

# **Transport Layer Security (TLS) Lab**

**Name: Neeraj Reddy Karnati**

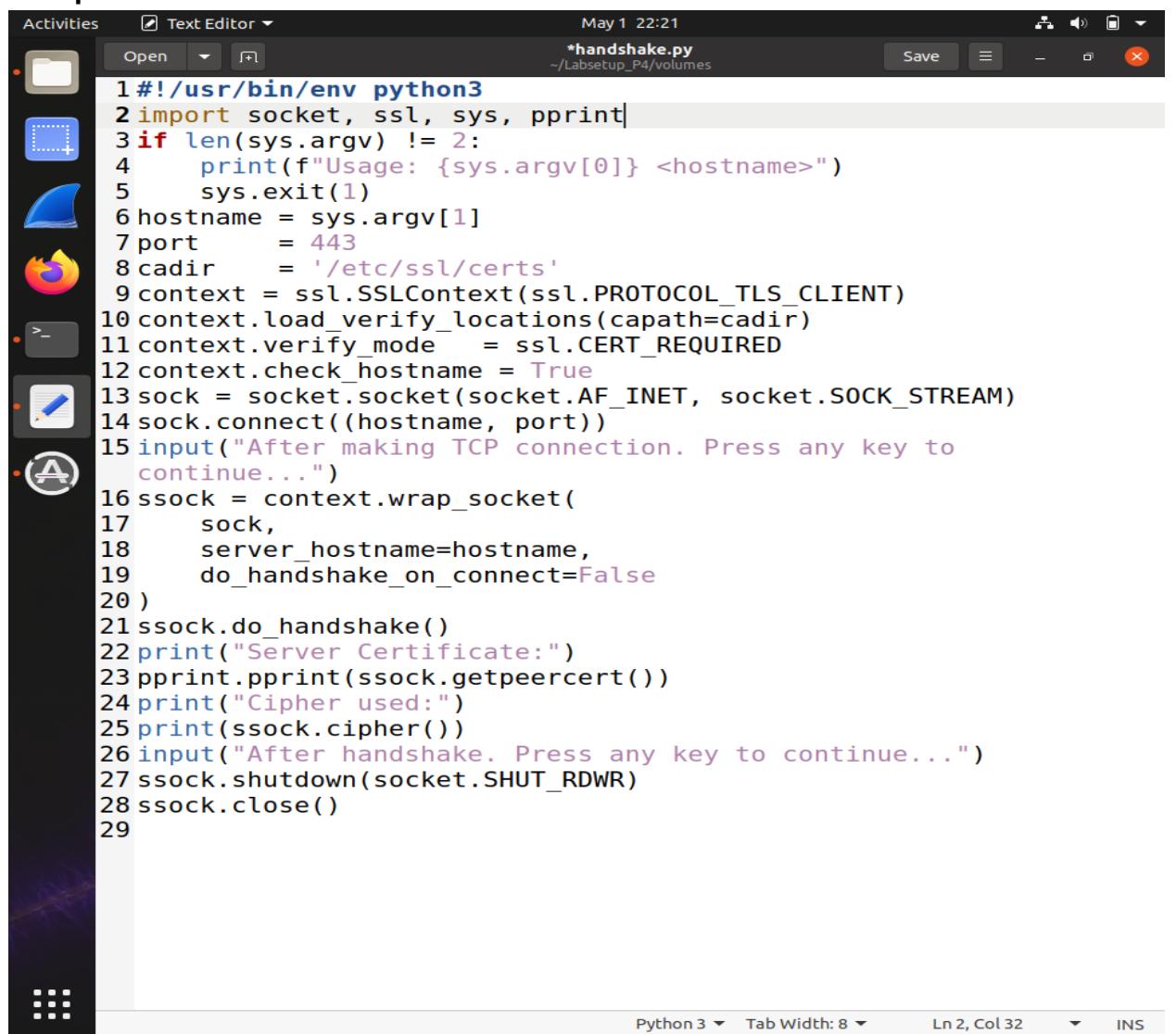
**G#: 01502689**

## **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

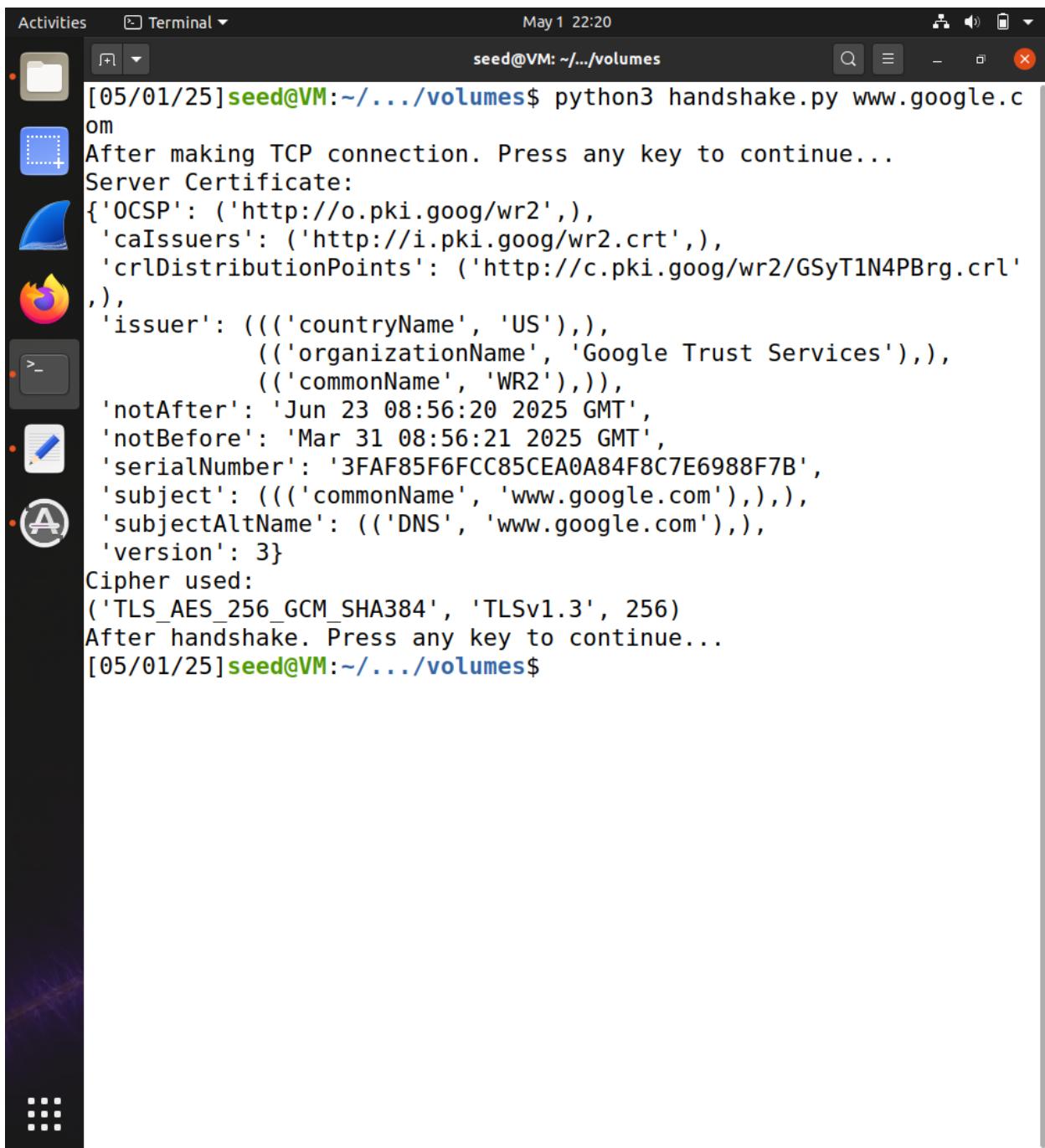
## Complete Client Code



The screenshot shows a terminal window titled "Text Editor" with the file name "handshake.py". The code is a Python script for performing a TLS handshake. It imports socket, ssl, sys, and pprint. It checks for command-line arguments and prints usage if there are not two. It then sets the hostname and port (443), loads certificates from /etc/ssl/certs, creates an SSLContext for TLS\_CLIENT, and connects to the specified host and port. It then performs a handshake, prints the server certificate, and lists the cipher used. Finally, it shuts down the connection and closes the socket.

```
1#!/usr/bin/env python3
2import socket, ssl, sys, pprint
3if len(sys.argv) != 2:
4    print(f"Usage: {sys.argv[0]} <hostname>")
5    sys.exit(1)
6hostname = sys.argv[1]
7port      = 443
8cadir     = '/etc/ssl/certs'
9context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
10context.load_verify_locations(capath=cadir)
11context.verify_mode = ssl.CERT_REQUIRED
12context.check_hostname = True
13sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14sock.connect((hostname, port))
15input("After making TCP connection. Press any key to continue...")
16ssock = context.wrap_socket(
17    sock,
18    server_hostname=hostname,
19    do_handshake_on_connect=False
20)
21ssock.do_handshake()
22print("Server Certificate:")
23pprint.pprint(ssock.getpeercert())
24print("Cipher used:")
25print(ssock.cipher())
26input("After handshake. Press any key to continue...")
27ssock.shutdown(socket.SHUT_RDWR)
28ssock.close()
29
```

**Execution:**



```
Activities Terminal May 1 22:20
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$
```

### 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 URLs, 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.

```
Cipher used:  
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)  
After handshake. Press any key to continue...  
[05/01/25] seed@VM:~/.../volumes$
```

**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 URLs: where to check revocation status Serial number and version

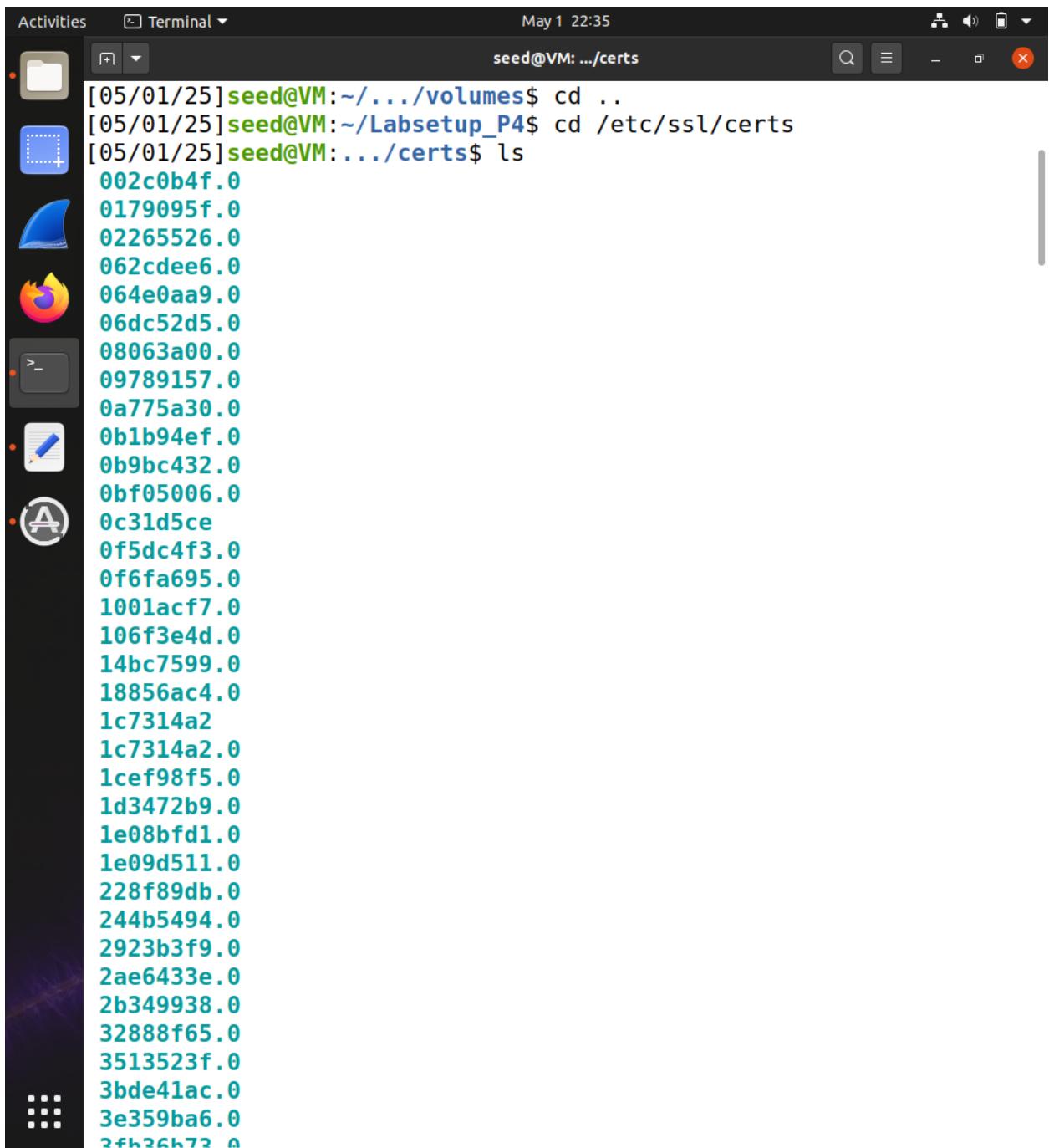
Activities Terminal May 1 22:32

```
[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.

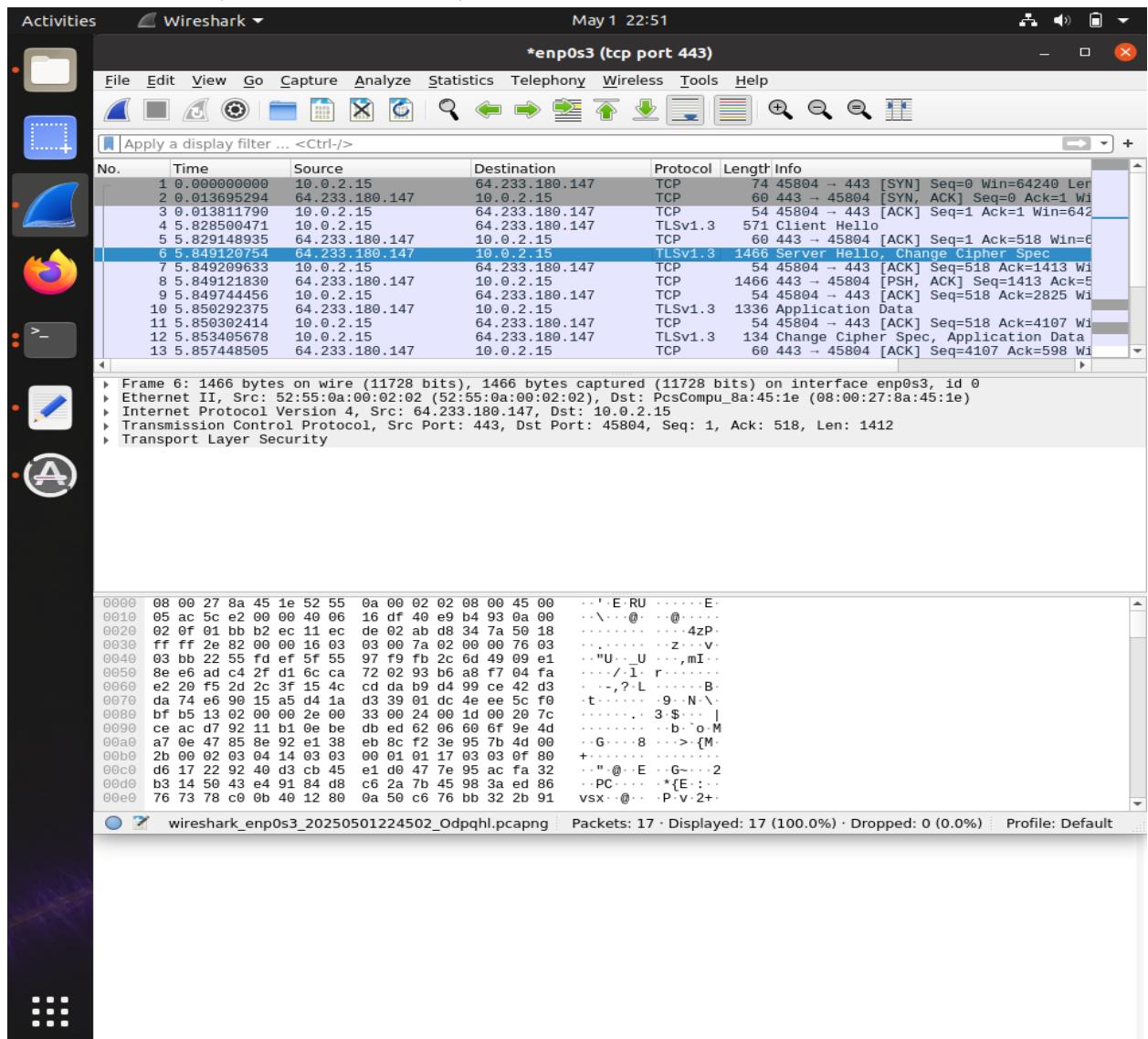


The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "Terminal" and the command prompt is "seed@VM: .../certs". The terminal displays the output of the "ls" command, listing numerous files in the "/etc/ssl/certs" directory, each ending in ".0". The files are listed in alphabetical order by name. The desktop background is visible on the left, showing various application icons in a dock.

```
[05/01/25] seed@VM:~/.../volumes$ cd ..
[05/01/25] seed@VM:~/Labsetup_P4$ cd /etc/ssl/certs
[05/01/25] seed@VM:.../certs$ ls
002c0b4f.0
0179095f.0
02265526.0
062cdee6.0
064e0aa9.0
06dc52d5.0
08063a00.0
09789157.0
0a775a30.0
0b1b94ef.0
0b9bc432.0
0bf05006.0
0c31d5ce
0f5dc4f3.0
0f6fa695.0
1001acf7.0
106f3e4d.0
14bc7599.0
18856ac4.0
1c7314a2
1c7314a2.0
1cef98f5.0
1d3472b9.0
1e08bfd1.0
1e09d511.0
228f89db.0
244b5494.0
2923b3f9.0
2ae6433e.0
2b349938.0
32888f65.0
3513523f.0
3bde41ac.0
3e359ba6.0
2fb26b72.0
```

**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 Telephoney 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=64
2	0.013695294	64.233.180.147	10.0.2.15	TCP	60	443 → 45804 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=64
3	0.013811790	10.0.2.15	64.233.180.147	TCP	54	45804 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=64
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=64240 Len=64
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 Win=64240 Len=64
8	5.849121830	64.233.180.147	10.0.2.15	TCP	1466	443 → 45804 [PSH, ACK] Seq=1413 Ack=518 Win=64240 Len=64
9	5.849744456	10.0.2.15	64.233.180.147	TCP	54	45804 → 443 [ACK] Seq=518 Ack=2825 Win=64240 Len=64
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 Win=64240 Len=64
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 Win=64240 Len=64

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 09 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.

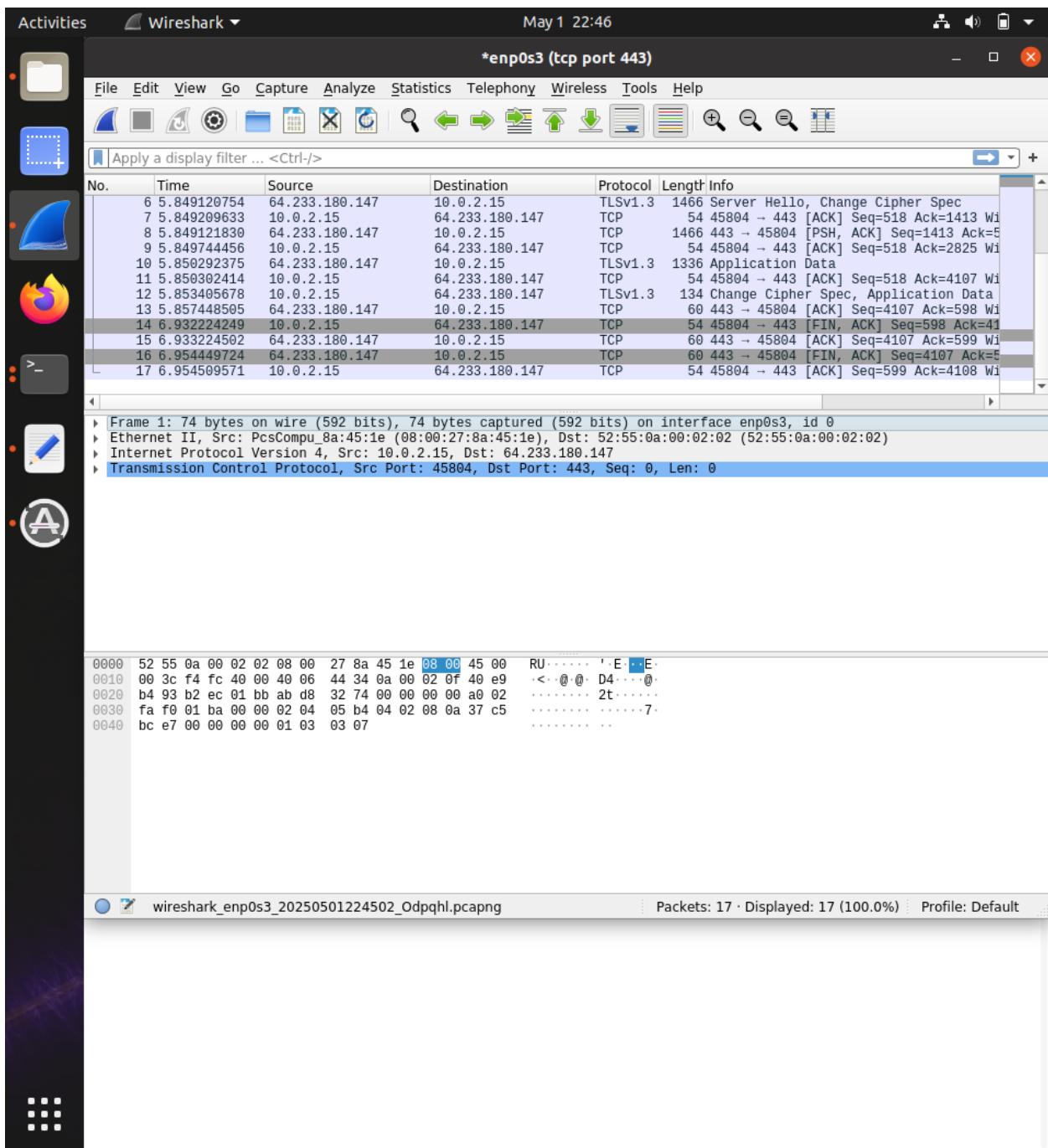
00b0 c0 13 00 33 00 9d 00 9c 00 3d 00 3c 00 35 00 2f ..3....=<5/

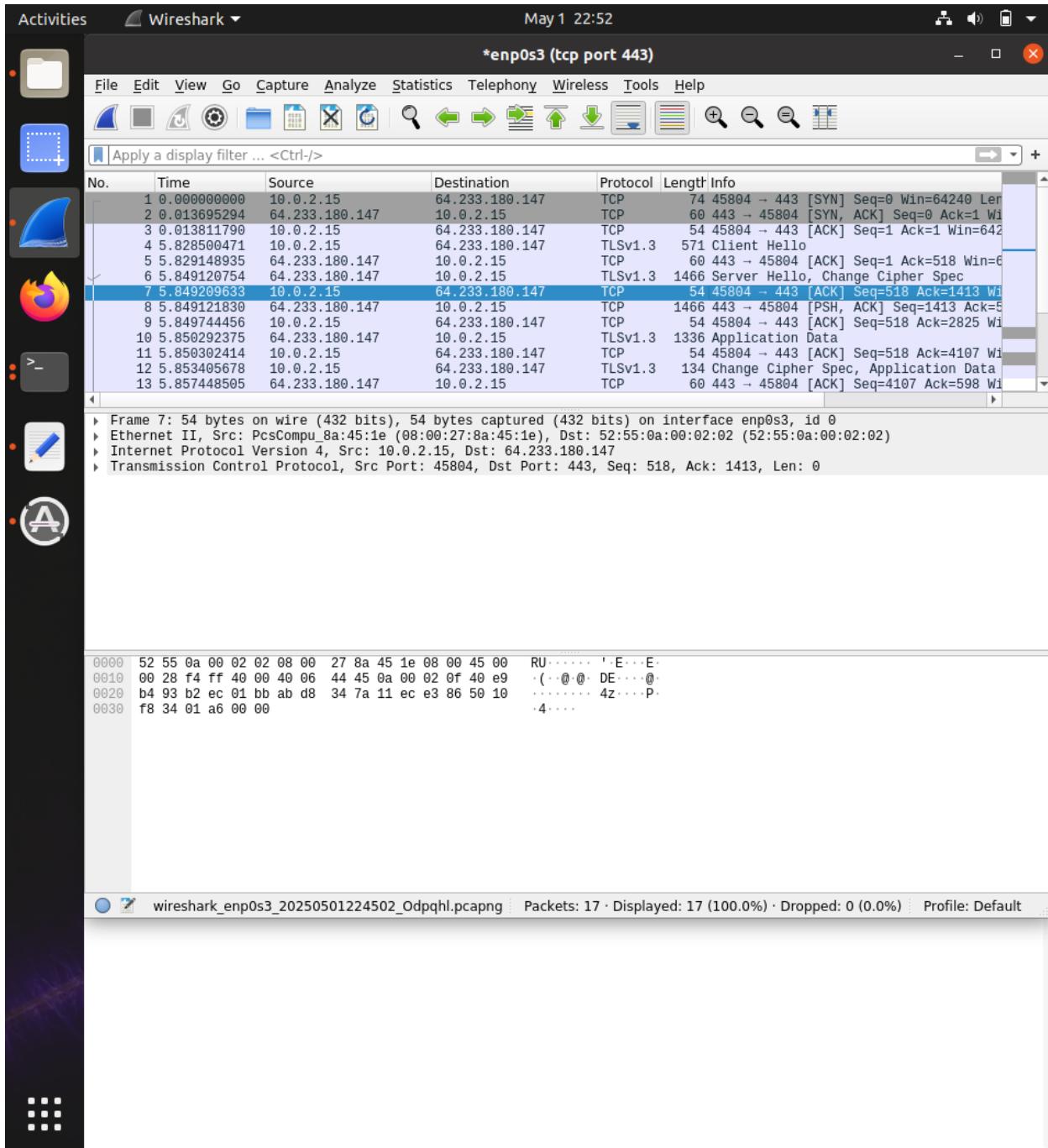
00c0 00 ff 01 00 01 75 00 00 00 13 00 11 00 00 0e 77 .....u.....w

00d0 77 77 2e 67 6f 6f 67 6c 65 2e 63 6f 6d 00 0b 00 www.google.com..

00e0 04 03 00 01 02 00 0a 00 0c 00 0a 00 1d 00 17 00 .....

wireshark\_enp0s3\_20250501224502\_Odpqlh.pcapng | Packets: 17 · Displayed: 17 (100.0%) · Dropped: 0 (0.0%) · 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](http://www.google.com), [github.com](http://github.com), [www.linkedin.com](http://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](http://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**

```
Activities Terminal May 3 22:49 seed@VM:~/.../client-certs$ openssl x509 -in GTS_Root_R1.pem -noout -subject_hash
dcb02fe2
[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$ python3 handshake_client_certs.py www.google.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake_client_certs.py", line 29, in <module>
    ssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)
```

```

[05/03/25]seed@VM:~/.../volumes$ python3 handshake.py www.google.com
After making TCP connection. Press any key to continue ...
==== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==== Server hostname: www.google.com
==== 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),
                           {'issuer': (((('countryName', 'US'),
                                         (('organizationName', 'Google Trust Services LLC'),),
                                         (('commonName', 'GTS Root R1'),)),
                                         ('notAfter': 'Jun 22 00:00:00 2036 GMT',
                                          'notBefore': 'Jun 22 00:00:00 2016 GMT',
                                          'serialNumber': '0203E5936F31B01349886BA217',
                                          'subject': (((('countryName', 'US'),),
                                                       ('organizationName', 'Google Trust Services LLC'),),
                                                       ('commonName', 'GTS Root R1'),),
                                                       'version': 3)}]
After TLS handshake. Press any key to continue ...^[[A

```

## 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`
  
`In -s githubCA.crt df5a3c34.0`

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

```

[05/03/25]seed@VM:~/.../client-certs$ python3 handshake_client_certs.py www.github.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake_client_certs.py", line 29, in <module>
    ssock.do_handshake()  # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)

```

```
[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
```

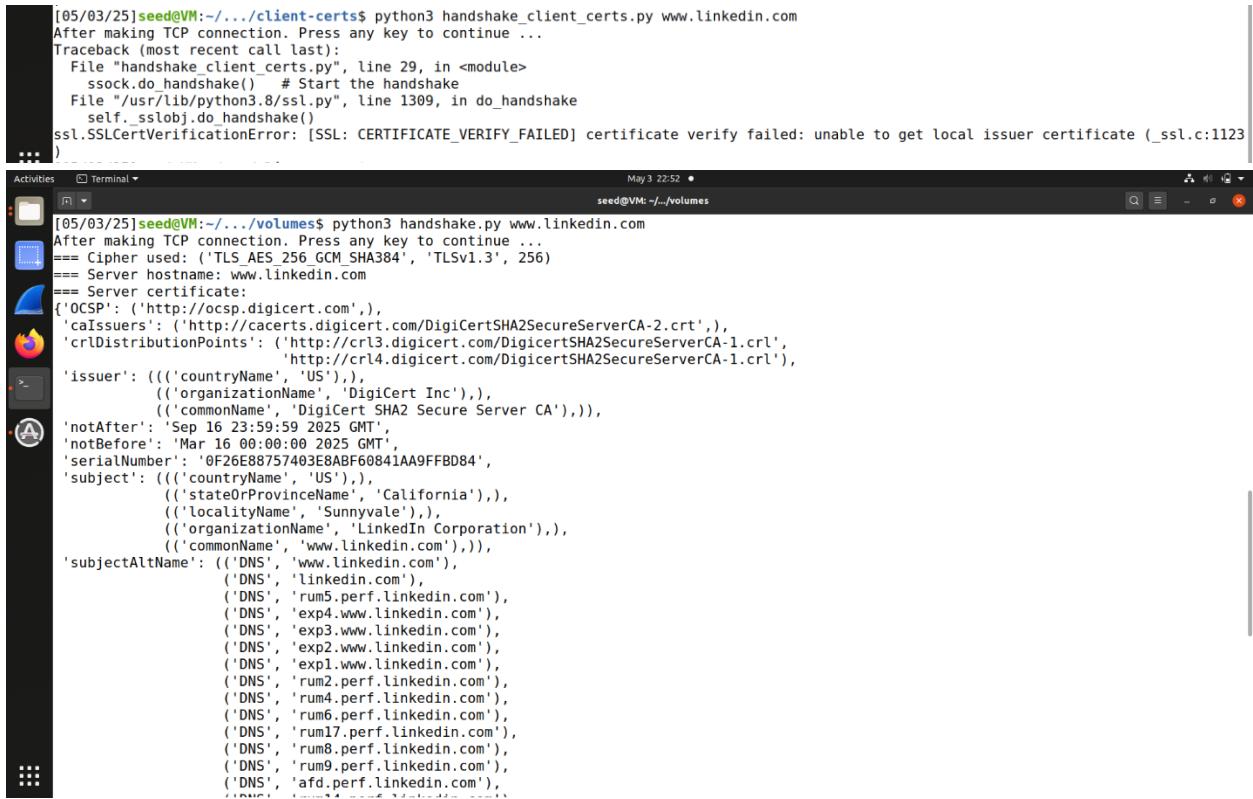
```
Activities Terminal May 3 22:52 • seed@VM: ~/volumes
[05/03/25]seed@VM:~/.../volumes$ python3 handshake.py www.github.com
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
==> Server hostname: www.github.com
==> Server certificate:
{'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': 'AB6686B5627BE805968213208649F5',
            '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': '5C8B99C55A94C5D271560ED8980CC26',
            '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](http://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-r--r-- 1 seed seed 103 Jan 2 2021 README.md
-rw-rw-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$
```



```
[05/03/25]seed@VM:~/.../client-certs$ python3 handshake_client_certs.py www.linkedin.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake_client_certs.py", line 29, in <module>
    sssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)
[05/03/25]seed@VM:~/.../volumes$ python3 handshake.py www.linkedin.com
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.linkedin.com
==> Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2SecureServerCA-2.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertSHA2SecureServerCA-1.crl',
                           'http://crl4.digicert.com/DigiCertSHA2SecureServerCA-1.crl'),
 'issuer': (((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('commonName', 'DigiCert SHA2 Secure Server CA'))),
            'notAfter': 'Sep 16 23:59:59 2025 GMT',
            'notBefore': 'Mar 16 00:00:00 2025 GMT',
            'serialNumber': '0F26E88757403E8ABF60841AA9FFBD84',
            'subject': (((('countryName', 'US'),),
                         (('stateOrProvinceName', 'California'),),
                         (('localityName', 'Sunnyvale'),),
                         (('organizationName', 'LinkedIn Corporation'),),
                         (('commonName', 'www.linkedin.com'))),
            'subjectAltName': (('DNS', 'www.linkedin.com'),
                               ('DNS', 'linkedin.com'),
                               ('DNS', 'rum5.perf.linkedin.com'),
                               ('DNS', 'exp4.www.linkedin.com'),
                               ('DNS', 'exp3.www.linkedin.com'),
                               ('DNS', 'exp2.www.linkedin.com'),
                               ('DNS', 'exp1.www.linkedin.com'),
                               ('DNS', 'rum2.perf.linkedin.com'),
                               ('DNS', 'rum4.perf.linkedin.com'),
                               ('DNS', 'rum6.perf.linkedin.com'),
                               ('DNS', 'rum17.perf.linkedin.com'),
                               ('DNS', 'rum8.perf.linkedin.com'),
                               ('DNS', 'rum9.perf.linkedin.com'),
                               ('DNS', 'afd.perf.linkedin.com'))..
```

**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.

```

[05/01/25] seed@VM:~/Labsetup_P4$ dig www.google.com
; <>> DiG 9.18.30-0ubuntu0.20.04.2-Ubuntu <>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 43061
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.google.com. IN A
;; ANSWER SECTION:
www.google.com. 79 IN A 142.251.16.104
www.google.com. 79 IN A 142.251.16.105
www.google.com. 79 IN A 142.251.16.147
www.google.com. 79 IN A 142.251.16.106
www.google.com. 79 IN A 142.251.16.99
www.google.com. 79 IN A 142.251.16.103
;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Thu May 01 23:35:04 EDT 2025
;; MSG SIZE rcvd: 139
[05/01/25] seed@VM:~/Labsetup_P4$

```

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](http://www.google2025.com)

```

Activities Terminal May 1 23:52
seed@VM: ~/volumes /etc/hosts
GNU nano 4.8
127.0.0.1 localhost
127.0.1.1 VM

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80 www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5 www.SeedLabSQLInjection.com

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

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

# For Shellshock Lab
10.9.0.80 www.seedlab-shellshock.com
142.251.16.104 www.google2025.com

[ Read 34 lines ]

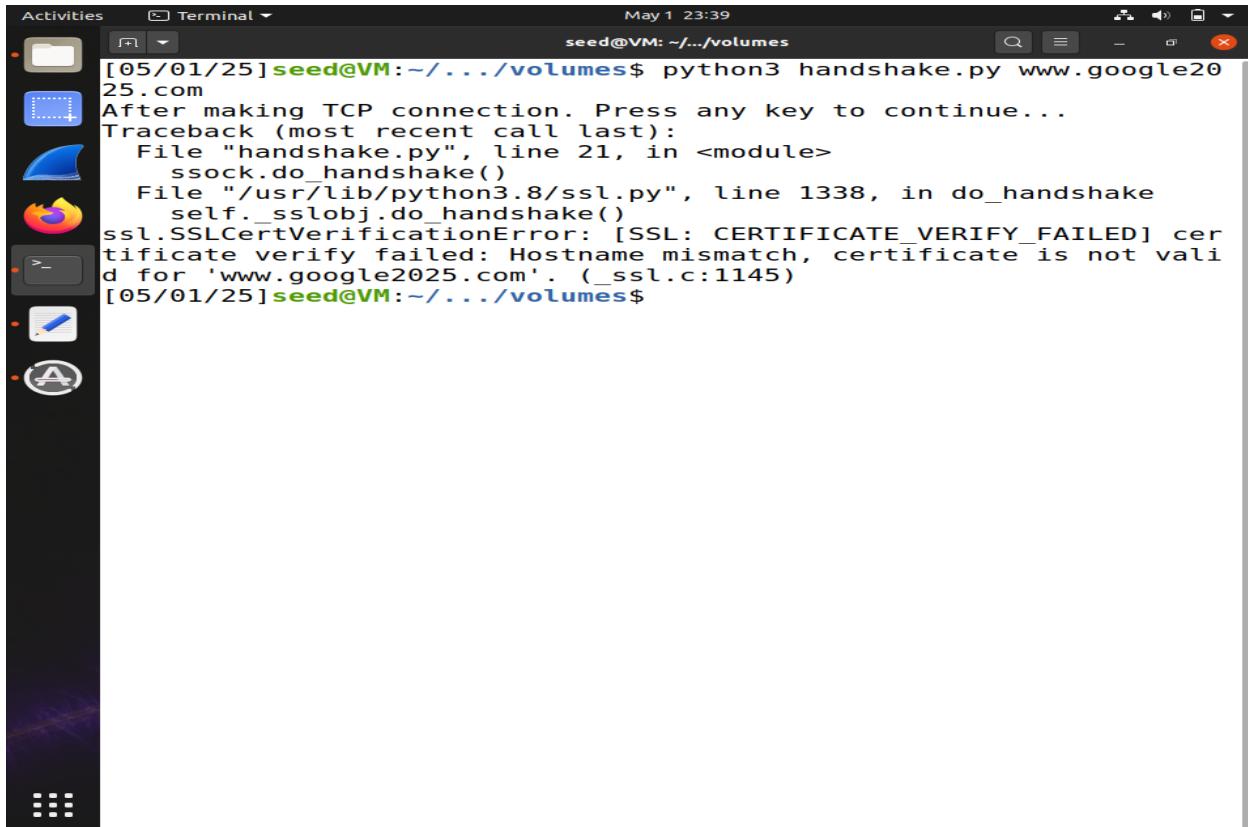
```

Bottom status bar:

- Get Help
- Write Out
- Where Is
- Cut Text
- Justify
- Cur Pos
- Undo
- Mark Text
- To Bracket
- Exit
- Read File
- Replace
- Paste Text
- To Spell
- Go To Line
- Redo
- Copy Text
- Where Was

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.

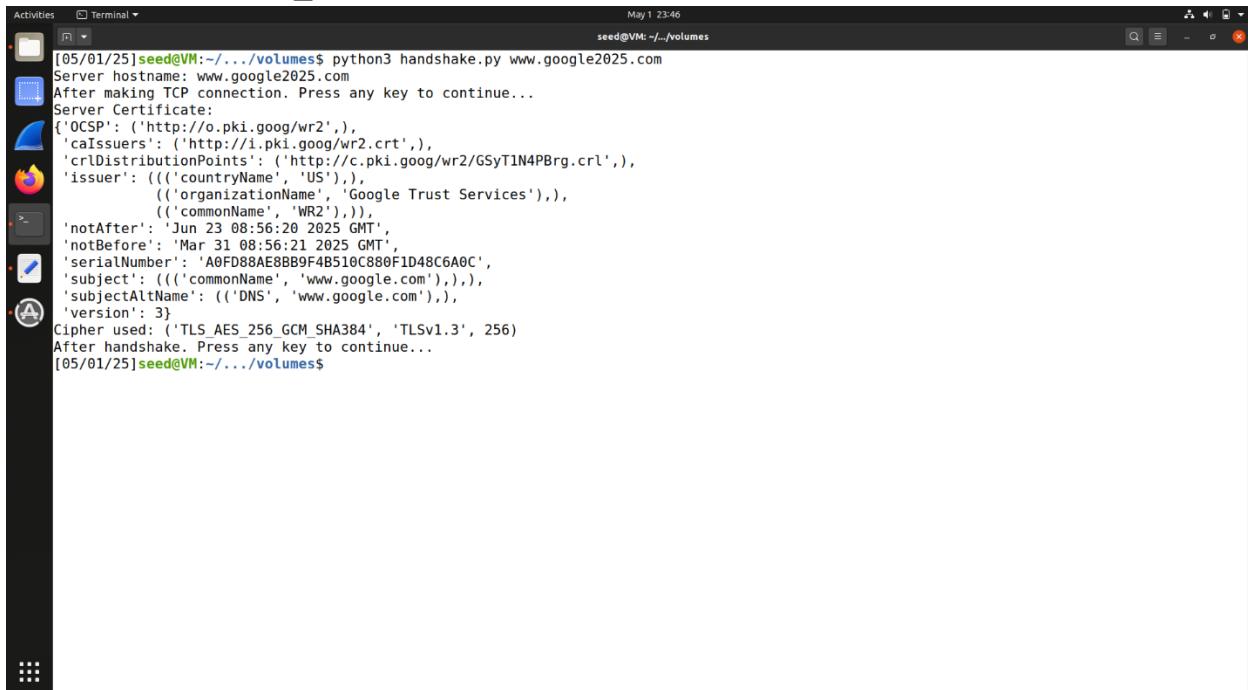
When context.check\_hostname = True



The screenshot shows a terminal window titled "Terminal" with the command "python3 handshake.py www.google2025.com" run by user "seed". The output indicates a TCP connection was made and a key was pressed to continue. A detailed traceback follows, showing the code flow from "handshake.py" to "ssl.py", specifically line 1338 of "ssl.py". The error is an "ssl.SSLCertVerificationError" with the message "[SSL: CERTIFICATE\_VERIFY\_FAILED] certificate verify failed: Hostname mismatch, certificate is not valid for 'www.google2025.com'. (\_ssl.c:1145)". The terminal window is part of a desktop environment with a sidebar containing icons for various applications like a file manager, terminal, and system settings.

```
Activities Terminal May 1 23:39
seed@VM: ~/.../volumes
[05/01/25] seed@VM:~/.../volumes$ python3 handshake.py www.google2025.com
After making TCP connection. Press any key to continue...
Traceback (most recent call last):
  File "handshake.py", line 21, in <module>
    ssock.do_handshake()
  File "/usr/lib/python3.8/ssl.py", line 1338, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: Hostname mismatch, certificate is not valid for 'www.google2025.com'. (_ssl.c:1145)
[05/01/25] seed@VM:~/.../volumes$
```

## When context.check\_hostname = False



The screenshot shows a terminal window titled "Terminal" with the command "python3 handshake.py www.google2025.com". The output indicates a successful TLS handshake with Google's certificate, despite the mismatched hostname. The certificate details shown include the server's name, its common name, and its expiration date.

```
[05/01/25]seed@VM:~/.../volumes$ python3 handshake.py www.google2025.com
Server hostname: www.google2025.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': 'A0FD88AE8BB89F4B510C880F1D48C6A0C',
              '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$
```

## 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.

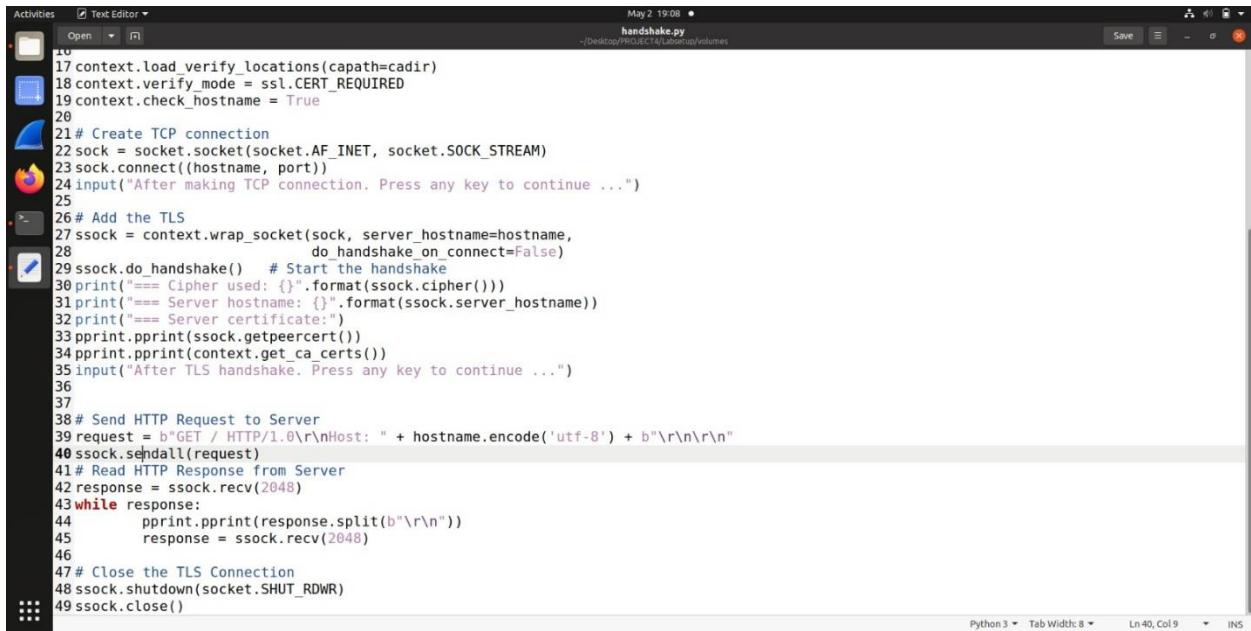
**Q) The importance of hostname check: Based on this experiment, please explain the importance of hostname check. If the client program does not perform the hostname check, what is the security consequence? Please explain.**

**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.**



```
Activities Text Editor May 2 19:08 • handshake.py -/Desktop/PROJECT4/LabSetup/volumes
Open F Save E X
10 context.load_verify_locations(capath=cadir)
11 context.verify_mode = ssl.CERT_REQUIRED
12 context.check_hostname = True
13
14# Create TCP connection
15sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16sock.connect((hostname, port))
17input("After making TCP connection. Press any key to continue ...")
18
19# Add the TLS
20ssock = context.wrap_socket(sock, server_hostname=hostname,
21                           do_handshake_on_connect=False)
22ssock.do_handshake() # Start the handshake
23print("==> Cipher used: {}".format(ssock.cipher()))
24print("==> Server hostname: {}".format(ssock.server_hostname))
25print("==> Server certificate:")
26pprint.pprint(ssock.getpeercert())
27pprint.pprint(context.get_ca_certs())
28input("After TLS handshake. Press any key to continue ...")
29
30# Send HTTP Request to Server
31request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
32ssock.sendall(request)
33# Read HTTP Response from Server
34response = ssock.recv(2048)
35while response:
36    pprint.pprint(response.split(b"\r\n"))
37    response = ssock.recv(2048)
38
39# Close the TLS Connection
40ssock.shutdown(socket.SHUT_RDWR)
41ssock.close()
```

## Observing the Results

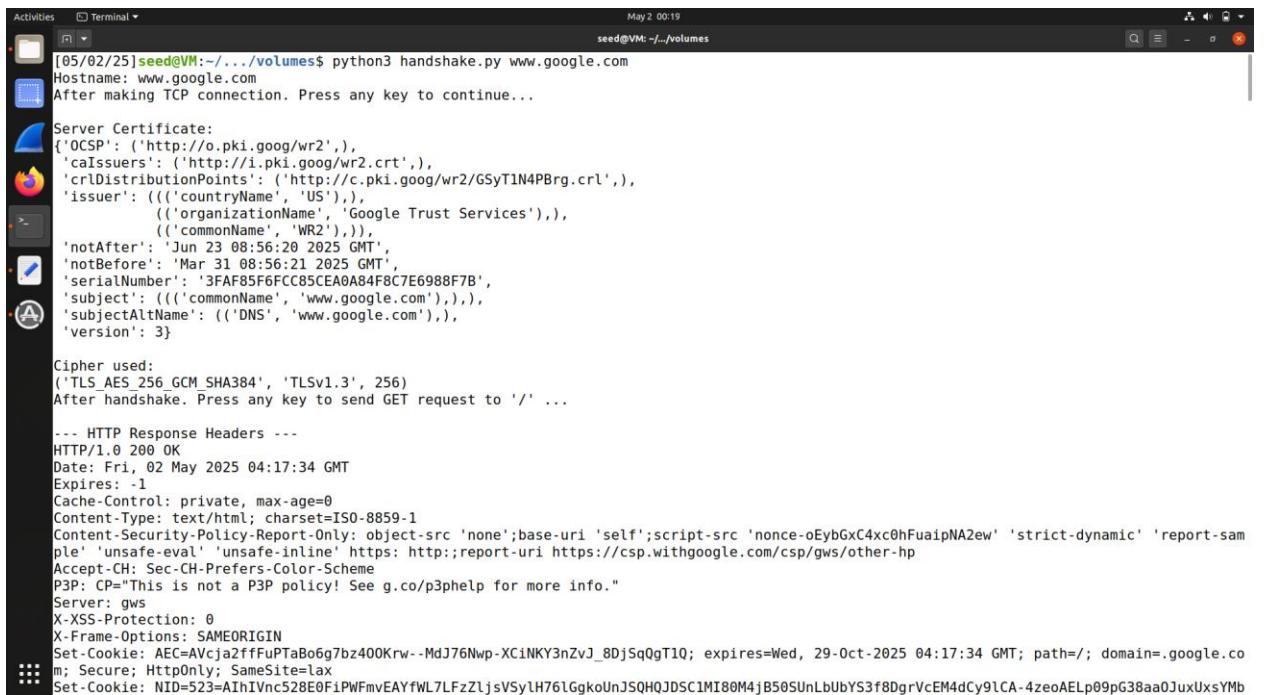
When I executed the program, the following observations were made:

1. Connection Established:
  - o The program successfully established a TCP connection to the specified server ([www.google.com](http://www.google.com)) and created a secure SSL/TLS connection.
2. HTTP Request Sent:
  - o 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.

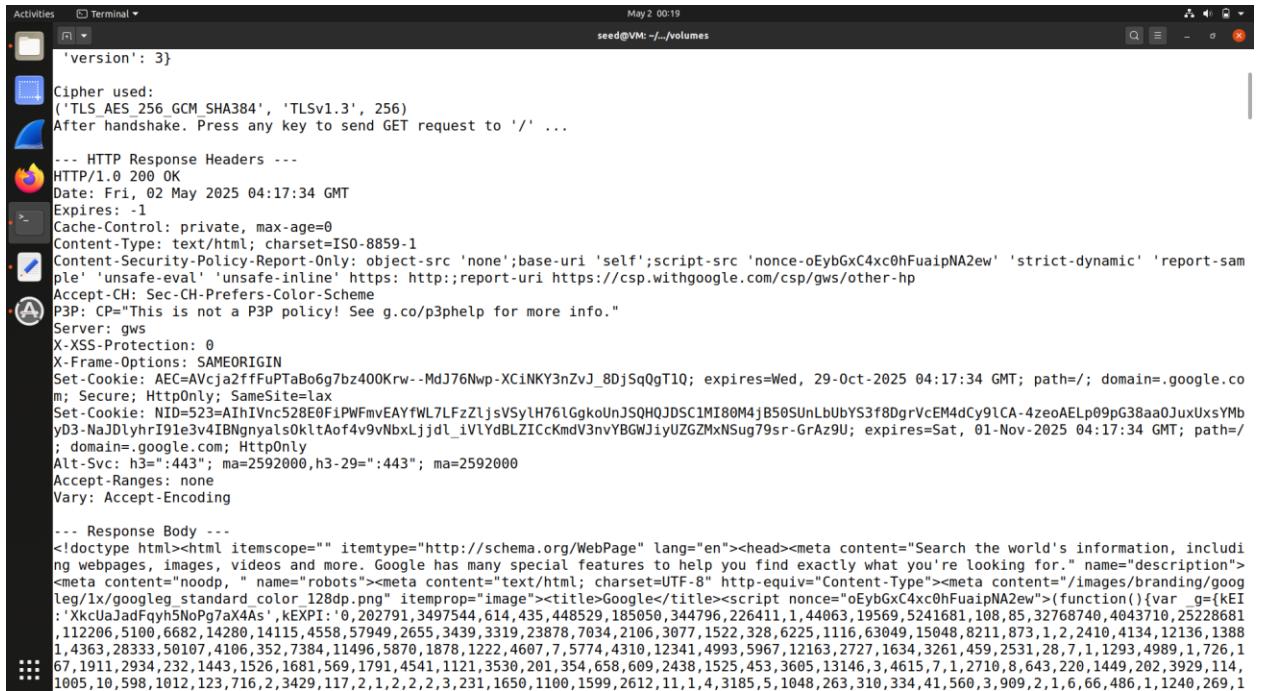


```
[05/02/25]seed@VM:~/.../volumes$ python3 handshake.py www.google.com
Hostname: www.google.com
After making TCP connection. Press any key to continue...

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 send GET request to '/' ...

--- HTTP Response Headers ---
HTTP/1.0 200 OK
Date: Fri, 02 May 2025 04:17:34 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-oEybGx4xc0hFuapNA2ew' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http://report-uri https://csp.withgoogle.com/csp/gws/other-hp
Accept-CH: Sec-CH-Prefers-Color-Scheme
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AVcja2ffFuPTaBo6g7bz400Krw--MdJ76Nwp-XCiNKY3nZvJ_8DjSgQgT1Q; expires=Wed, 29-Oct-2025 04:17:34 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=523=AIhIVnc528E0F1PWFmvEAYfWL7LFzZljsVSylH76lgkoUnJSQHQJDSC1MI80M4jB50SUlnLbUbYS3f8DgrVcEM4dCy9lCA-4zeoAELp09pG38aa0JuxUxsYMb
```



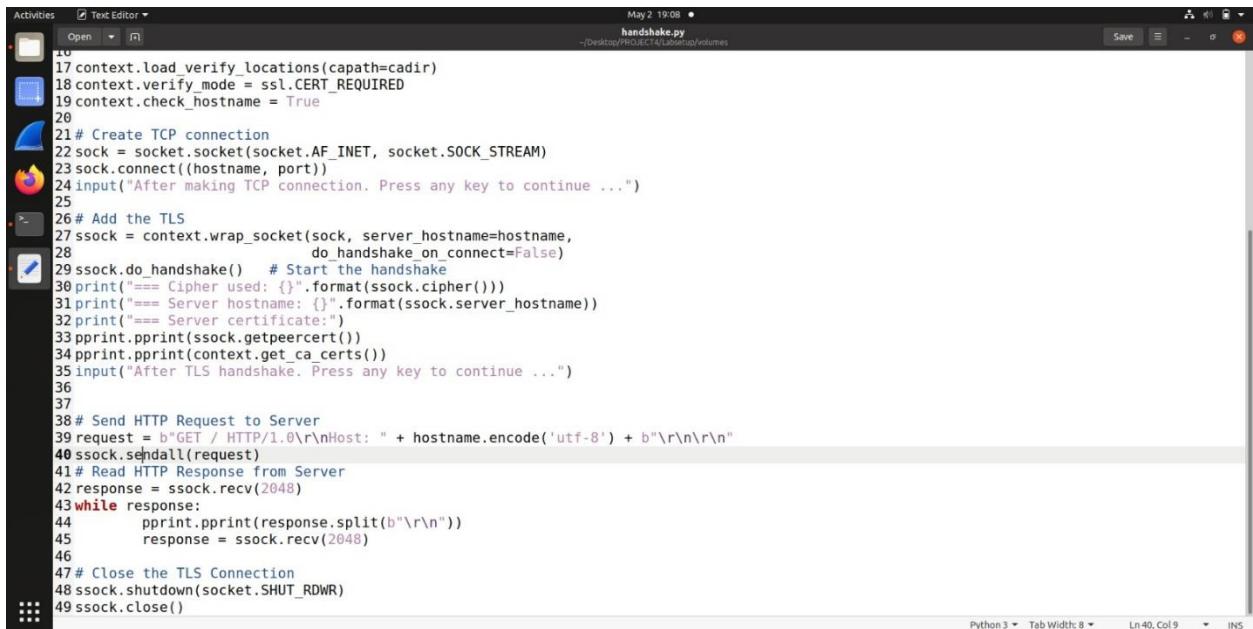
```

Activities Terminal May 2 00:19
seed@VM: ~/volumes

'session': 3}
Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
After handshake. Press any key to send GET request to '/' ...
--- HTTP Response Headers ---
HTTP/1.0 200 OK
Date: Fri, 02 May 2025 04:17:34 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-oEybGxC4xc0hFuaipNet' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http://report-uri https://csp.withgoogle.com/csp/gws/other-hp
Accept-CH: Sec-CH-Prefers-Color-Scheme
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AVcja2ffFuPTaB06g7bz400Krw-MdJ76Nwp-XC1NKY3nZvJ_8DjsQgTl0; expires=Wed, 29-Oct-2025 04:17:34 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=523=AIhIVnc528E0F1PWFmvEAYfWL7LFzZljsVSYLH761GgkoUnJSQHQJDSC1MI80M4jB50SUlbUbYS3f8DgrVcEM4dCy9lCA-4zeoAEIp09pG38aa0JuxUxsYMbzb3-NaJDlyhrI91e3v4IBNgnyslsOkltAof4v9vNbxEjjdl_iVLYdBLZICcKmdV3nvYBGWJiyUZGZMxNSug79sr-GrAzU; expires=Sat, 01-Nov-2025 04:17:34 GMT; path=/; domain=.google.com; HttpOnly
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Accept-Ranges: none
Vary: Accept-Encoding

--- Response Body ---
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noop, " name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleleg/1x/googleleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="oEybGxC4xc0hFuaipNet">(function(){var _g={kEI:'XkcUaJadFqyh5NoPg7aX4As',kEXPI:'0,202791,3497544,614,435,448529,185050,344796,226411,1,44063,19569,5241681,108,85,32768740,4043710,252286811,112206,5100,6682,14280,14115,4558,57949,2655,3439,3319,23878,7034,2106,3077,1522,328,6225,1116,63049,15048,8211,873,1,2,2410,4134,12136,13881,1,4363,28333,50107,4106,352,7384,11496,5870,1878,1222,4607,7,5774,4310,12341,4993,5967,12163,2727,1634,3261,459,2531,28,7,1,1293,4989,1,726,167,1911,2934,232,1443,1526,1681,569,1791,4541,1121,3530,201,354,658,609,2438,1525,453,3605,13146,3,4615,7,1,2710,8,643,220,1449,202,3929,114,1005,10,598,1012,123,716,2,3429,117,2,1,2,2,2,3,231,1650,1100,1599,2612,11,1,4,3185,5,1048,263,310,334,41,560,3,909,2,1,6,66,486,1,1240,269,1
...[redacted]
```

**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).**



```

Activities Text Editor May 2 19:08
handshake.py
~/Desktop/PycharmProjects/volumes

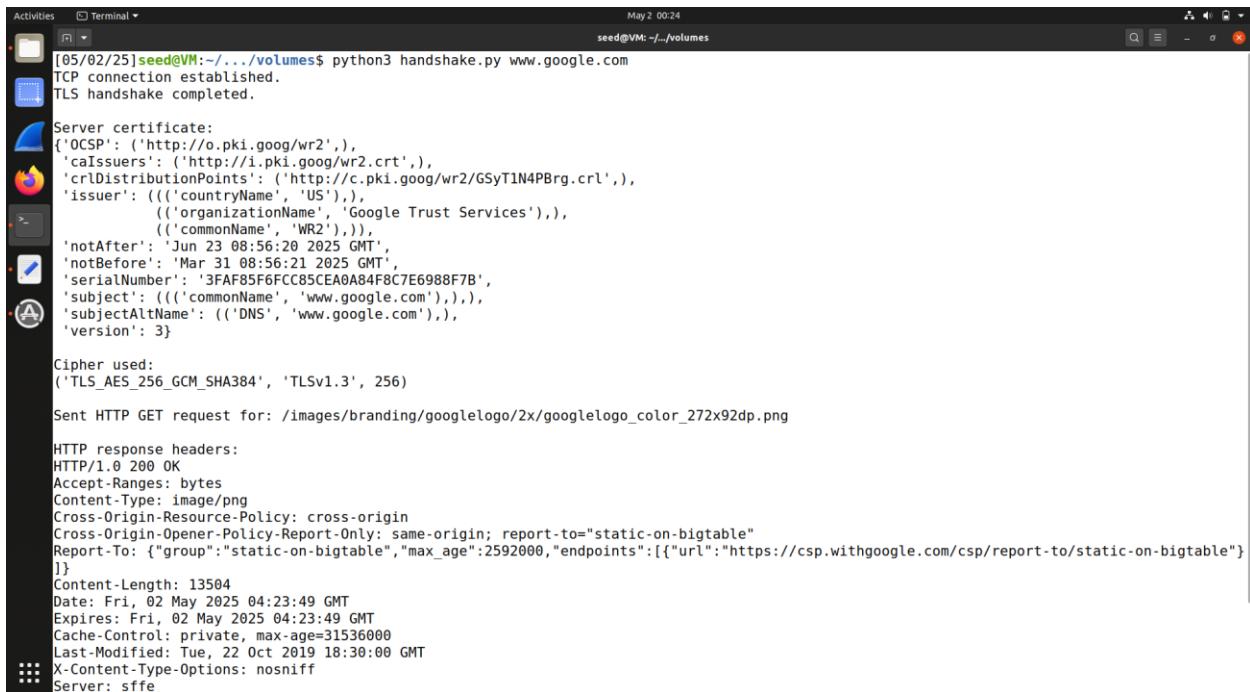
10 context.load_verify_locations(capath=cadir)
11 context.verify_mode = ssl.CERT_REQUIRED
12 context.check_hostname = True
13
14 # Create TCP connection
15 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16 sock.connect((hostname, port))
17 input("After making TCP connection. Press any key to continue ...")
18
19 # Add the TLS
20 ssock = context.wrap_socket(sock, server_hostname=hostname,
21                             do_handshake_on_connect=False)
22 ssock.do_handshake() # Start the handshake
23 print("==> Cipher used: {}".format(ssock.cipher()))
24 print("==> Server hostname: {}".format(ssock.server_hostname))
25 print("==> Server certificate:")
26 pprint.pprint(ssock.getpeercert())
27 pprint.pprint(context.get_ca_certs())
28 input("After TLS handshake. Press any key to continue ...")
29
30 # Send HTTP Request to Server
31 request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
32 ssock.sendall(request)
33 # Read HTTP Response from Server
34 response = ssock.recv(2048)
35 while response:
36     pprint.pprint(response.split(b"\r\n"))
37     response = ssock.recv(2048)
38
39 # Close the TLS Connection
40 ssock.shutdown(socket.SHUT_RDWR)
41 ssock.close()
42
```

**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.



The screenshot shows a terminal window on a Linux desktop environment. The title bar says "Activities Terminal" and the status bar indicates "May 2 00:24 seed@VM: ~/volumes". The terminal command is "[05/02/25] seed@VM:~/volumes\$ python3 handshake.py www.google.com". The output shows the program establishing a TCP connection, performing a TLS handshake, and then displaying the server certificate details. It includes the OCSP endpoint, certificate issuers, CRL distribution points, issuer information, validity period, serial number, subject, subject alternate names, and version. The cipher used is TLS\_AES\_256\_GCM\_SHA384. The program then sends an HTTP GET request for the Google logo image at /images/branding/googlelogo/2x/googlelogo\_color\_272x92dp.png. The response headers are shown, including the Content-Type as image/png, and various security and reporting headers like Cross-Origin-Resource-Policy, Cross-Origin-Opener-Policy-Report-Only, Report-To, and Cache-Control. The response body contains the image data.

```
[05/02/25] seed@VM:~/volumes$ python3 handshake.py www.google.com
TCP connection established.
TLS handshake completed.

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': '3FAF85F6FC85CEA0A84F8C7E6988F7B',
 'subject': (('commonName', 'www.google.com'),),
 'subjectAltName': ('DNS', 'www.google.com'),
 'version': 3}

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

Sent HTTP GET request for: /images/branding/googlelogo/2x/googlelogo_color_272x92dp.png

HTTP response headers:
HTTP/1.0 200 OK
Accept-Ranges: bytes
Content-Type: image/png
Cross-Origin-Resource-Policy: cross-origin
Cross-Origin-Opener-Policy-Report-Only: same-origin; report-to="static-on-bigtable"
Report-To: {"group":"static-on-bigtable","max_age":2592000,"endpoints":[{"url":"https://csp.withgoogle.com/csp/report-to/static-on-bigtable"}]}
Content-Length: 13504
Date: Fri, 02 May 2025 04:23:49 GMT
Expires: Fri, 02 May 2025 04:23:49 GMT
Cache-Control: private, max-age=31536000
Last-Modified: Tue, 22 Oct 2019 18:30:00 GMT
X-Content-Type-Options: nosniff
Server: sffe
```

```
Activities Terminal May 2 00:24
seed@VM: ~/volumes
'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': '3FAF85F6FC85CEA0A84F8C7E6988F7B',
            'subject': ((('commonName', 'www.google.com'),),
            'subjectAltName': ((('DNS', 'www.google.com'),),
            'version': 3}

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

[A] Sent HTTP GET request for: /images/branding/googlelogo/2x/googlelogo_color_272x92dp.png

HTTP response headers:
HTTP/1.0 200 OK
Accept-Ranges: bytes
Content-Type: image/png
Cross-Origin-Resource-Policy: cross-origin
Cross-Origin-Opener-Policy-Report-Only: same-origin; report-to="static-on-bigtable"
Report-To: {"group":"static-on-bigtable","max_age":2592000,"endpoints":[{"url":"https://csp.withgoogle.com/csp/report-to/static-on-bigtable"}]}
Content-Length: 13504
Date: Fri, 02 May 2025 04:23:49 GMT
Expires: Fri, 02 May 2025 04:23:49 GMT
Cache-Control: private, max-age=31536000
Last-Modified: Tue, 22 Oct 2019 18:30:00 GMT
X-Content-Type-Options: nosniff
Server: sffe
X-XSS-Protection: 0
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

Image saved as 'google_logo.png'
[05/02/25]seed@VM:~/volumes$
```

## Observation of Results

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

1. Connection Established:
  - o The program successfully established a secure HTTPS connection with the specified server (www.google.com).
2. Modified HTTP Request Sent:
  - o 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:
  - o 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:
  - o 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', 'neeraj'),),
    (('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'),)),
```

```

'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', 'neeraj'),),
              (('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'),),
              (('organizationName', 'GMU'),),
              (('organizationalUnitName', 'CS'),),
              (('commonName', 'neeraj'),),
              (('emailAddress', 'nkarnati@gmu.edu'))),
'version': 3}]
==== Finished TLS handshake. Press any key to continue ...

==== HTTP Headers ====
HTTP/1.1 200 OK
Date: Mar 24 2025 GMT
Content-Type: text/html
Content-Length: 102
Connection: close
Downloaded content saved as 'downloaded_file.bin'.
[05/03/25]seed@VM:~/.../volumes$ █

```

## 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.

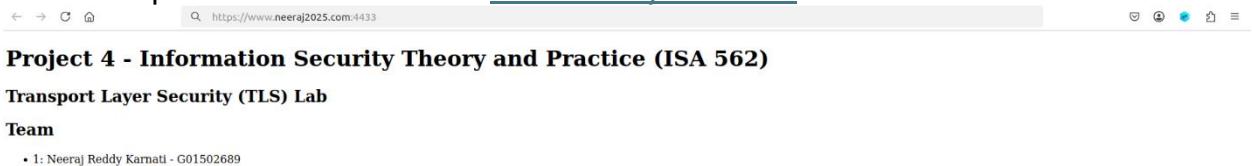
## Overall Observation:

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](https://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

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

### **Transport Layer Security (TLS) Lab**

#### **Team**

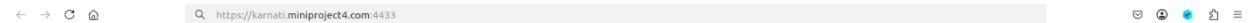
- 1: Neeraj Reddy Karnati - G01502689

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

### **Transport Layer Security (TLS) Lab**

#### **Team**

- 1: Neeraj Reddy Karnati - G01502689

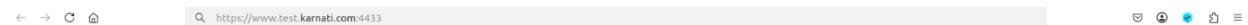


## Project 4 - Information Security Theory and Practice (ISA 562)

### Transport Layer Security (TLS) Lab

#### Team

- 1: Neeraj Reddy Karnati - G01502689



## 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.