ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

Rapport du Public-Key Infrastructure (PKI) Lab

RÉALISÉ PAR:
IMAD AFIF
YOUSSEF AZILI

ENCADRÉ PAR:
PR. YASSINE MALEH

SOMMAIRE

Introduction

Public key cryptography is a critical component of secure digital communication. However, it faces vulnerabilities, such as man-in-the-middle attacks, particularly during the exchange of public keys. The main challenge lies in verifying the ownership of a public key—ensuring that it genuinely belongs to the claimed owner. To address this issue, the Public Key Infrastructure (PKI) provides a structured and reliable framework for establishing trust in public key exchanges. This lab focuses on exploring PKI and its role in securing web communication. Through practical tasks, students will gain insights into the workings of PKI, its application in mitigating man-in-the-middle attacks, and the importance of the root of trust in the infrastructure. Additionally, the lab introduces key concepts, such as certificate authorities, X.509 certificates, and HTTPS, offering a comprehensive understanding of PKI's role in modern cybersecurity.

1. TASK 1

```
[12/23/24]seed@VM:~/.../Labsetup$  docker-compose build
Building web-server
Step 1/7 : FROM handsonsecurity/seed-server:apache-php
apache-php: Pulling from handsonsecurity/seed-server
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
d801bb9d0b6c: Downloading [==========================>                        ]  39.73d801bb9d0b6c: Downloading
 [============================>                      ]  43.49MB/71.99MB6c: Downloading [======================
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 b4672f07cc95
Removing intermediate container b4672f07cc95
 ---> e5178b0adfdd
Step 3/7 : COPY ./index.html ./index_red.html $WWWDIR/
 ---> fb314df77b05
Step 4/7 : COPY ./bank32_apache_ssl.conf /etc/apache2/sites-available
 ---> 2453908a629b
Step 5/7 : COPY ./certs/bank32.crt ./certs/bank32.key  /certs/
 ---> b19de31e3029
Step 6/7 : RUN  chmod 400 /certs/bank32.key      && chmod 644 $WWWDIR/index.html      && chmod 644 $WWWDIR/inde
```

Initial laboratory environment setup showing the Docker container configuration and network settings.

```
[12/23/24]seed@VM:~/.../Labsetup$  docker-compose up -d
Recreating www-10.9.0.80 ... done
[12/23/24]seed@VM:~/.../Labsetup$ dockps
8c43e4c4d38f  www-10.9.0.80
```

here we are composing our docker container to make it ready to use on the following addresses

```
10.9.0.80        www.bank32.com
10.9.0.80        www.smith2020.com
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
```

we use www.bank32.com as an example to show how to set up an HTTPS
web server with this name. Students need to use a different name for their lab. Unless the name is specified
by the instructors, students should include their last name and the current year in the server name.

1. TASK 1

```
[12/23/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/Labsetup
[12/23/24]seed@VM:~/.../Labsetup$ sudo cp "/usr/lib/ssl/openssl.cnf" "/home/seed
/Desktop/Labsetup/pki"
[12/23/24]seed@VM:~/.../Labsetup$ cd pki
[12/23/24]seed@VM:~/.../pki$ ls
openssl.cnf
[12/23/24]seed@VM:~/.../pki$ mkdir democa
[12/23/24]seed@VM:~/.../pki$ cd democa
[12/23/24]seed@VM:~/.../democa$ mkdir certs crl newcerts
[12/23/24]seed@VM:~/.../democa$ echo 1000 > serial
[12/23/24]seed@VM:~/.../democa$ touch index.txt
[12/23/24]seed@VM:~/.../democa$
```

```
23/24]seed@VM:~/.../pki$ openssl req -x509 -newkey rsa:4096 -sha256 -days
keyout ca.key -out ca.crt
rating a RSA private key
.....
.++++
ing new private key to 'ca.key'
r PEM pass phrase:
fying - Enter PEM pass phrase:
-
are about to be asked to enter information that will be incorporated
 your certificate request.
```

In order to use OpenSSL to create certificates, you have to
have a configuration file. The configuration file usually has an extension .cnf. It is used by three OpenSSL
commands: ca, req and x509. The manual page of openssl.conf can be found from online resources.
By default, OpenSSL use the configuration file from /usr/lib/ssl/openssl.cnf. Since we need to make changes to this file, we will copy it into our current directory, and instruct OpenSSL to use this copy instead.

1. TASK 1

```
[12/23/24]seed@VM:~/.../pki$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7e:de:de:ae:5b:9b:9c:e3:12:9a:10:a8:0e:a8:ee:38:a0:cf:8c:5c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Dec 23 11:08:51 2024 GMT
            Not After : Dec 21 11:08:51 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:c2:83:7d:0f:7b:7c:99:95:79:e1:69:db:0f:a3:
                    d7:8c:6d:72:46:a8:ad:61:b4:21:23:ba:8f:71:0c:
                    84:bf:00:a6:cd:d5:1f:1a:b2:de:f0:b7:a5:a5:6c:
                    c1:2f:7c:05:fa:58:93:9f:c4:94:16:a1:fc:75:f0:
                    f8:4a:90:c4:6b:b6:78:e1:90:e9:d1:de:18:aa:c6:
                    52:05:4b:4a:1b:d1:bd:47:f1:6c:cc:6c:8d:be:88:
                    c0:8f:b3:47:f5:d1:e2:cc:4c:01:b0:c8:bf:0b:43:
                    00:69:d2:01:35:8e:08:52:ce:95:4f:22:a8:ee:b2:
                    fc:f4:e5:9c:c3:5e:67:f8:4d:df:53:58:ce:b5:8a:
                    21:71:c9:b8:b8:e7:82:a1:1d:45:e4:45:6d:ba:0d:
                    28:48:2f:47:9d:37:8b:80:ac:e1:1a:9c:03:0b:eb:
                    e8:93:8c:48:11:16:e9:1e:3e:d0:e9:19:0b:b9:f2:
                    45:0b:15:79:69:93:9a:57:87:fe:c5:80:1c:d9:bf:
                    21:1f:e7:2c:02:e9:c6:36:42:d6:b4:c9:93:91:12:
                    55:ea:58:c3:14:32:66:07:a6:ce:30:ec:0c:e3:5d:
                    59:0a:f3:f9:e1:5e:bf:1c:40:cc:0a:fd:da:26:7e:
                    fb:82:dd:de:c6:98:b8:fc:70:f6:52:c0:7b:6d:fd:
```

As we described before, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate.

```
[12/23/24]seed@VM:~/.../pki$ openssl rsa -in ca.key -text -noout
Enter pass phrase for ca.key:
RSA Private-Key: (4096 bit, 2 primes)
modulus:
    00:c2:83:7d:0f:7b:7c:99:95:79:e1:69:db:0f:a3:
    d7:8c:6d:72:46:a8:ad:61:b4:21:23:ba:8f:71:0c:
    84:bf:00:a6:cd:d5:1f:1a:b2:de:f0:b7:a5:a5:6c:
    c1:2f:7c:05:fa:58:93:9f:c4:94:16:a1:fc:75:f0:
    f8:4a:90:c4:6b:b6:78:e1:90:e9:d1:de:18:aa:c6:
    52:05:4b:4a:1b:d1:bd:47:f1:6c:cc:6c:8d:be:88:
    c0:8f:b3:47:f5:d1:e2:cc:4c:01:b0:c8:bf:0b:43:
    00:69:d2:01:35:8e:08:52:ce:95:4f:22:a8:ee:b2:
    fc:f4:e5:9c:c3:5e:67:f8:4d:df:53:58:ce:b5:8a:
    21:71:c9:b8:b8:e7:82:a1:1d:45:e4:45:6d:ba:0d:
    28:48:2f:47:9d:37:8b:80:ac:e1:1a:9c:03:0b:eb:
    e8:93:8c:48:11:16:e9:1e:3e:d0:e9:19:0b:b9:f2:
    45:0b:15:79:69:93:9a:57:87:fe:c5:80:1c:d9:bf:
    21:1f:e7:2c:02:e9:c6:36:42:d6:b4:c9:93:91:12:
    55:ea:58:c3:14:32:66:07:a6:ce:30:ec:0c:e3:5d:
    59:0a:f3:f9:e1:5e:bf:1c:40:cc:0a:fd:da:26:7e:
    fb:82:dd:de:c6:98:b8:fc:70:f6:52:c0:7b:6d:fd:
    db:3c:f2:97:da:d4:10:e0:f6:58:24:f7:4b:8d:47:
    dc:67:69:90:30:49:89:fb:f7:0f:fd:a7:1a:52:a9:
    4f:df:44:f7:14:66:d4:eb:8d:8c:54:45:7e:76:4f:
    4e:b4:4e:7f:01:5d:ce:6a:c3:85:3b:ef:32:98:49:
```

The output of the command are stored in two files: ca.key and ca.crt. The file ca.key contains the CA's private key, while ca.crt contains the public-key certificate.

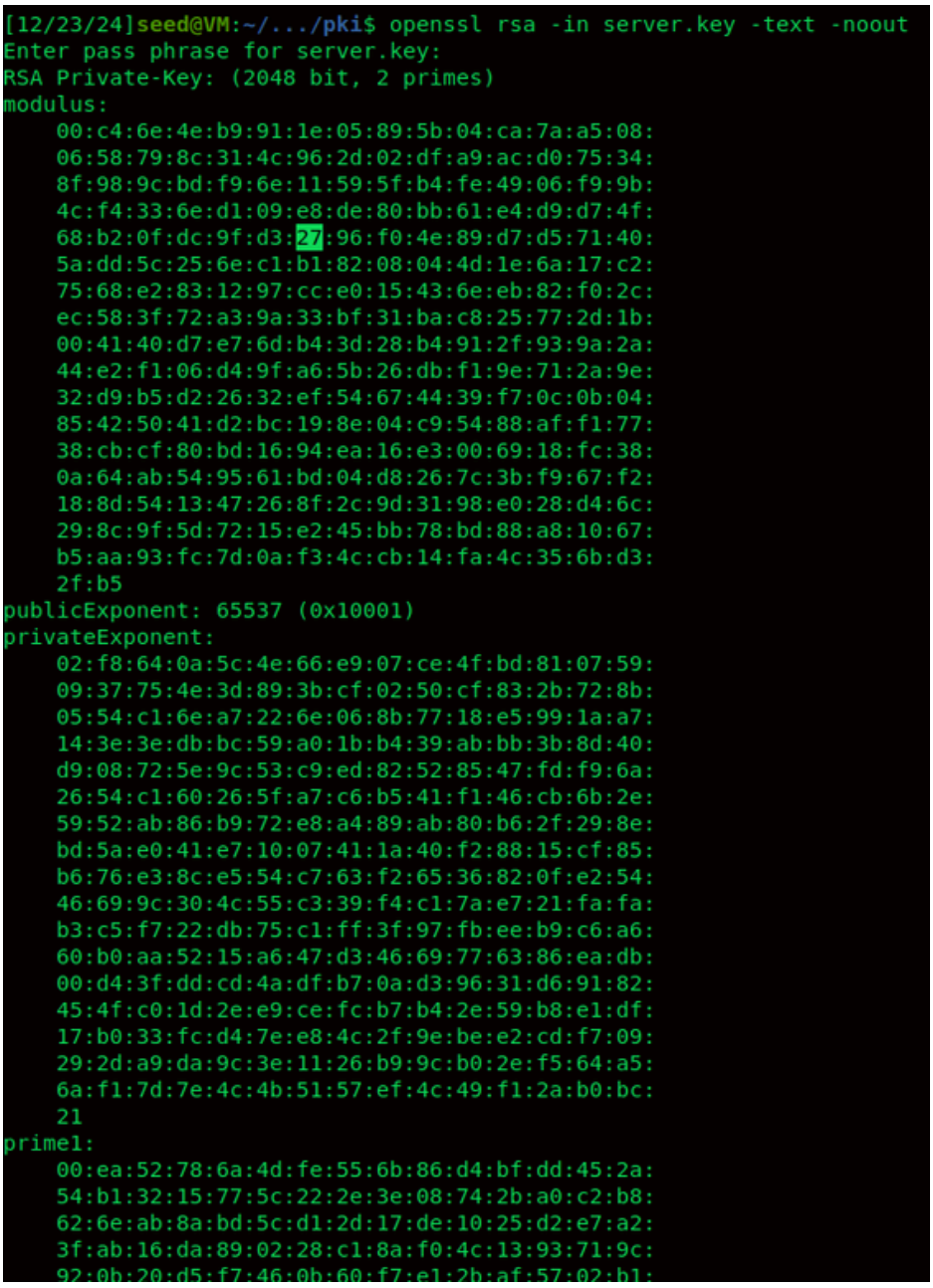
2. TASK 2

```
[12/23/24]seed@VM:~/.../pki$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=www.bank32.com/O=Bank32 Inc./
C=US" -passout pass:dees
Generating a RSA private key
.............+++++
...+++++
writing new private key to 'server.key'
-----
```

```
[12/23/24]seed@VM:~/.../pki$ openssl req -in server.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = www.bank32.com, O = Bank32 Inc., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c4:6e:4e:b9:91:1e:05:89:5b:04:ca:7a:a5:08:
                    06:58:79:8c:31:4c:96:2d:02:df:a9:ac:d0:75:34:
                    8f:98:9c:bd:f9:6e:11:59:5f:b4:fe:49:06:f9:9b:
                    4c:f4:33:6e:d1:09:e8:de:80:bb:61:e4:d9:d7:4f:
                    68:b2:0f:dc:9f:d3:27:96:f0:4e:89:d7:d5:71:40:
                    5a:dd:5c:25:6e:c1:b1:82:08:04:4d:1e:6a:17:c2:
                    75:68:e2:83:12:97:cc:e0:15:43:6e:eb:82:f0:2c:
                    ec:58:3f:72:a3:9a:33:bf:31:ba:c8:25:77:2d:1b:
                    00:41:40:d7:e7:6d:b4:3d:28:b4:91:2f:93:9a:2a:
                    44:e2:f1:06:d4:9f:a6:5b:26:db:f1:9e:71:2a:9e:
                    32:d9:b5:d2:26:32:ef:54:67:44:39:f7:0c:0b:04:
                    85:42:50:41:d2:bc:19:8e:04:c9:54:88:af:f1:77:
                    38:cb:cf:80:bd:16:94:ea:16:e3:00:69:18:fc:38:
                    0a:64:ab:54:95:61:bd:04:d8:26:7c:3b:f9:67:f2:
                    18:8d:54:13:47:26:8f:2c:9d:31:98:e0:28:d4:6c:
                    29:8c:9f:5d:72:15:e2:45:bb:78:bd:88:a8:10:67:
                    b5:aa:93:fc:7d:0a:f3:4c:cb:14:fa:4c:35:6b:d3:
                    2f:b5
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
```

The command to generate a CSR is quite similar to the one we used in creating the self-signed certificate for the CA. The only difference is the -x509 option. Without it, the command generates a re-quest; with it, the command generates a self-signed certificate. The following command generate a CSR for www.bank32.com (you should use your own server name):

2. TASK 2



The command will generate a pair of public/private key, and then create a certificate signing request
from the public key. We can use the following command to look at the decoded content of the CSR and
private key files:

2. TASK 2

```
[12/23/24]seed@VM:~/.../pki$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=www.bank32.com/O=Bank32 Inc./C=US" -addext "subjectAlt
Name = DNS:www.bank32.com, DNS:www.bank32A.com, DNS:www.bank32B.com" -passout pass:dees
Generating a RSA private key
..+++++
.....+++++
writing new private key to 'server.key'
-----
```

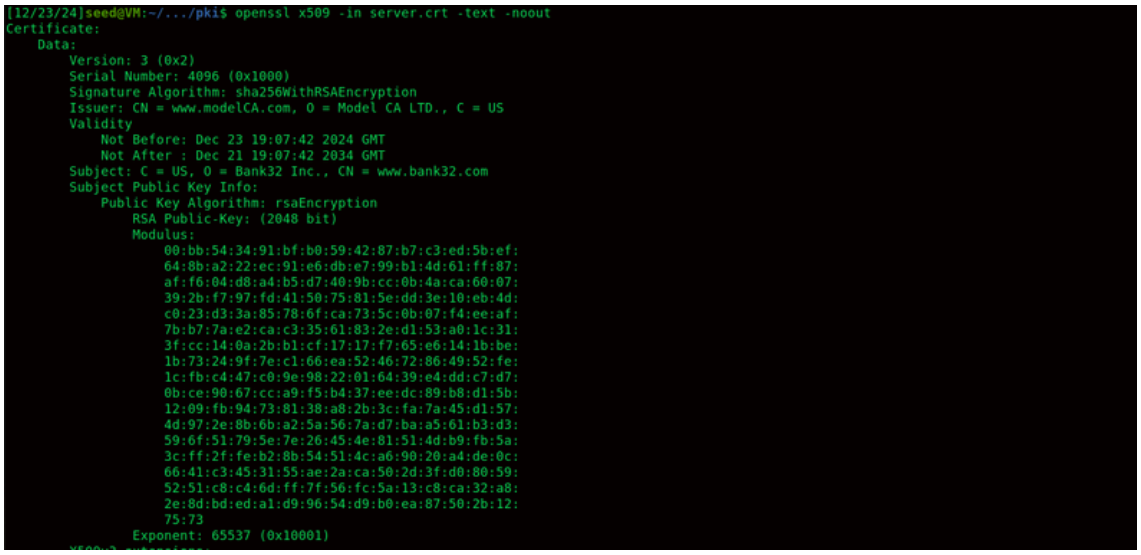
To generate a certificate signing request with such a field, we can put all the necessary information in a configuration file or at the command line. We will use the command-line approach in this task (the configuration file approach is used in another SEED lab, the TLS lab).

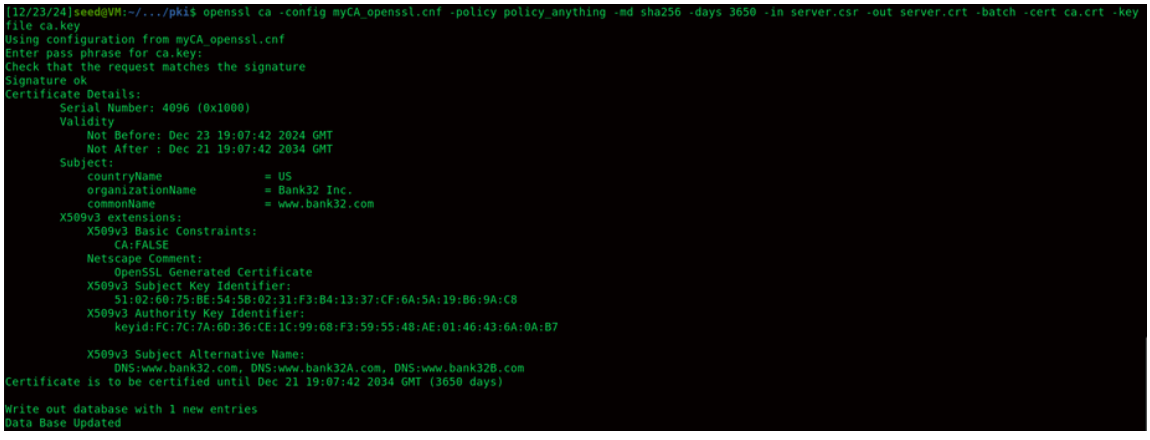
3. TASK 3

```
# Extension copying option: use with caution.
copy_extensions = copy
```

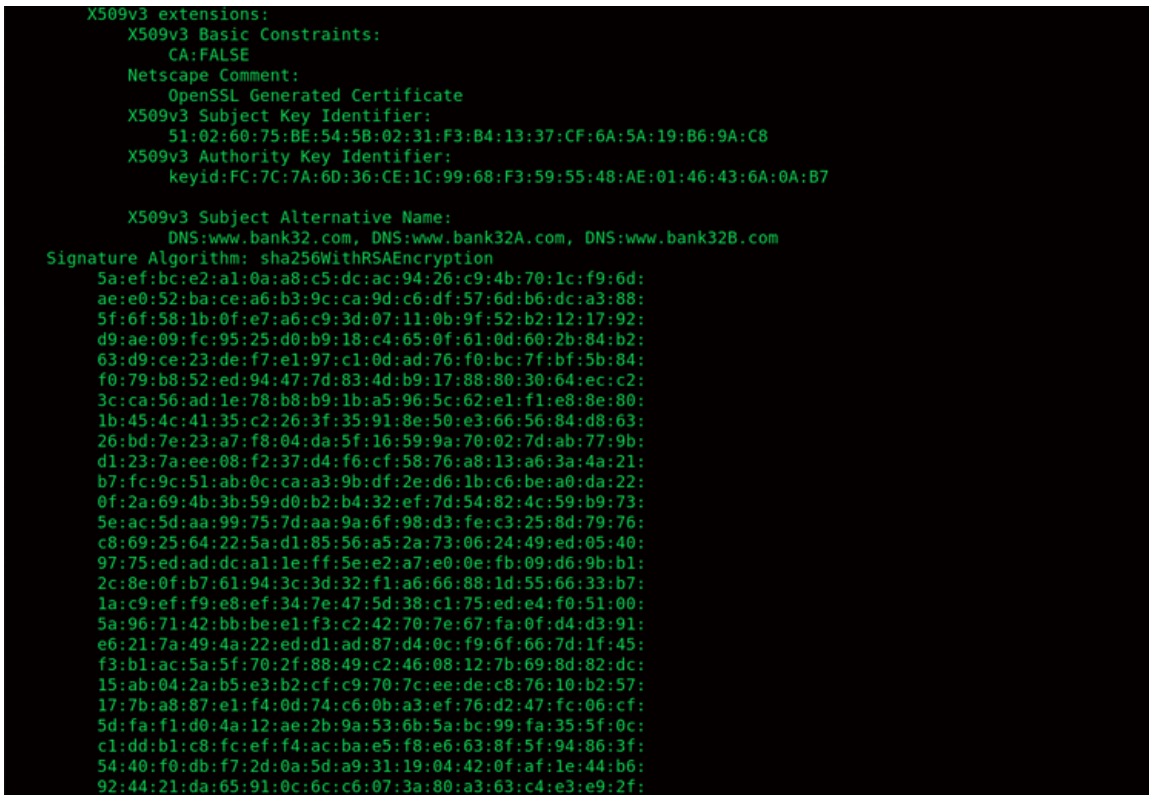
For security reasons, the default setting in openssl.cnf does not allow the "openssl ca" command to copy the extension field from the request to the final certificate

```
[12/23/24]seed@VM:~/.../pki$ cp /etc/ssl/openssl.cnf myCA_openssl.cnf
[12/23/24]seed@VM:~/.../pki$ ls
ca.crt   ca.key   democa   myCA_openssl.cnf   openssl.cnf   server.csr   server.key
```





The generated server certificate showing the CA's signature and the inherited subject alternative names."

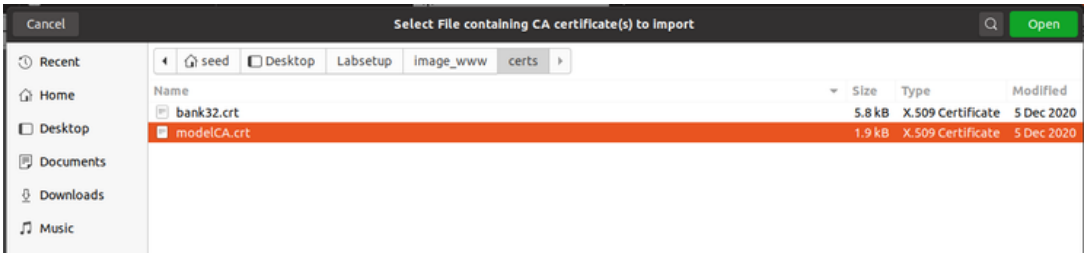


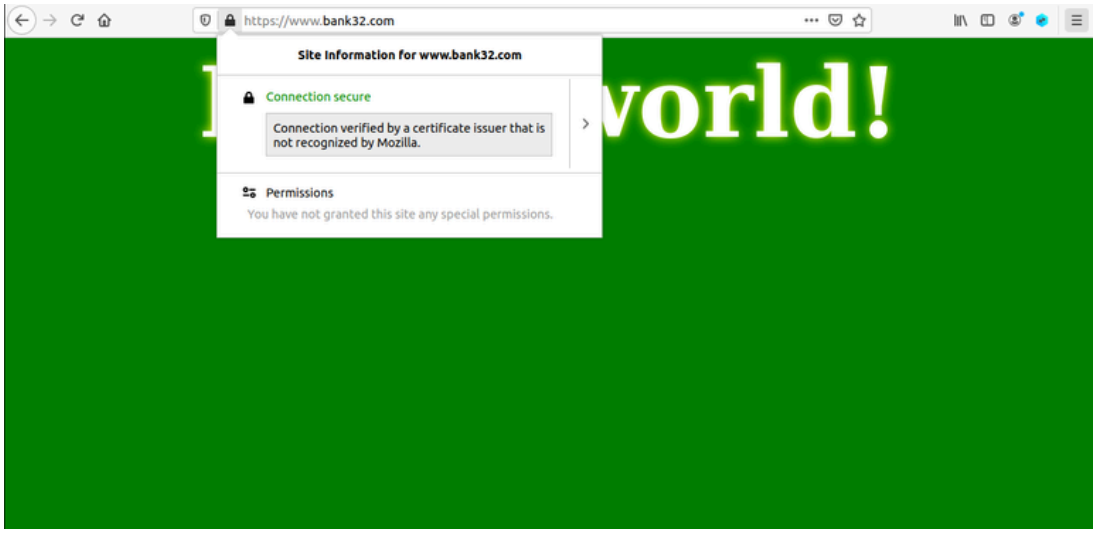
Verification of the certificate chain showing our server certificate was signed by our CA.

4. TASK 4

```
Successfully built d60c4606af98
Successfully tagged seed-image-www-pki:latest
[12/23/24]seed@VM:~/.../Labsetup$ docker-compose up -d
www-10.9.0.80 is up-to-date
[12/23/24]seed@VM:~/.../Labsetup$ dockps
8c43e4c4d38f  www-10.9.0.80
[12/23/24]seed@VM:~/.../Labsetup$ docksh 8c^C
[12/23/24]seed@VM:~/.../Labsetup$ docksh 8c43e4c4d38f
root@8c43e4c4d38f:/# service apache2 start
 * Starting Apache httpd web server apache2
Enter passphrase for SSL/TLS keys for www.bank32.com:443 (RSA):
 *
root@8c43e4c4d38f:/# service apache2 status
 * apache2 is running
root@8c43e4c4d38f:/#
```

In this task, we will see how public-key certificates are used by websites to secure web browsing. We will set up an HTTPS website based Apache. The Apache server, which is already installed in our container, supports the HTTPS protocol.

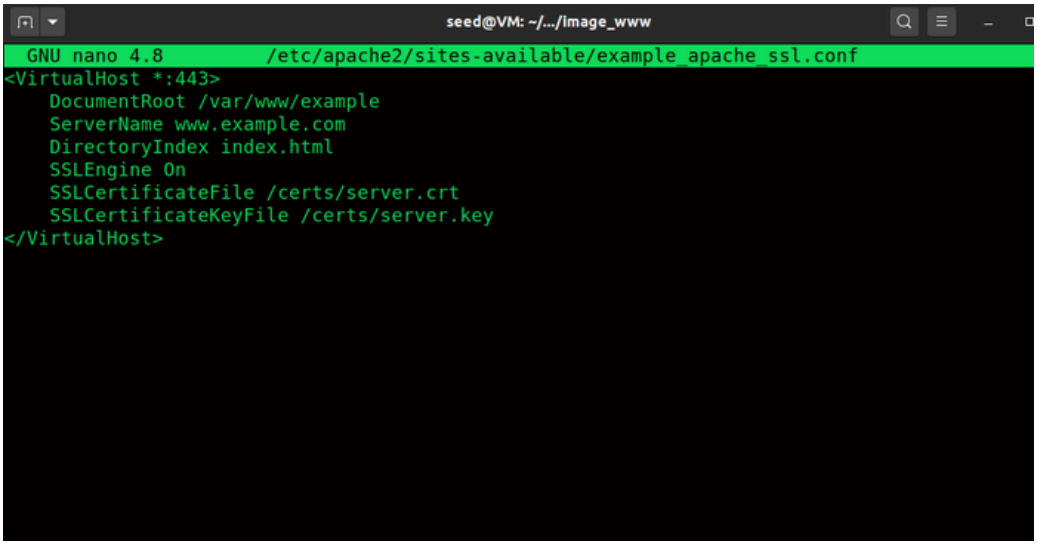


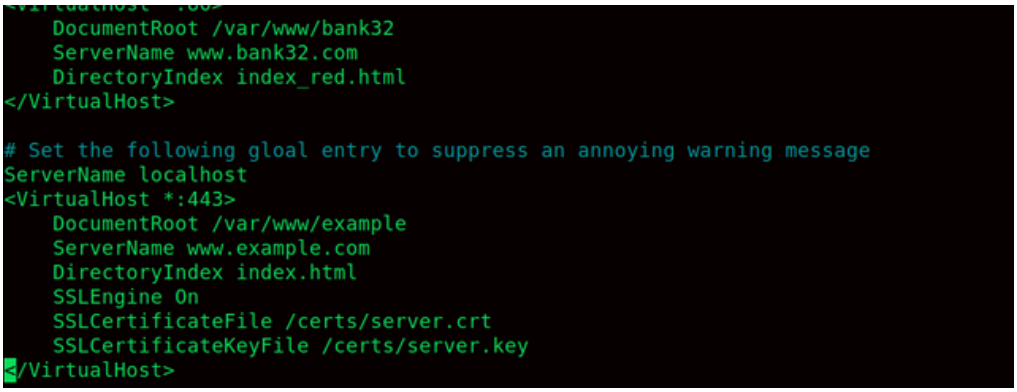


Deploying the Certificate in an Apache-Based HTTPS Website

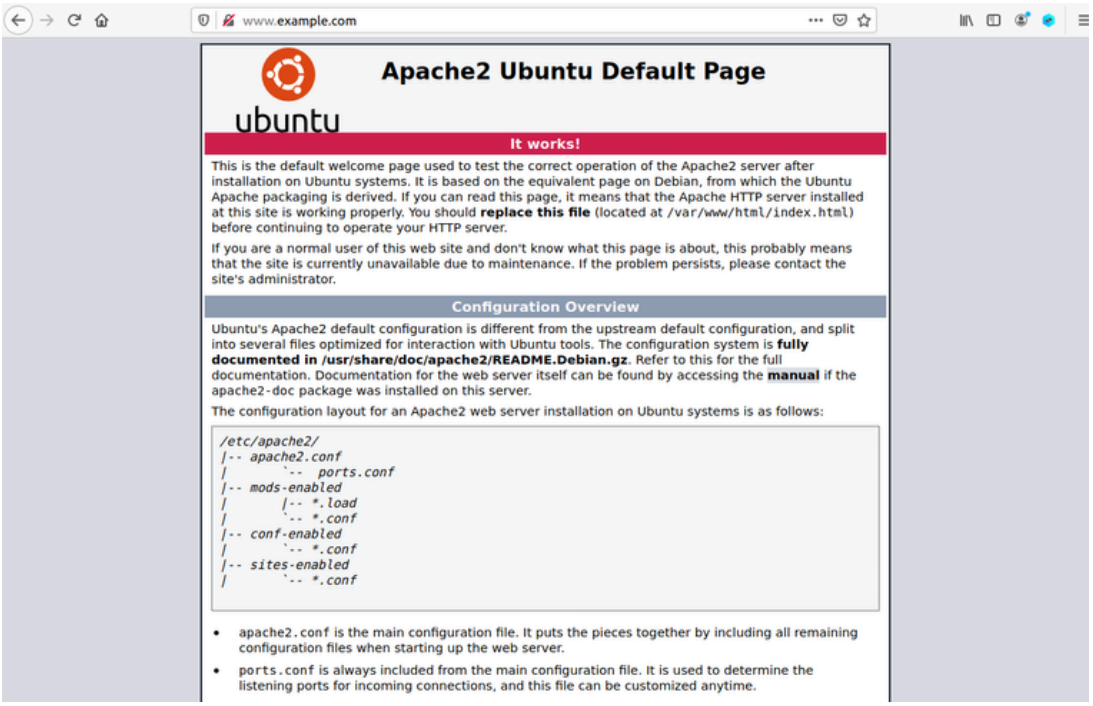
We configured an Apache web server to use the generated certificate for HTTPS. Our process included:
Updating the Apache VirtualHost configuration file to include the certificate and private key paths.
Enabling Apache's SSL module and the site's SSL configuration using a2enmod ssl and a2ensite.
Starting the Apache server and verifying the HTTPS connection in a browser.

5. TASK 5


seed@VM: ~/.../image_www
GNU nano 4.8 /etc/apache2/sites-available/example_apache_ssl.conf
<VirtualHost *:443>
DocumentRoot /var/www/example
ServerName www.example.com
DirectoryIndex index.html
SSLEngine On
SSLCertificateFile /certs/server.crt
SSLCertificateKeyFile /certs/server.key
</VirtualHost>


DocumentRoot /var/www/bank32
ServerName www.bank32.com
DirectoryIndex index_red.html
</VirtualHost>
Set the following gloal entry to suppress an annoying warning message
ServerName localhost
<VirtualHost *:443>
DocumentRoot /var/www/example
ServerName www.example.com
DirectoryIndex index.html
SSLEngine On
SSLCertificateFile /certs/server.crt
SSLCertificateKeyFile /certs/server.key
</VirtualHost>

5. TASK 5



Launching a Man-in-the-Middle (MITM) Attack

We simulated a MITM attack to understand how PKI mitigates such threats. The steps were:
Setting up a malicious HTTPS server impersonating a legitimate website.
Redirecting the victim’s traffic to the malicious server using a simulated DNS spoofing attack.
Observing the browser's behavior when it detected the untrusted certificate.

6. TASK 6

```
[12/23/24]seed@VM:~/.../image_www$ openssl req -newkey rsa:2048 -sha256 -keyout compromised.ke
y -out compromised.csr -subj "/CN=www.example.com/O=Fake Inc./C=US" -passout pass:dees
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'compromised.key'
-----
```

```
[12/23/24]seed@VM:~/.../pki$ openssl ca -config openssl.cnf -policy policy_anything -md sha256
 -days 3650 -in compromised.csr -out compromised.crt -batch -cert ca.crt -keyfile ca.key
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Dec 23 21:39:15 2024 GMT
            Not After : Dec 21 21:39:15 2034 GMT
        Subject:
            countryName               = US
            organizationName          = Fake Inc.
            commonName                = www.example.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                28:C5:29:AF:3A:84:11:12:4E:34:18:AA:62:E1:59:ED:CA:D3:07:E9
            X509v3 Authority Key Identifier:
                keyid:FC:7C:7A:6D:36:CE:1C:99:68:F3:59:55:48:AE:01:46:43:6A:0A:B7

Certificate is to be certified until Dec 21 21:39:15 2034 GMT (3650 days)
```

Launching a MITM Attack with a Compromised CA

To explore the consequences of a compromised CA, we assumed the CA's private key was stolen. The steps included:
Using the compromised private key to generate a fraudulent certificate for the malicious server.
Demonstrating how the browser trusted the fraudulent certificate, enabling a successful MITM attack.