PKI Lab Seed Lab

Task 1: Becoming a CA Authority

First we have **cd** to the **Labsetup** folder. Then we have to make a directory called demoCA and its subdirectory. We also copy the **openssl.cnf** file from **/usr/lib/ssl** to the **LabSetup** file. This is the open ssl configuration file which is needed for the following tasks.

In this picture below we do all the necessary setups needed to do task 1 according to the question material below.

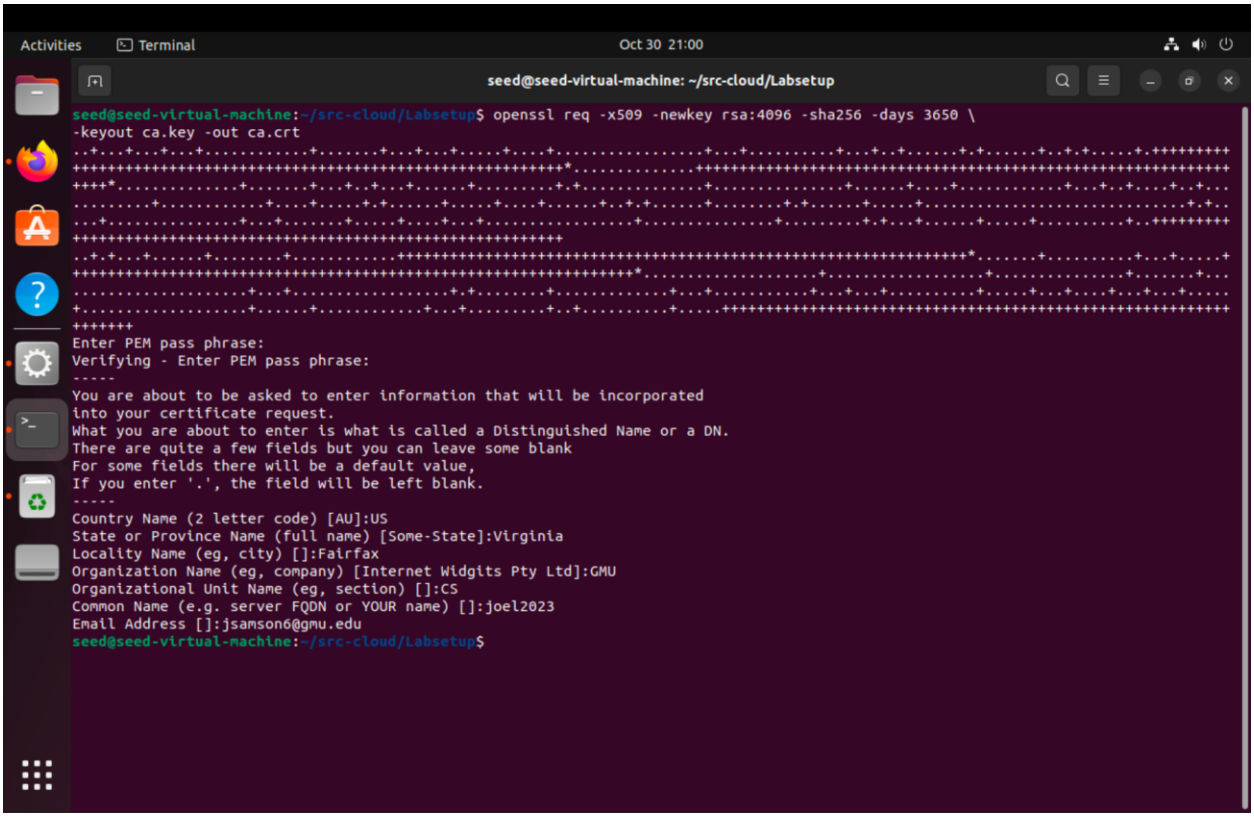
[CA_default]
dir = ./demoCA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of # several certs with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.
serial = $dir/serial # The current serial number

In **index.txt** we leave it empty and for the file string we store any value
I ran the command **echo 01 > serial** to store the value 01 into serial.

Now we need to generate a self signed certificate for our CA. we do so by running the following command

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \ -keyout ca.key -out ca.crt

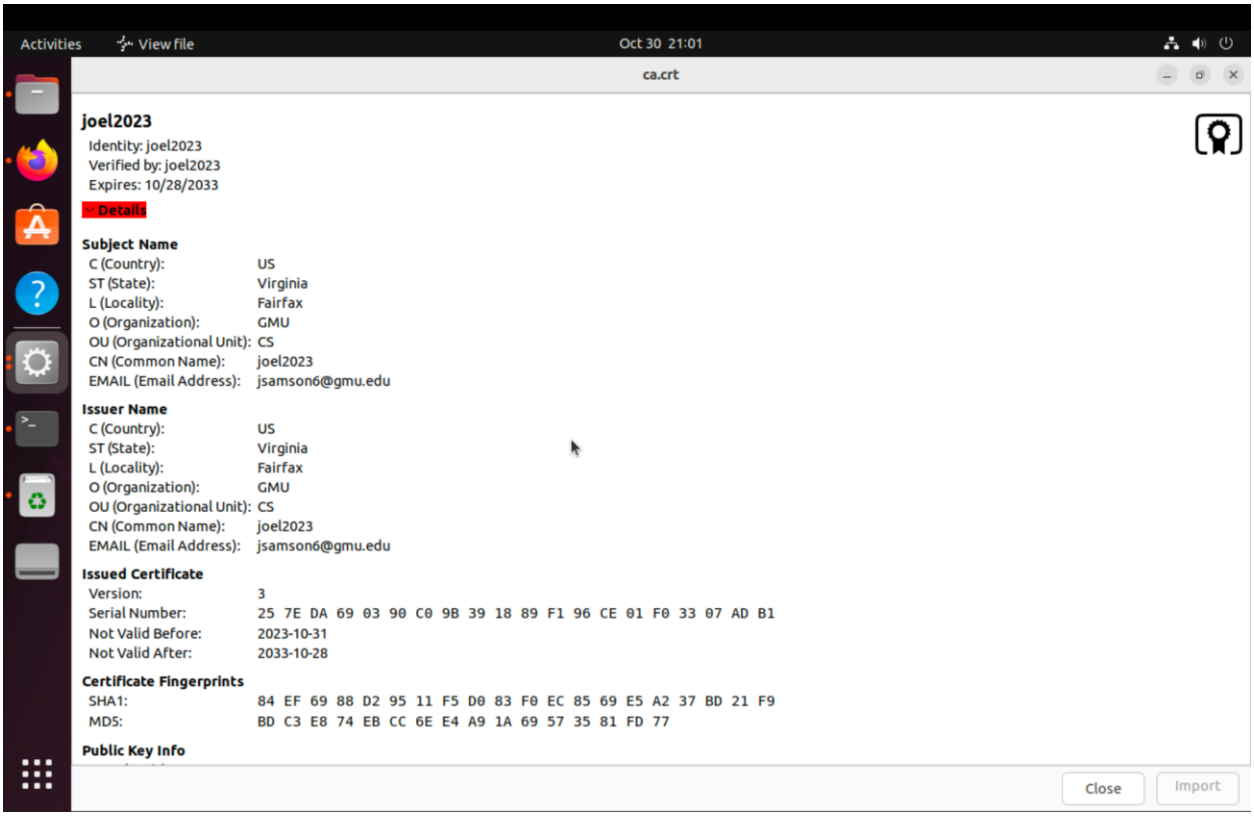
We will be asked to enter some details for which I have given the information below
Password: joel
Country: US
State: Virginia
City: Fairfax
Organization Name: GMU
Organization Unit Name: CS
Company Name: joel2023
Email: jsamson6@gmu.edu

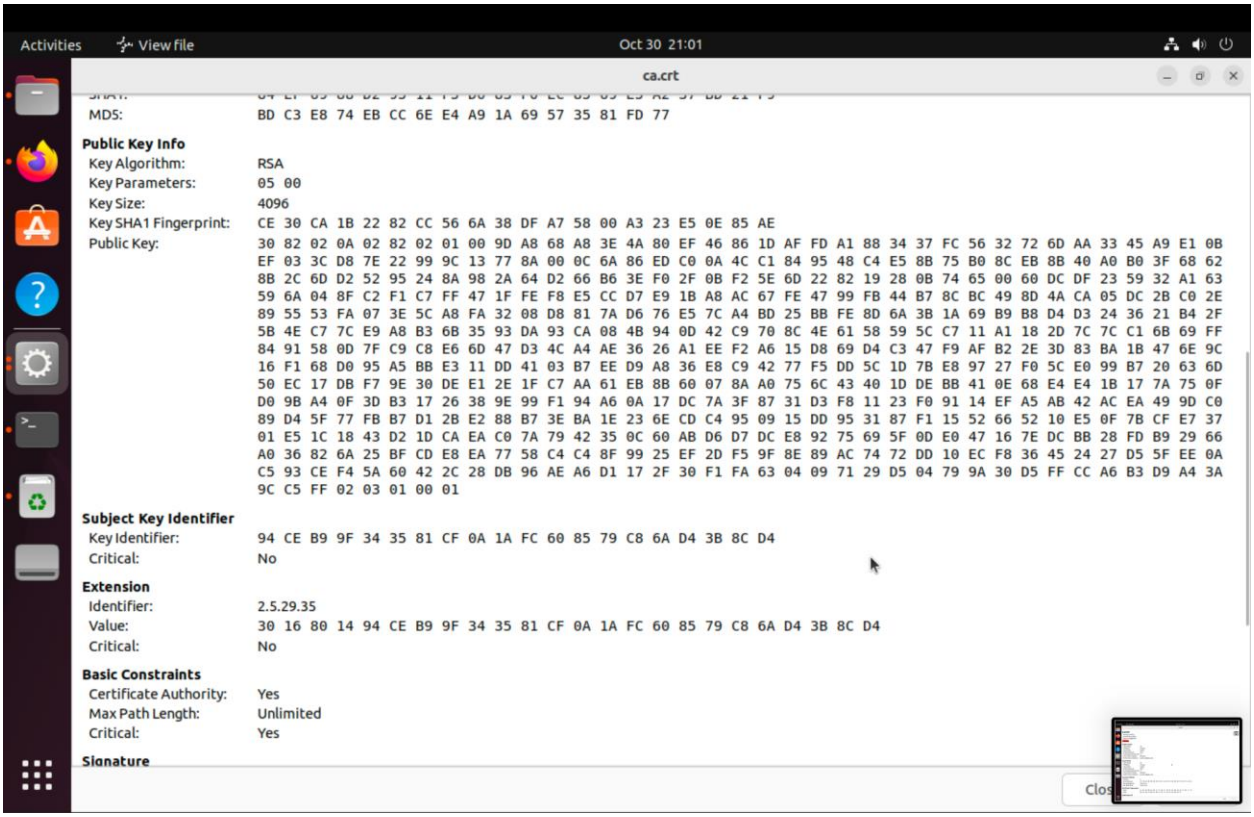


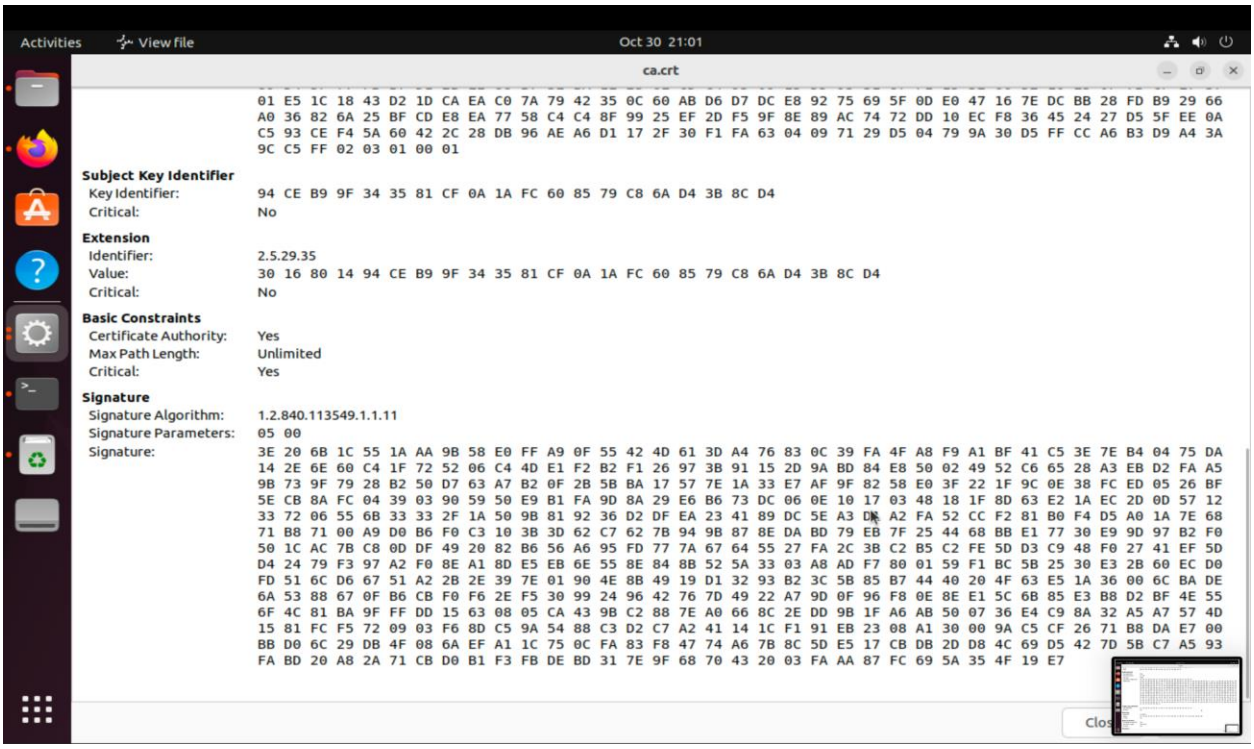
The output of the command are stored in two files: **ca.key** and ca.crt. The file ca.key contains the CA's private key, while ca.crt contains the public-key certificate

Files after command:----------

ca.crt file details:—---



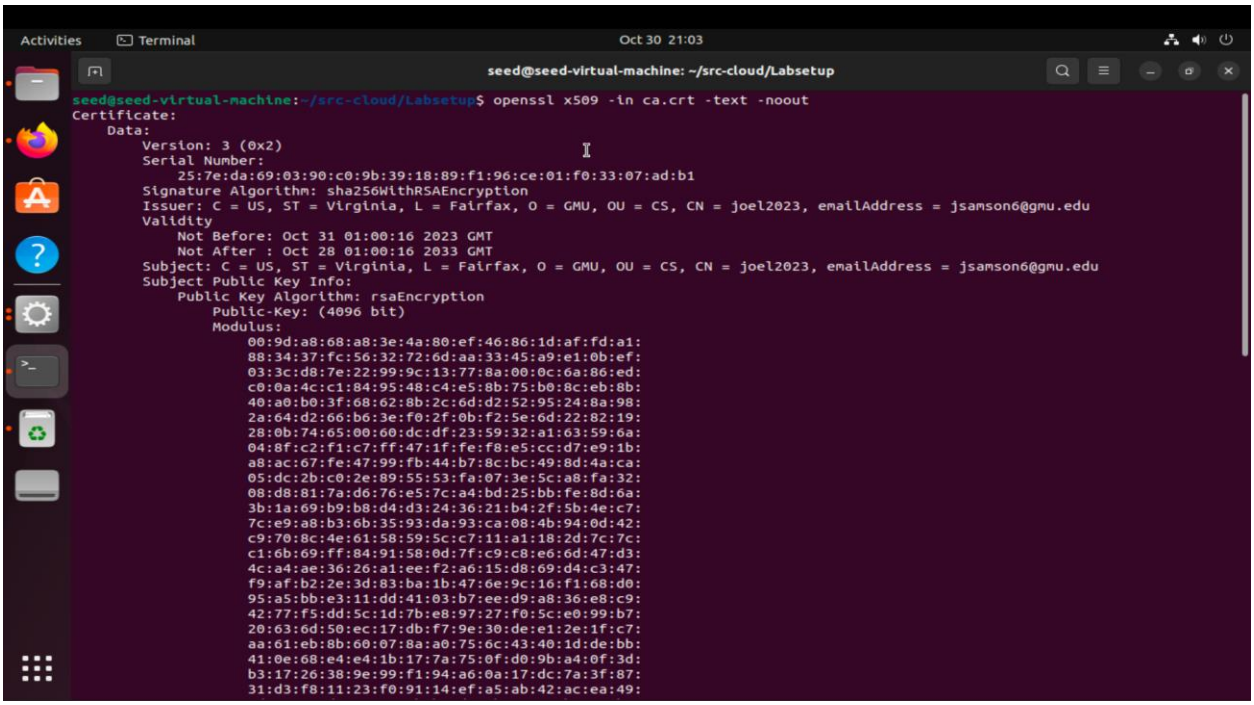


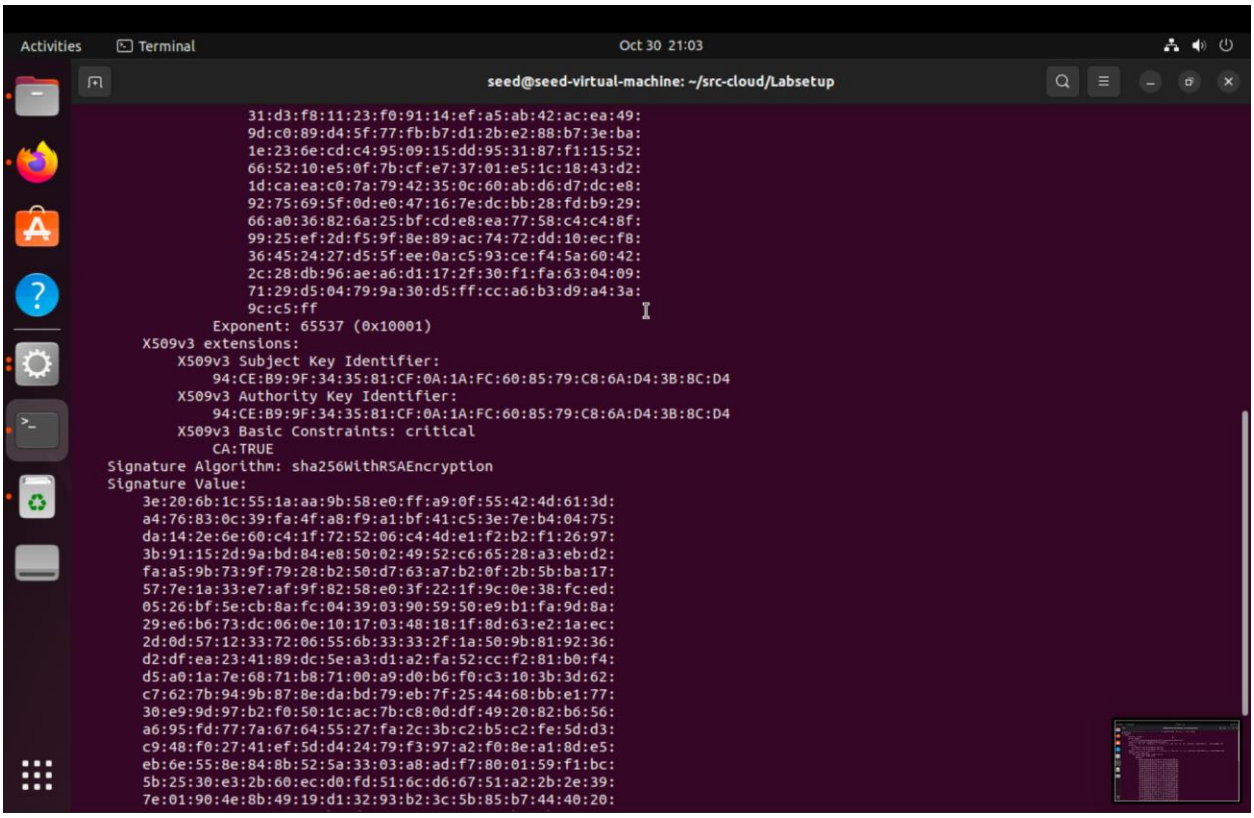


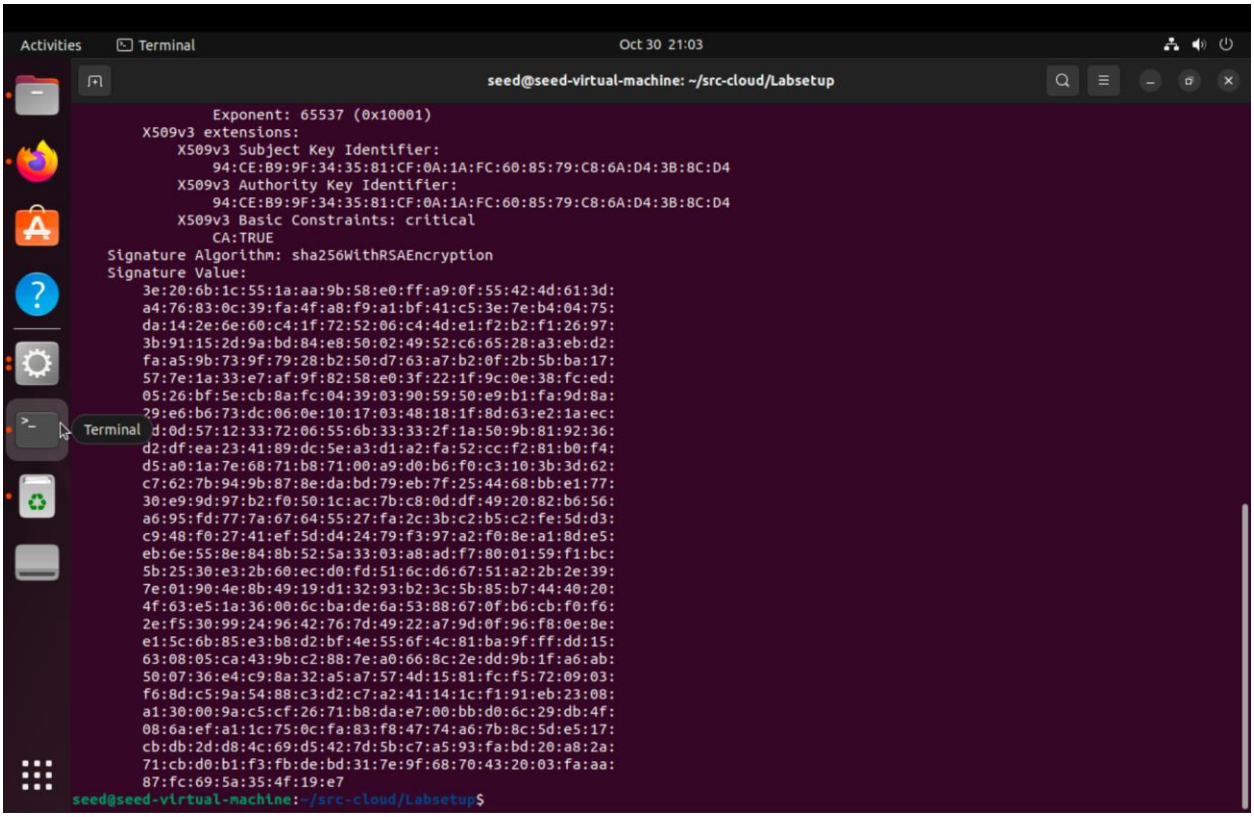
We can use the following command to look at the decoded content of the X509 certificate

openssl x509 -in ca.crt -text -noout

Output as follows

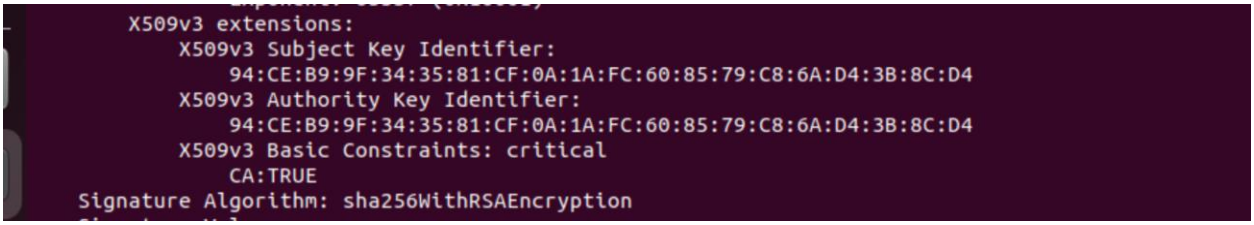



Activities
Terminal
Oct 30 21:03
seed@seed-virtual-machine: ~/src-cloud/Labsetup
31:d3:f8:11:23:f0:91:14:ef:a5:ab:42:ac:ea:49:
9d:c0:89:d4:5f:77:fb:b7:d1:2b:e2:88:b7:3e:ba:
1e:23:6e:cd:c4:95:09:15:dd:95:31:87:f1:15:52:
66:52:10:e5:0f:7b:cf:e7:37:01:e5:1c:18:43:d2:
1d:ca:ea:c0:7a:79:42:35:0c:60:ab:d6:d7:dc:e8:
92:75:69:5f:0d:e0:47:16:7e:dc:bb:28:fd:b9:29:
66:a0:36:82:6a:25:bf:cd:e8:ea:77:58:c4:c4:8f:
99:25:ef:2d:f5:9f:8e:89:ac:74:72:dd:10:ec:f8:
36:45:24:27:d5:5f:ee:0a:c5:93:ce:f4:5a:60:42:
2c:28:db:96:ae:a6:d1:17:2f:30:f1:fa:63:04:09:
71:29:d5:04:79:9a:30:d5:ff:cc:a6:b3:d9:a4:3a:
9c:c5:ff
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Key Identifier:
94:CE:B9:9F:34:35:81:CF:0A:1A:FC:60:85:79:C8:6A:D4:3B:8C:D4
X509v3 Authority Key Identifier:
94:CE:B9:9F:34:35:81:CF:0A:1A:FC:60:85:79:C8:6A:D4:3B:8C:D4
X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
3e:20:6b:1c:55:1a:aa:9b:58:e0:ff:a9:0f:55:42:4d:61:3d:
a4:76:83:0c:39:fa:4f:a8:f9:a1:bf:41:c5:3e:7e:b4:04:75:
da:14:2e:6e:60:c4:1f:72:52:06:c4:4d:e1:f2:b2:f1:26:97:
3b:91:15:2d:9a:bd:84:e8:50:02:49:52:c6:65:28:a3:eb:d2:
fa:a5:9b:73:9f:79:28:b2:50:d7:63:a7:b2:0f:2b:5b:ba:17:
57:7e:1a:33:e7:af:9f:82:58:e0:3f:22:1f:9c:0e:38:fc:ed:
05:26:bf:5e:cb:8a:fc:04:39:03:90:59:50:e9:b1:fa:9d:8a:
29:e6:b6:73:dc:06:0e:10:17:03:48:18:1f:8d:63:e2:1a:ec:
2d:0d:57:12:33:72:06:55:6b:33:33:2f:1a:50:9b:81:92:36:
d2:df:ea:23:41:89:dc:5e:a3:d1:a2:fa:52:cc:f2:81:b0:f4:
d5:a0:1a:7e:68:71:b8:71:00:a9:d0:b6:f0:c3:10:3b:3d:62:
c7:62:7b:94:9b:87:8e:da:bd:79:eb:7f:25:44:68:bb:e1:77:
30:e9:9d:97:b2:f0:50:1c:ac:7b:c8:0d:df:49:20:82:b6:56:
a6:95:fd:77:7a:67:64:55:27:fa:2c:3b:c2:b5:c2:fe:5d:d3:
c9:48:f0:27:41:ef:5d:d4:24:79:f3:97:a2:f0:8e:a1:8d:e5:
eb:6e:55:8e:84:8b:52:5a:33:03:a8:ad:f7:80:01:59:f1:bc:
5b:25:30:e3:2b:60:ec:d0:fd:51:6c:d6:67:51:a2:2b:2e:39:
7e:01:90:4e:8b:49:19:d1:32:93:b2:3c:5b:85:b7:44:40:20:


Activities
Terminal
Oct 30 21:03
seed@seed-virtual-machine: ~/src-cloud/Labsetup
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Key Identifier:
94:CE:B9:9F:34:35:81:CF:0A:1A:FC:60:85:79:C8:6A:D4:3B:8C:D4
X509v3 Authority Key Identifier:
94:CE:B9:9F:34:35:81:CF:0A:1A:FC:60:85:79:C8:6A:D4:3B:8C:D4
X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
3e:20:6b:1c:55:1a:aa:9b:58:e0:ff:a9:0f:55:42:4d:61:3d:
a4:76:83:0c:39:fa:4f:a8:f9:a1:bf:41:c5:3e:7e:b4:04:75:
da:14:2e:6e:60:c4:1f:72:52:06:c4:4d:e1:f2:b2:f1:26:97:
3b:91:15:2d:9a:bd:84:e8:50:02:49:52:c6:65:28:a3:eb:d2:
fa:a5:9b:73:9f:79:28:b2:50:d7:63:a7:b2:0f:2b:5b:ba:17:
57:7e:1a:33:e7:af:9f:82:58:e0:3f:22:1f:9c:0e:38:fc:ed:
05:26:bf:5e:cb:8a:fc:04:39:03:90:59:50:e9:b1:fa:9d:8a:
29:e6:b6:73:dc:06:0e:10:17:03:48:18:1f:8d:63:e2:1a:ec:
Terminal
d:0d:57:12:33:72:06:55:6b:33:33:2f:1a:50:9b:81:92:36:
d2:df:ea:23:41:89:dc:5e:a3:d1:a2:fa:52:cc:f2:81:b0:f4:
d5:a0:1a:7e:68:71:b8:71:00:a9:d0:b6:f0:c3:10:3b:3d:62:
c7:62:7b:94:9b:87:8e:da:bd:79:eb:7f:25:44:68:bb:e1:77:
30:e9:9d:97:b2:f0:50:1c:ac:7b:c8:0d:df:49:20:82:b6:56:
a6:95:fd:77:7a:67:64:55:27:fa:2c:3b:c2:b5:c2:fe:5d:d3:
c9:48:f0:27:41:ef:5d:d4:24:79:f3:97:a2:f0:8e:a1:8d:e5:
eb:6e:55:8e:84:8b:52:5a:33:03:a8:ad:f7:80:01:59:f1:bc:
5b:25:30:e3:2b:60:ec:d0:fd:51:6c:d6:67:51:a2:2b:2e:39:
7e:01:90:4e:8b:49:19:d1:32:93:b2:3c:5b:85:b7:44:40:20:
4f:63:e5:1a:36:00:6c:ba:de:6a:53:88:67:0f:b6:cb:f0:f6:
2e:f5:30:99:24:96:42:76:7d:49:22:a7:9d:0f:96:f8:0e:8e:
e1:5c:6b:85:e3:b8:d2:bf:4e:55:6f:4c:81:ba:9f:ff:dd:15:
63:08:05:ca:43:9b:c2:88:7e:a0:66:8c:2e:dd:9b:1f:a6:ab:
50:07:36:e4:c9:8a:32:a5:a7:57:4d:15:81:fc:f5:72:09:03:
f6:8d:c5:9a:54:88:c3:d2:c7:a2:41:14:1c:f1:91:eb:23:08:
a1:30:00:9a:c5:cf:26:71:b8:da:e7:00:bb:d0:6c:29:db:4f:
08:6a:ef:a1:1c:75:0c:fa:83:f8:47:74:a6:7b:8c:5d:e5:17:
cb:db:2d:d8:4c:69:d5:42:7d:5b:c7:a5:93:fa:bd:20:a8:2a:
71:cb:d0:b1:f3:fb:de:bd:31:7e:9f:68:70:43:20:03:fa:aa:
87:fc:69:5a:35:4f:19:e7
seed@seed-virtual-machine:~/src-cloud/Labsetup$

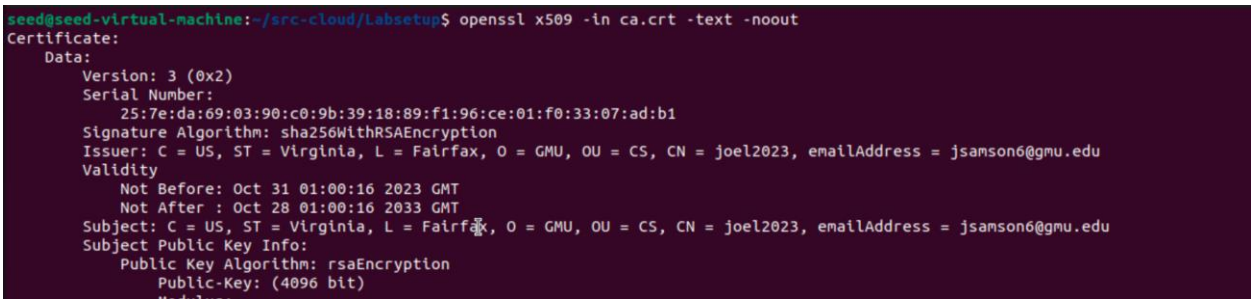
1) **What part of the certificate indicates this is a CA's certificate?**

After running the command to look at the decoded content of the X509 certificate
This part of the decoded X509 certificate indicates the CA certificate.
The part where it is written CA: True



2) **What part of the certificate indicates this is a self-signed certificate?**

After running the command to look at the decoded content of the X509 certificate
If the issuer and subject fields are identical as shown in the following figure then we know it is a self signed certificate.

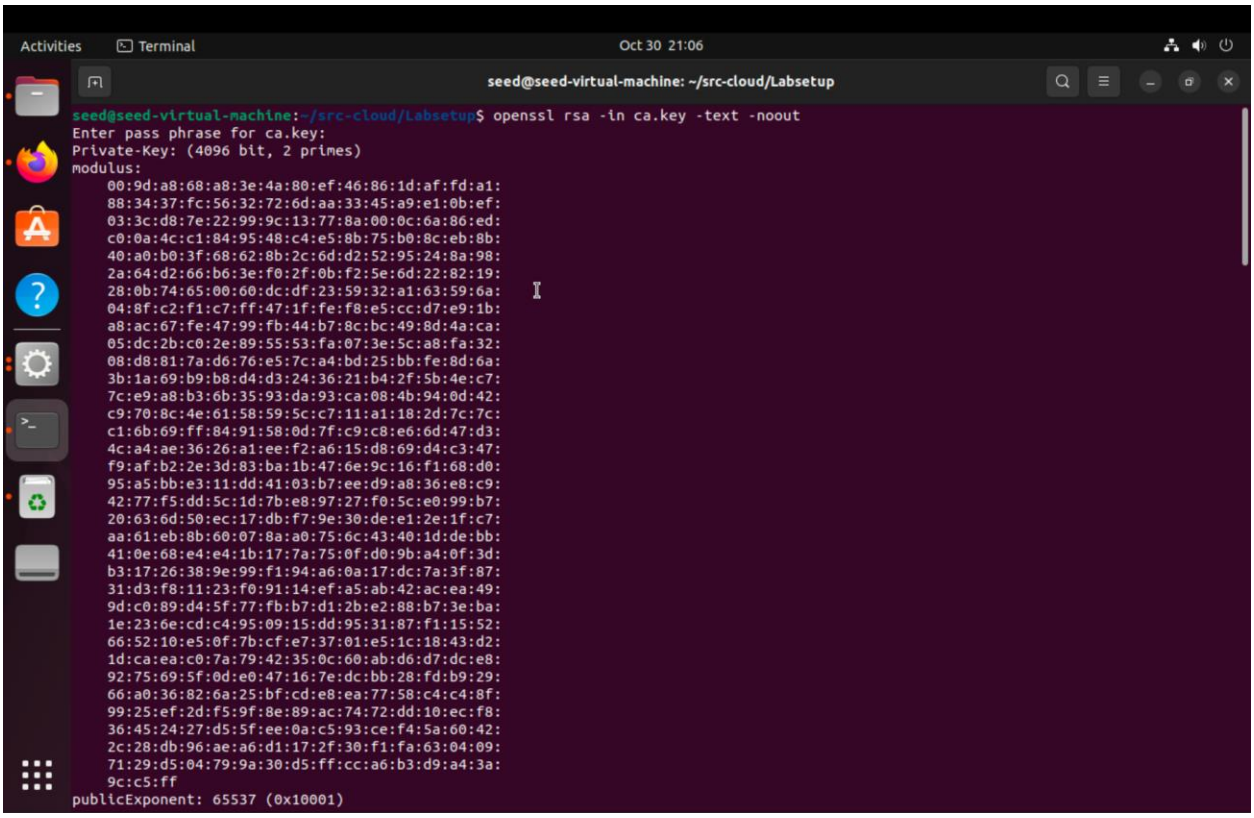


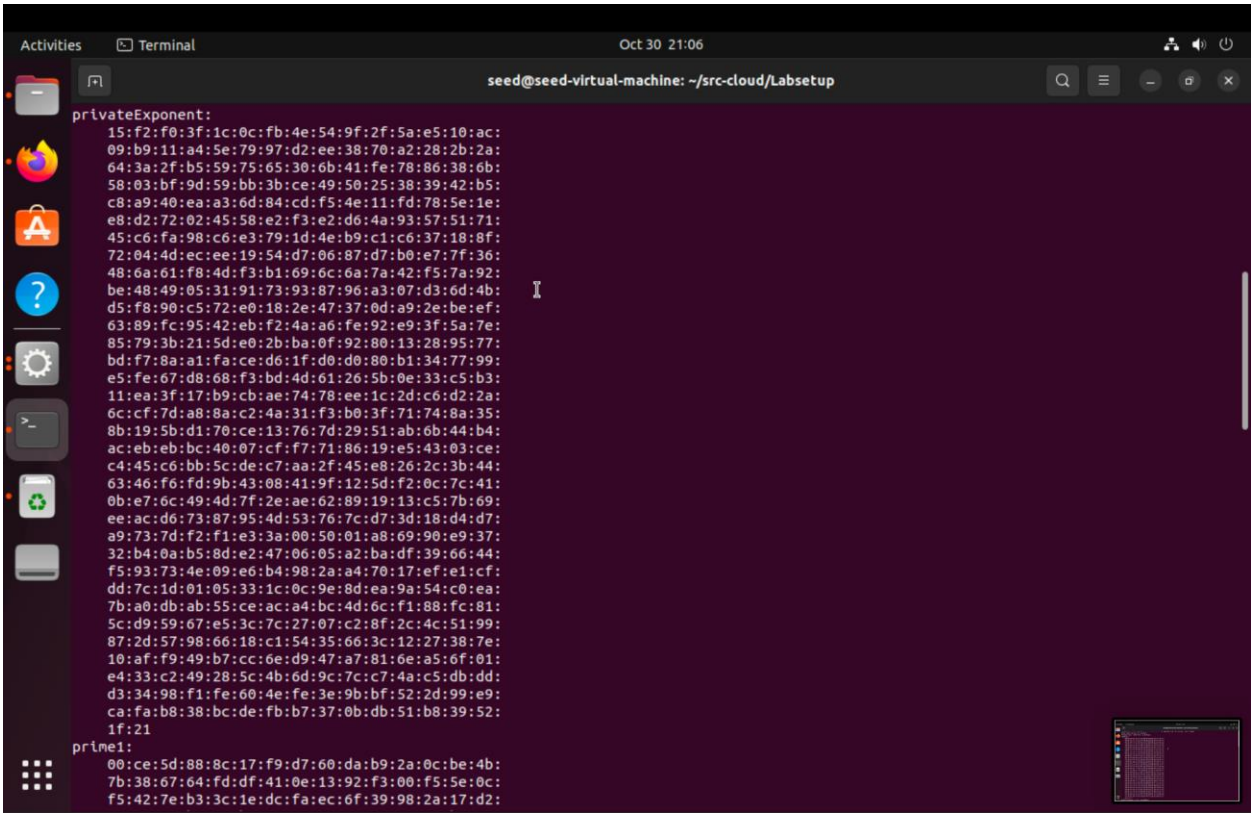
3) **In the RSA algorithm, we have a public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq. Please identify the values for these elements in your certificate and key files.**

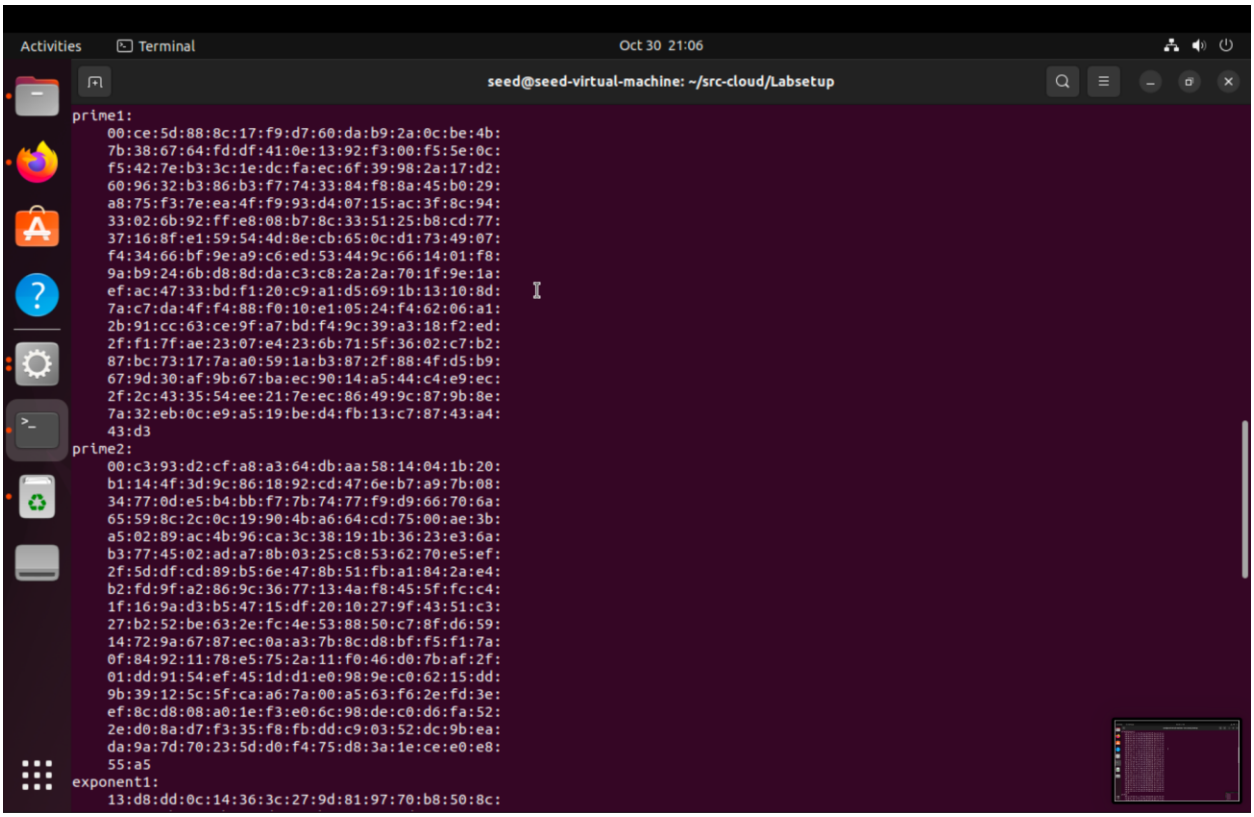
After running the the following commands to look at the decoded content the RSA key

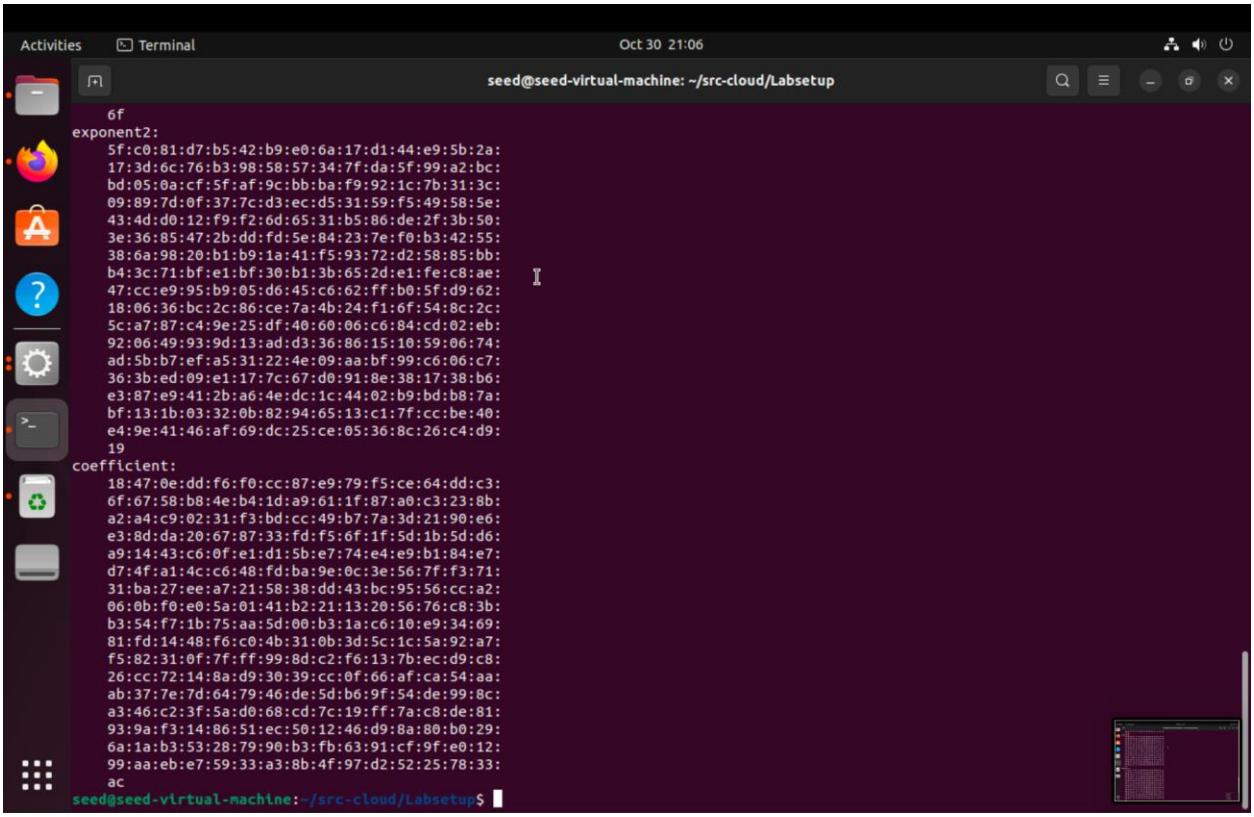
openssl rsa -in ca.key -text -noout

We get the following output
This shows us the values of the public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq.


Activities
Terminal
Oct 30 21:06
seed@seed-virtual-machine: ~/src-cloud/Labsetup
seed@seed-virtual-machine:~/src-cloud/Labsetup$ openssl rsa -in ca.key -text -noout
Enter pass phrase for ca.key:
Private-Key: (4096 bit, 2 primes)
modulus:
00:9d:a8:68:a8:3e:4a:80:ef:46:86:1d:af:fd:a1:
88:34:37:fc:56:32:72:6d:aa:33:45:a9:e1:0b:ef:
03:3c:d8:7e:22:99:9c:13:77:8a:00:0c:6a:86:ed:
c0:0a:4c:c1:84:95:48:c4:e5:8b:75:b0:8c:eb:8b:
40:a0:b0:3f:68:62:8b:2c:6d:d2:52:95:24:8a:98:
2a:64:d2:66:b6:3e:f0:2f:0b:f2:5e:6d:22:82:19:
28:0b:74:65:00:60:dc:df:23:59:32:a1:63:59:6a:
04:8f:c2:f1:c7:ff:47:1f:fe:f8:e5:cc:d7:e9:1b:
a8:ac:67:fe:47:99:fb:44:b7:8c:bc:49:8d:4a:ca:
05:dc:2b:c0:2e:89:55:53:fa:07:3e:5c:a8:fa:32:
08:d8:81:7a:d6:76:e5:7c:a4:bd:25:bb:fe:8d:6a:
3b:1a:69:b9:b8:d4:d3:24:36:21:b4:2f:5b:4e:c7:
7c:e9:a8:b3:6b:35:93:da:93:ca:08:4b:94:0d:42:
c9:70:8c:4e:61:58:59:5c:c7:11:a1:18:2d:7c:7c:
c1:6b:69:ff:84:91:58:0d:7f:c9:c8:e6:6d:47:d3:
4c:a4:ae:36:26:a1:ee:f2:a6:15:d8:69:d4:c3:47:
f9:af:b2:2e:3d:83:ba:1b:47:6e:9c:16:f1:68:d0:
95:a5:bb:e3:11:dd:41:03:b7:ee:d9:a8:36:e8:c9:
42:77:f5:dd:5c:1d:7b:e8:97:27:f0:5c:e0:99:b7:
20:63:6d:50:ec:17:db:f7:9e:30:de:e1:2e:1f:c7:
aa:61:eb:8b:60:07:8a:a0:75:6c:43:40:1d:de:bb:
41:0e:68:e4:e4:1b:17:7a:75:0f:d0:9b:a4:0f:3d:
b3:17:26:38:9e:99:f1:94:a6:0a:17:dc:7a:3f:87:
31:d3:f8:11:23:f0:91:14:ef:a5:ab:42:ac:ea:49:
9d:c0:89:d4:5f:77:fb:b7:d1:2b:e2:88:b7:3e:ba:
1e:23:6e:cd:c4:95:09:15:dd:95:31:87:f1:15:52:
66:52:10:e5:0f:7b:cf:e7:37:01:e5:1c:18:43:d2:
1d:ca:ea:c0:7a:79:42:35:0c:60:ab:d6:d7:dc:e8:
92:75:69:5f:0d:e0:47:16:7e:dc:bb:28:fd:b9:29:
66:a0:36:82:6a:25:bf:cd:e8:ea:77:58:c4:c4:8f:
99:25:ef:2d:f5:9f:8e:89:ac:74:72:dd:10:ec:f8:
36:45:24:27:d5:5f:ee:0a:c5:93:ce:f4:5a:60:42:
2c:28:db:96:ae:a6:d1:17:2f:30:f1:fa:63:04:09:
71:29:d5:04:79:9a:30:d5:ff:cc:a6:b3:d9:a4:3a:
9c:c5:ff
publicExponent: 65537 (0x10001)


Activities
Terminal
Oct 30 21:06
seed@seed-virtual-machine: ~/src-cloud/Labsetup
privateExponent:
15:f2:f0:3f:1c:0c:fb:4e:54:9f:2f:5a:e5:10:ac:
09:b9:11:a4:5e:79:97:d2:ee:38:70:a2:28:2b:2a:
64:3a:2f:b5:59:75:65:30:6b:41:fe:78:86:38:6b:
58:03:bf:9d:59:bb:3b:ce:49:50:25:38:39:42:b5:
c8:a9:40:ea:a3:6d:84:cd:f5:4e:11:fd:78:5e:1e:
e8:d2:72:02:45:58:e2:f3:e2:d6:4a:93:57:51:71:
45:c6:fa:98:c6:e3:79:1d:4e:b9:c1:c6:37:18:8f:
72:04:4d:ec:ee:19:54:d7:06:87:d7:b0:e7:7f:36:
48:6a:61:f8:4d:f3:b1:69:6c:6a:7a:42:f5:7a:92:
be:48:49:05:31:91:73:93:87:96:a3:07:d3:6d:4b:
d5:f8:90:c5:72:e0:18:2e:47:37:0d:a9:2e:be:ef:
63:89:fc:95:42:eb:f2:4a:a6:fe:92:e9:3f:5a:7e:
85:79:3b:21:5d:e0:2b:ba:0f:92:80:13:28:95:77:
bd:f7:8a:a1:fa:ce:d6:1f:d0:d0:80:b1:34:77:99:
e5:fe:67:d8:68:f3:bd:4d:61:26:5b:0e:33:c5:b3:
11:ea:3f:17:b9:cb:ae:74:78:ee:1c:2d:c6:d2:2a:
6c:cf:7d:a8:8a:c2:4a:31:f3:b0:3f:71:74:8a:35:
8b:19:5b:d1:70:ce:13:76:7d:29:51:ab:6b:44:b4:
ac:eb:eb:bc:40:07:cf:f7:71:86:19:e5:43:03:ce:
c4:45:c6:bb:5c:de:c7:aa:2f:45:e8:26:2c:3b:44:
63:46:f6:fd:9b:43:08:41:9f:12:5d:f2:0c:7c:41:
0b:e7:6c:49:4d:7f:2e:ae:62:89:19:13:c5:7b:69:
ee:ac:d6:73:87:95:4d:53:76:7c:d7:3d:18:d4:d7:
a9:73:7d:f2:f1:e3:3a:00:50:01:a8:69:90:e9:37:
32:b4:0a:b5:8d:e2:47:06:05:a2:ba:df:39:66:44:
f5:93:73:4e:09:e6:b4:98:2a:a4:70:17:ef:e1:cf:
dd:7c:1d:01:05:33:1c:0c:9e:8d:ea:9a:54:c0:ea:
7b:a0:db:ab:55:ce:ac:a4:bc:4d:6c:f1:88:fc:81:
5c:d9:59:67:e5:3c:7c:27:07:c2:8f:2c:4c:51:99:
87:2d:57:98:66:18:c1:54:35:66:3c:12:27:38:7e:
10:af:f9:49:b7:cc:6e:d9:47:a7:81:6e:a5:6f:01:
e4:33:c2:49:28:5c:4b:6d:9c:7c:c7:4a:c5:db:dd:
d3:34:98:f1:fe:60:4e:fe:3e:9b:bf:52:2d:99:e9:
ca:fa:b8:38:bc:de:fb:b7:37:0b:db:51:b8:39:52:
1f:21
prime1:
00:ce:5d:88:8c:17:f9:d7:60:da:b9:2a:0c:be:4b:
7b:38:67:64:fd:df:41:0e:13:92:f3:00:f5:5e:0c:
f5:42:7e:b3:3c:1e:dc:fa:ec:6f:39:98:2a:17:d2:


Activities
Terminal
Oct 30 21:06
seed@seed-virtual-machine: ~/src-cloud/Labsetup
prime1:
00:ce:5d:88:8c:17:f9:d7:60:da:b9:2a:0c:be:4b:
7b:38:67:64:fd:df:41:0e:13:92:f3:00:f5:5e:0c:
f5:42:7e:b3:3c:1e:dc:fa:ec:6f:39:98:2a:17:d2:
60:96:32:b3:86:b3:f7:74:33:84:f8:8a:45:b0:29:
a8:75:f3:7e:ea:4f:f9:93:d4:07:15:ac:3f:8c:94:
33:02:6b:92:ff:e8:08:b7:8c:33:51:25:b8:cd:77:
37:16:8f:e1:59:54:4d:8e:cb:65:0c:d1:73:49:07:
f4:34:66:bf:9e:a9:c6:ed:53:44:9c:66:14:01:f8:
9a:b9:24:6b:d8:8d:da:c3:c8:2a:2a:70:1f:9e:1a:
ef:ac:47:33:bd:f1:20:c9:a1:d5:69:1b:13:10:8d:
7a:c7:da:4f:f4:88:f0:10:e1:05:24:f4:62:06:a1:
2b:91:cc:63:ce:9f:a7:bd:f4:9c:39:a3:18:f2:ed:
2f:f1:7f:ae:23:07:e4:23:6b:71:5f:36:02:c7:b2:
87:bc:73:17:7a:a0:59:1a:b3:87:2f:88:4f:d5:b9:
67:9d:30:af:9b:67:ba:ec:90:14:a5:44:c4:e9:ec:
2f:2c:43:35:54:ee:21:7e:ec:86:49:9c:87:9b:8e:
7a:32:eb:0c:e9:a5:19:be:d4:fb:13:c7:87:43:a4:
43:d3
prime2:
00:c3:93:d2:cf:a8:a3:64:db:aa:58:14:04:1b:20:
b1:14:4f:3d:9c:86:18:92:cd:47:6e:b7:a9:7b:08:
34:77:0d:e5:b4:bb:f7:7b:74:77:f9:d9:66:70:6a:
65:59:8c:2c:0c:19:90:4b:a6:64:cd:75:00:ae:3b:
a5:02:89:ac:4b:96:ca:3c:38:19:1b:36:23:e3:6a:
b3:77:45:02:ad:a7:8b:03:25:c8:53:62:70:e5:ef:
2f:5d:df:cd:89:b5:6e:47:8b:51:fb:a1:84:2a:e4:
b2:fd:9f:a2:86:9c:36:77:13:4a:f8:45:5f:fc:c4:
1f:16:9a:d3:b5:47:15:df:20:10:27:9f:43:51:c3:
27:b2:52:be:63:2e:fc:4e:53:88:50:c7:8f:d6:59:
14:72:9a:67:87:ec:0a:a3:7b:8c:d8:bf:f5:f1:7a:
0f:84:92:11:78:e5:75:2a:11:f0:46:d0:7b:af:2f:
01:dd:91:54:ef:45:1d:d1:e0:98:9e:c0:62:15:dd:
9b:39:12:5c:5f:ca:a6:7a:00:a5:63:f6:2e:fd:3e:
ef:8c:d8:08:a0:1e:f3:e0:6c:98:de:c0:d6:fa:52:
2e:d0:8a:d7:f3:35:f8:fb:dd:c9:03:52:dc:9b:ea:
da:9a:7d:70:23:5d:d0:f4:75:d8:3a:1e:ce:e0:e8:
55:a5
exponent1:
13:d8:dd:0c:14:36:3c:27:9d:81:97:70:b8:50:8c:


Activities
Terminal
Oct 30 21:06
seed@seed-virtual-machine: ~/src-cloud/Labsetup
6f
exponent2:
5f:c0:81:d7:b5:42:b9:e0:6a:17:d1:44:e9:5b:2a:
17:3d:6c:76:b3:98:58:57:34:7f:da:5f:99:a2:bc:
bd:05:0a:cf:5f:af:9c:bb:ba:f9:92:1c:7b:31:3c:
09:89:7d:0f:37:7c:d3:ec:d5:31:59:f5:49:58:5e:
43:4d:d0:12:f9:f2:6d:65:31:b5:86:de:2f:3b:50:
3e:36:85:47:2b:dd:fd:5e:84:23:7e:f0:b3:42:55:
38:6a:98:20:b1:b9:1a:41:f5:93:72:d2:58:85:bb:
b4:3c:71:bf:e1:bf:30:b1:3b:65:2d:e1:fe:c8:ae:
47:cc:e9:95:b9:05:d6:45:c6:62:ff:b0:5f:d9:62:
18:06:36:bc:2c:86:ce:7a:4b:24:f1:6f:54:8c:2c:
5c:a7:87:c4:9e:25:df:40:60:06:c6:84:cd:02:eb:
92:06:49:93:9d:13:ad:d3:36:86:15:10:59:06:74:
ad:5b:b7:ef:a5:31:22:4e:09:aa:bf:99:c6:06:c7:
36:3b:ed:09:e1:17:7c:67:d0:91:8e:38:17:38:b6:
e3:87:e9:41:2b:a6:4e:dc:1c:44:02:b9:bd:b8:7a:
bf:13:1b:03:32:0b:82:94:65:13:c1:7f:cc:be:40:
e4:9e:41:46:af:69:dc:25:ce:05:36:8c:26:c4:d9:
19
coefficient:
18:47:0e:dd:f6:f0:cc:87:e9:79:f5:ce:64:dd:c3:
6f:67:58:b8:4e:b4:1d:a9:61:1f:87:a0:c3:23:8b:
a2:a4:c9:02:31:f3:bd:cc:49:b7:7a:3d:21:90:e6:
e3:8d:da:20:67:87:33:fd:f5:6f:1f:5d:1b:5d:d6:
a9:14:43:c6:0f:e1:d1:5b:e7:74:e4:e9:b1:84:e7:
d7:4f:a1:4c:c6:48:fd:ba:9e:0c:3e:56:7f:f3:71:
31:ba:27:ee:a7:21:58:38:dd:43:bc:95:56:cc:a2:
06:0b:f0:e0:5a:01:41:b2:21:13:20:56:76:c8:3b:
b3:54:f7:1b:75:aa:5d:00:b3:1a:c6:10:e9:34:69:
81:fd:14:48:f6:c0:4b:31:0b:3d:5c:1c:5a:92:a7:
f5:82:31:0f:7f:ff:99:8d:c2:f6:13:7b:ec:d9:c8:
26:cc:72:14:8a:d9:30:39:cc:0f:66:af:ca:54:aa:
ab:37:7e:7d:64:79:46:de:5d:b6:9f:54:de:99:8c:
a3:46:c2:3f:5a:d0:68:cd:7c:19:ff:7a:c8:de:81:
93:9a:f3:14:86:51:ec:50:12:46:d9:8a:80:b0:29:
6a:1a:b3:53:28:79:90:b3:fb:63:91:cf:9f:e0:12:
99:aa:eb:e7:59:33:a3:8b:4f:97:d2:52:25:78:33:
ac
seed@seed-virtual-machine:~/src-cloud/Labsetup$

Task 2: Generating a Certificate Request for Your Web Server

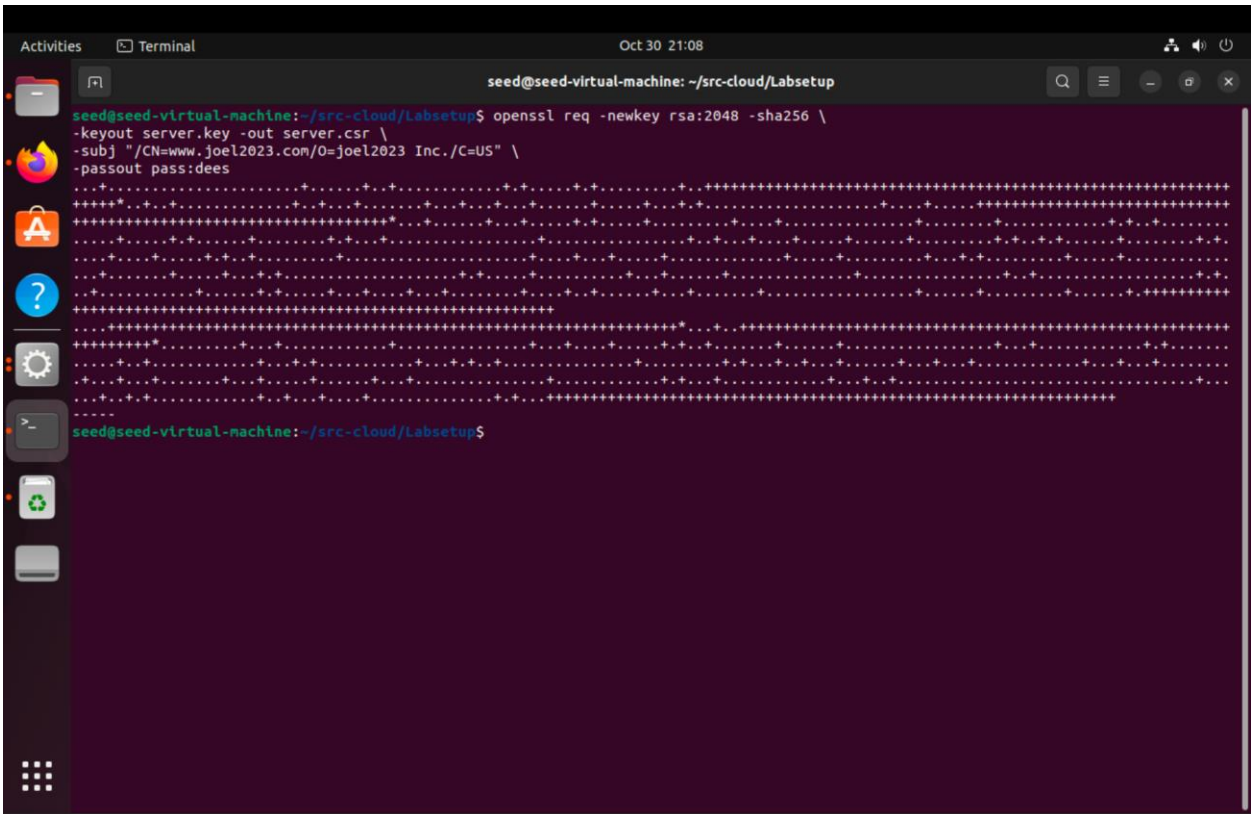
First it needs to generate a Certificate Signing Request (CSR), which basically includes the company's public key and identity information. The CSR will be sent to the CA, who will verify the identity information in the request, and then generate a certificate.

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**

The command to generate a CSR for my own server name **joel2023** is

openssl req -newkey rsa:2048 -sha256 \ -keyout server.key -out server.csr \ -subj "/CN=www.joel2023.com/O=joel2023 Inc./C=US" \ -passout pass:dees

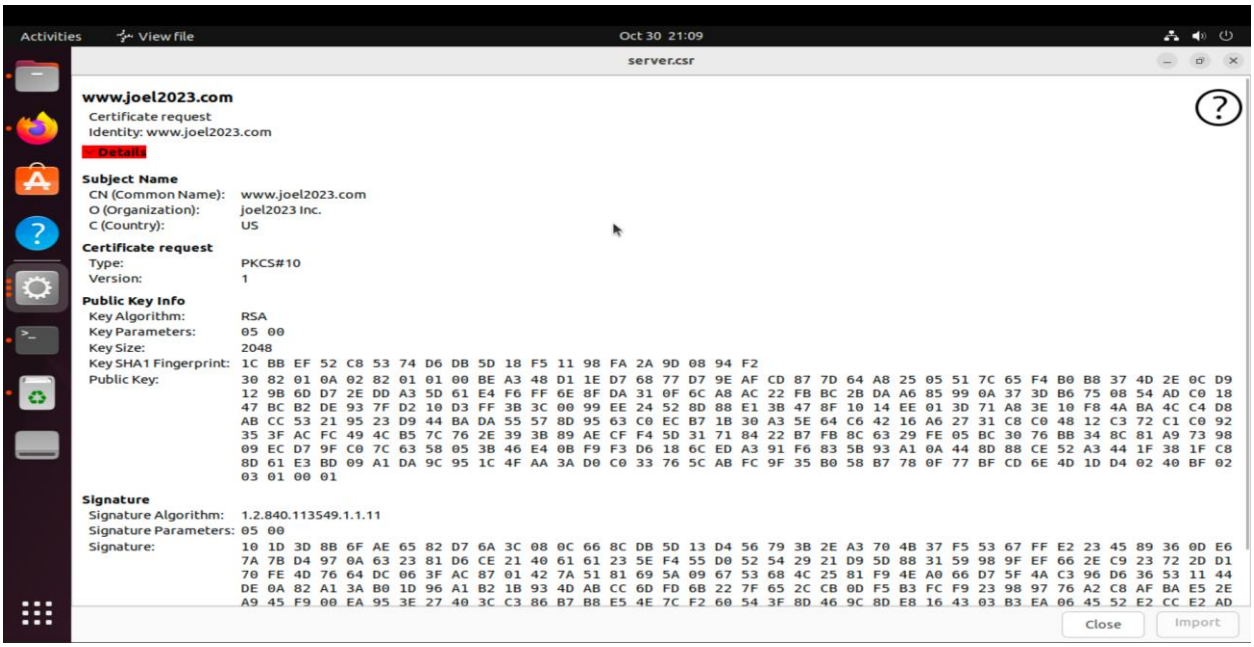
Output as follows:



Files after running the command:
The command will generate a pair of **public/private key**, and then create a certificate signing request from the public key.
ie the **server.csr** and the **server.key**

```
seed@seed-virtual-machine:~/src-cloud/Labsetup$ ls
ca.crt  ca.key  demoCA  docker-compose.yml  image_www  openssl.cnf  serial  server.csr  server.key  volumes
seed@seed-virtual-machine:~/src-cloud/Labsetup$
```

The content of the **server.csr(public key)**



Note: This method is without adding the alternative names so I removed the **server.csr and server.key** generated and added the alternative name through the -addtext option below.

Adding Alternative names:

To add alternative names we have to add another option to the the command done before that is
-addext "subjectAltName = DNS:www.joel2023.com, \ DNS:www.joel2023A.com, \ DNS:www.joel2023B.com"

So the command becomes:

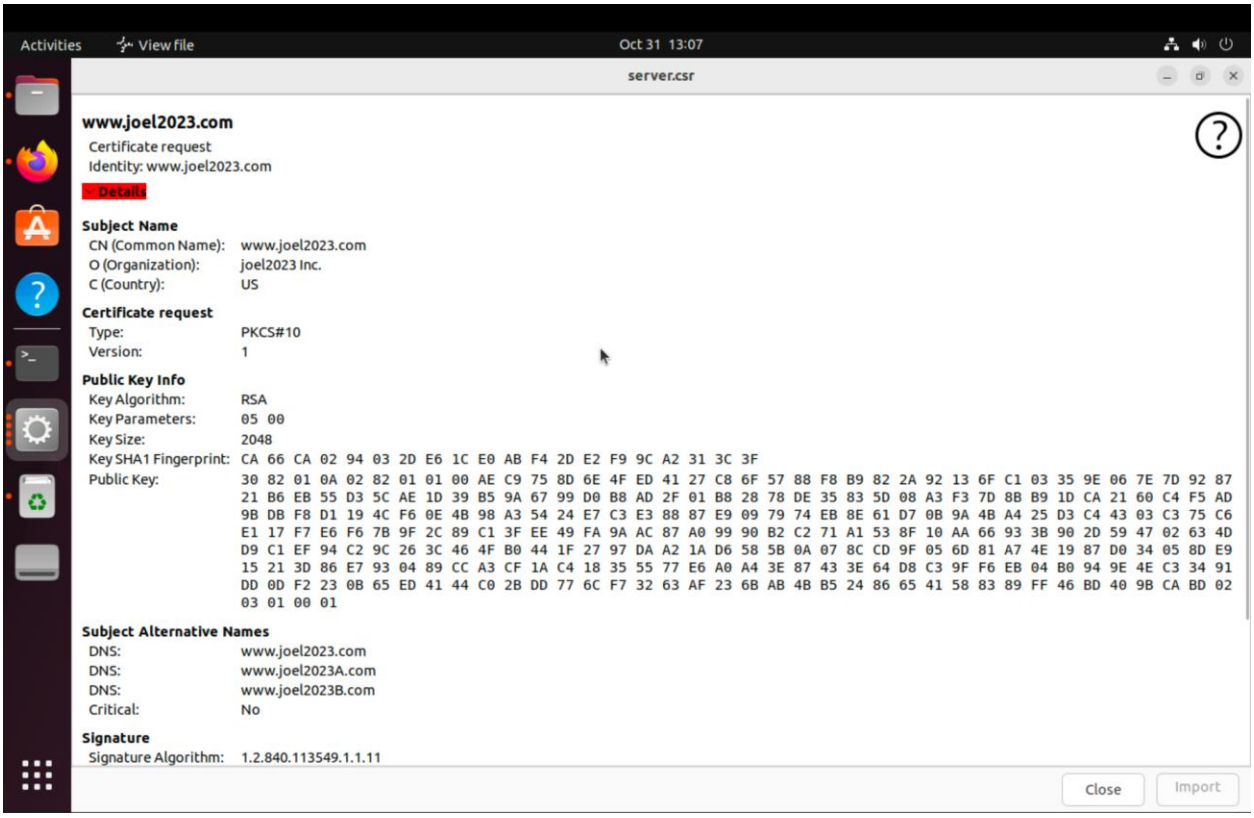
**openssl req -newkey rsa:2048 -sha256 \ -keyout server.key -out server.csr \ -subj "/CN=www.joel2023..com/O=joel2023 Inc./C=US" **
**-addext "subjectAltName = DNS:www.joel2023.com, \ DNS:www.joel2023A.com, \ DNS:www.joel2023B.com" **
-passout pass:dees

Here we added 2 alternative names to the certificate

```
seed@seed-virtual-machine:~/src-cloud/Labsetup$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=w
ww.joel2023.com/O=joel2023 Inc./C=US" -addext "subjectAltName = DNS:www.joel2023.com, \
DNS:www.joel2023A.com, \
DNS:www.joel2023B.com" -passout pass:dees
```

A look at the public key ie the **server.csr** generated:

We can see the alternative names of the website




```
seed@seed-virtual-machine:~/src-cloud/Labsetup$ ls
backup  ca.crt  ca.key  demoCA  docker-compose.yml  image_www  openssl.cnf  serial  server.csr  server.key  volumes
seed@seed-virtual-machine:~/src-cloud/Labsetup$
```

3) Task 3: Generating a Certificate for your server

The **CSR file** needs to have the CA's signature to form a certificate.

Here in the lab we will be using our own trusted CA to generate the certificate for the server. In real world scenarios the CSR files are usually sent to a trusted CA for their signature.

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:

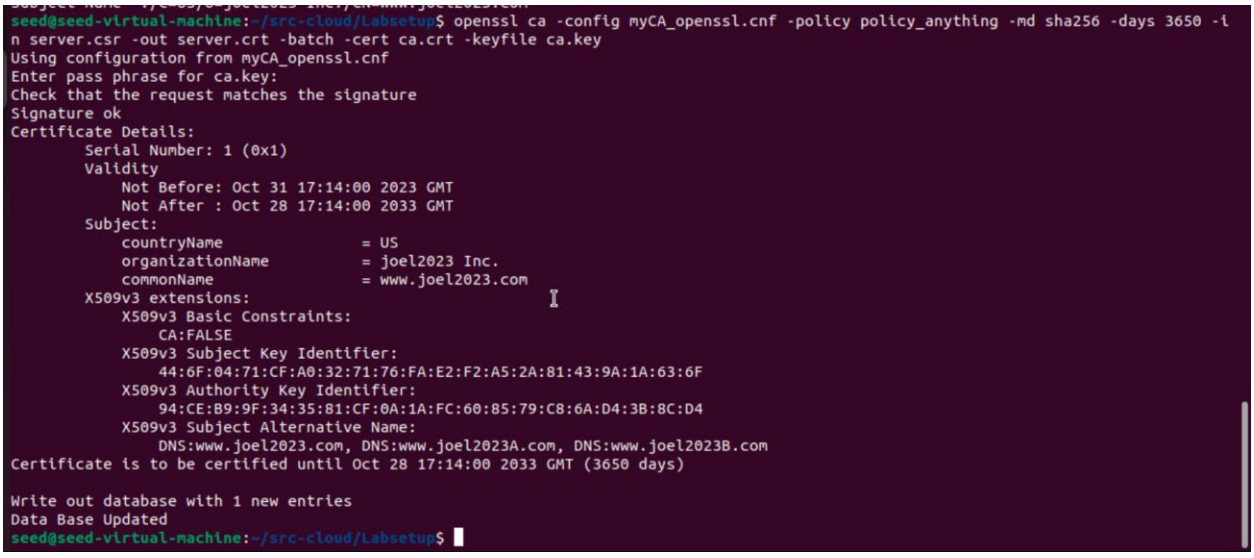
openssl ca -config myCA_openssl.cnf -policy policy_anything \ -md sha256 -days 3650 \ -in server.csr -out server.crt -batch \ -cert ca.crt -keyfile ca.key

We use the **policy_anything** policy defined in the configuration file.

If we do this we won't be able to see the alternative names to enable this we have to go to the config file or copy the same config file and change some values in it ie uncomment the line:

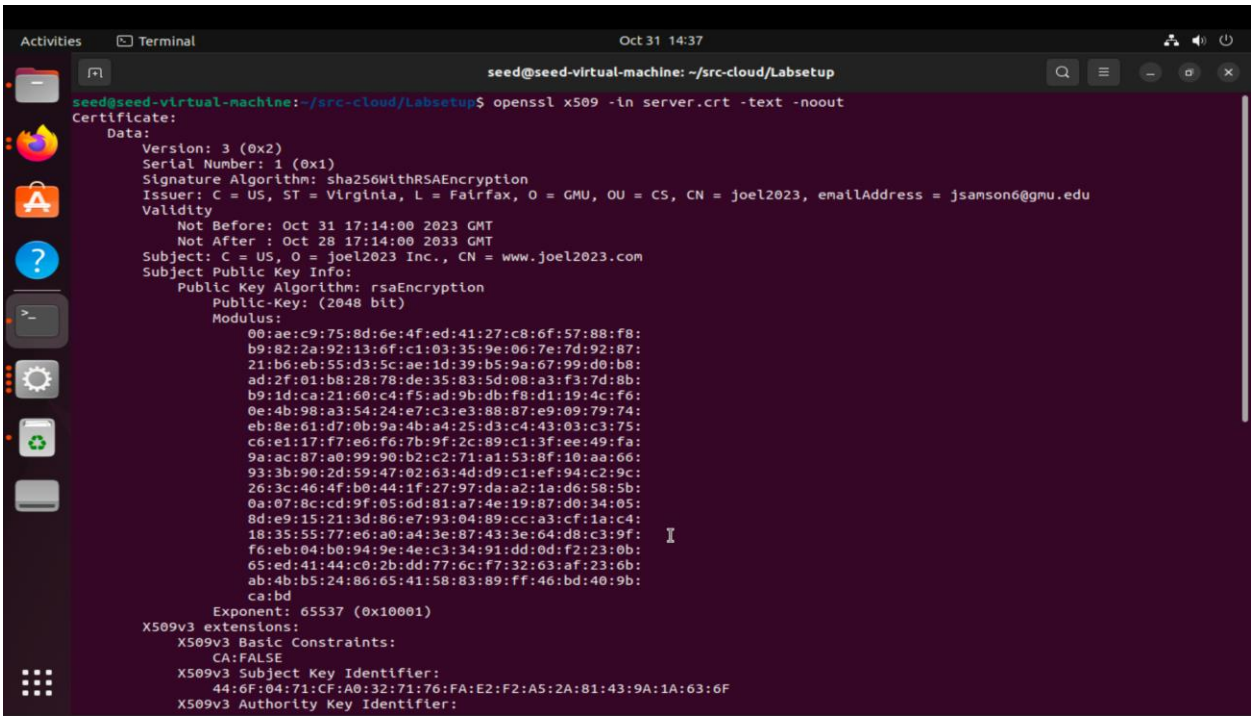
 # Extension copying option: use with caution.
copy_extensions = copy

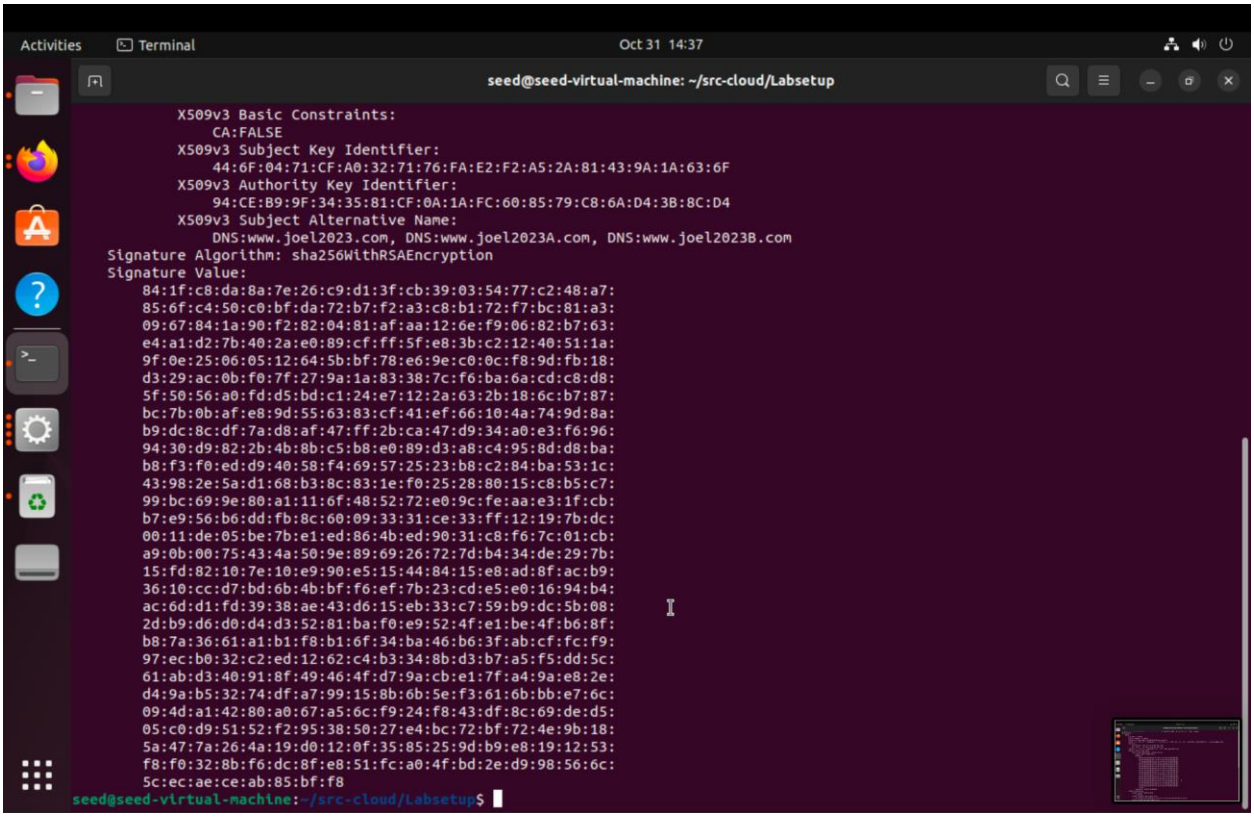
Command execution as follows:



After signing the certificate, please use the following command to print out the decoded content of the certificate, and check whether the alternative names are included.

openssl x509 -in server.crt -text -noout





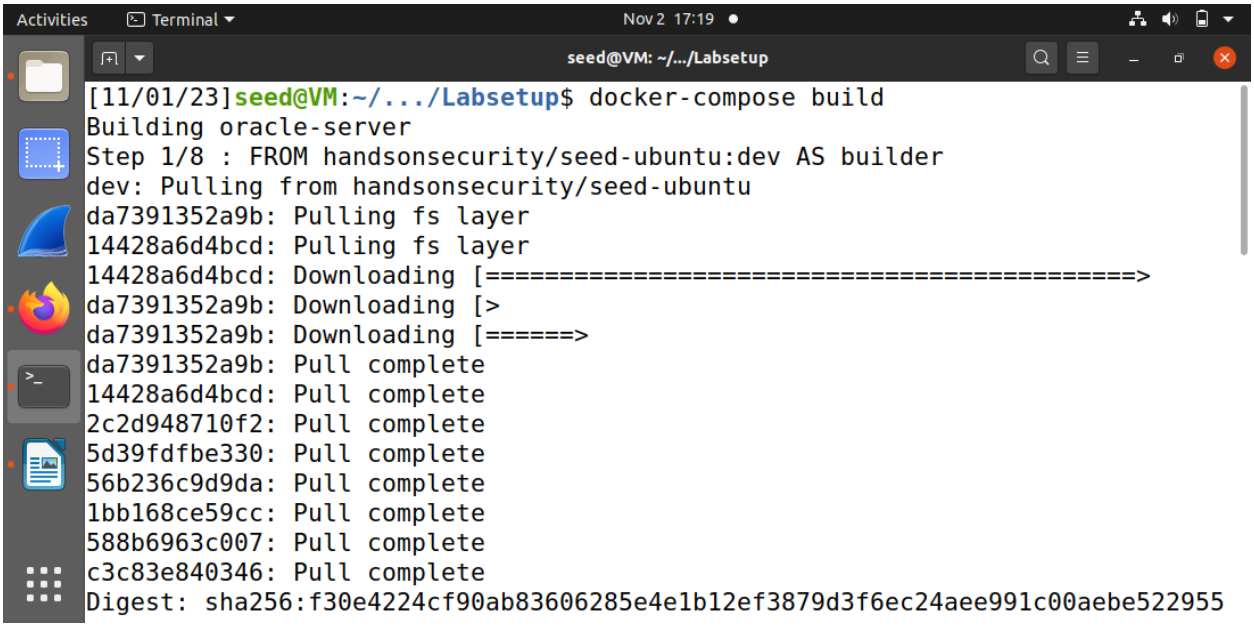
Here we can see that it's successful and the alternative names are also present so the generation of the certificate for the server is complete.

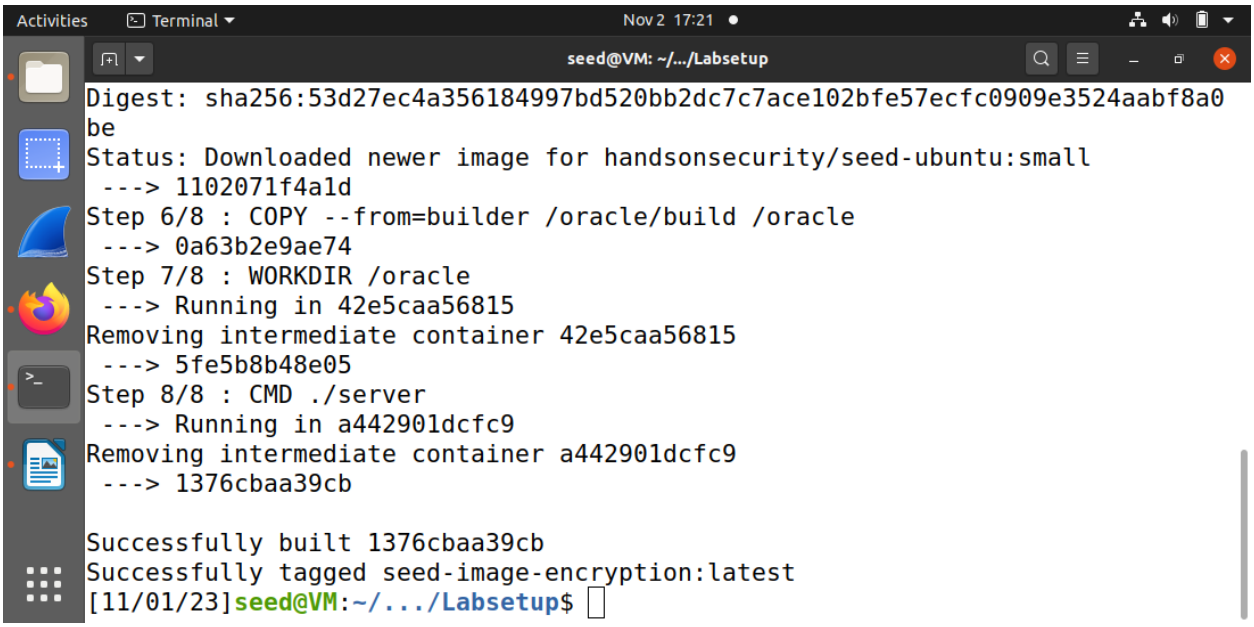
Container Setup:

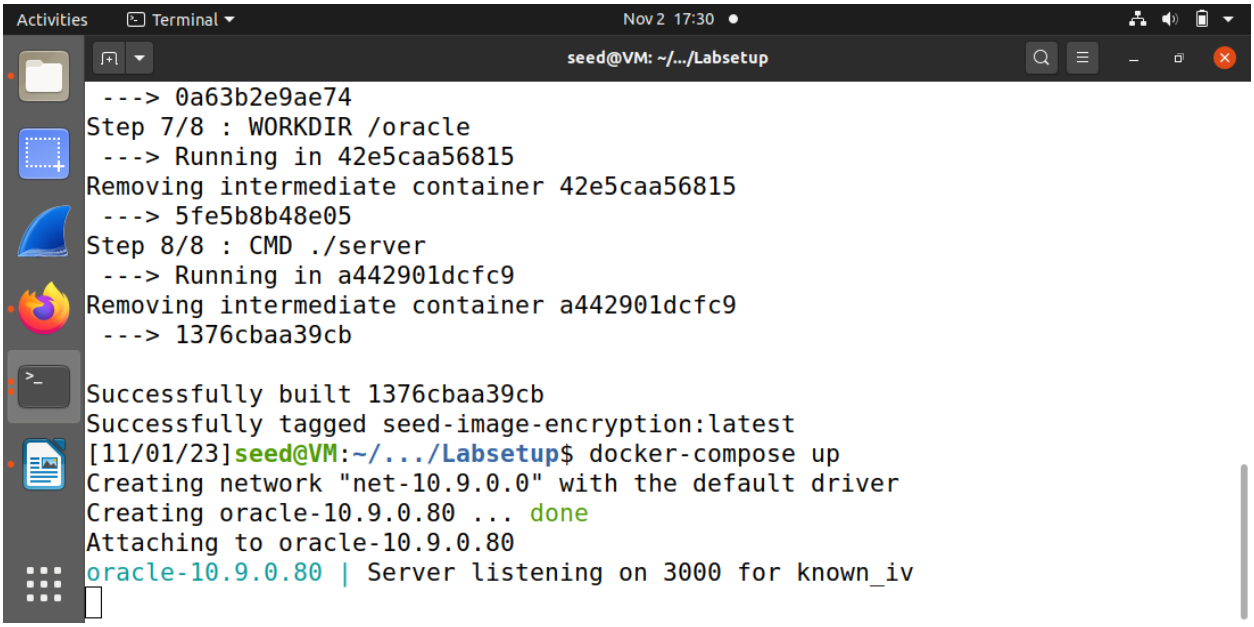
The following commands used to get the docker container running, these commands were executed in the same folder where the docker-compose.yml file is located.

$ docker-compose build # Command to build the container image

$ docker-compose up # Command to start the container



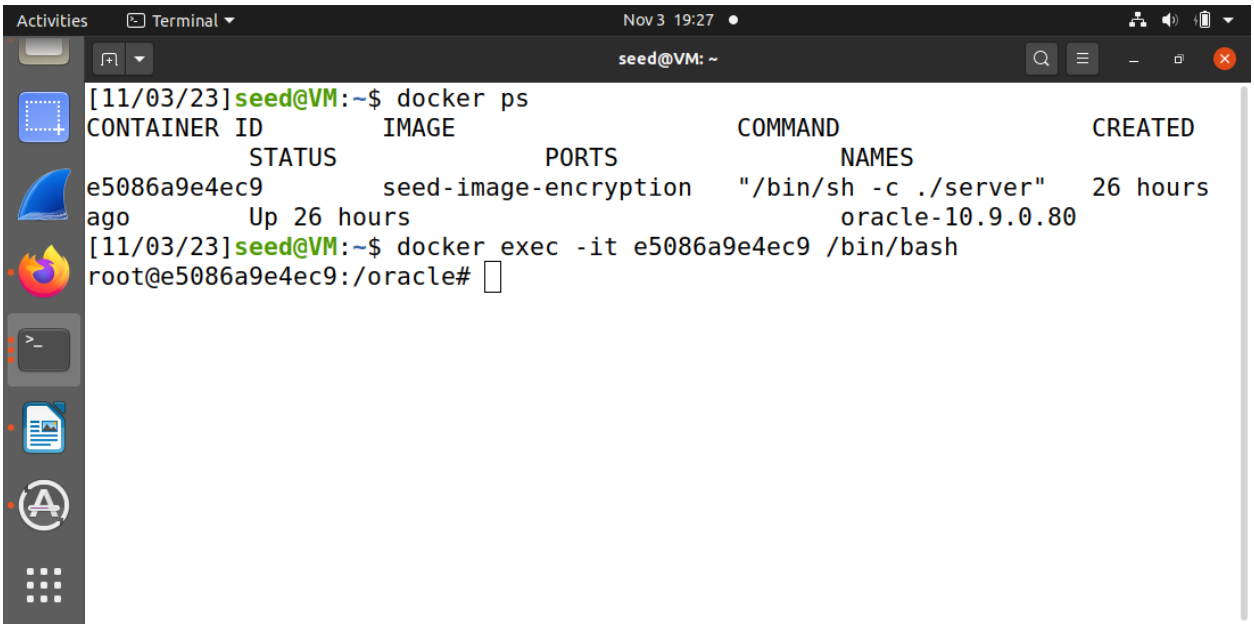




To run shell on the container we use the following commands:

$ docker ps

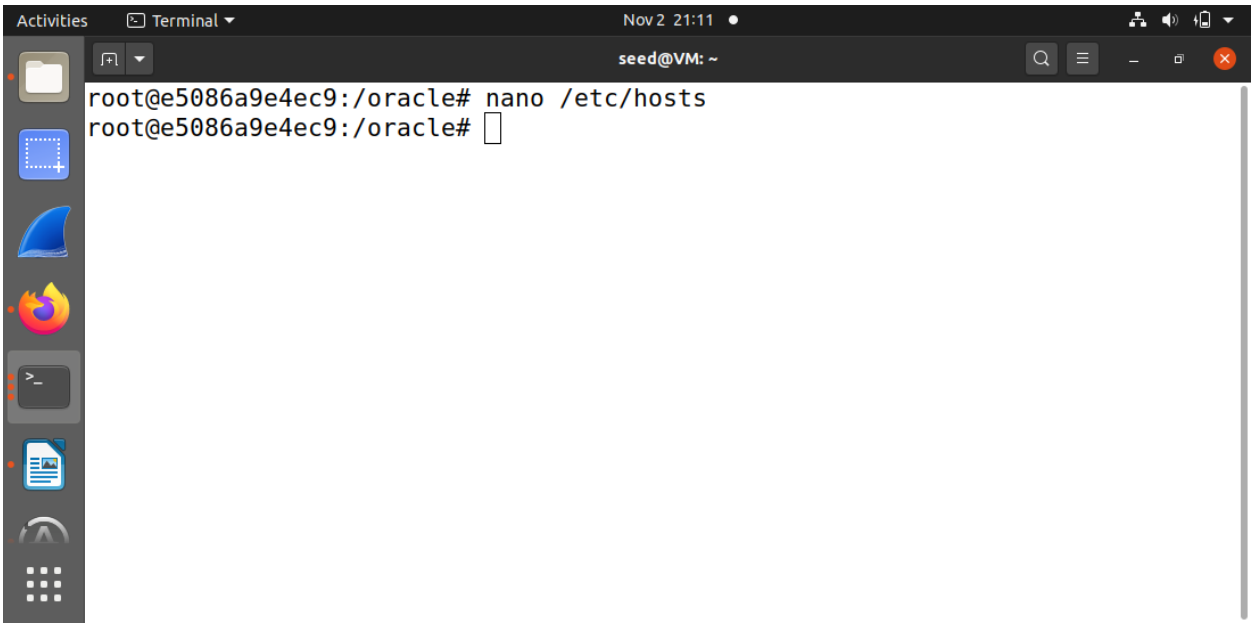
$ docker exec -it e5086a9e4ec9 /bin/bash



DNS setup:

The following command was used to access and add DNS entries to the /etc/hosts file:

$ nano /etc/hosts

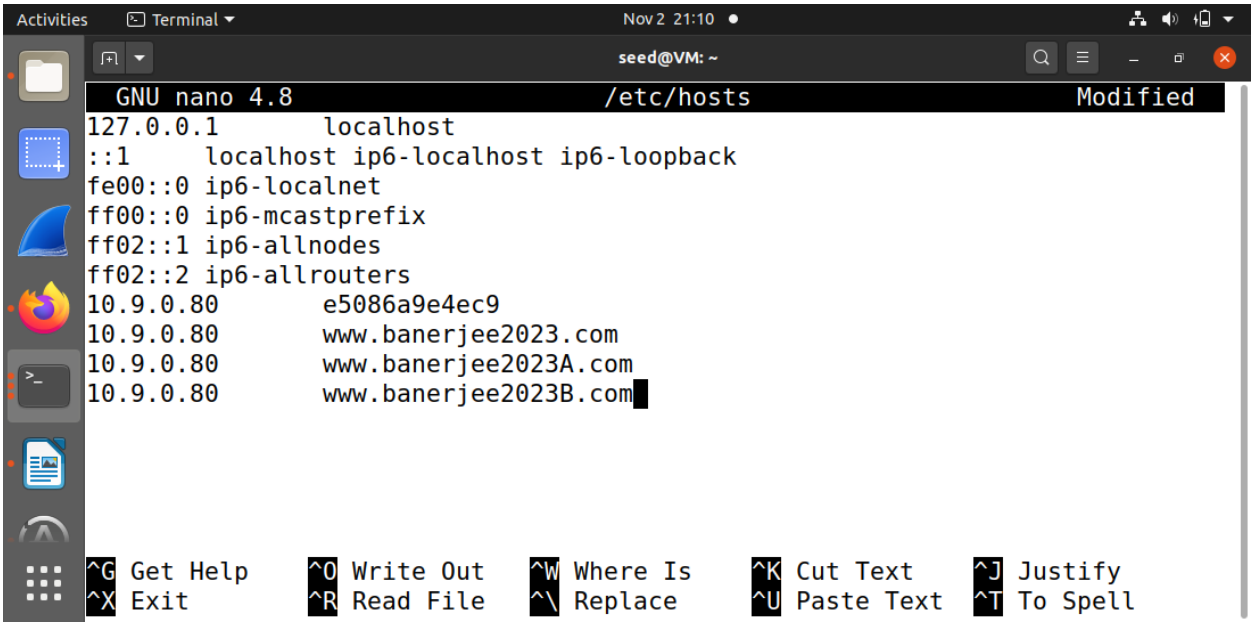


The following entries were added:

10.9.0.80 www.banerjee2023.com

10.9.0.80 www.banerjee2023A.com

10.9.0.80 www.banerjee2023B.com



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

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. To create an HTTPS website, we just need to configure the Apache server, so it knows where to get the private key and certificates.

An Apache server can simultaneously host multiple websites. It needs to know the directory where a website's files are stored. This is done via its VirtualHost file, located in the /etc/apache2/ sites-available directory. In the sites-available folder, a file named 000-default.conf which contains the information about VirtualHost.

The following command was used to access the .conf file:

$ nano /etc/apache2/sites-available/000-default.conf

At the end of the file the following VirtualHost entry was added.

*<VirtualHost *: 443>*

DocumentRoot /var/www/banerjee2023

ServerName www.banerjee2023.com

ServerAlias www.banerjee2023A.com

ServerAlias www.banerjee2023B.com

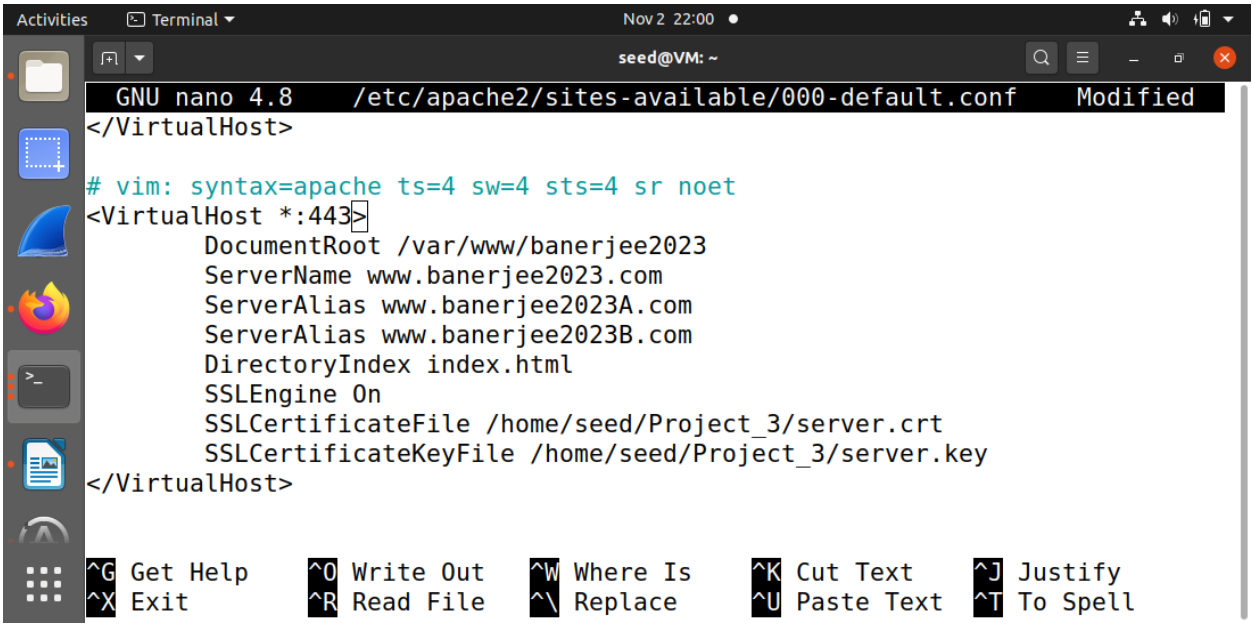
DirectoryIndex index.html

SSLEngine On

SSLCertificateFile /home/seed/Project_3/server.crt

SSLCertificateKeyFile /home/seed/Project_3/server.key

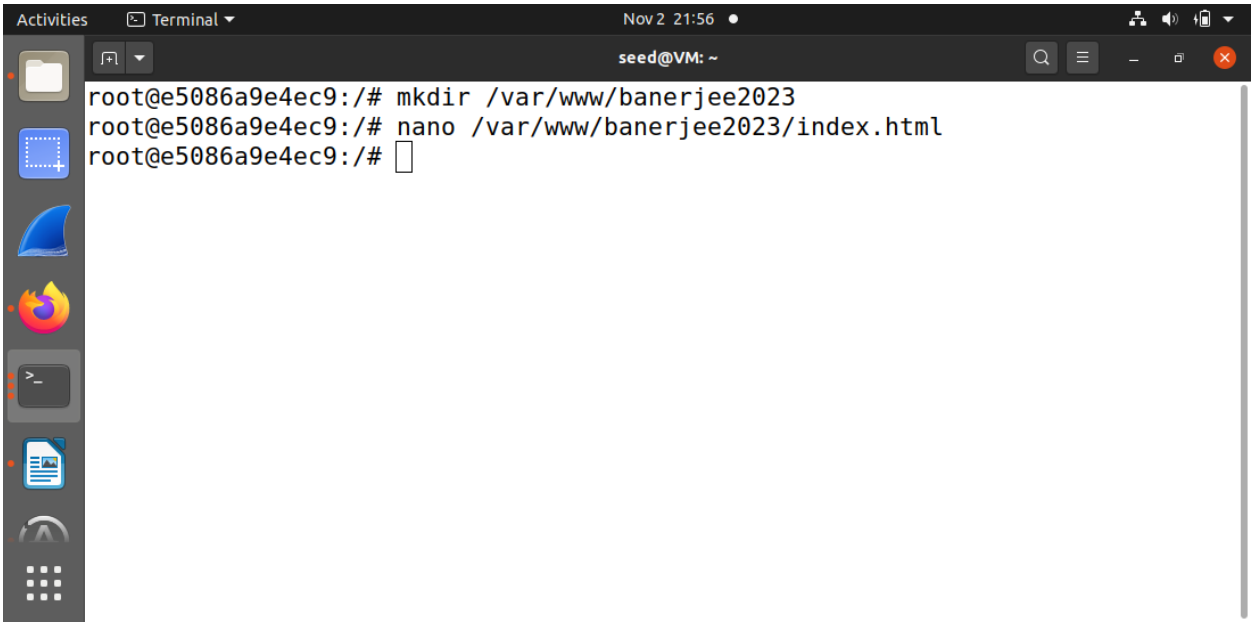
</VirtualHost>



According to the VirtualHost file the index.html file will be shown which is stored in the directory /var/www/banerjee2023. Now we need to create such a file in the mentioned directory. The commands are given below:

$ mkdir /var/www/banerjee2023 # to create the directory.

$ nano /var/www/banerjee2023/index.html # to create a html file and edit it.

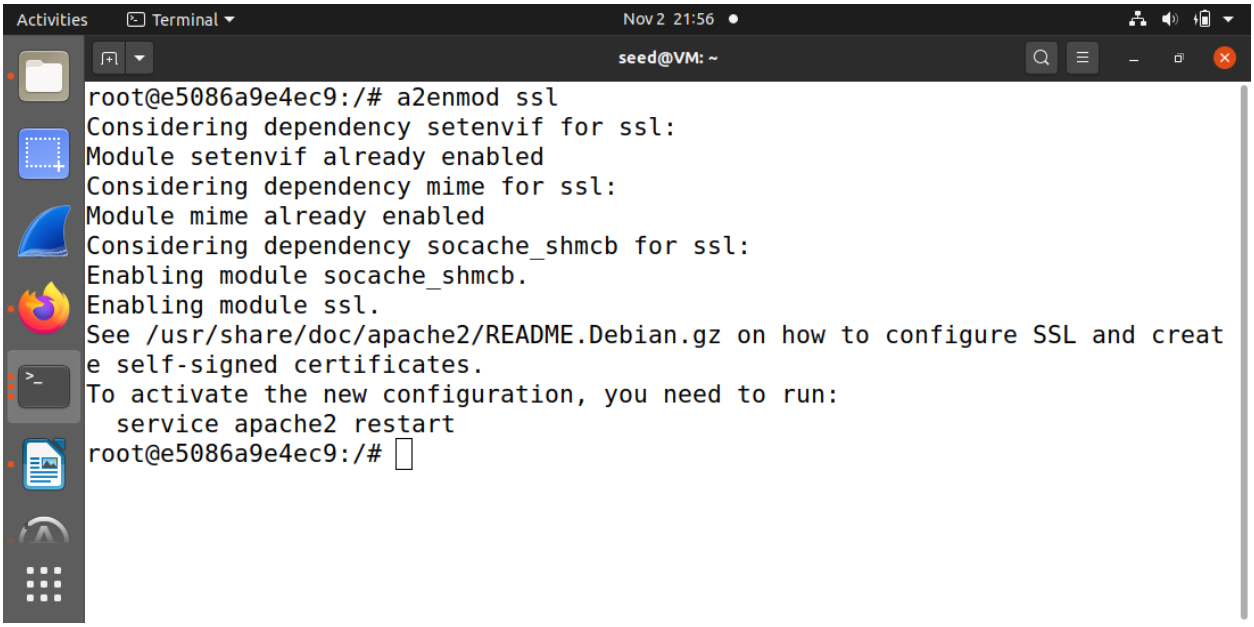


The following code was placed inside the html file, and this will be the content of our page:

```
<!DOCTYPE html>
<html>
        <body>
                <h1>Hello World!!!</h1>
                <p>This is a part of Mini Project 3 in ISA 562</p>
        </body>
</html>
```

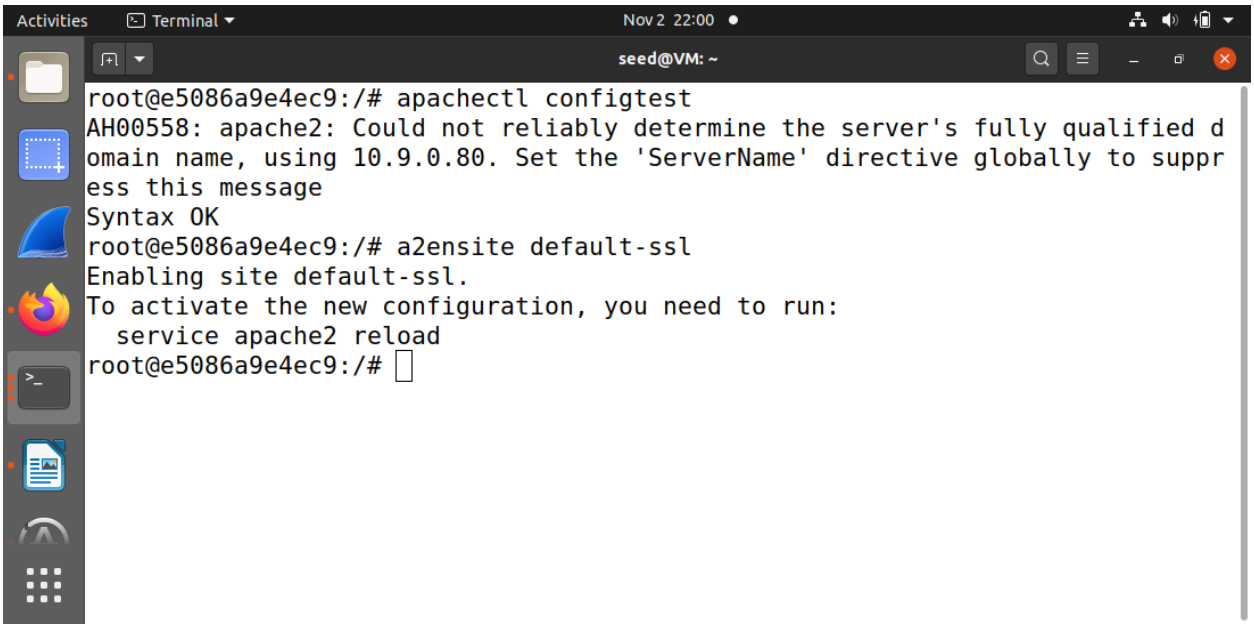
So, to make this website work, we need to. Enable Apache's ssl module and then enable this site. The commands used are as follows:

```
$ a2enmod ssl                    # Enable the SSL module
```



$ apachectl configtest # Test the Apache configuration for errors

$ a2ensite default-ssl # Enable the site

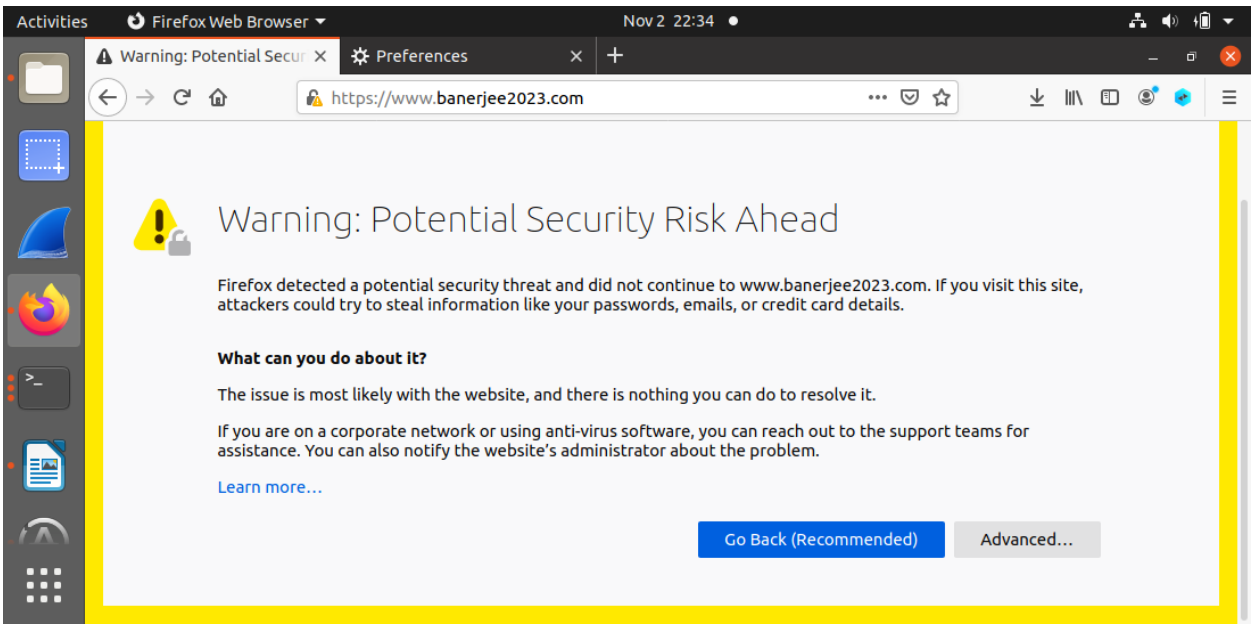


Starting the Apache server: The Apache server is not automatically started in the container, because of the need to type the password to unlock the private key.

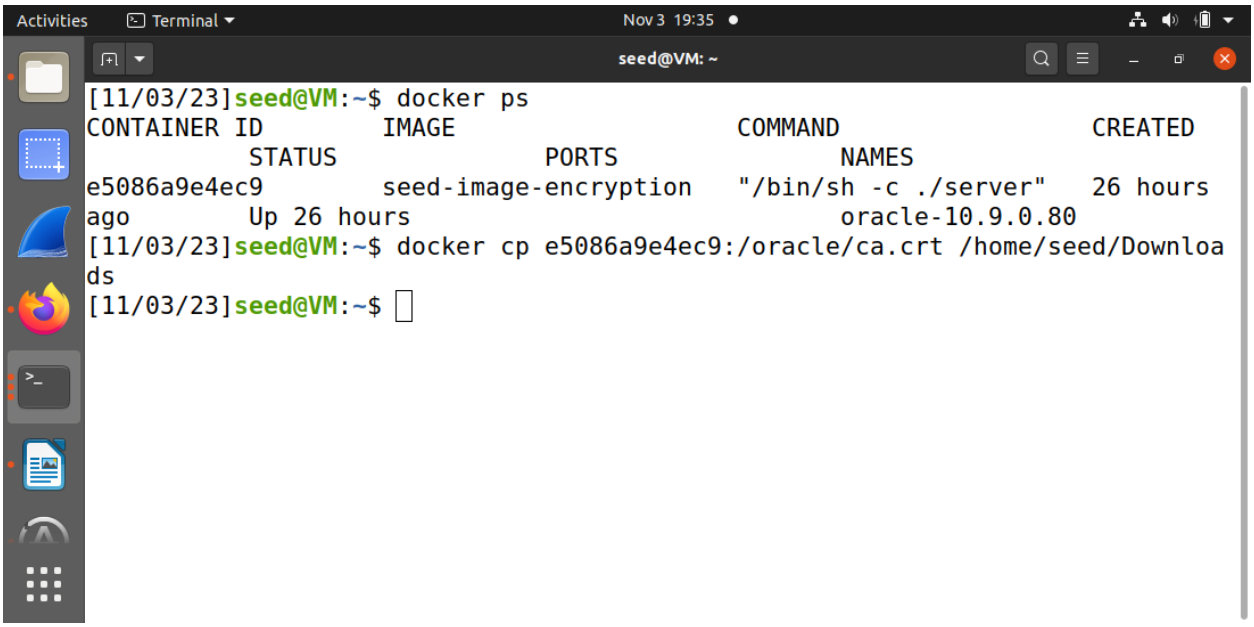
$ service apache2 start # Start the server

When Apache starts, it needs to load the private key for each HTTPS site. Our private key is encrypted, so Apache will ask us to type the password for decryption. Inside the container, the password used for banerjee2023 is dees. Once everything is set up properly, we can browse the web site.

Browsing the website: When we point the browser to your web server, (https://www.banerjee2023.com) we are not able to access the contents of the site. A "Warning" appears saying "Potential Security Risk Ahead". The reason being, the site is using a self-signed certificate, which is not automatically trusted by browser because it isn't signed by a recognized certificate authority (CA).



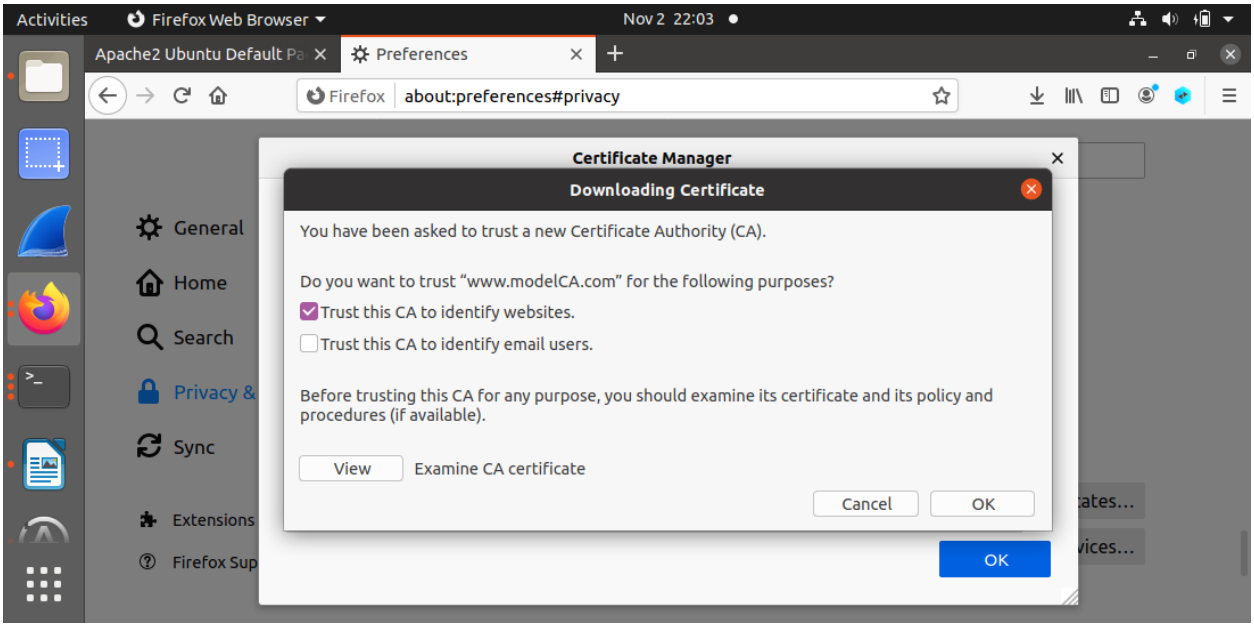
Now to make the warning go away we need to load the certificate to the browser and make the browser trust the new certificate for this website. For this we first need to get the certificate from inside the container into the host directory. We use the following command to do so:
Command executed in Host CLI.
$ docker cp e5086a9e4ec9:/oracle/ca.crt /home/seed/Downloads

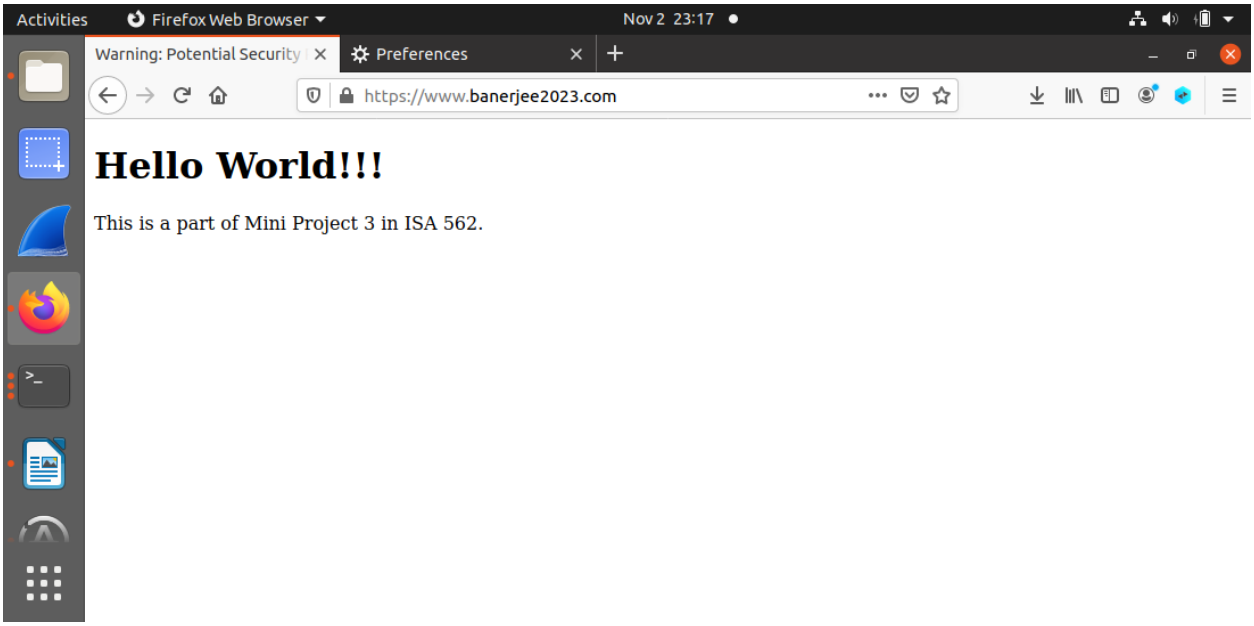


This command copies the ca.crt file from this folder “oracle” which is inside the container to “Downloads” folder which is in Host.

Now that we have the certificate in Host we do this because the browser can only access the files in Host and not inside the container.

To load the certificate, we enter the following command in the search bar, “about:preference#privacy”, this takes us to the Privacy and Security section of the settings menu, which contains the “Certificates” section, in this section we click on “View Certificates…” Which then gives is the option to scroll through all the certificates present and gives us an option to import new certificate. So, we click “Import” then navigate to the directory where our certificate, in this case “ca.crt”, is located and select it and click on the green import button. This will open a “Downloading Certificate” tab where we need to select “Trust this CA to identify websites.” And click OK. We will see that our certificate is now in Firefox’s list of accepted certificates. Then we can reload the website and see that the content of https://www.banerjee2023.com can be accessed. And we can also see that the connection is secure and verified by Model CA LTD.





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

In this task, we will show how PKI can defeat Man-In-The-Middle (MITM) attacks. The goal of this task is to help students understand how PKI can defeat such MITM attacks. In the task, we will emulate an MITM attack, and see how exactly PKI can defeat it. We will select a target website first. In this task, we use www.instagram.com as the target website.

Step 1 - Setting up the malicious website:

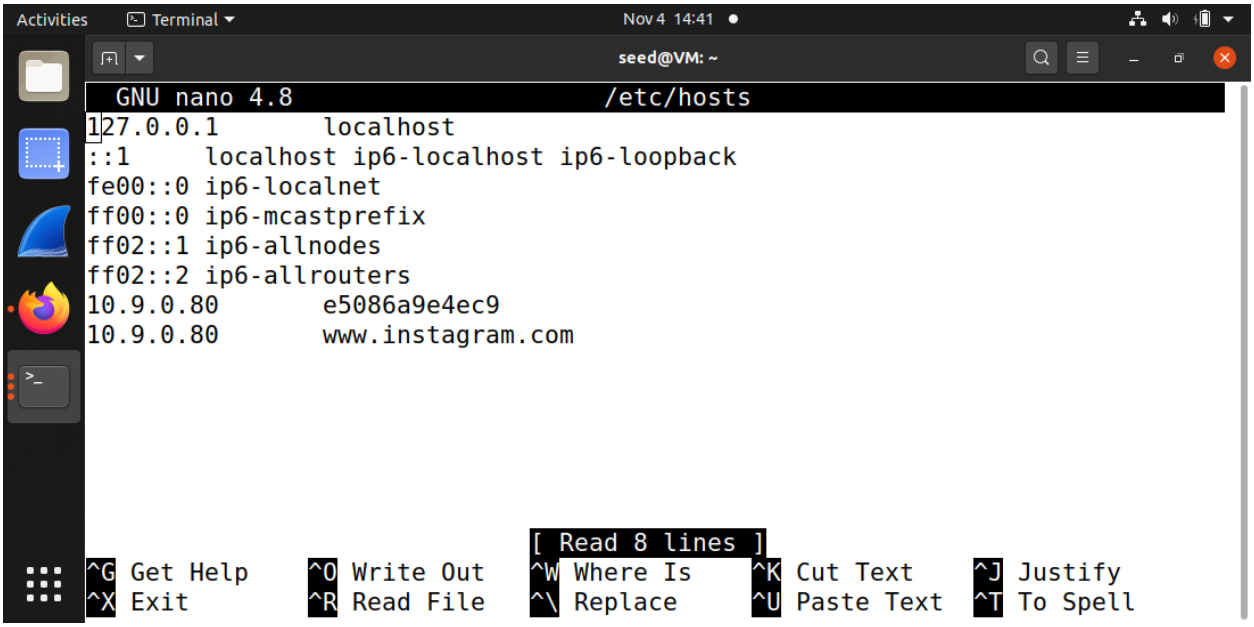
In Task 4, we have already set up an HTTPS website. We will use the same Apache server to impersonate www.instagram.com. To achieve that, we have followed the instruction in Task 4 to add a VirtualHost entry to Apache's SSL configuration file: the ServerName should be www.instagram.com, but the rest of the configuration will be the same as that used in Task 4.

In this step, we configured the DNS on the attacker machine, which in our case is the docker container using the following command:

$ nano /etc/hosts

And add our target website name as:

10.9.0.80 www.instagram.com



Then we add the website as a VirtualHost entry in 000-default.conf using the following command:

$ nano /etc/apache2/sites-available/000-default.conf

The following VirtualHost entry was add in the end on the file:

*<VirtualHost *:443>*

DocumentRoot /var/www/Instagram

ServerName www.instagram.com

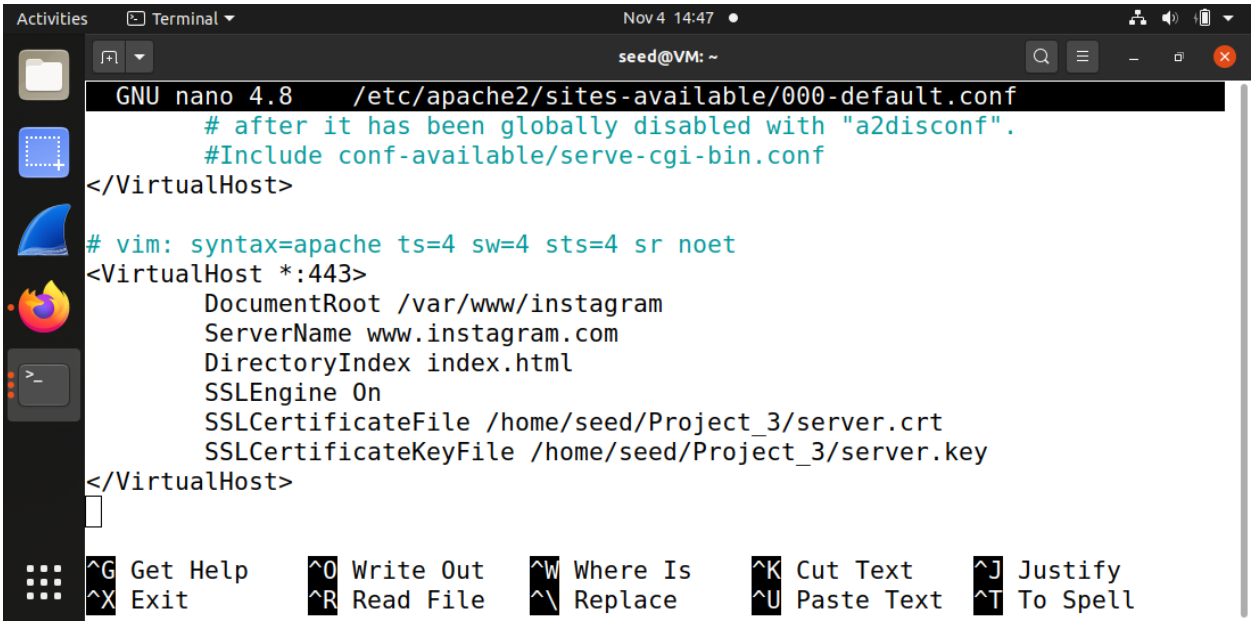
DirectoryIndex index.html

SSLEngine On

SSLCertificateFile /home/seed/Project_3/server.crt

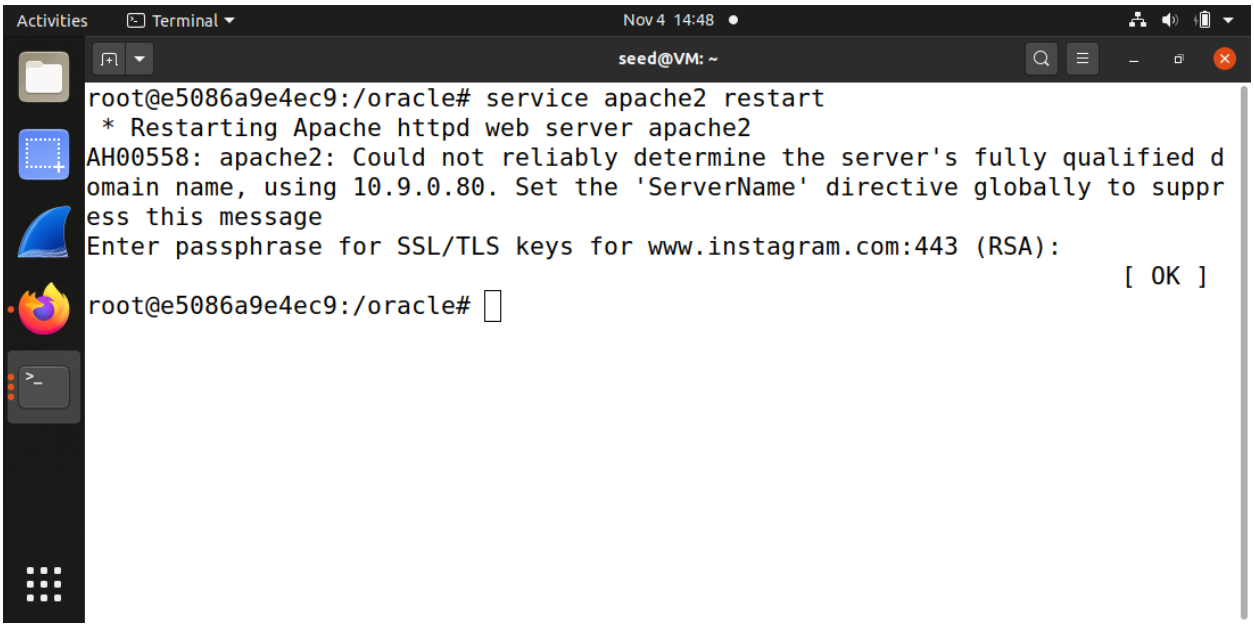
SSLCertificateKeyFile /home/seed/Project_3/server.key

</VritualHost>



Then we use restart the Apache server to save the changes:

$ service apache2 restart

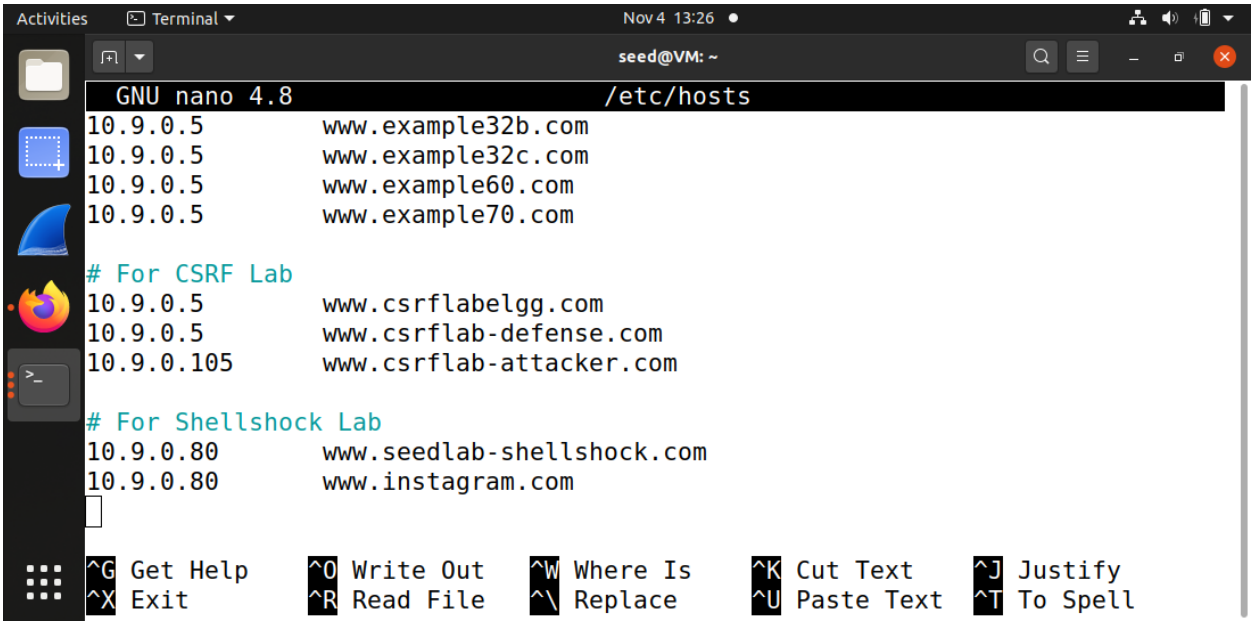


Step 2 - Becoming the man in the middle:

In this task, we simulate the attack-DNS approach. Instead of launching an actual DNS cache poisoning attack, we simply modify the victim's machine's /etc/hosts file to emulate the result

of a DNS cache positing attack by mapping the hostname www.instagram.com to our malicious web server.

10.9.0.80 www.instagram.com



After this step, we tried accessing our target website through the browser, but our secure connection failed. So, we copied the ca certificate from the docker container i.e., the attacker machine and placed it in the Host i.e., victims' machine and loaded the certificate into the browser and added it as an authority, by following the steps described in the previous task. Following command was used to copy ca.crt for container to host machine:

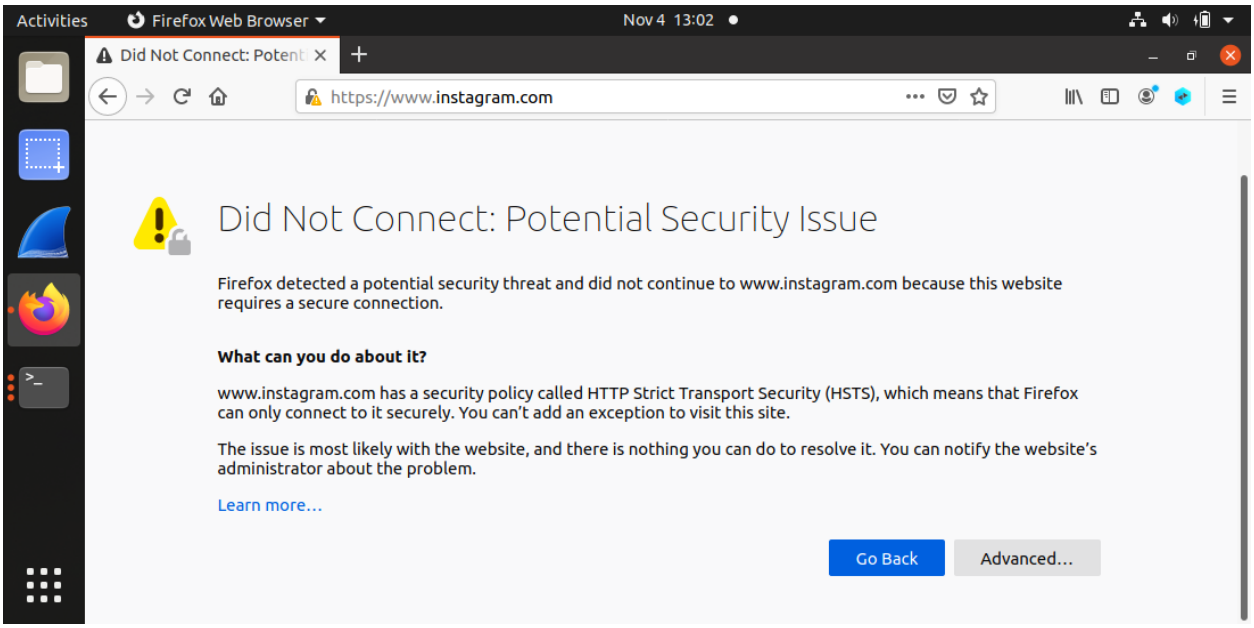
$ docker cp e5086a9e4ec9:/oracle/ca.crt /home/seed/Download

Step 3 - Browse the target website:

With everything set up, now visit the target real website.

We observe that, “Did Not Connect: Potential Security Risk” which is a warning that our browser gives us because the SSL certificate does not match the domain name www.instagram.com.

The browser alerts the user that there is some issue with the website’s security certificate. This is the first line of defense in PKI.



Team Member:
Joel Sadanand Samson: Task 1,2 and 3
G01352483

Abhijeet Amitava Banerjee: Task 4 and 5
G01349260