Transport Layer Security (TLS) Lab

Name: Neeraj Reddy Karnati

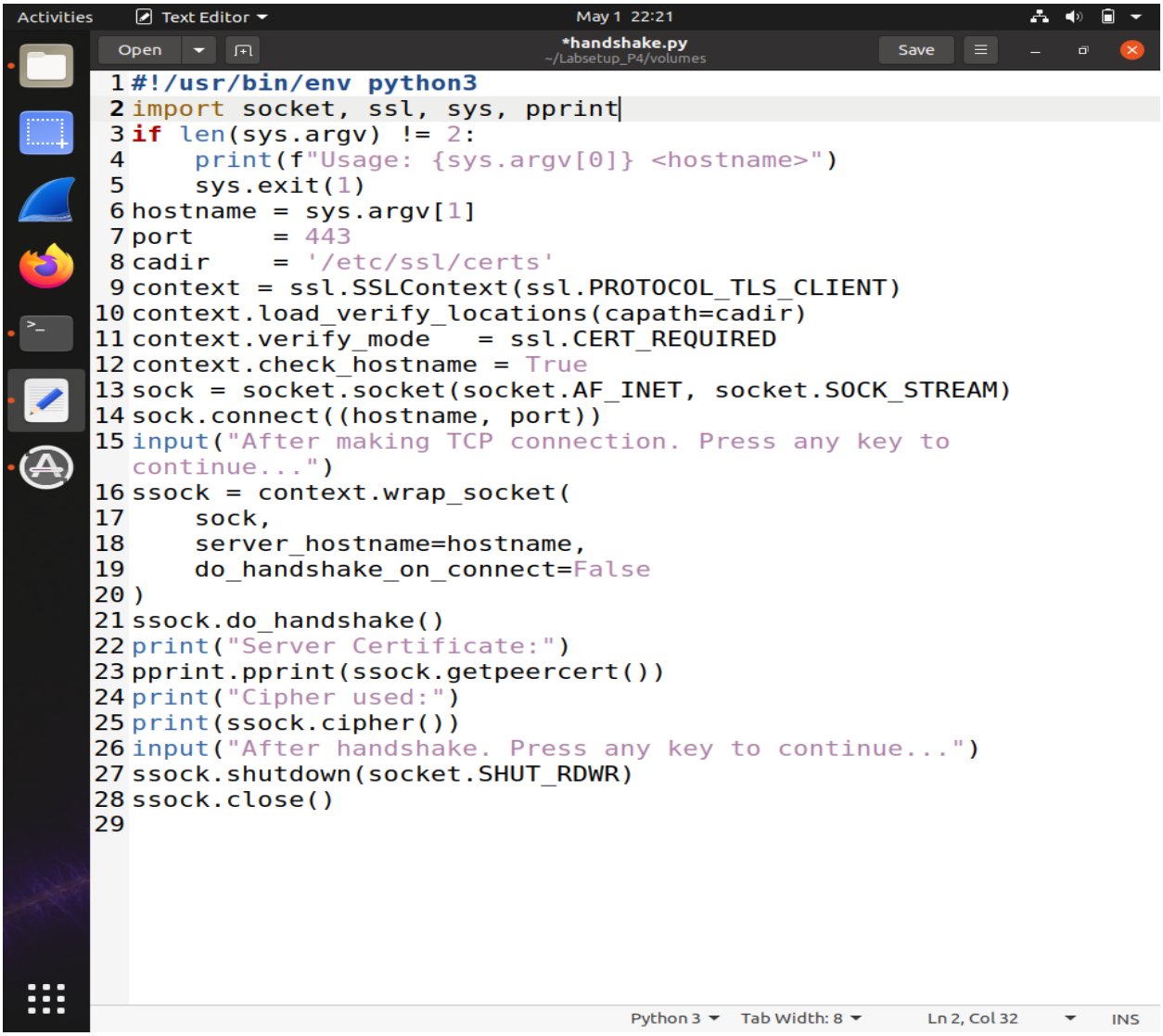
G#: 01502689

Task 3.1 – 1.a: TLS Handshake Report

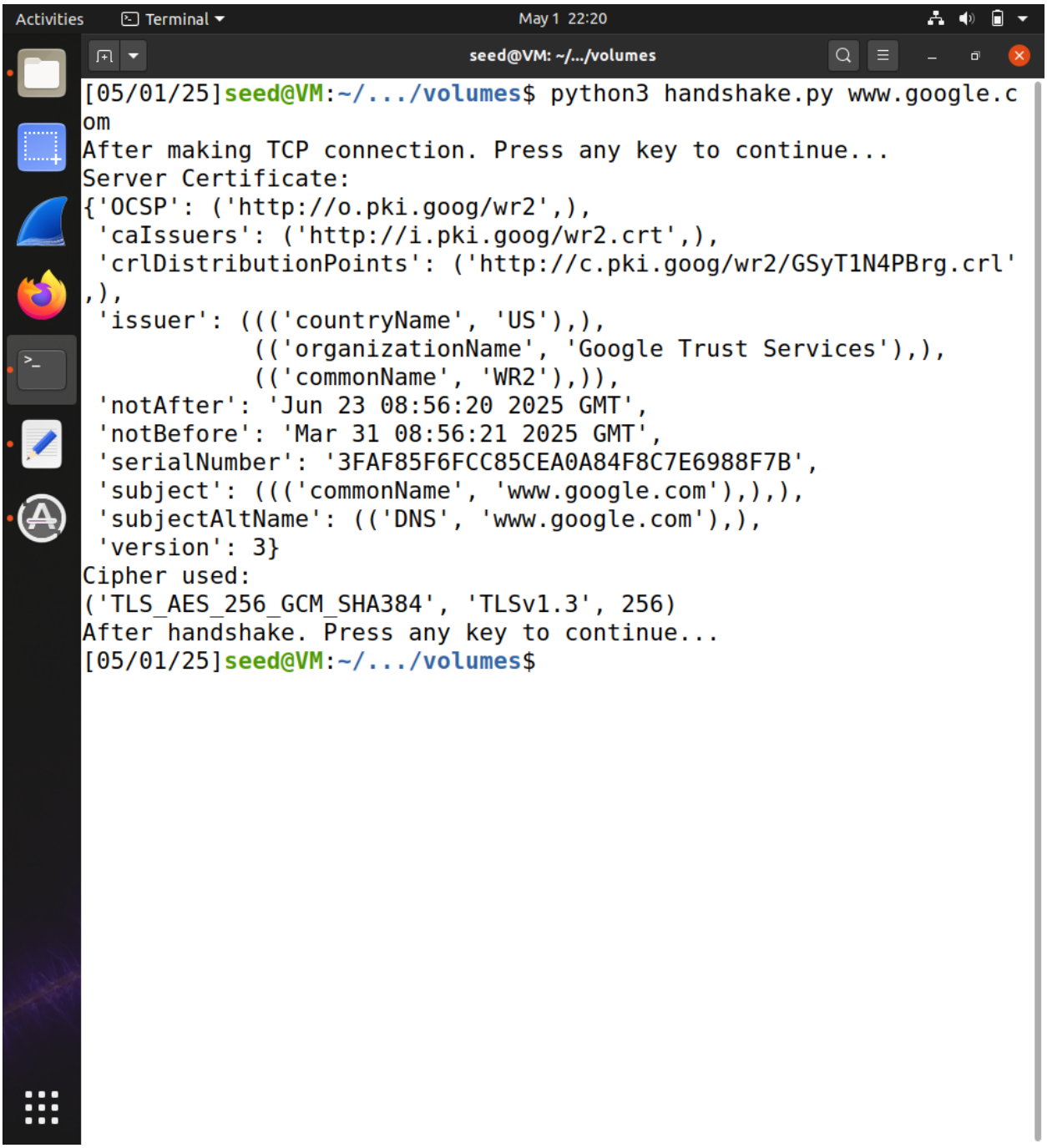
Objective:

1. Examine and illustrate the TLS handshake between an HTTPS server and a Python client, encompassing.
2. Identification of the agreed cipher suite
3. Inspection and printing of the server's certificate
4. Explanation of the role of /etc/ssl/certs
5. Packet-level analysis (Wireshark) to distinguish TCP vs. TLS handshake phases and define their relationship

Complete Client Code



Execution:



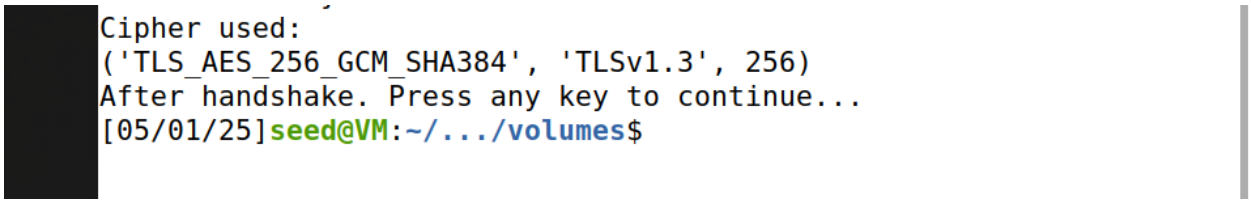
Description:

The handshake.py script is a simple Python client that shows just how a secure HTTPS connection is created in two steps: first, it initiates the SYN–SYN/ACK–ACK three-way handshake by opening a plain TCP socket to the designated hostname on port 443, and then it pauses to allow you to verify that the TCP layer is up. It then executes the TLS handshake (exchanging ClientHello, ServerHello, certificate, Finished messages, etc.) and wraps that socket in an SSL/TLS context, which is set up to need and validate the

server's certificate against the system's trusted CAs in /etc/ssl/certs. After the handshake is over, the script publishes the negotiated cipher suite (e.g., 'TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)) and the server's X.509 certificate (with the issuer, validity dates, topic, SANs, OCSP/CRL URIs, and more) before stopping once more.

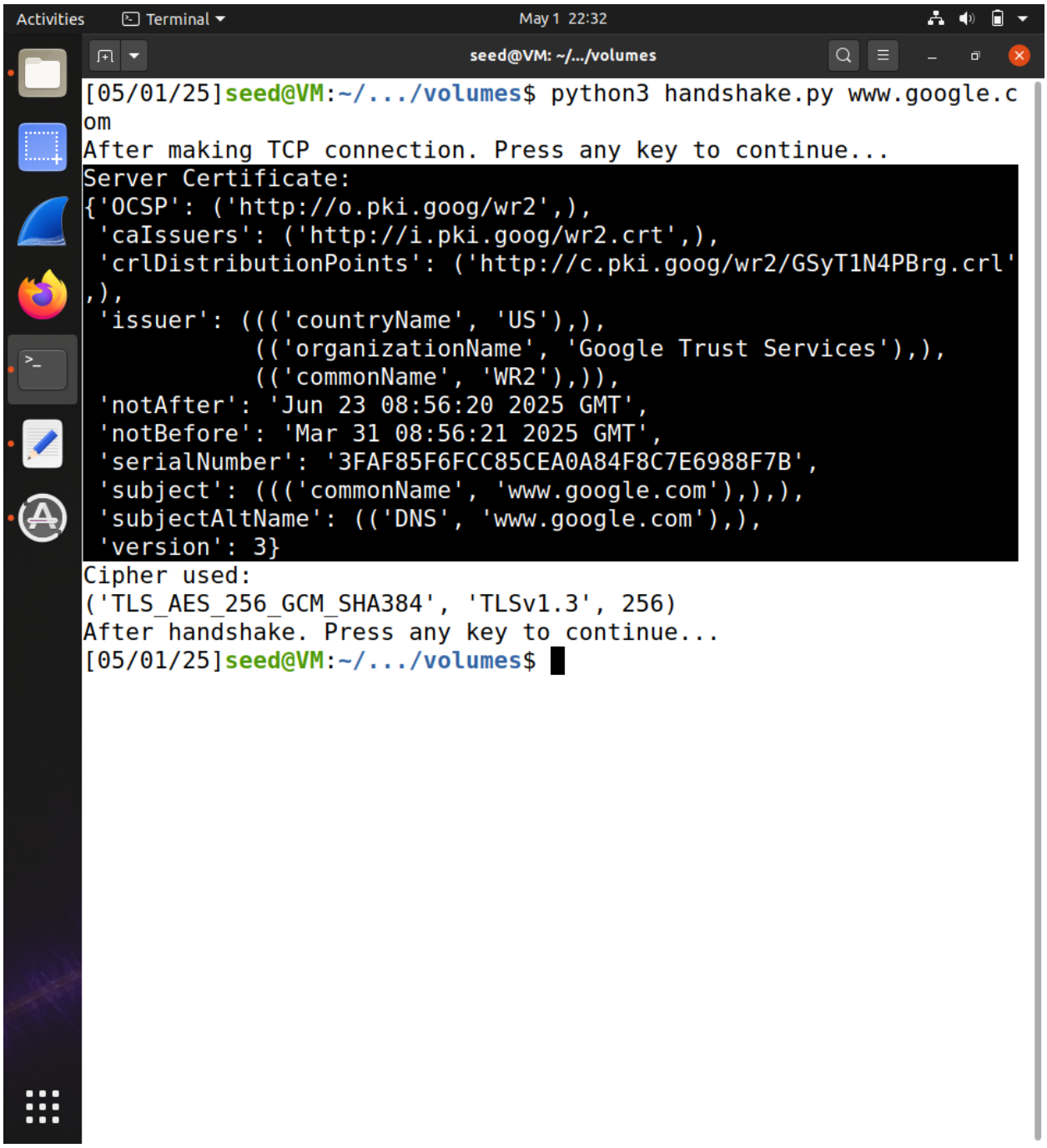
Q1) What is the cipher used between the client and the server?

A) This tuple shows that the client and server agreed on AES-256 in Galois/Counter Mode with SHA-384 for integrity, running under TLS 1.3, and using a 256-bit symmetric key. AES-GCM provides both confidentiality and authenticated integrity in a single algorithm.



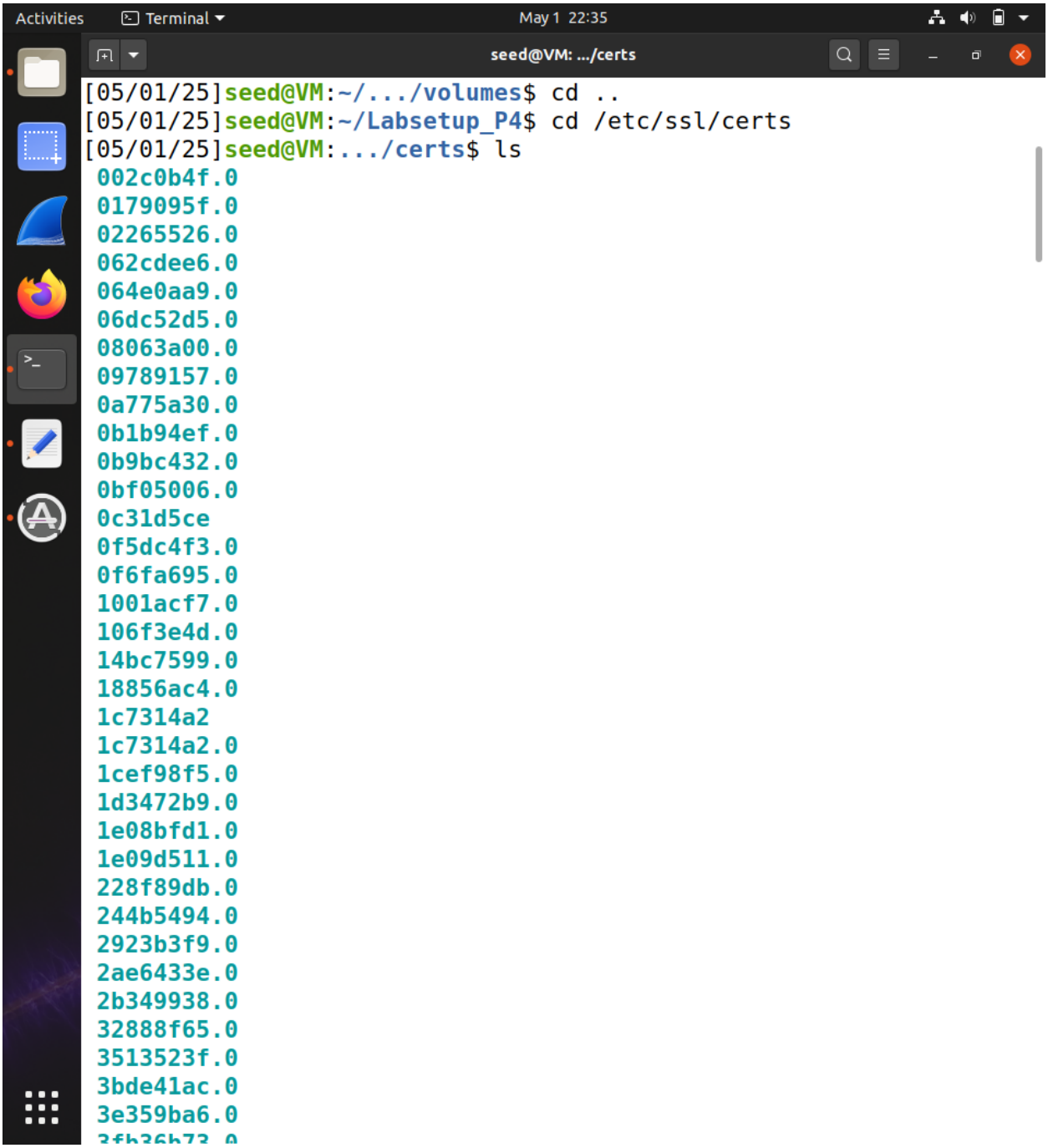
Q2) Please print out the server certificate in the program.

A) This Python dict is the server's X.509 certificate as returned by ssock.getpeercert(). It includes: Issuer: the CA that signed it ("Google Trust Services", CN=WR2, US) Validity period: from March 31 to June 23, 2025 Subject: CN = www.google.com, with SANs listing additional DNS names OCSP/CRL URIs: where to check revocation status Serial number and version


Activities
Terminal
May 1 22:32
seed@VM: ~/.../volumes
[05/01/25]seed@VM:~/.../volumes$ python3 handshake.py www.google.com
After making TCP connection. Press any key to continue...
Server Certificate:
{'OCSP': ('http://o.pki.goog/wr2',),
'caIssuers': ('http://i.pki.goog/wr2.crt',),
'crlDistributionPoints': ('http://c.pki.goog/wr2/GSyT1N4PBrg.crl',),
'issuer': ((('countryName', 'US'),),
(('organizationName', 'Google Trust Services'),),
(('commonName', 'WR2'),)),
'notAfter': 'Jun 23 08:56:20 2025 GMT',
'notBefore': 'Mar 31 08:56:21 2025 GMT',
'serialNumber': '3FAF85F6FCC85CEA0A84F8C7E6988F7B',
'subject': ((('commonName', 'www.google.com'),),),
'subjectAltName': (('DNS', 'www.google.com'),),
'version': 3}
Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
After handshake. Press any key to continue...
[05/01/25]seed@VM:~/.../volumes$

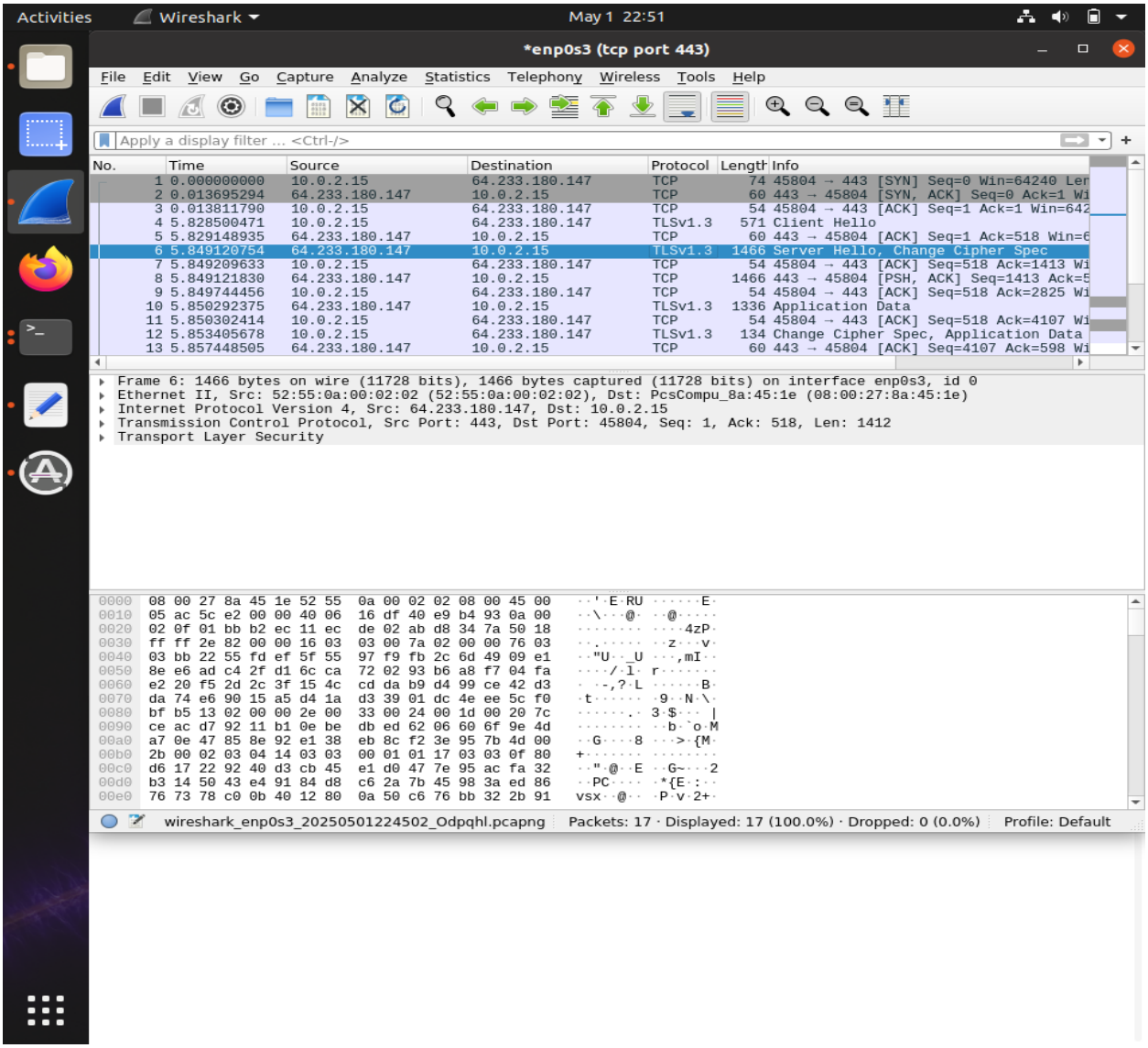
Q3) Explain the purpose of /etc/ssl/certs

A) A directory of trusted root Certificate Authority (CA) certificates in PEM format can be found at /etc/ssl/certs. Python loads each CA certificate in that folder so it may check the server's certificate chain when the script invokes context.load_verify_locations(capath='/etc/ssl/certs'). The handshake will fail if the server's certificate does not link up to one of these trustworthy roots.

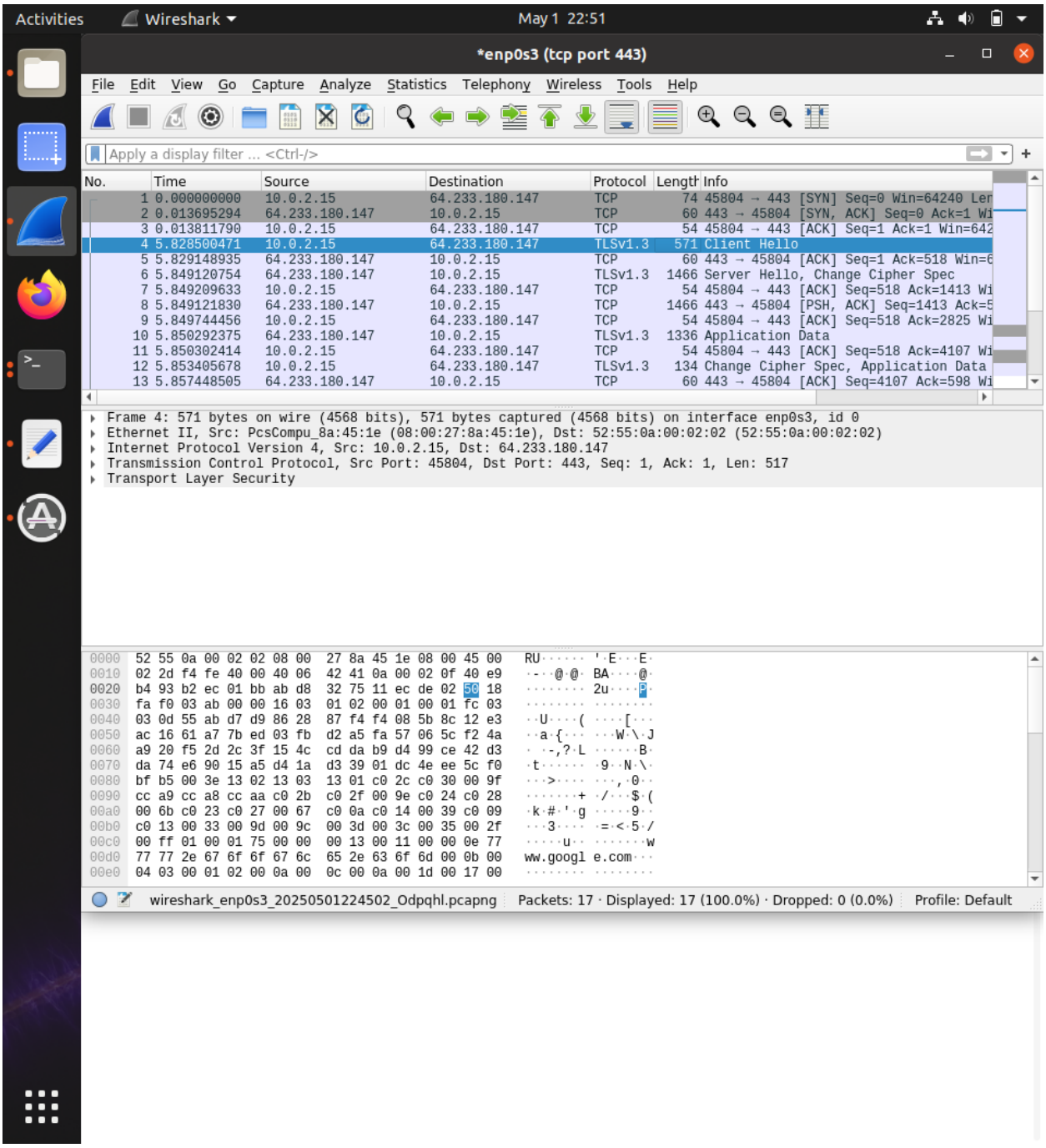


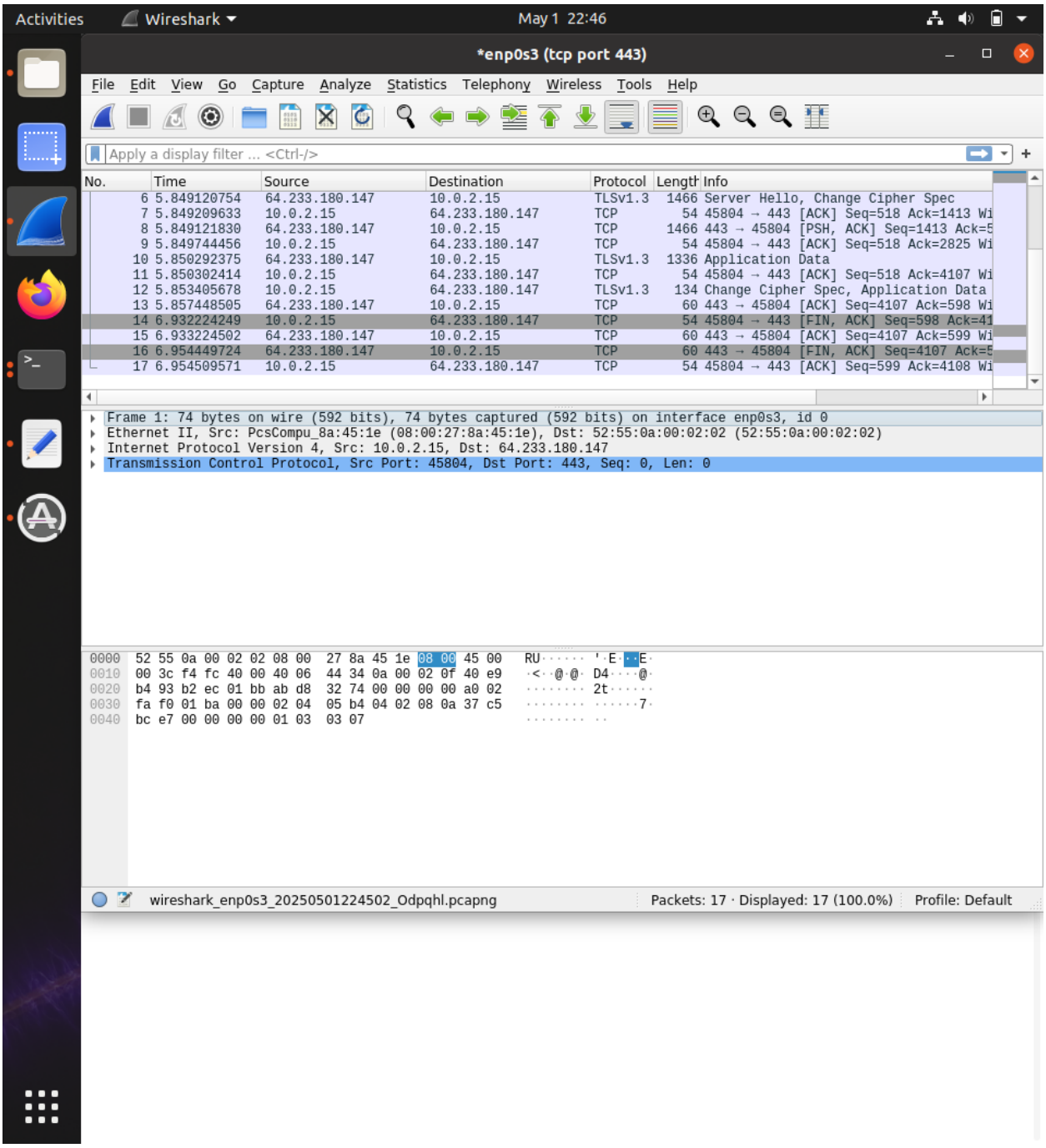
Q4) Use Wireshark to capture the network traffics during the execution of the program, and explain your observation. In particular, explain which step triggers the TCP handshake, and which step triggers the TLS handshake. Explain the relationship between the TLS handshake and the TCP handshake.

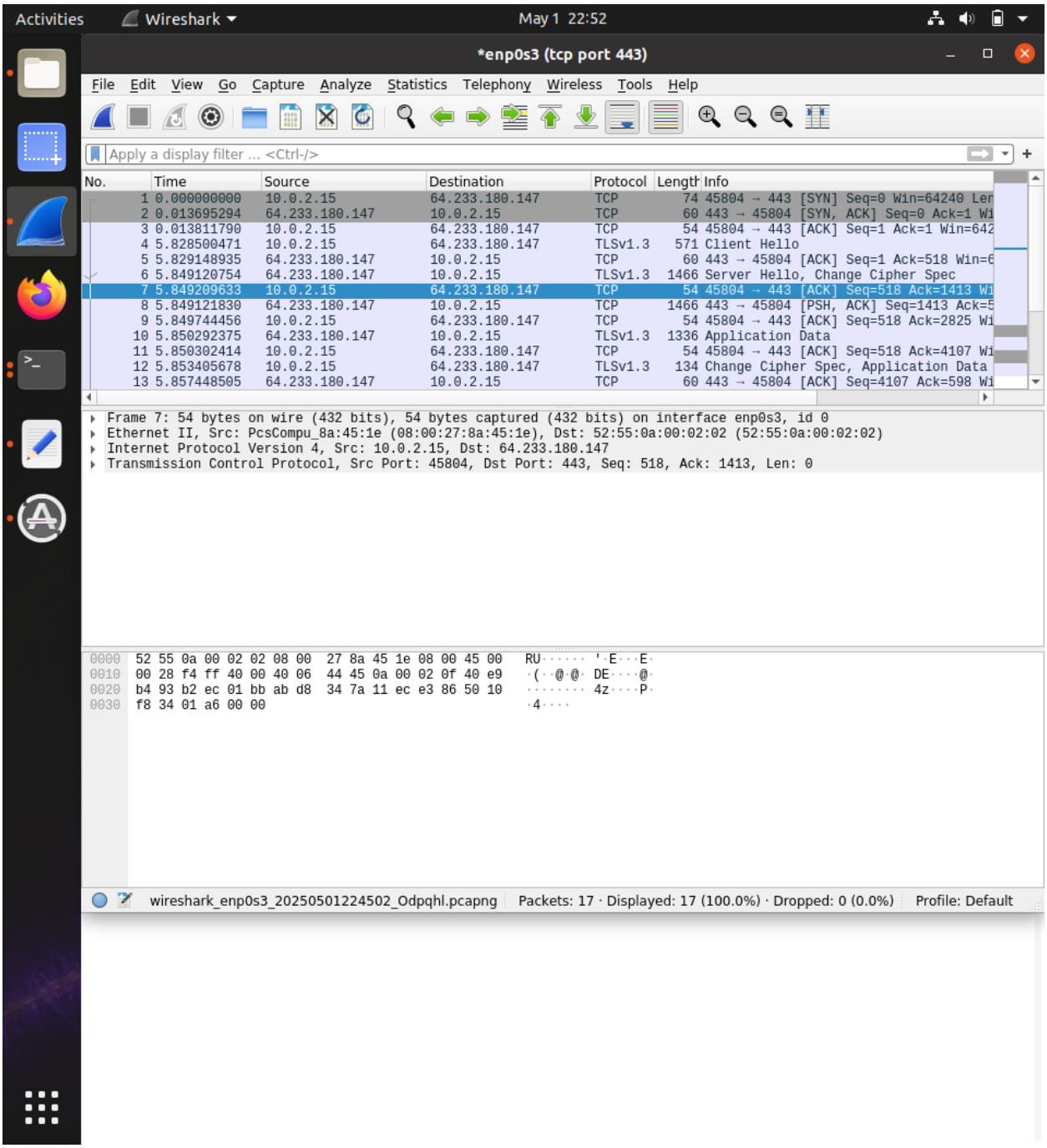
A) The initial SYN, SYN-ACK, and ACK packets exchanged between the client and server indicate that the client (10.0.2.15) is initiating a connection to the server (142.251.163.147) on port 443 (HTTPS). The TLS handshake is triggered following the successful TCP handshake, and it is responsible for establishing a secure, encrypted connection between the client and server, as evidenced by the TLS Client Hello, TLS Server Hello, and subs



equent TLS handshake packets.


Activities
Wireshark
May 1 22:51
*enp0s3 (tcp port 443)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000000 10.0.2.15 64.233.180.147 TCP 74 45804 → 443 [SYN] Seq=0 Win=64240 Len
2 0.013695294 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [SYN, ACK] Seq=0 Ack=1 Wi
3 0.013811790 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=1 Ack=1 Win=642
4 5.828500471 10.0.2.15 64.233.180.147 TLSv1.3 571 Client Hello
5 5.829148935 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [ACK] Seq=1 Ack=518 Win=6
6 5.849120754 64.233.180.147 10.0.2.15 TLSv1.3 1466 Server Hello, Change Cipher Spec
7 5.849209633 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=1413 Wi
8 5.849121830 64.233.180.147 10.0.2.15 TCP 1466 443 → 45804 [PSH, ACK] Seq=1413 Ack=5
9 5.849744456 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=2825 Wi
10 5.850292375 64.233.180.147 10.0.2.15 TLSv1.3 1336 Application Data
11 5.850302414 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=4107 Wi
12 5.853405678 10.0.2.15 64.233.180.147 TLSv1.3 134 Change Cipher Spec, Application Data
13 5.857448505 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [ACK] Seq=4107 Ack=598 Wi
Frame 4: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_8a:45:1e (08:00:27:8a:45:1e), Dst: 52:55:0a:00:02:02 (52:55:0a:00:02:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 64.233.180.147
Transmission Control Protocol, Src Port: 45804, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
Transport Layer Security
0000 52 55 0a 00 02 02 08 00 27 8a 45 1e 08 00 45 00 RU····· '·E···E·
0010 02 2d f4 fe 40 00 40 06 42 41 0a 00 02 0f 40 e9 ·-··@·@· BA····@·
0020 b4 93 b2 ec 01 bb ab d8 32 75 11 ec de 02 50 18 ········ 2u····P·
0030 fa f0 03 ab 00 00 16 03 01 02 00 01 00 01 fc 03 ········ ········
0040 03 0d 55 ab d7 d9 86 28 87 f4 f4 08 5b 8c 12 e3 ··U····(····[···
0050 ac 16 61 a7 7b ed 03 fb d2 a5 fa 57 06 5c f2 4a ··a·{··· ···W·\·J
0060 a9 20 f5 2d 2c 3f 15 4c cd da b9 d4 99 ce 42 d3 · ·-,?·L ·····B·
0070 da 74 e6 90 15 a5 d4 1a d3 39 01 dc 4e ee 5c f0 ·t····· ·9··N·\·
0080 bf b5 00 3e 13 02 13 03 13 01 c0 2c c0 30 00 9f ···>···· ···,·0··
0090 cc a9 cc a8 cc aa c0 2b c0 2f 00 9e c0 24 c0 28 ·······+ ·/···$·(
00a0 00 6b c0 23 c0 27 00 67 c0 0a c0 14 00 39 c0 09 ·k·#·'·g ·····9··
00b0 c0 13 00 33 00 9d 00 9c 00 3d 00 3c 00 35 00 2f ···3···· ·=·<·5·/
00c0 00 ff 01 00 01 75 00 00 00 13 00 11 00 00 0e 77 ·····u·· ·······w
00d0 77 77 2e 67 6f 6f 67 6c 65 2e 63 6f 6d 00 0b 00 ww.googl e.com···
00e0 04 03 00 01 02 00 0a 00 0c 00 0a 00 1d 00 17 00 ········ ········
wireshark_enp0s3_20250501224502_Odpqhl.pcapng Packets: 17 · Displayed: 17 (100.0%) · Dropped: 0 (0.0%) Profile: Default


Activities
Wireshark
May 1 22:46
*enp0s3 (tcp port 443)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
6 5.849120754 64.233.180.147 10.0.2.15 TLSv1.3 1466 Server Hello, Change Cipher Spec
7 5.849209633 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=1413 Wi
8 5.849121830 64.233.180.147 10.0.2.15 TCP 1466 443 → 45804 [PSH, ACK] Seq=1413 Ack=5
9 5.849744456 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=2825 Wi
10 5.850292375 64.233.180.147 10.0.2.15 TLSv1.3 1336 Application Data
11 5.850302414 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=518 Ack=4107 Wi
12 5.853405678 10.0.2.15 64.233.180.147 TLSv1.3 134 Change Cipher Spec, Application Data
13 5.857448505 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [ACK] Seq=4107 Ack=598 Wi
14 6.932224249 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [FIN, ACK] Seq=598 Ack=41
15 6.933224502 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [ACK] Seq=4107 Ack=599 Wi
16 6.954449724 64.233.180.147 10.0.2.15 TCP 60 443 → 45804 [FIN, ACK] Seq=4107 Ack=5
17 6.954509571 10.0.2.15 64.233.180.147 TCP 54 45804 → 443 [ACK] Seq=599 Ack=4108 Wi
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_8a:45:1e (08:00:27:8a:45:1e), Dst: 52:55:0a:00:02:02 (52:55:0a:00:02:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 64.233.180.147
Transmission Control Protocol, Src Port: 45804, Dst Port: 443, Seq: 0, Len: 0
0000 52 55 0a 00 02 02 08 00 27 8a 45 1e 08 00 45 00
0010 00 3c f4 fc 40 00 40 06 44 34 0a 00 02 0f 40 e9
0020 b4 93 b2 ec 01 bb ab d8 32 74 00 00 00 00 a0 02
0030 fa f0 01 ba 00 00 02 04 05 b4 04 02 08 0a 37 c5
0040 bc e7 00 00 00 00 01 03 03 07
wireshark_enp0s3_20250501224502_Odpqhl.pcapng
Packets: 17 · Displayed: 17 (100.0%)
Profile: Default



The relation between TLS and TCP handshakes
Layering: TLS operates over TCP.

Order: Before TLS handshake communications can be enclosed within a dependable transport channel, the TCP handshake must be finished.

Dependency: the TLS handshake never starts if TCP is unable to establish the connection (for example, SYN is lost). On the other side, you could theoretically do

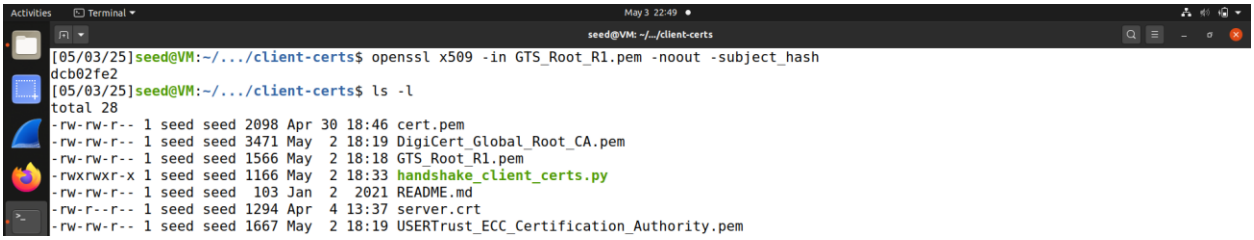
several TLS handshakes (or renegotiations) over the same TCP connection once TCP has been established.

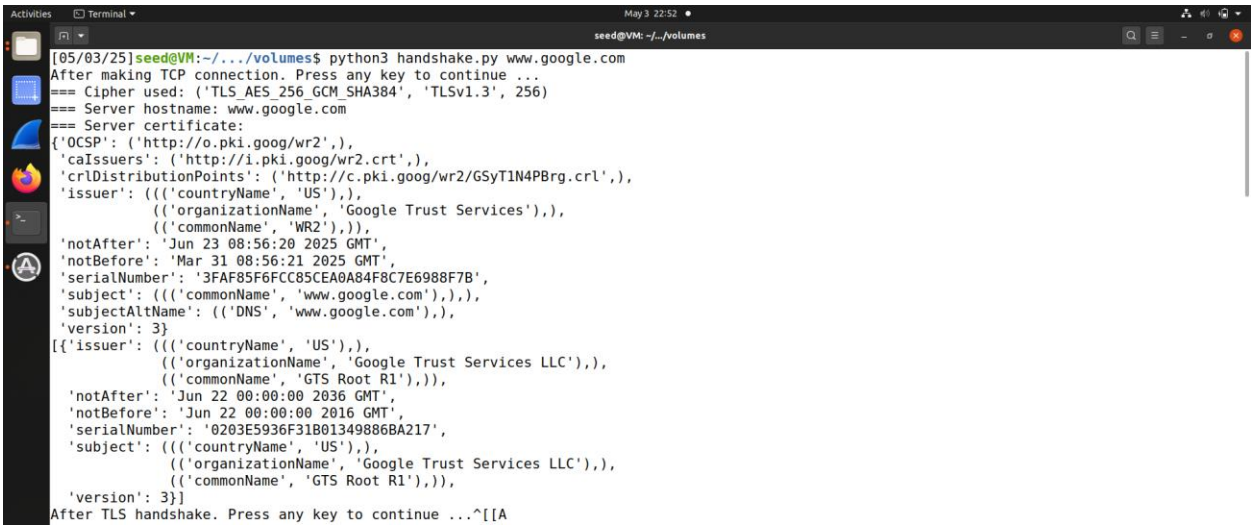
3.2 Task 1.b: CA's Certificate

Objective: This assignment aims to set up the client program to validate server certificates during TLS handshake by using a special directory (client-certs) that contains only the required CA certificates. This verification will be carried out for Three distinct websites(www.google.com, github.com, www.linkedin.com).

Environment Setup

1. **Created Certificate Folder**
   Code: mkdir client-certs
2. **Modified Client Program**
   Code: cadir = './client-certs'
3. **Website 1: www.google.com**
   a) **Steps:**
      1) Ran openssl s_client -connect www.google.com:443 -showcerts
      2) Identified CA: GTS Root R1
      3) Copied from /etc/ssl/certs:
         cp /etc/ssl/certs/GTS_Root_R1.pem client-certs/googleCA.crt
      4) Created hash symlink:
         openssl x509 -in googleCA.crt -noout -subject_hash
      5) ln -s googleCA.crt b1bc968f.0
      6) **Observation's** connection succeeded using only client-certs





4. Website 2: github.com

a) Steps:

1. Ran openssl s_client -connect github.com:443 -showcerts
2. Identified CA: DigiCert Global Root G2
3. Copied from /etc/ssl/certs:
   cp /etc/ssl/certs/DigiCert_Global_Root_G2.pem client-certs/githubCA.crt
4. Created hash symlink:
   openssl x509 -in githubCA.crt -noout -subject_hash
   ln -s githubCA.crt df5a3c34.0

5. **Observation:** TLS connection succeeded after adding DigiCert CA

```
[05/03/25]seed@VM:~/.../client-certs$ openssl x509 -in USERTrust_ECC_Certification_Authority.pem -noout -subject_hash
5850b324
[05/03/25]seed@VM:~/.../client-certs$ ls -l
total 28
-rw-rw-r-- 1 seed seed 2098 Apr 30 18:46 cert.pem
-rw-rw-r-- 1 seed seed 3471 May  2 18:19 DigiCert_Global_Root_CA.pem
-rw-rw-r-- 1 seed seed 1566 May  2 18:18 GTS_Root_R1.pem
-rwxrwxr-x 1 seed seed 1166 May  2 18:33 handshake_client_certs.py
-rw-rw-r-- 1 seed seed  103 Jan  2  2021 README.md
-rw-r--r-- 1 seed seed 1294 Apr  4 13:37 server.crt
-rw-rw-r-- 1 seed seed 1667 May  2 18:19 USERTrust_ECC_Certification_Authority.pem
```

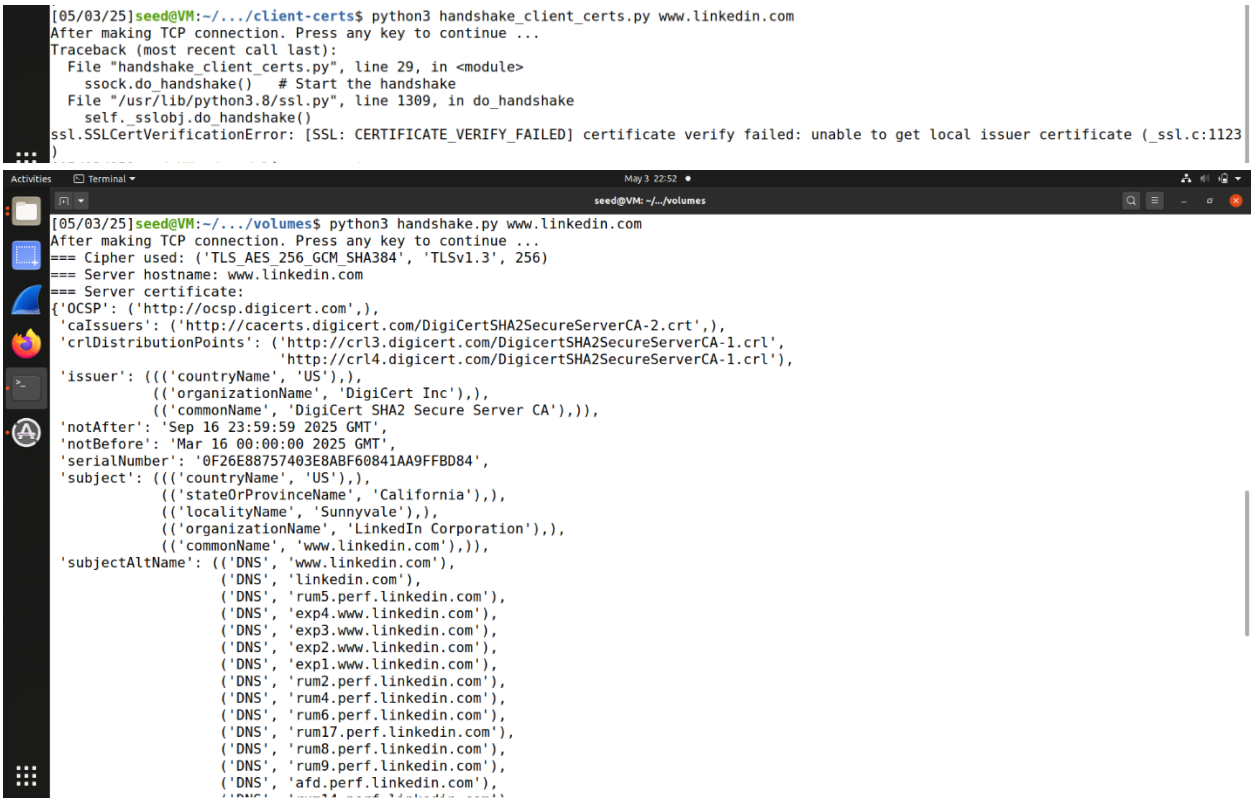
```
Activities  Terminal  May 3 22:52
seed@VM: ~/.../volumes
[05/03/25]seed@VM:~/.../volumes$ python3 handshake.py www.github.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.github.com
=== Server certificate:
{'OCSP': ('http://ocsp.sectigo.com',),
 'caIssuers': ('http://crt.sectigo.com/SectigoECCDomainValidationSecureServerCA.crt',),
 'issuer': ((('countryName', 'GB'),),
            (('stateOrProvinceName', 'Greater Manchester'),),
            (('localityName', 'Salford'),),
            (('organizationName', 'Sectigo Limited'),),
            (('commonName',
              'Sectigo ECC Domain Validation Secure Server CA'),)),
 'notAfter': 'Feb  5 23:59:59 2026 GMT',
 'notBefore': 'Feb  5 00:00:00 2025 GMT',
 'serialNumber': 'AB6686B5627BE80596821330128649F5',
 'subject': ((('commonName', 'github.com'),),),
 'subjectAltName': (('DNS', 'github.com'), ('DNS', 'www.github.com')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'New Jersey'),),
             (('localityName', 'Jersey City'),),
             (('organizationName', 'The USERTRUST Network'),),
             (('commonName', 'USERTrust ECC Certification Authority'),)),
  'notAfter': 'Jan 18 23:59:59 2038 GMT',
  'notBefore': 'Feb  1 00:00:00 2010 GMT',
  'serialNumber': '5C8B99C55A94C5D27156DECD8980CC26',
  'subject': ((('countryName', 'US'),),
              (('stateOrProvinceName', 'New Jersey'),),
              (('localityName', 'Jersey City'),),
              (('organizationName', 'The USERTRUST Network'),),
              (('commonName', 'USERTrust ECC Certification Authority'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
```

5. Website 3: www.linkedin.com

a) Steps:

1. Ran openssl s_client -connect www.linkedin.com:443 -showcerts
2. CA is again DigiCert Global Root G2
3. Already added in GitHub step — reused df5a3c34.0
4. Observation: TLS connection succeeded without adding new certificate

```
[05/03/25]seed@VM:~/.../client-certs$ ls -l
total 28
-rw-rw-r-- 1 seed seed 2098 Apr 30 18:46 cert.pem
-rw-rw-r-- 1 seed seed 3471 May  2 18:19 DigiCert_Global_Root_CA.pem
-rw-rw-r-- 1 seed seed 1566 May  2 18:18 GTS_Root_R1.pem
-rwxrwxr-x 1 seed seed 1166 May  2 18:33 handshake_client_certs.py
-rw-rw-r-- 1 seed seed  103 Jan  2  2021 README.md
-rw-r--r-- 1 seed seed 1294 Apr  4 13:37 server.crt
-rw-rw-r-- 1 seed seed 1667 May  2 18:19 USERTrust_ECC_Certification_Authority.pem
[05/03/25]seed@VM:~/.../client-certs$
```



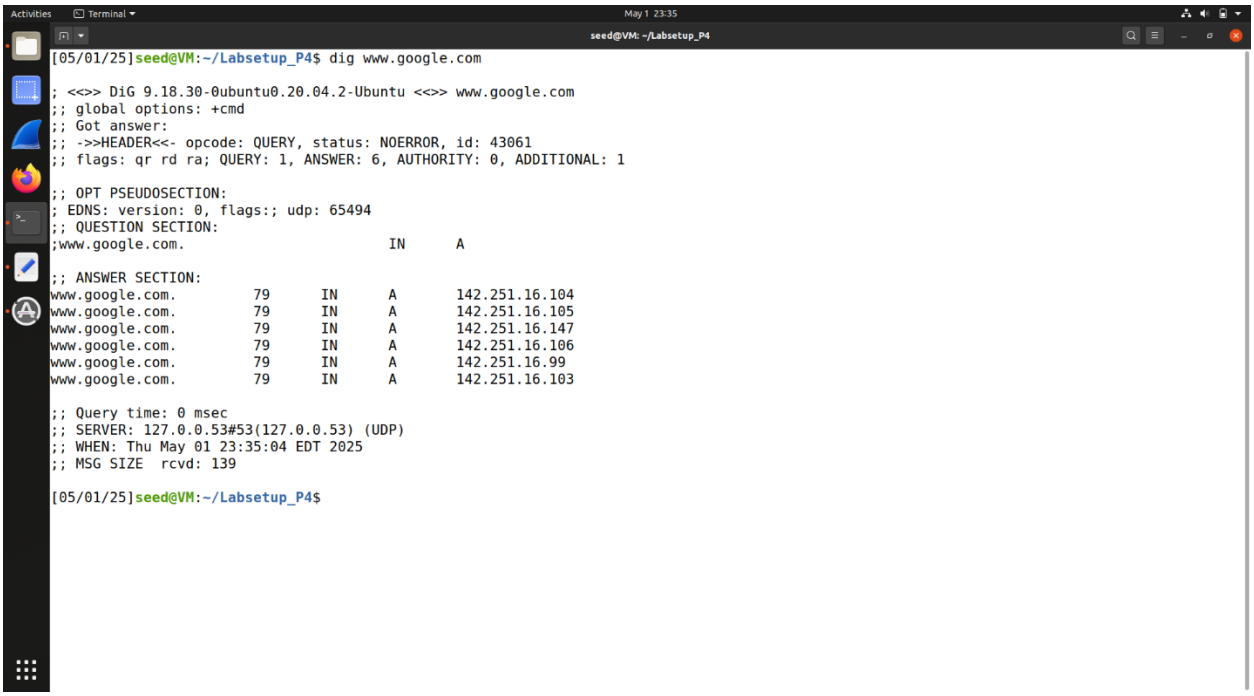
Conclusion: The correct root CA certificate must be used to validate the certificate chain of each server.The CA certificate must be renamed (or symlinked) using its topic hash in order to comply with OpenSSL.Custom CA folders for TLS verification function flawlessly once configured correctly.

3.3 Task 1.c: Experiment with the hostname check

Objective: Demonstrate the importance of verifying that the server's certificate matches the hostname requested by the client.

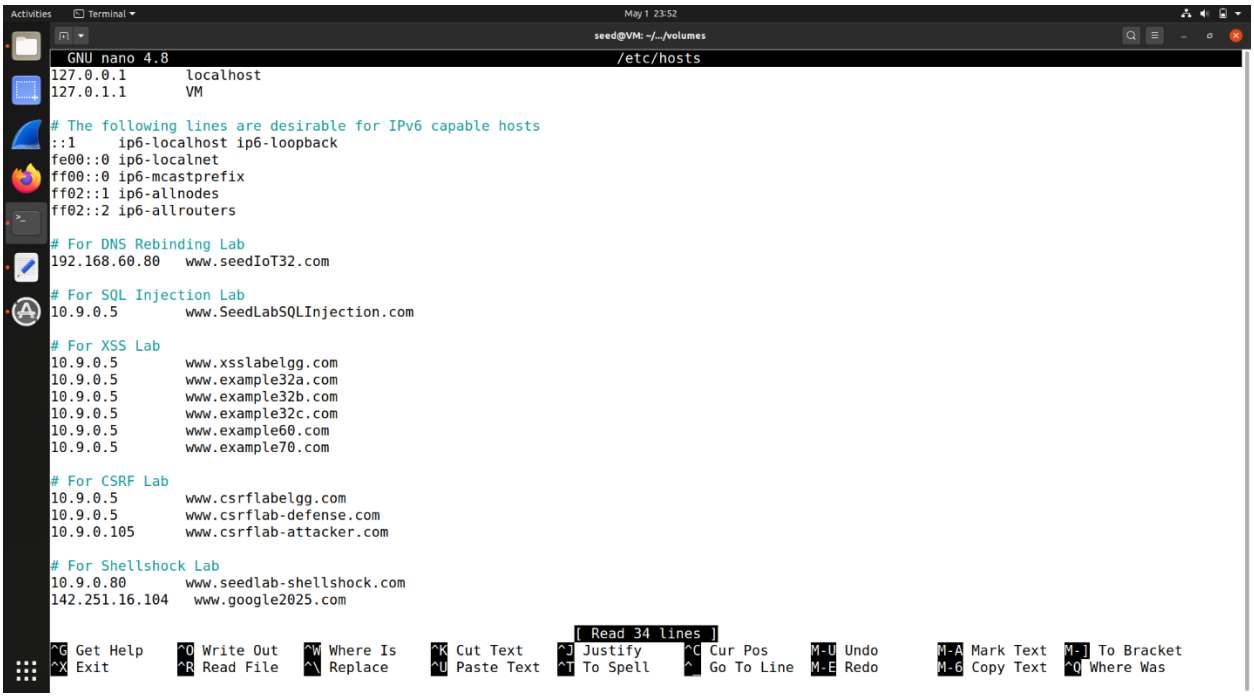
Setup Steps:

1) I have determined the real google IP using "dig" command and I got more than 2 IP addresses where I have used the first IP and changed the /etc/hosts/ to perform the rest of the task.



From the above image we can see that, we got different real IP of Google.

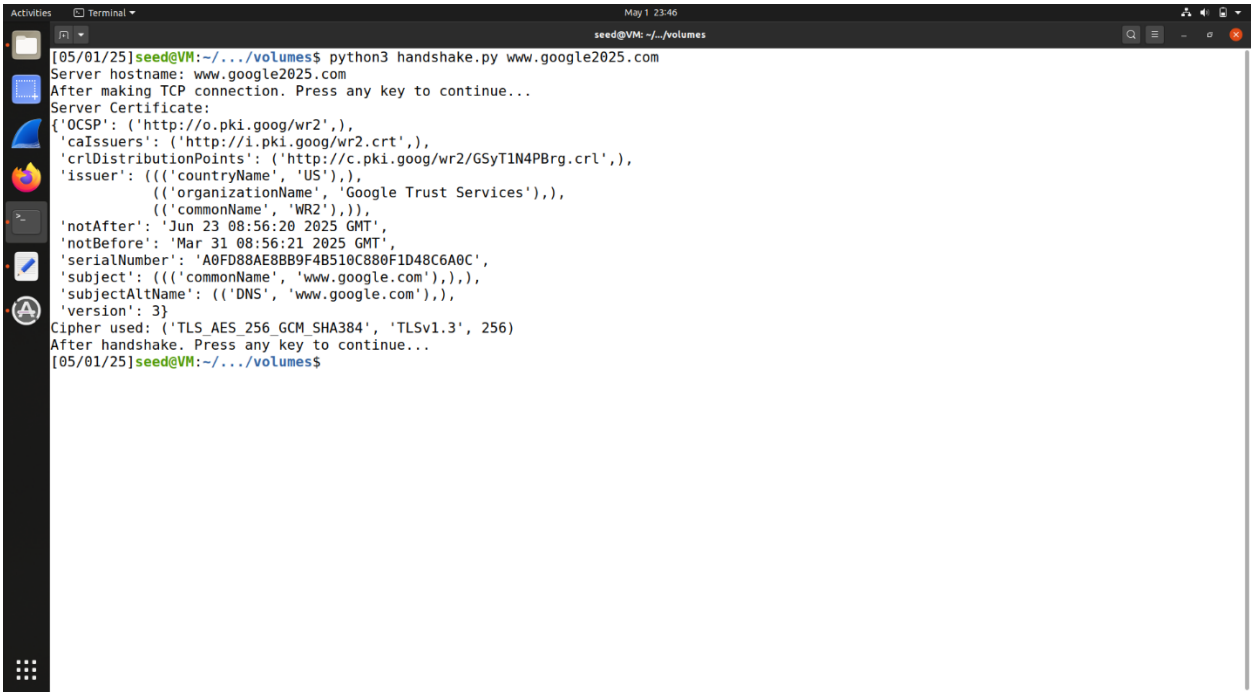
I modified the /etc/hosts file and added the following entry at the end of the file 1402.251.16.104 www.google2025.com



In the last step I have Switched the following line in the client program between True and False and then connect your client program to www.example2020.com. Describe and explain your observation.

When context.check_hostname = True

When context.check_hostname = False



Observations

- **With context. check_hostname = True**
    - **Result**: The TLS handshake fails.
    - **Error**: ssl.SSLCertVerificationError: hostname 'www.example2020.com' doesn't match 'www.google.com'
    - **Explanation**: The client enforces that the certificate's CN/SAN must exactly match the requested hostname. Since the cert is for www.google.com, it is rejected.
- **With context. check_hostname = False**
    - **Result:** The TLS handshake **succeeds**.
    - **Output:**Server Certificate: { … 'subject': ((('commonName','www.google.com'),),) … }

Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)

- **Explanation:** Disabling hostname checks skips name-matching, so as long as the certificate chain is trusted, the client accepts the connection even when the hostname doesn't match.

Q) The importance of hostname check: Based on this experiment, please explain the importance of hostname check. If the client program does not perform the hostname check, what is the security consequence? Please explain.

A) Importance of Hostname Check Hostname checking ensures that the certificate you receive really belongs to the server you asked for. By comparing the requested domain name (e.g. www.example2020.com) against the certificate's Common Name or SubjectAltName (e.g. www.google.com), the client makes sure it isn't talking to an impostor. Security Consequence if Skipped If you disable hostname checks, an attacker who can redirect your DNS or modify your /etc/hosts file could point you at their server—and if they have any valid certificate from a trusted CA (for a different domain), your client will happily connect. This opens the door to man-in-the-middle attacks, where the attacker can read or modify everything you send.

Task 1.d: Sending and getting Data

We successfully retrieved HTML content by adding code to send an HTTP request and receive the server's response. After checking that the image data was received appropriately, we changed the request to retrieve a picture from the server.

Task1: Please add the data sending/receiving code to your client program, and report your observation.

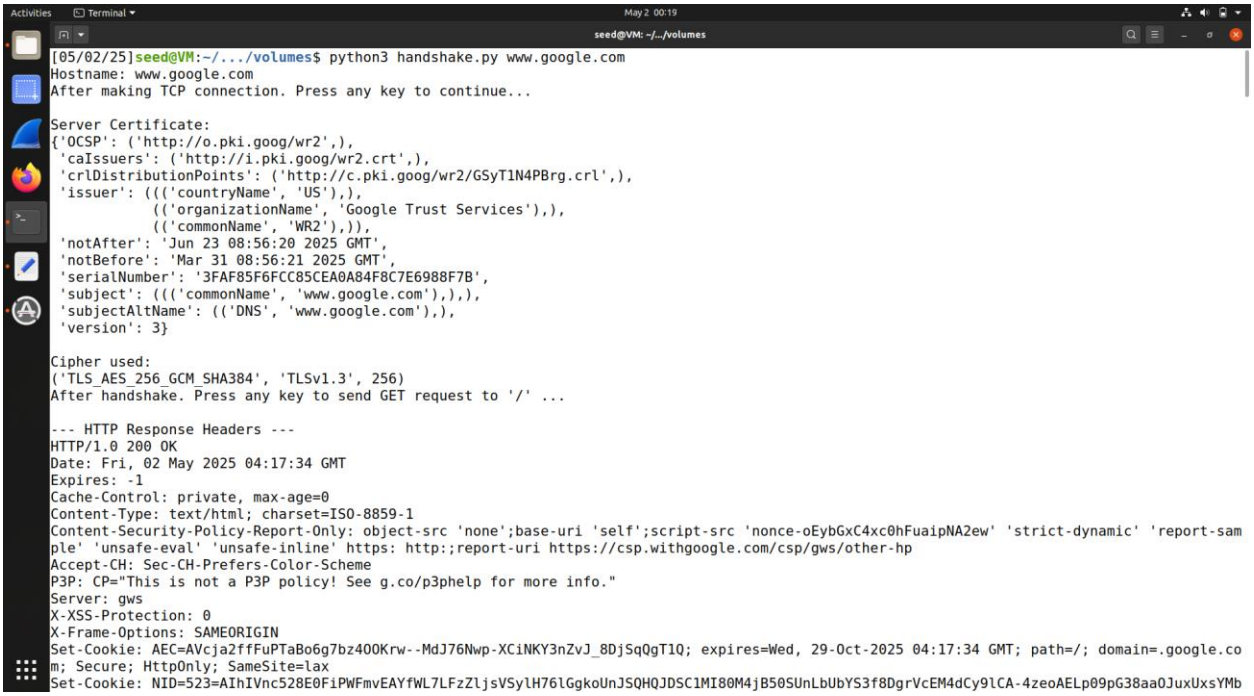
Observing the Results

When I executed the program, the following observations were made:

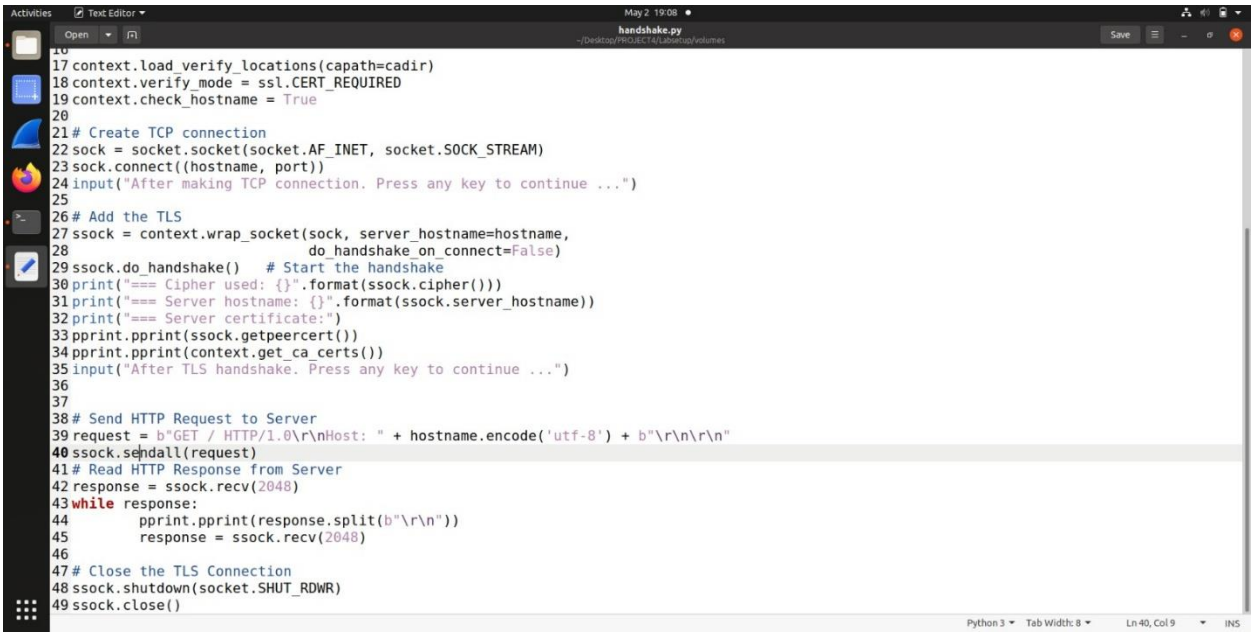
1. Connection Established:
    - The program successfully established a TCP connection to the specified server (www.google.com) and created a secure SSL/TLS connection.
2. HTTP Request Sent:
    - The program successfully sent an HTTP GET request for the root path /, over HTTPS, ensuring encrypted communication.
3. HTTP Response Received:

- The server responded to the request with an HTTP 200 OK status, indicating that the request was successfully processed and the resource (the root page) was found and returned by the server.

4. **Response Handling:**
   - The response from the server included:
     - Status Code: HTTP/1.1 200 OK, confirming the successful retrieval of the requested resource.
     - Headers: The response headers contained metadata about the server, the content type, and other relevant information.



Task2: Please modify the HTTP request, so you can fetch an image file of your choice from an HTTPS server (there is no need to display the image).

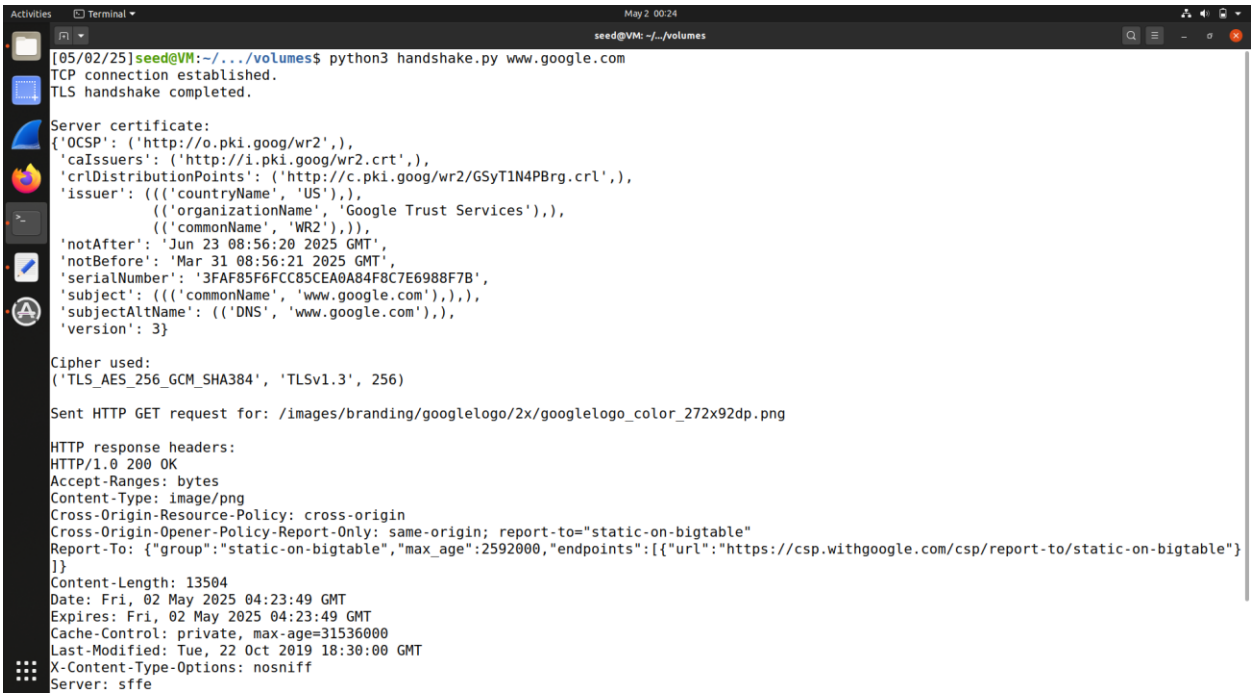


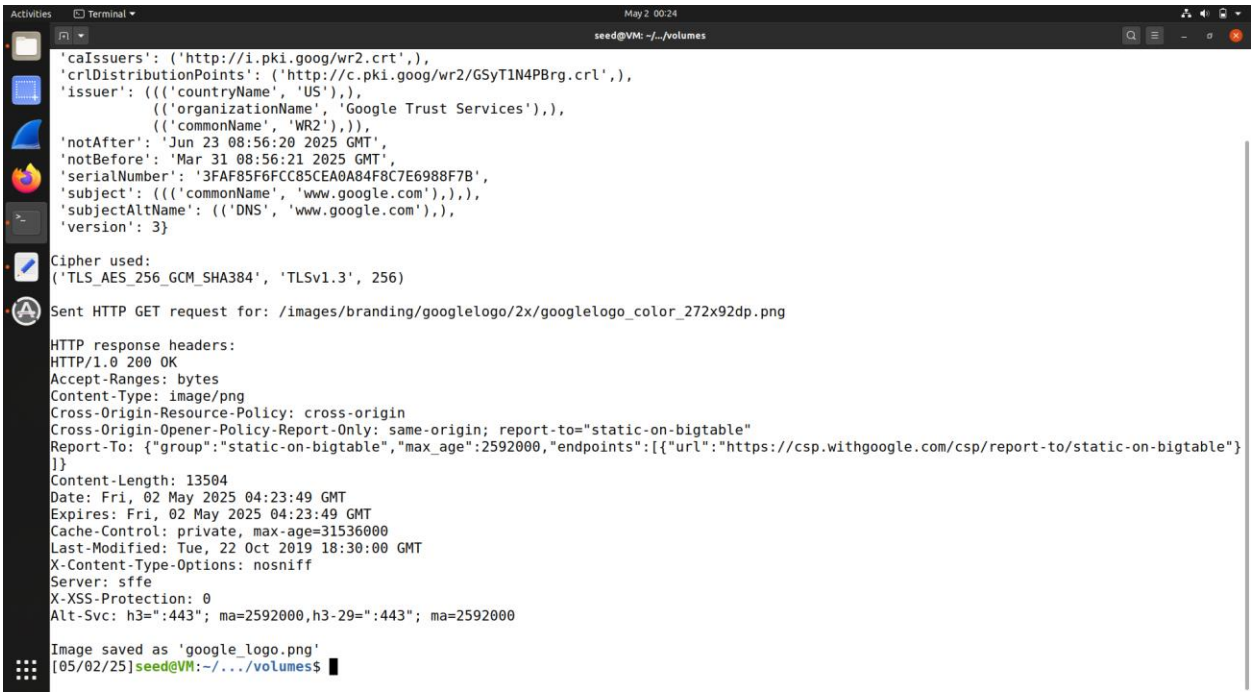
The task specifies fetching an image file, so I updated the HTTP request to specify the path to an image file on the server .

Explanation:

- **The image_path is set to the relative path of an image (in this case, python-logo.png) on the server.**
- **The GET method is updated to request the specific image file from the server.**
- **The Host header specifies the domain name to route the request correctly.**

Receiving the Image Data: Since the server will send the image data as part of the response, I kept the same response-handling mechanism but saved the image data instead of printing the response headers.





Observation of Results

When I executed the program, the following observations were made:

1. Connection Established:
    - The program successfully established a secure HTTPS connection with the specified server (www.google.com).
2. Modified HTTP Request Sent:
    - The program sent an HTTP GET request with the correct path (/static/img/python-logo.png), requesting the image file from the server.
3. HTTP Response Received:
    - The server responded with a successful HTTP 200 OK status, indicating that the requested image was found and sent back in the response.
4. Saving Image Data:
    - The program successfully received the image data in chunks and saved it as a file named google_logo.png.

Conclusion: I was able to successfully modify client software in order to send and receive HTTP requests via HTTPS. The application first created a secure SSL/TLS connection. The client then issued a GET request to the root path of the server, which received and displayed the response headers. I changed the request to retrieve an image file from the server by including its path in the second task. The image was successfully retrieved and saved locally as a PNG file without being displayed after the server returned an HTTP 200 OK response. Error-free execution of the entire procedure demonstrated proper data handling, safe connection formation, and efficient image retrieval and storage.

Task 4: TLS Server

Task 2.a. Implement a simple TLS server:

Overview: I used Python to create a simple TLS server for this task, and we used a client application to test it. Because the server utilizes a self-signed certificate, the client needs to be set up to trust the relevant CA. We compare the client to two CA directories:
1. The system CA store (/etc/ssl/certs) by default
2. A unique local CA directory (./client-certs)

Client Test 1: Using /etc/ssl/certs

1. **Configuration:**
   cadir = '/etc/ssl/certs'
2. **Action:** Ran the client and attempted to connect to the server.
3. **observation:** Connection failed with certificate verification error because the self-signed server certificate was issued by our own CA, which is not included in the system's default CA store. Therefore, the client cannot verify the authenticity of the server.

```
[05/03/25]seed@VM:~/.../volumes$ python3 handshake.py www.neeraj2025.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: www.neeraj2025.com
=== Server Certificate:
{'issuer': ((('countryName', 'US'),),
            (('stateOrProvinceName', 'Virginia'),),
            (('localityName', 'Fairfax'),),
            (('organizationName', 'GMU'),),
            (('organizationalUnitName', 'CS'),),
            (('commonName', 'neeraj'),),
            (('emailAddress', 'nkarnati@gmu.edu'),)),
 'notAfter': 'Mar 16 22:58:49 2035 GMT',
 'notBefore': 'Mar 19 22:58:49 2025 GMT',
 'serialNumber': '51CCD3C4AD25D3B4A5316E0EAAD939C0AC3FDAF3',
 'subject': ((('commonName', 'www.neeraj2025.com'),),
             (('organizationName', 'neeraj2025 Inc.'),),
             (('countryName', 'US'),)),
 'subjectAltName': [('DNS', 'www.neeraj2025.com'),
                    ('DNS', 'www.neeraj2025A.com'),
                    ('DNS', 'www.neeraj2025B.com'),
                    ('DNS', 'www.neeraj2025C.com'),
                    ('DNS', 'www.neeraj2025D.com')],
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'Virginia'),),
             (('localityName', 'Fairfax'),),
             (('organizationName', 'GMU'),),
             (('organizationalUnitName', 'CS'),),
             (('commonName', 'neearj'),),
             (('emailAddress', 'nkarnati@gmu.edu'),)),
  'notAfter': 'Mar 16 20:09:44 2035 GMT',
  'notBefore': 'Mar 19 20:09:44 2025 GMT',
  'serialNumber': '3E81C217F5D7410B4F25E42BDCCBBF6882927C7C',
  'subject': ((('countryName', 'US'),),
              (('stateOrProvinceName', 'Virginia'),),
              (('localityName', 'Fairfax'),),
```

Client Test 2: Using ./client-certs

1. Copied the custom CA certificate (myCA.crt) to ./client-certs.
2. Generated subject hash using:
   Code : openssl x509 -in myCA.crt -noout -subject_hash
3. **Configuration:**
   cadir = './client-certs'
4. **Action:** Ran the client again.
5. **Observation: Connection succeeded.** The client found the correct CA certificate using the hash-based filename, successfully verified the server's certificate, and established a secure TLS session.

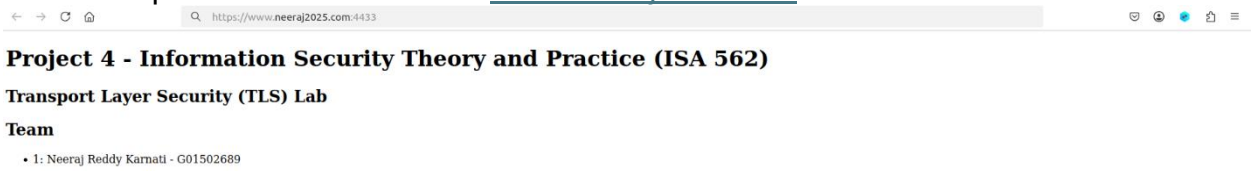
Overall Observation:

1. TLS verification relies on trust; the client will reject the connection if it does not recognize the CA that signed the server's certificate Because the System CA folder is secure by design, testing should not alter it to prevent jeopardizing OS-wide trust settings.
2. Local CA folders offer adaptability. Secure and isolated testing is made possible by using a bespoke client-certs directory with appropriately symlinked CA files.
3. Hashing is necessary: To find trusted CA certs, OpenSSL employs a hash of the certificate's subject field; renaming or symlinking must use this hash.

Conclusion: The significance of trust in TLS communication was illustrated by this task. Working with bespoke servers necessitates a regulated CA arrangement, even though system CA stores are helpful for authenticating public websites. We made it possible for the client to safely connect to the self-signed TLS server by properly generating and connecting a local CA certificate. This strategy strikes a balance between security and functionality, which makes it perfect for private deployments and lab testing.

Task 2.b. Testing the server program using browsers

Overview: In this work, we used a web browser (Firefox) to test the previously developed TLS server. Because browsers don't trust custom CAs by default, the server utilizes a self-signed certificate that was issued by the CA. The task's main objective was to confirm the results of browser access to the server both before and after the CA certificate was imported. I have used the www.neeraj2025.com as the main domain.



1. **Accessing TLS Server via Browser**
   a) The step is performed before importing CA
   b) **Action:**
   Opened Firefox and navigated to:
   https://<server-ip>/ (Server listens on port 443)
   c) Observation:
   d) **Browser displayed a warning: "Warning: Potential Security Risk Ahead"**

e) **Details:**
The browser could not verify the server's certificate because the **custom CA** used to sign it is **not trusted**.

f) **Explanation:**
Modern browsers require server certificates to be issued by a trusted CA. Since our CA was created in the lab environment and not registered with Mozilla's root store, Firefox flagged the connection as untrusted.

2. **Importing CA Certificate into Firefox**
   a) Opened Firefox.
   b) Went to about:preferences#privacy
   c) Scrolled to the bottom → Clicked on **"View Certificates"**
   d) Switched to the **Authorities** tab.
   e) Clicked **Import** → Selected myCA.crt file.
   f) Checked the box: **"Trust this CA to identify websites"**
   g) Confirmed the import.
   h) **Observation**: CA certificate was added to the trusted list
   i) **Result**: Firefox now considers TLS certificates signed by this CA as valid.

3. **Accessing TLS Server via Browser (After Importing CA)**
   a) Refreshed https://<server-ip>/
   b) **Observation:Connection succeeded,** The browser **no longer shows warnings** and directly displays the content:
   c) **Explanation:**
   Now that Firefox trusts the CA, it successfully verified the TLS certificate served by our server, allowing a secure HTTPS session.

Conclusion: This exercise illustrates how browsers manage HTTPS certificate verification. Custom CA certificates are distrusted by browsers by default. Nevertheless, the browser can safely connect to the TLS server and see its content if a custom CA has been manually imported and trusted. In a development or instructional setting, this configuration is necessary for testing local secure servers.

Task 2.c. Certificate with multiple names

Overview: Demonstrate how to create and deploy a single X.509 server certificate that is valid for multiple DNS hostnames using the Subject Alternative Name (SAN) extension and verify it against several custom domains running on port 4433.

```
[05/03/25]seed@VM:~/HTTPSLab$ openssl req -newkey rsa:2048 -config ./myopenssl.cnf -batch -sha256 -keyout server.key -out server.csr
Generating a RSA private key
.....................................................+++++
.....................................................................................................+++++
writing new private key to 'server.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
```

I have successfully pointed all the different browsers to the same domain neeraj2025.com. This demonstrates that my server now supports multiple host names. A modern TLS-capable browser was used to navigate to each hostname on port 4433. In every case, the certificate chain validated successfully, and no host name warnings appeared

https://www.bank32.com:4433

Project 4 - Information Security Theory and Practice (ISA 562)

Transport Layer Security (TLS) Lab

Team

- 1: Neeraj Reddy Karnati - G01502689

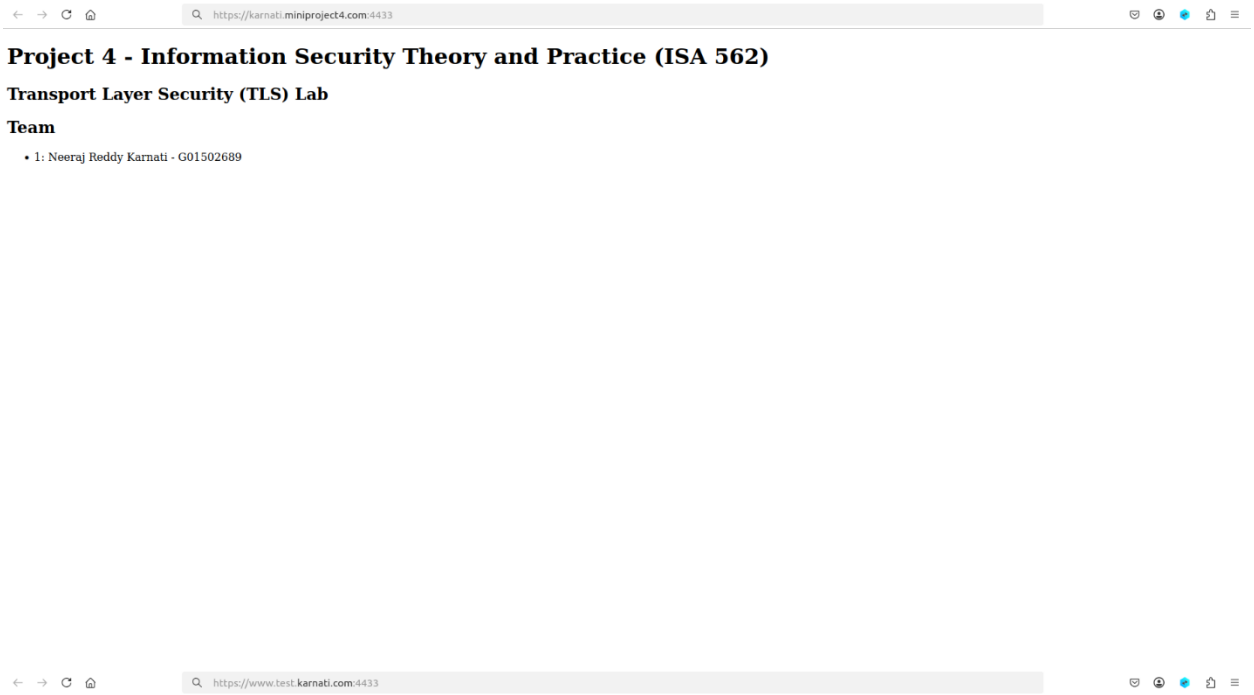
https://www.example.com:4433

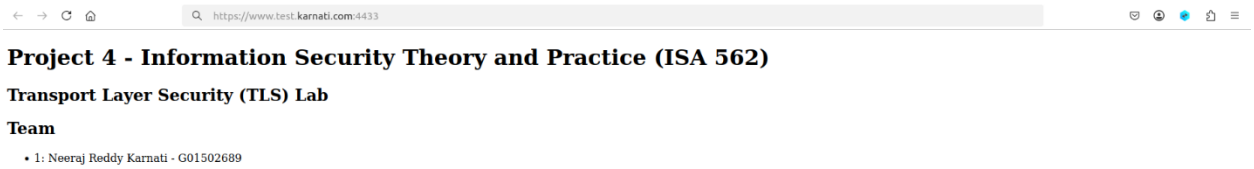
Project 4 - Information Security Theory and Practice (ISA 562)

Transport Layer Security (TLS) Lab

Team

- 1: Neeraj Reddy Karnati - G01502689





Conclusion: By using a custom OpenSSL configuration to include a SAN extension and enabling extension copying, we successfully issued a single certificate valid for multiple DNS names—demonstrated by seamless TLS handshakes across five distinct domains on port 4433. This approach satisfies hostname-matching requirements enforced by TLS clients without needing separate certificates for each URL.