

We used same commands to generate the keys and by using “subjectAltNAme” we added the extra websites. We gave the Hash function SHA256.
We successfully generated the .csr file.

```
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl req -in server.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = www.kala2024.com, O = kala2024 Inc., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a3:e2:bf:ac:f3:14:4e:bf:63:5a:da:6b:23:e6:
                    38:f6:60:2a:be:89:a2:94:27:33:8b:49:52:c8:c5:
                    a3:56:fb:8c:e1:c3:8b:97:0b:ff:0c:e8:4d:90:39:
                    f8:2b:65:68:e6:3f:df:b3:93:d4:60:a4:21:f9:6c:
                    f5:ab:ad:bc:62:4f:a5:11:04:b9:57:76:76:b1:4c:
                    0f:6e:35:74:b4:06:07:ad:4a:8e:86:38:a4:30:58:
                    4e:ae:6f:25:d0:12:4d:e8:6b:2a:92:c4:c7:2f:39:
                    5e:af:93:27:ce:ad:ab:bf:65:95:24:65:52:a7:ce:
                    2a:62:9b:96:18:c3:cb:70:7c:7a:da:e7:ca:84:ac:
                    5c:60:b0:ad:61:7e:13:ee:6d:f3:58:92:34:fb:10:
```

To check the .csr file, we used “openssl req -in server.csr -text -noout” command and everything looks good.

And here is our private Key



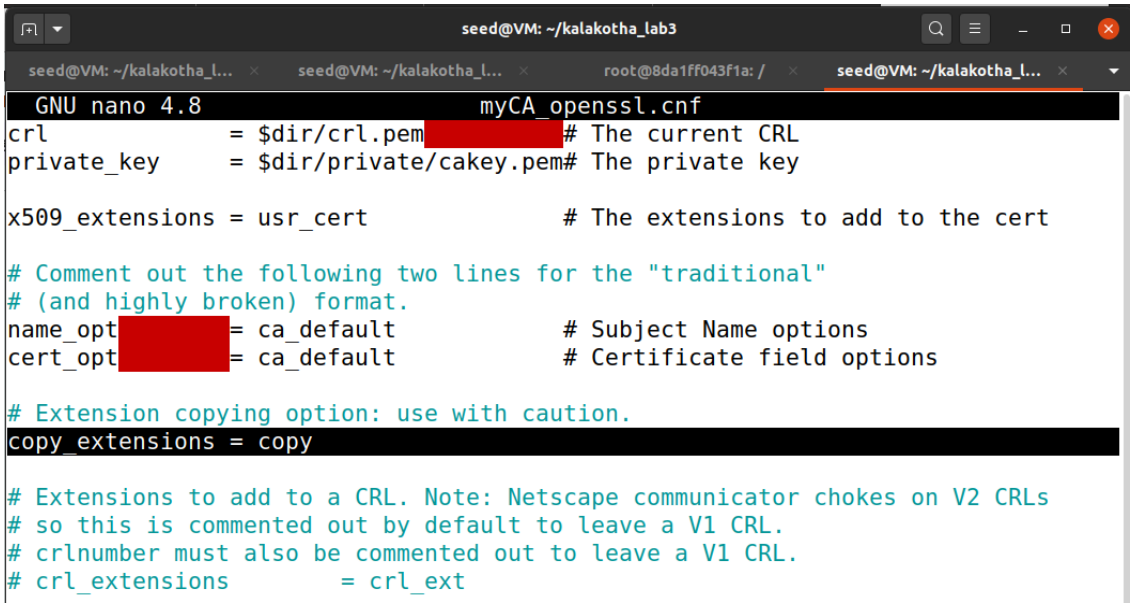
Takeaways: We learned how to create a public key certificate by first generating a Certificate Signing Request (CSR) for a web server, such as "www.kala204.com." This CSR includes the server's public key and identity details, which we send to the Certificate Authority (CA) for approval. We also discovered that to allow the certificate to support multiple website addresses, we can use the Subject Alternative Name (SAN) extension. This lets us list several hostnames in the certificate, ensuring that browsers accept the server's identity correctly.

3.3 Task 3: Generating a Certificate for your server

To Generate certificate, we need .csr file which will be verified and signed by CA.

```
[09/14/24]seed@VM:~/kalakotha_lab3$
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl ca -config myCA_openssl.cnf -policy
policy_anything -md sha256 -days 3650 -in server.csr -out server.crt -batch -cer
t ca.crt -keyfile ca.key
Using configuration from myCA_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: Sep 14 05:24:37 2024 GMT
            Not After : Sep 12 05:24:37 2034 GMT
        Subject:
            countryName               = US
            organizationName          = kala2024 Inc.
            commonName                = www.kala2024.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                4E:AE:BF:92:D7:C7:51:FA:55:A2:19:6C:51:85:A5:69:4C:EA:F3:52
```

Now our website is verified and signed by the CA. which means we can use our website in the browser by deploying it.



And now we went to myCA_openssl.cnf file and uncommented the copy_extensions.

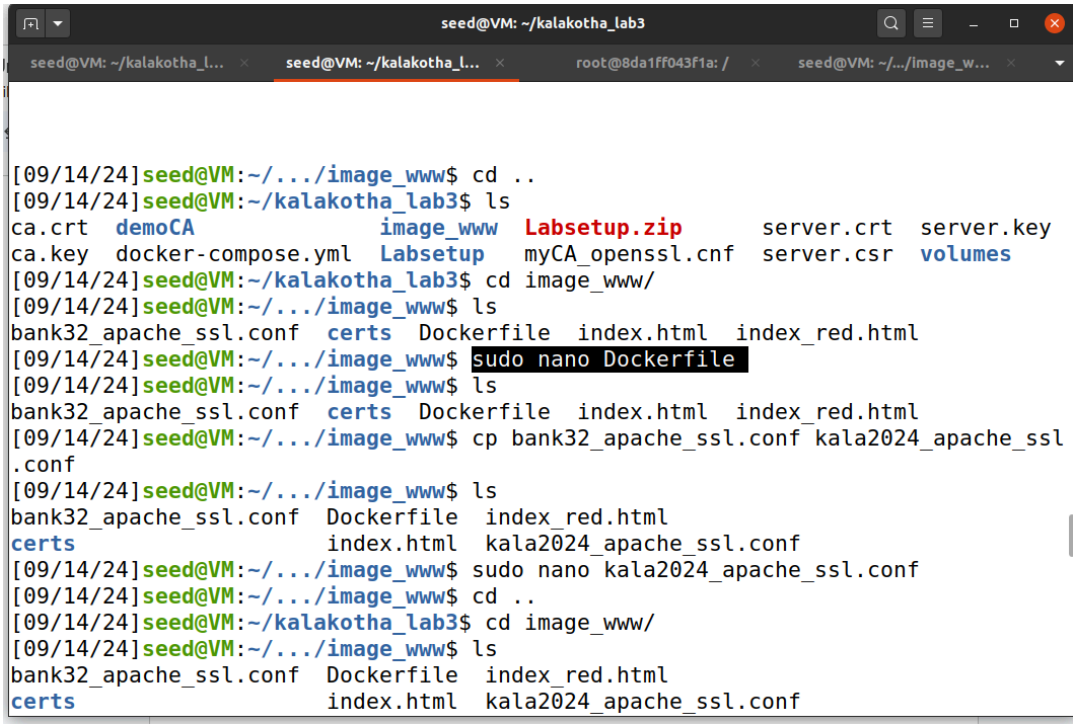
```
[09/14/24]seed@VM:~/kalakotha_lab3$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Sep 14 05:24:37 2024 GMT
            Not After : Sep 12 05:24:37 2034 GMT
        Subject: C = US, O = kala2024 Inc., CN = www.kala2024.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a3:e2:bf:ac:f3:14:4e:bf:63:5a:da:6b:23:e6:
                    38:f6:60:2a:be:89:a2:94:27:33:8b:49:52:c8:c5:
                    a3:56:fb:8c:e1:c3:8b:97:0b:ff:0c:e8:4d:90:39:
                    f8:2b:65:68:e6:3f:df:b3:93:d4:60:a4:21:f9:6c:
                    f5:ab:ad:bc:62:4f:a5:11:04:b9:57:76:76:b1:4c:
                    0f:6e:35:74:b4:06:07:ad:4a:8e:86:38:a4:30:58:
                    4e:ae:6f:25:d0:12:4d:e8:6b:2a:92:c4:c7:2f:39:
```

Above is our server certificate which is ready to deploy.

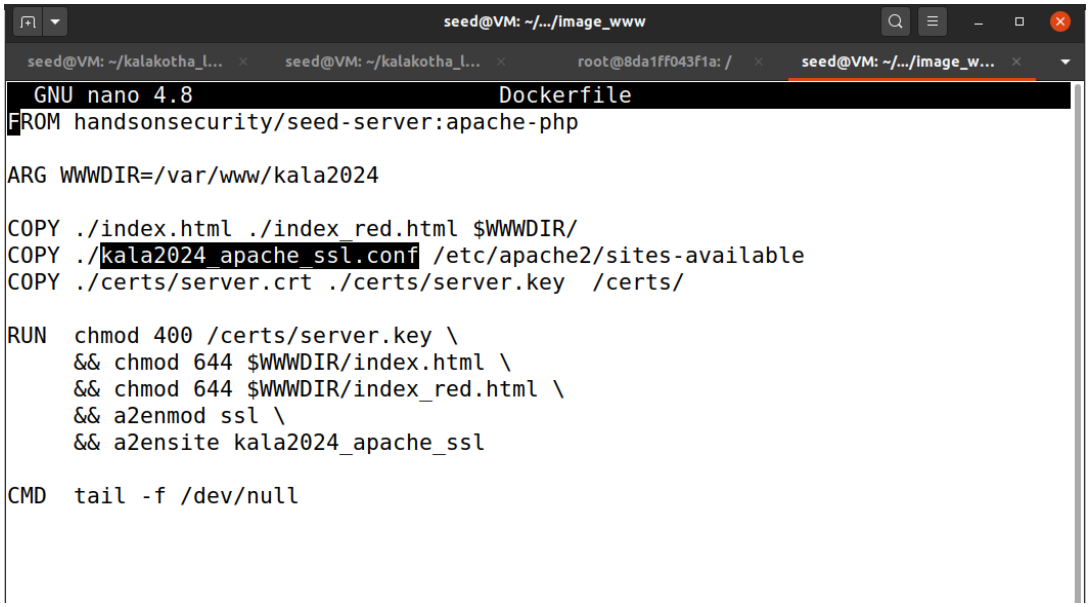
Takeaways: In this part of the Task, we created a certificate from the Certificate Signing Request (CSR) we made earlier by using our own Certificate Authority (CA) to sign it. We executed a command that converted our CSR file (server.csr) into an X.509 certificate (server.crt) using our CA's certificate and key. We ensured to use a specific configuration file we set up previously. Additionally, we adjusted the settings to allow copying extra information from the CSR to the new certificate. Finally, we checked the newly created certificate to confirm that it included the alternative names we specified.

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

Before deploying, we have ca.crt, ca.key and server.crt, server.key which means we are now ready to deploy. We initially started Task 4 from Dockerfile which is present in the image_www directory.



We replaced every bank32 word with the kala2024. Because we are using a different website this time



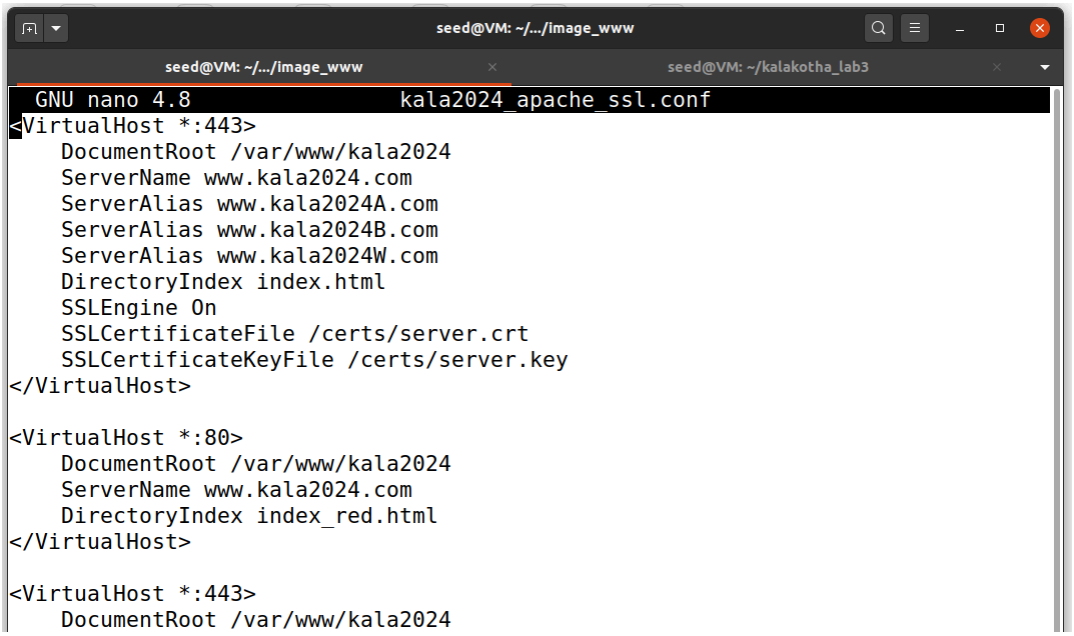
After making changes in the DockerFile, we copied bank32.conf to the kala2024_apache_ssl.conf file. Basically, Apache will use this .conf file when we start the Apache browser.
We didn't know what is chmod 400 and 644 so we asked ChatGpt

Question: what is chmod 400 and 644?
Answer given by ChatGpt: "The 'chmod' command in Unix and Linux is used to change file or directory permissions. For instance, 'chmod 400' sets permissions so that only the owner can read the file, while no one else has any access. This means the owner has read permission (4), but the group and others have none (0). On the other hand, 'chmod 644' allows the owner to read and write the file, while the group and others can only read it. In this case, the owner has read and write permissions (6), and both the group and others have read permission (4). These commands are essential for controlling who can access and modify files."

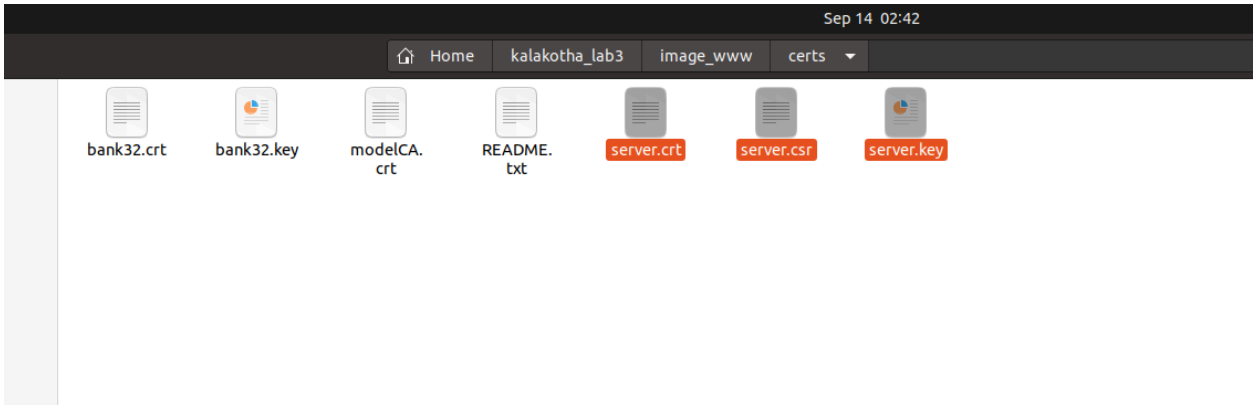
```
[09/14/24]seed@VM:~/.../image_www$ cp bank32_apache_ssl.conf kala2024_apache_ssl
.conf
[09/14/24]seed@VM:~/.../image_www$ ls
bank32_apache_ssl.conf  Dockerfile  index_red.html
certs                   index.html  kala2024_apache_ssl.conf
[09/14/24]seed@VM:~/.../image_www$ sudo nano kala2024_apache_ssl.conf
[09/14/24]seed@VM:~/.../image_www$ cd ..
[09/14/24]seed@VM:~/kalakotha_lab3$ cd image_www/
[09/14/24]seed@VM:~/.../image_www$ ls
bank32_apache_ssl.conf  Dockerfile  index_red.html
certs                   index.html  kala2024_apache_ssl.conf
```

Then we opened the kala2024_apache_ssl.conf file and gave our website details, serverName and the location of keys and certificates.

VirtualHost *:443 - HTTPS
VirtualHost *:80 - HTTP



We changed everything with kala2024 and saved it successfully. Copied the server.key, server.csr, server.crt to the certs folder



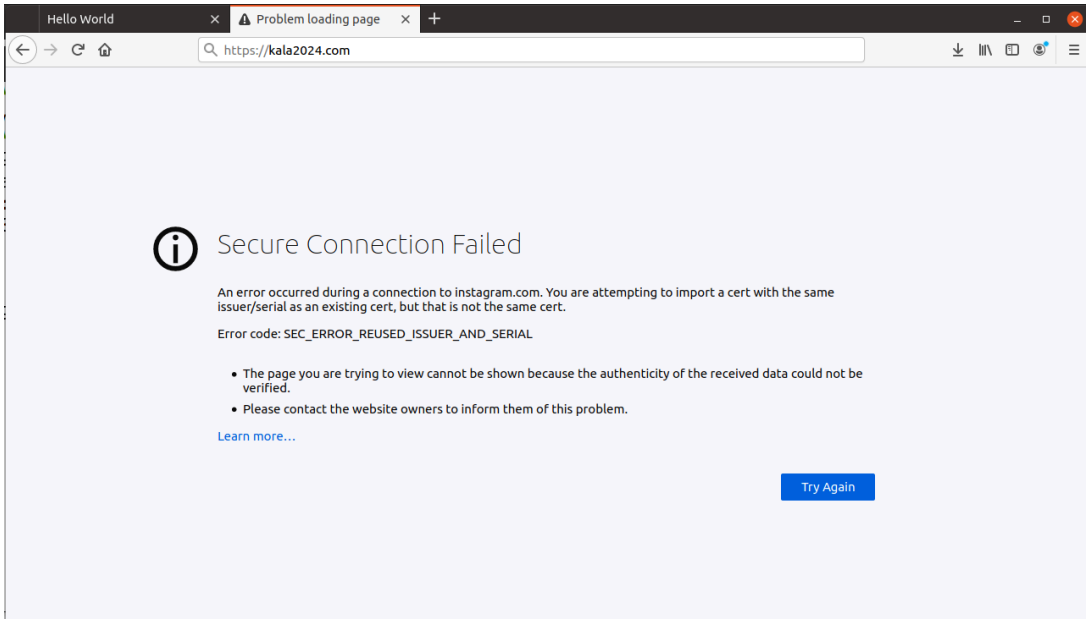
Now all set !!
After Pasting the files we build the Docker followed by the Dcup

```
[09/14/24]seed@VM:~/kalakotha_lab3$ dcbuild
Building web-server
Step 1/7 : FROM handsonsecurity/seed-server:apache-php
 ---> 2365d0ed3ad9
Step 2/7 : ARG WWWDIR=/var/www/kala2024
 ---> Using cache
 ---> 85cef2427d9e
Step 3/7 : COPY ./index.html ./index_red.html $WWWDIR/
 ---> Using cache
 ---> a84054c51786
Step 4/7 : COPY ./kala2024_apache_ssl.conf /etc/apache2/sites-available
 ---> Using cache
 ---> cf66426bdaa1
Step 5/7 : COPY ./certs/server.crt ./certs/server.key  /certs/
 ---> Using cache
 ---> 1d047c7c067b
Step 6/7 : RUN  chmod 400 /certs/server.key      && chmod 644 $WWWDIR/index.html
      && chmod 644 $WWWDIR/index_red.html      && a2enmod ssl      && a2ensite k
ala2024_apache_ssl
 ---> Using cache
 ---> a0f1e7736918
Step 7/7 : CMD  tail -f /dev/null


Step 7/7 : CMD  tail -f /dev/null
 ---> Using cache
 ---> cb69c80e5cac

Successfully built cb69c80e5cac
Successfully tagged seed-image-www-pki:latest
[09/14/24]seed@VM:~/kalakotha_lab3$ dcup
Starting www-10.9.0.80 ... done
Attaching to www-10.9.0.80
```

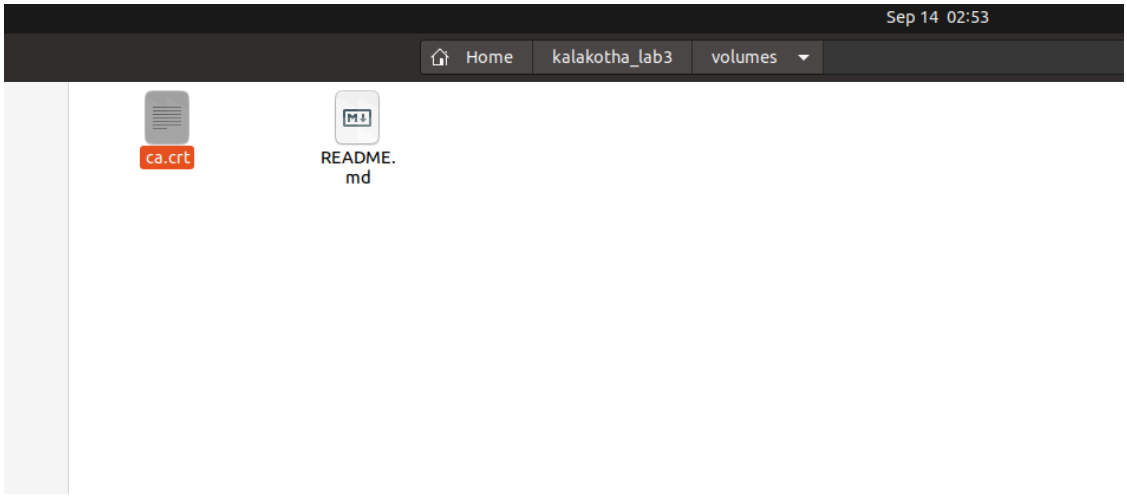
After Building and starting the Docker. We need to know the which container is running so to check that we used “dockps” which shows us the present running container. Using “docksh” we ran the docker container.

```
[09/14/24]seed@VM:~/kalakotha_lab3$ dockps
8da1ff043f1a  www-10.9.0.80
[09/14/24]seed@VM:~/kalakotha_lab3$ docksh 8d
root@8da1ff043f1a:/# service apache2 start
 * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.80. Set the 'ServerName' directive globally to suppress th
is message
Enter passphrase for SSL/TLS keys for www.kala2024.com:443 (RSA):
 *
root@8da1ff043f1a:/#
```

Using “service apache2 start” we started the apache server.we went to the browser and given my website. It said, Not secure

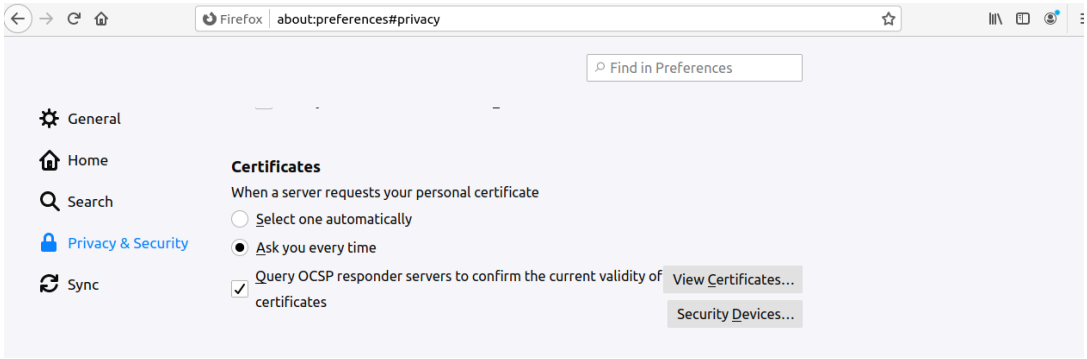


Then we copied the ca.crt to the volumes (shared folder between VM and Container).

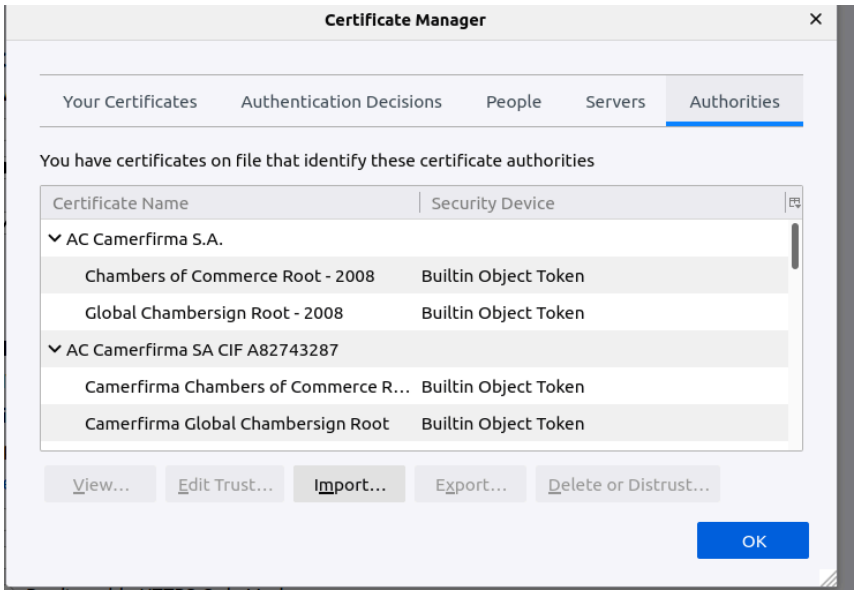


After copying the ca.crt . we uploaded the ca.crt in the browser -> options -> preferences -> privacy&security -> certificates.

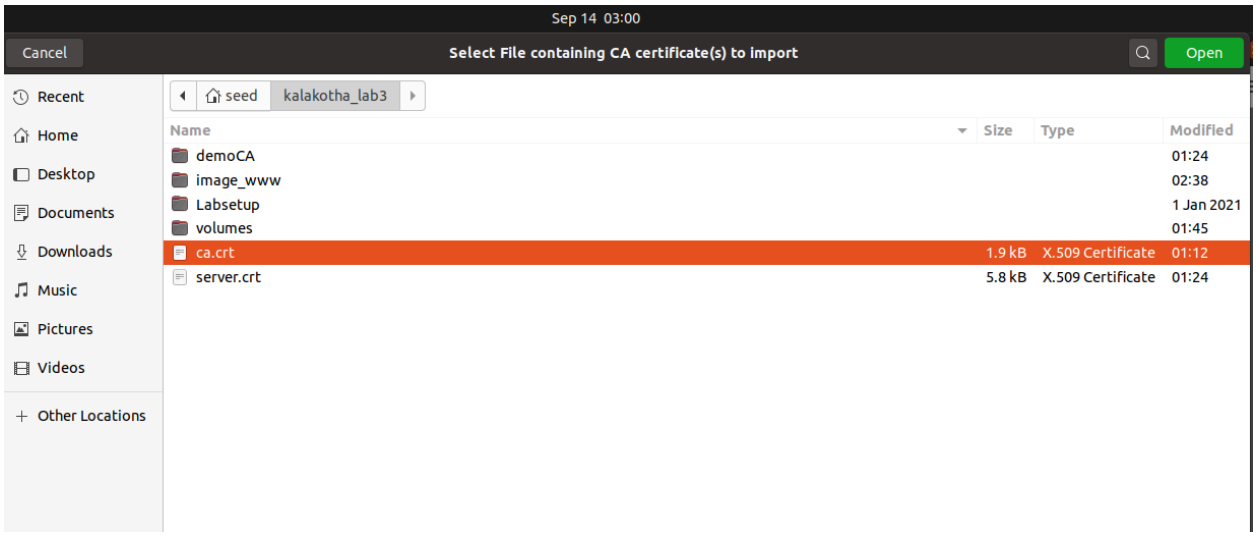
Here we uploaded my certificate.



View certificates

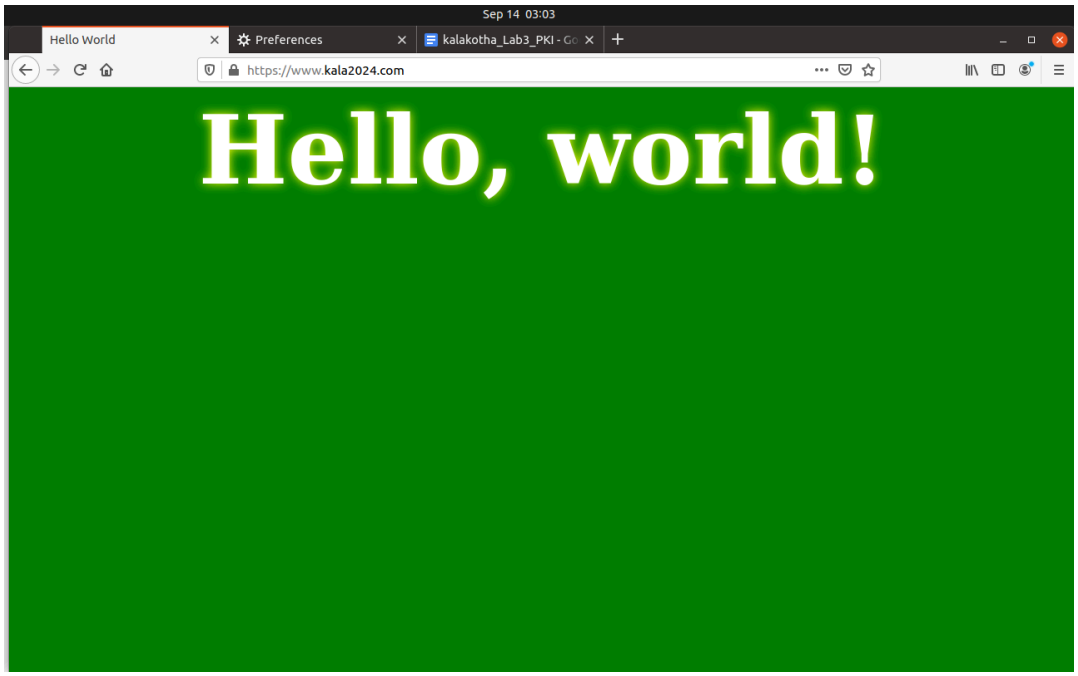


Import



Now we successfully imported the CA certificate in the browser.

At 1st we used **https:**www.kala2024.com
Refresh the page output:



And then we used **http:**//www.kala2024.com

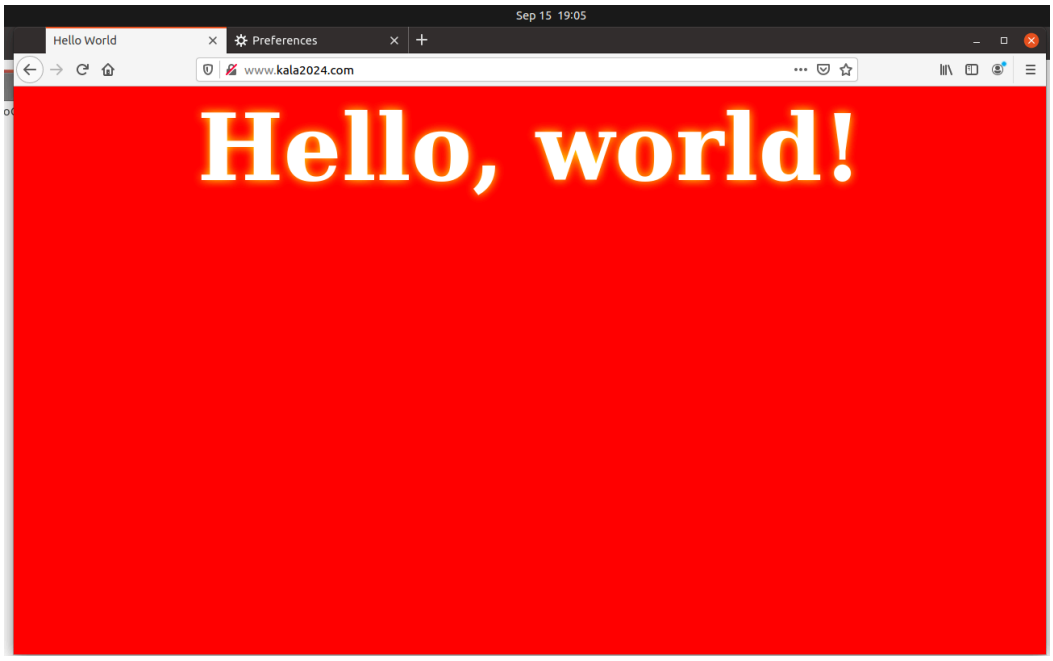
It redirected me to default ubuntu page, we made some small changes in the Dockerfile. We disabled the default ubuntu page so that now it will show the red_index.html file.

Open

Dockerfile
~/kalakotha_lab3/image_www

```
1 FROM handsonsecurity/seed-server:apache-php
2
3 ARG WWWDIR=/var/www/kala2024
4
5 COPY ./index.html ./index_red.html $WWWDIR/
6 COPY ./kala2024_apache_ssl.conf /etc/apache2/sites-available
7 COPY ./certs/server.crt ./certs/server.key  /certs/
8
9 RUN  chmod 400 /certs/server.key \
10     && chmod 644 $WWWDIR/index.html \
11     && chmod 644 $WWWDIR/index_red.html \
12     && a2enmod ssl \
13     && a2ensite kala2024_apache_ssl  \
14     && a2dissite 000-default
15
16 CMD  tail -f /dev/null
17
```

This is the output image image we got for HTTP:

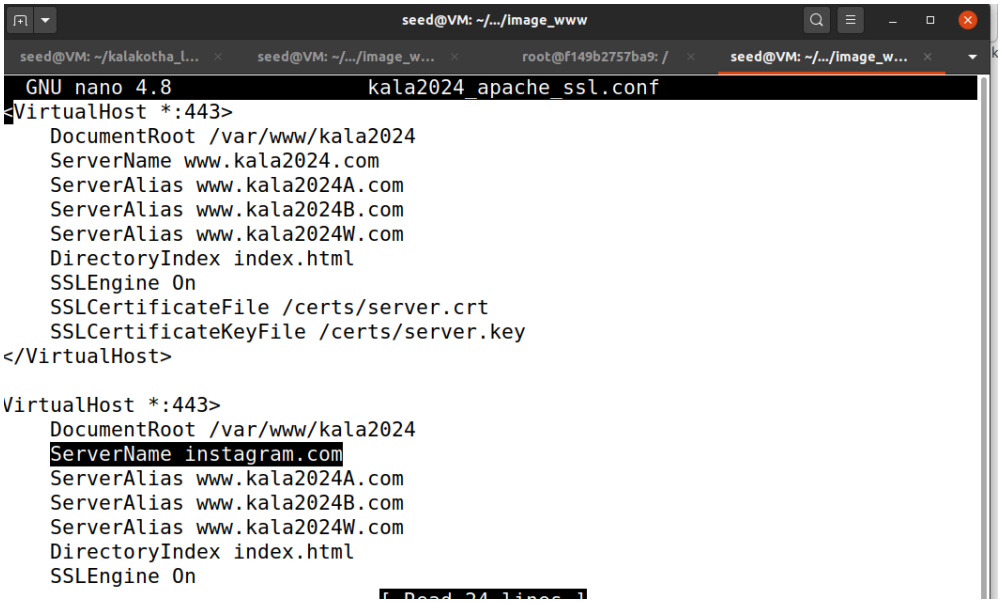


Takeaways: In this task, we set up an HTTPS website using Apache in a container. We configured the Apache server with a VirtualHost entry for the website, specifying the document root and server name, along with multiple aliases. We enabled the SSL module and the site configuration, then started the Apache server, entering the password for the encrypted private key. To access the site, we used "https" in the URL.

We also shared a folder between the VM and container for easy file management and imported our certificate into Firefox to ensure a secure connection.

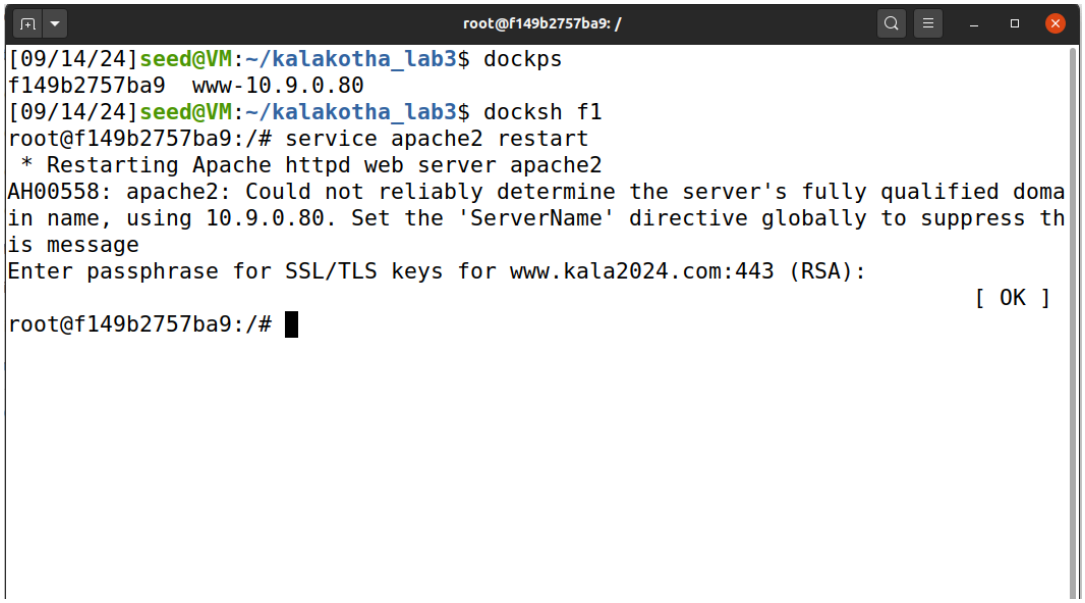
3.5 Task 5: Launching a Man-In-The-Middle Attack

To do this task, we have added one more Virtual host with the same configuration of Kala2024 in the kala2024_apache_ssl.conf file.
Target website : instagram.com

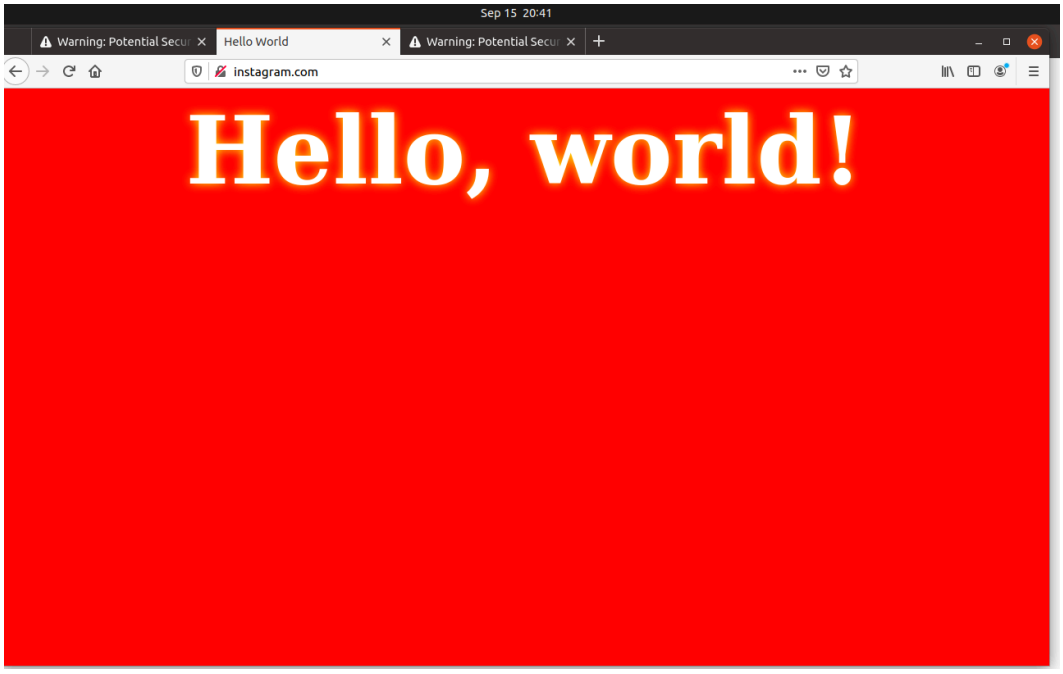


Then aslo added this website (instagram.com) in /etc/hosts

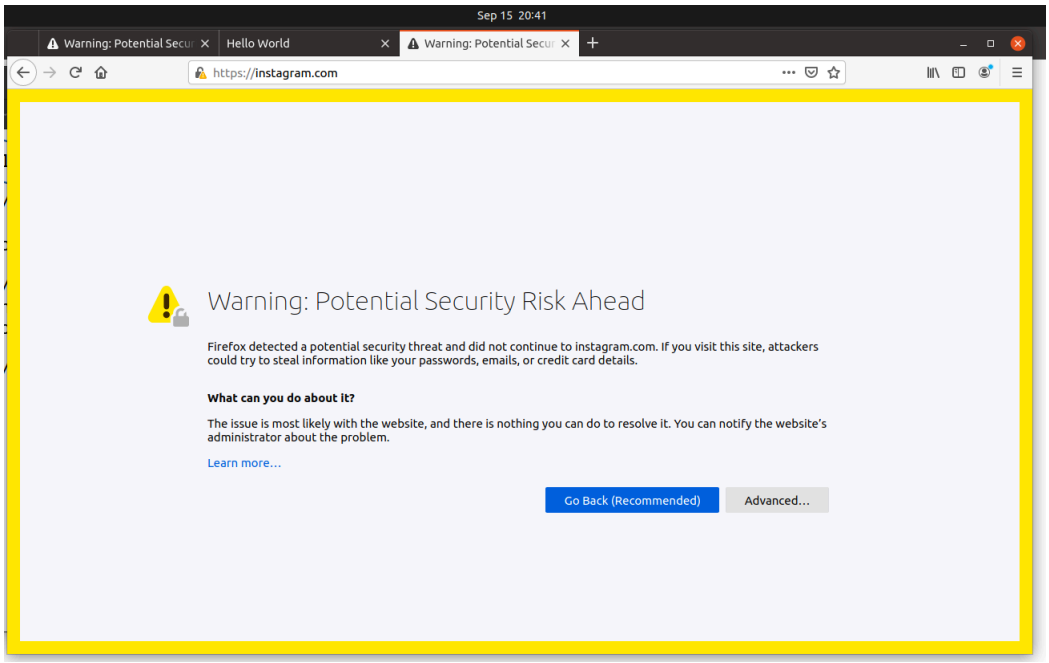
Next restarted the apache 2



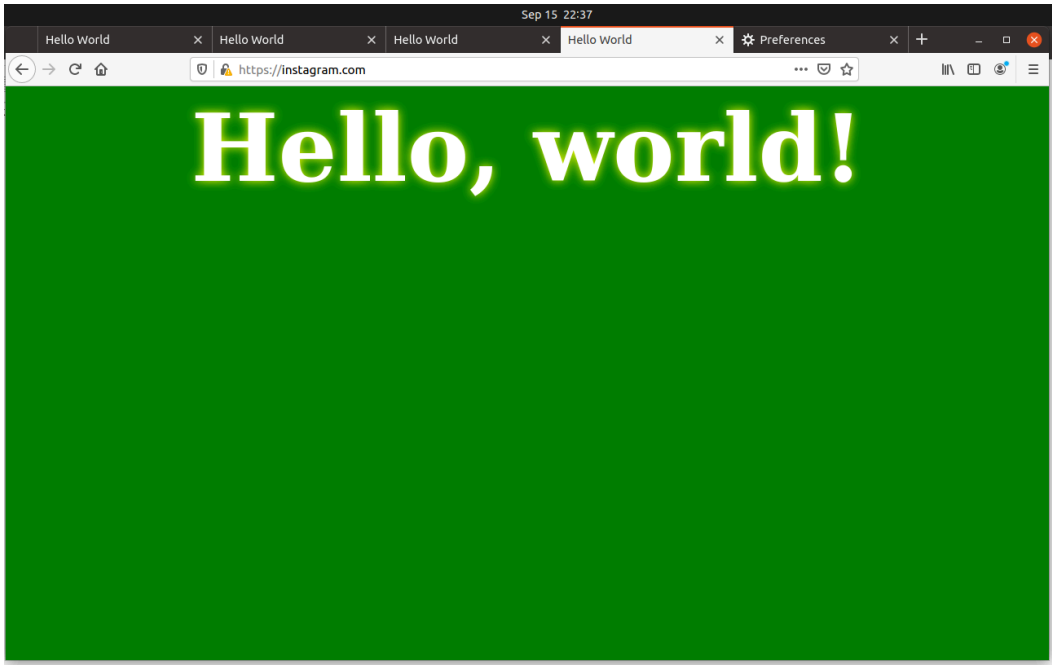
After restarting the apache2, we browsed the instagram website without https



After we used https,



we went to advanced and accepted the risk, it took us to the fake page. But if you observe, it says certificate is not verified.



Question: With everything set up, now visit the target real website, and see what your browser would say. Please explain what you have observed.
Answer: when we are launching this MITM attack using DNS approach, we knew only the website. So to hack it we used our own Virtual host details and then we tested in web browser Here is what we observed,

For HTTP: the instagram.com successfully shows our page (hacker page) without any warnings.
For HTTPS: But when it comes to HTTPS, initially it warned us saying that this is not a trusted website. After accepting the risk it successfully showed our web page but it shows that the website is not verified (warning symbol). Which means PKI can stop the MITM attack.

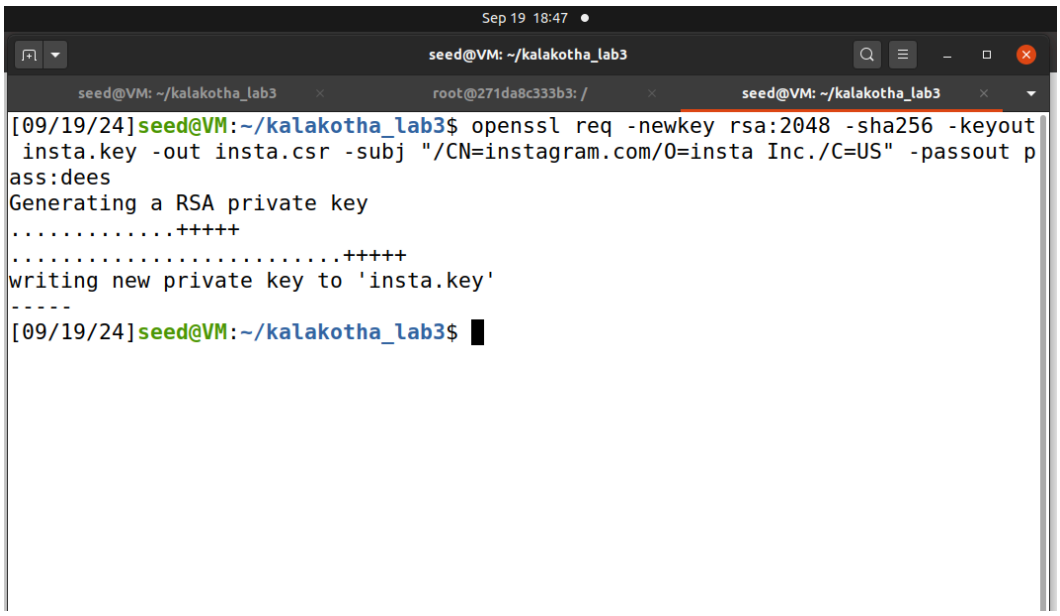
Takeaways: In this task, we showed how Public Key Infrastructure (PKI) can stop Man-In-The-Middle (MITM) attacks. We set up a fake website using Apache to look like instagram.com. First, we changed the Apache settings to use our own certificate for the fake site. Then, to make users visit our fake site instead of the real one, we edited the victim's computer's /etc/hosts file to point instagram.com to our server's IP address. This setup demonstrated how an attacker could trick users into giving away their login details if they couldn't tell the fake site from the real one.

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

In the above task we attacked the instagram without the CA certifcate, but this time we got the CA private which means we can easily generate a CA.crt using the private key. We are created a CA private key and CA.crt in Task1. I'll assume that i got the private key from the CA. and generate the CA.certificate.

Now ill use this CA certificate to sign my Fake certificate.

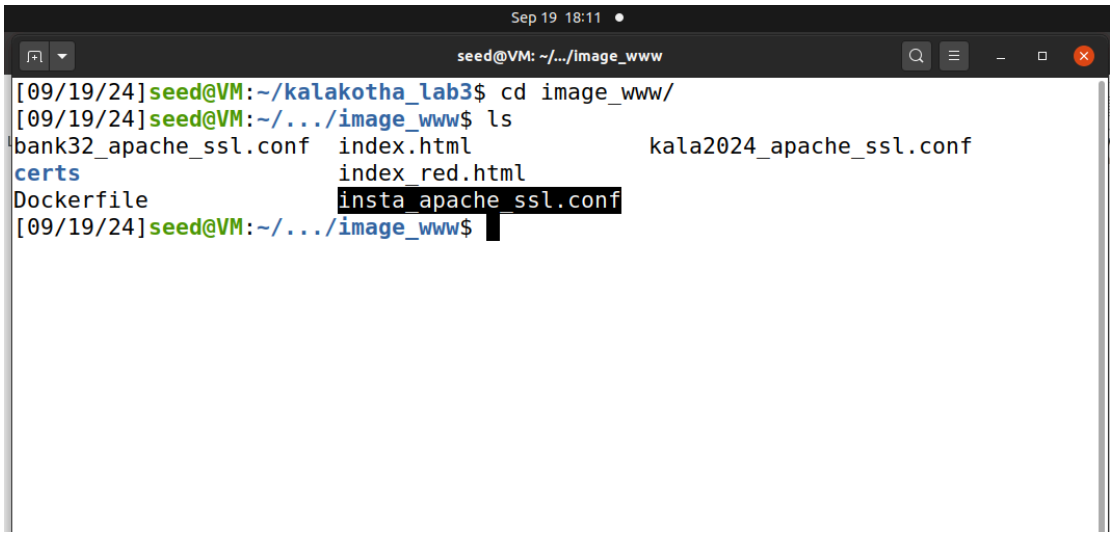
Intially we generated insta.csr file and ready to sign by CA.



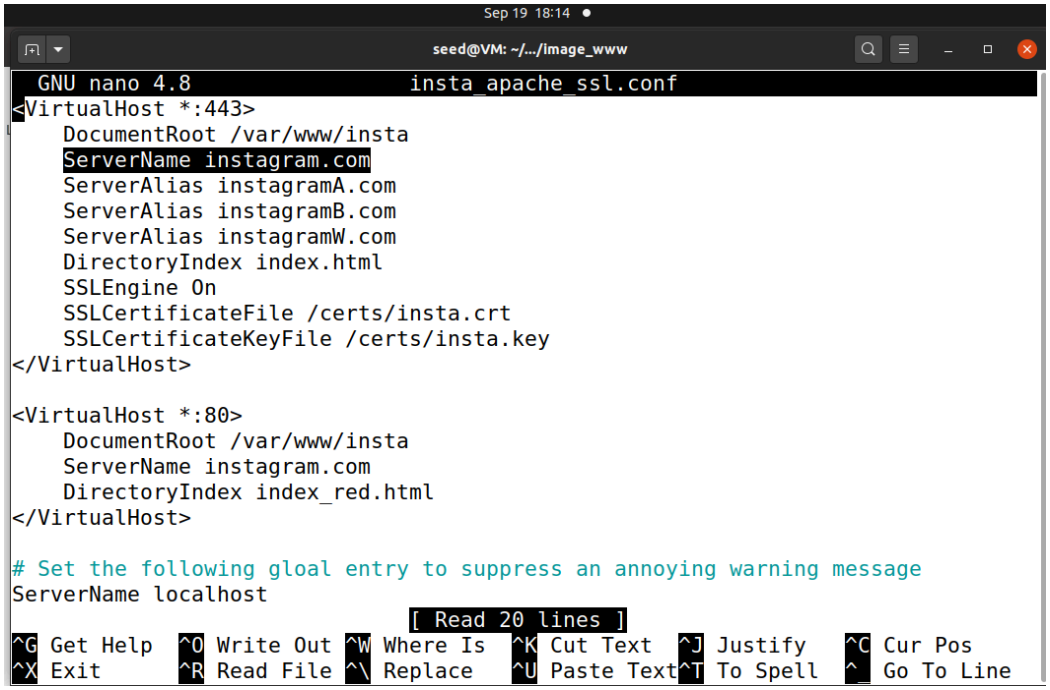
Now as a Hacker im signing my fake website. We assumed that, we stolen the private key of CAmodel (which we created in Task1).

```
-----
[09/19/24]seed@VM:~/kalakotha_lab3$ openssl ca -config myCA_openssl.cnf -policy
policy_anything -md sha256 -days 3650 -in insta.csr -out insta.crt -batch -cert
ca.crt -keyfile ca.key
Using configuration from myCA_openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4101 (0x1005)
        Validity
            Not Before: Sep 19 23:00:11 2024 GMT
            Not After : Sep 17 23:00:11 2034 GMT
        Subject:
            countryName               = US
            organizationName          = insta Inc.
            commonName                = instagram.com
```

Now we created another apache file for instagram website.

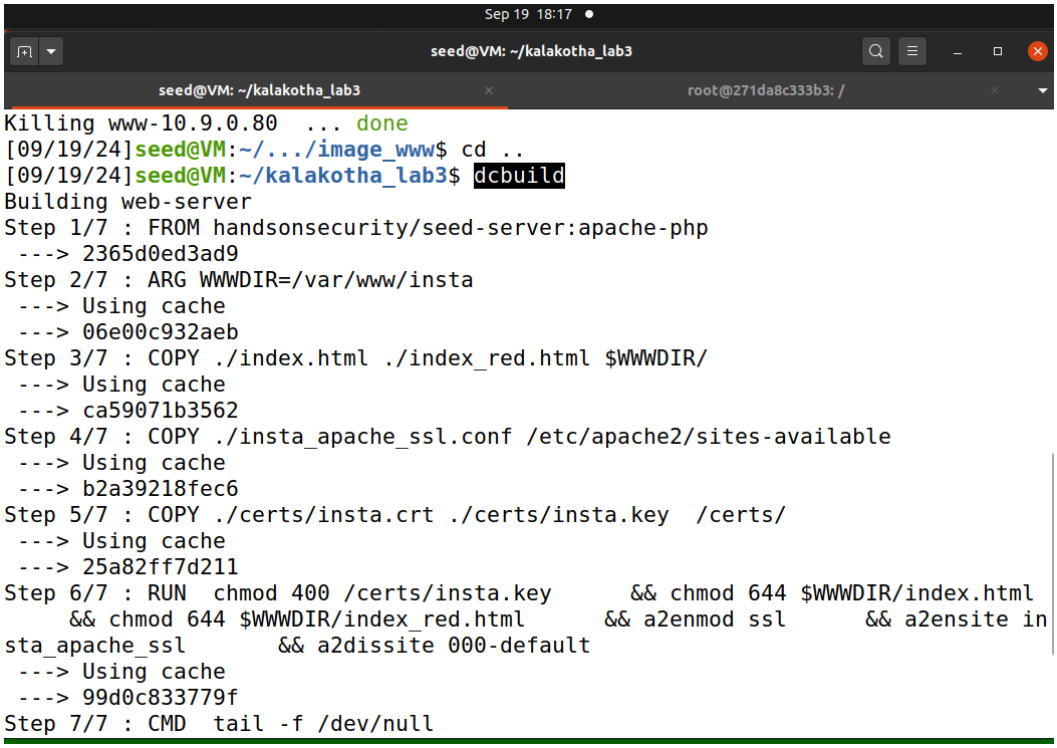


And made this changes in the .conf file

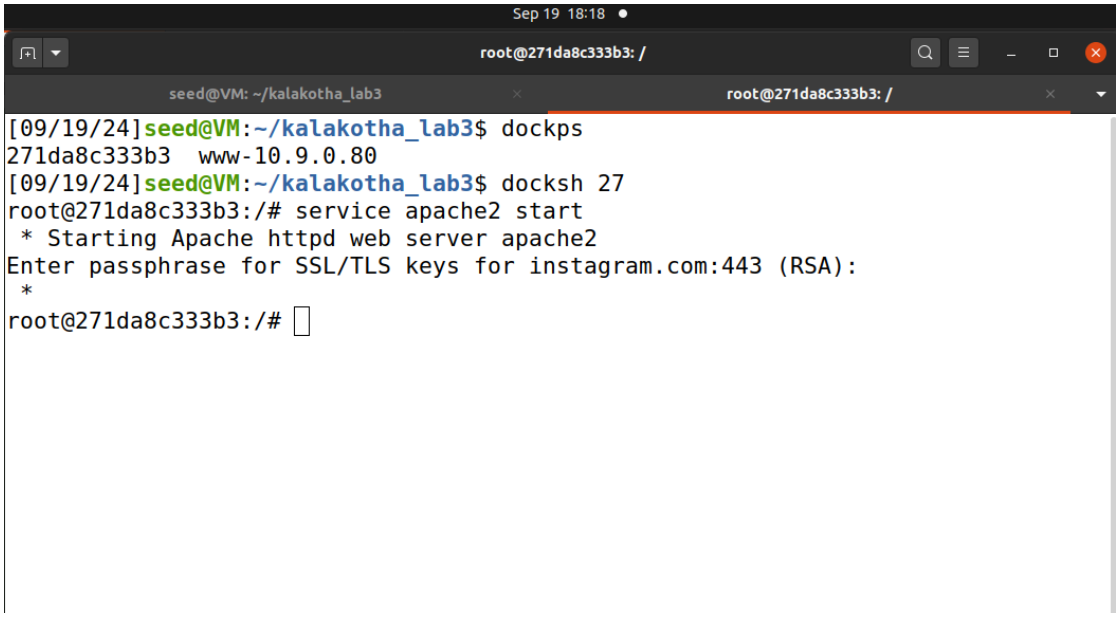


And then made some changes in DockerFile

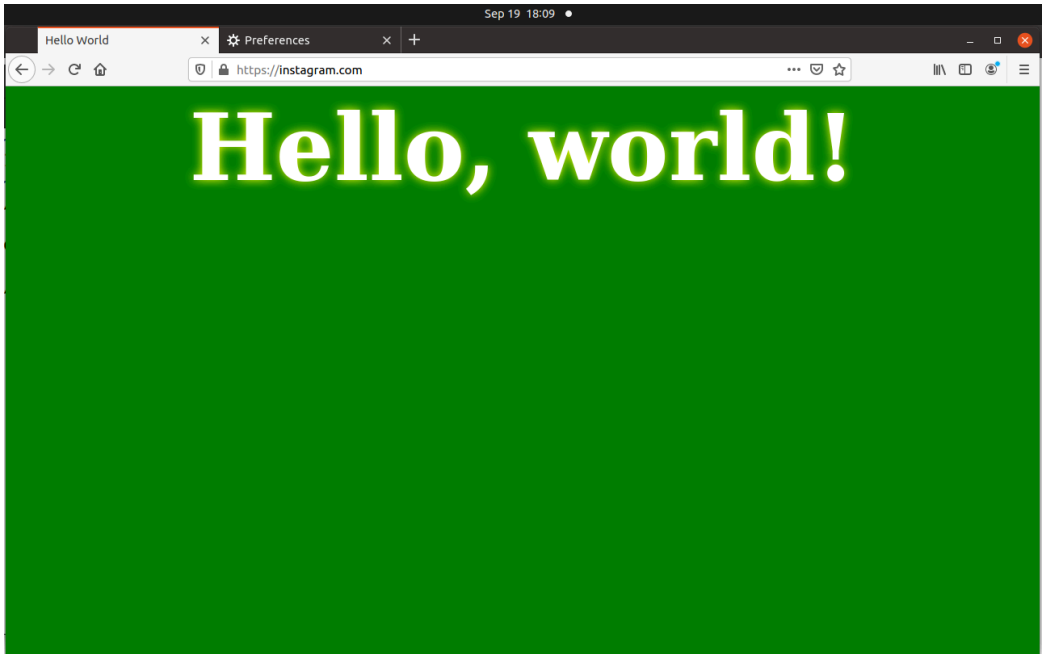
Now lets build the server using the dcbuild and dcup to start



Start the apache server by looking the present running container



Now we googled https://instagram.com and here is the output. Which means, MITM attack is performed.



Takeaways: In this task, we stole the CA private Key from Task-1, and performed the MITM attack. Since we knew the CA private key we can generated public and private keys sign and the CSR file of instagram.com (for Hacker's website) and sign it using the CA private key. And using that certificate, we (hacker) can deployed the ca certificate in browser and performed MITM attack. This time our fake website is trusted by the browser and the victim don't know about the website is real or fake. So they can easily loose their logins or important information.