SIL765: Networks and System Security
Assignment-5 Report

Rohit Patidar
Roll No: 2024JCS2042
IIT Delhi

April 20, 2025

Problem 1: Transport Layer Security (40 marks)

Task 1: TLS Handshake

1. **Cipher used between client and server:**

```
rohit@rohit:~/Desktop/assignment_5_SIL765$ python3 tls_client.py task_1 www.google.com
TCP connection established. Press Enter to start TLS handshake...
=== Cipher Used ===
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
```

Figure 1: Cipher used between client and server

2. **Server certificate output:**

3. **Purpose of /etc/ssl/certs:**

   This directory serves as the system-wide trust store for Certificate Authority (CA).

   - Stores root and intermediate CA certificates used by TLS clients to verify server certificates.
   - Enables validation of a server's certificate chain during the TLS handshake.
   - Protects against man-in-the-middle and spoofing attacks by ensuring the server's identity is authenticated by a trusted CA.

```
rohit@rohit:~/Desktop/assignment_5_SIL765$ python3 tls_client.py task_1 www.google.com
TCP connection established. Press Enter to start TLS handshake...
=== Cipher Used ===
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)

=== 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}
TLS handshake complete. Press Enter to close connection...
rohit@rohit:~/Desktop/assignment_5_SIL765$
```

Figure 2: Server's Certificate

4. **Wireshark capture observations:**
   The capture shows:

   - The TCP three-way handshake (SYN, SYN-ACK, ACK) is initiated when `sock.connect((hostname, port))` is called.Only the three-way handshake packets while using display filter..

```
(tcp.flags.syn == 1 && tcp.flags.ack == 0) ||
(tcp.flags.syn == 1 && tcp.flags.ack == 1) ||
(tcp.flags.ack == 1 && tcp.flags.syn == 0 && tcp.len ==
   0)
```

   .All packet that include in 3 way hand shake is show in below figure

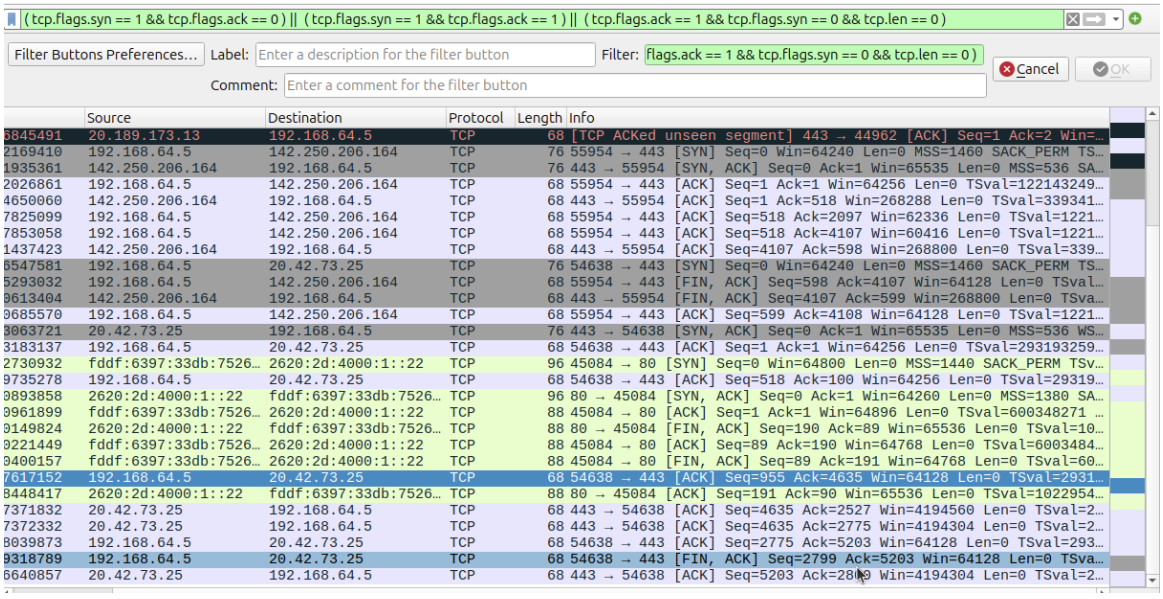
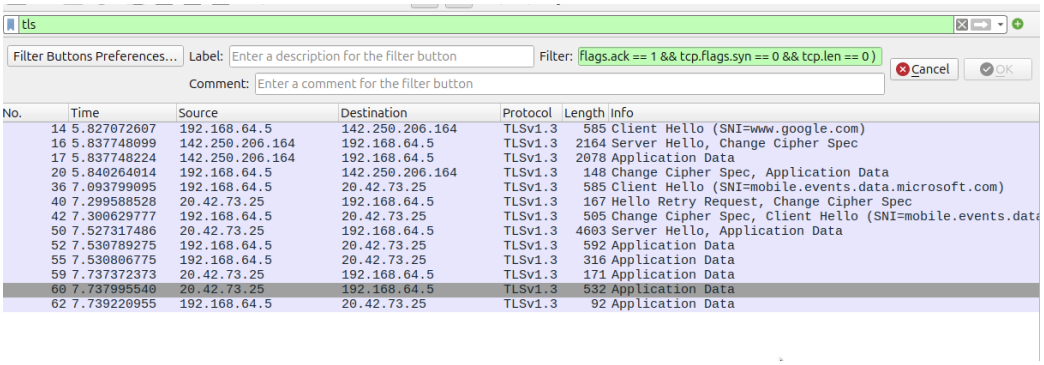


Figure 3: three-way handshake

   - The TLS handshake messages (ClientHello, ServerHello, Certificate exchange, etc.) begin when `ssock.do_handshake()` is executed.



- The TLS handshake occurs over the established TCP connection; without completion of the TCP handshake, the TLS handshake cannot start.

  The TLS handshake is an application-layer protocol that runs on top of an already-established TCP connection. In practice, this means:

1. TCP Three-Way Handshake

(a) The client sends a `SYN` packet.
(b) The server replies with `SYN{ACK`.
(c) The client completes with `ACK`.

Only once this exchange finishes do the endpoints have a reliable, ordered byte stream.

2. TLS Handshake

Over that TCP stream:

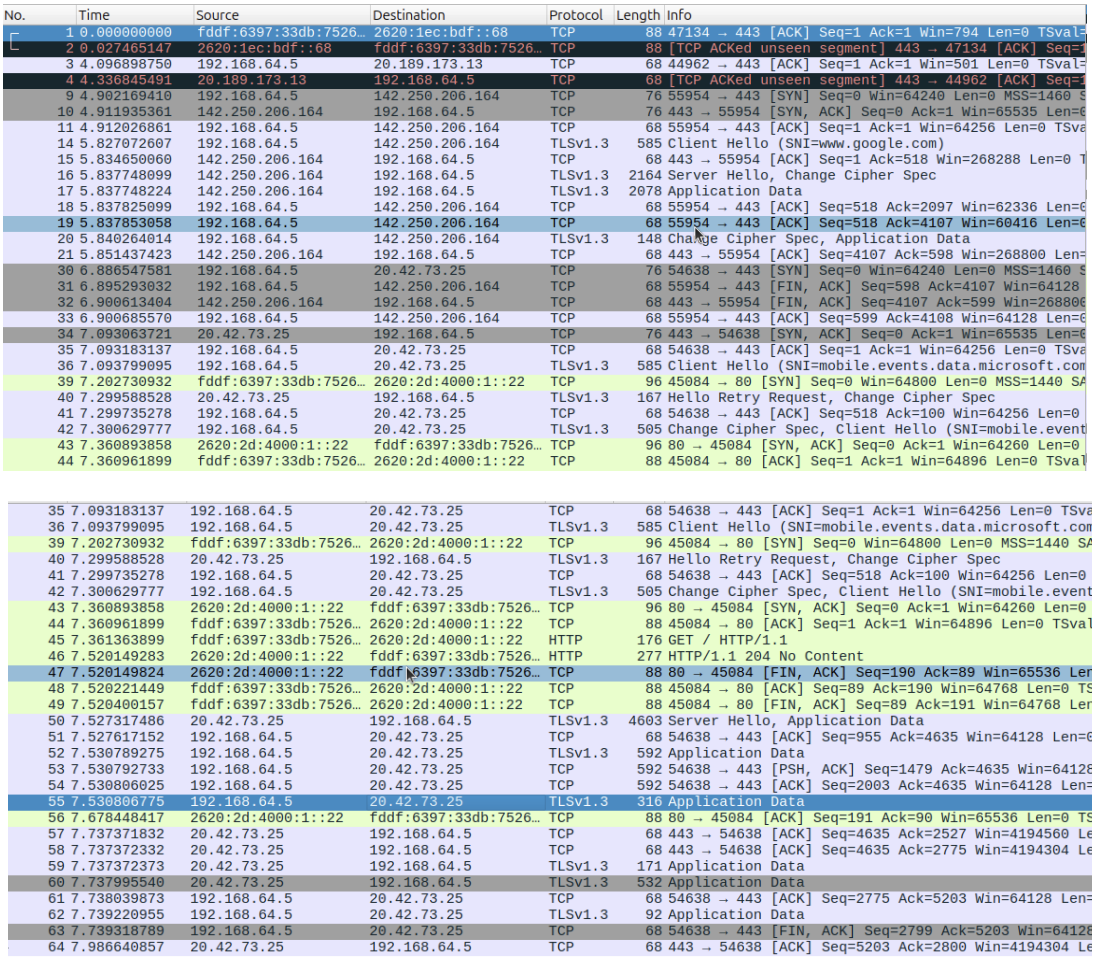
– The client sends a `ClientHello` message.
– The server replies with `ServerHello`, certificate messages, key-exchange messages, etc.

All of these cryptographic negotiation messages are simply framed as data carried by TCP.

Why It Matters

– TLS relies on TCP's reliability and in-order delivery to ensure its own handshake messages aren't lost or reordered.
– If the TCP handshake doesn't complete, the TLS handshake packets would never be delivered.
– Conversely, once the TLS handshake succeeds, the two sides can switch to sending encrypted application data over the same TCP connection.

In short: TCP sets up the channel; TLS then secures it.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fddf:6397:33db:7526…	2620:1ec:bdf::68	TCP	88	47134 → 443 [ACK] Seq=1 Ack=1 Win=794 Len=0 TSval=
2	0.027465147	2620:1ec:bdf::68	fddf:6397:33db:7526…	TCP	88	[TCP ACKed unseen segment] 443 → 47134 [ACK] Seq=
3	4.096898750	192.168.64.5	20.189.173.13	TCP	68	44962 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=
4	4.336845491	20.189.173.13	192.168.64.5	TCP	68	[TCP ACKed unseen segment] 443 → 44962 [ACK] Seq=
9	4.902169410	192.168.64.5	142.250.206.164	TCP	76	55954 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
10	4.911935361	142.250.206.164	192.168.64.5	TCP	76	443 → 55954 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
11	4.912026861	192.168.64.5	142.250.206.164	TCP	68	55954 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
14	5.827072607	192.168.64.5	142.250.206.164	TLSv1.3	585	Client Hello (SNI=www.google.com)
15	5.834650060	142.250.206.164	192.168.64.5	TCP	68	443 → 55954 [ACK] Seq=1 Ack=518 Win=268288 Len=0 T
16	5.837748099	142.250.206.164	192.168.64.5	TLSv1.3	2164	Server Hello, Change Cipher Spec
17	5.837748224	142.250.206.164	192.168.64.5	TLSv1.3	2078	Application Data
18	5.837825099	192.168.64.5	142.250.206.164	TCP	68	55954 → 443 [ACK] Seq=518 Ack=2097 Win=62336 Len=0
19	5.837853058	192.168.64.5	142.250.206.164	TCP	68	55954 → 443 [ACK] Seq=518 Ack=4107 Win=60416 Len=0
20	5.840264014	192.168.64.5	142.250.206.164	TLSv1.3	148	Change Cipher Spec, Application Data
21	5.851437423	142.250.206.164	192.168.64.5	TCP	68	443 → 55954 [ACK] Seq=4107 Ack=598 Win=268800 Len=
30	6.886547581	192.168.64.5	20.42.73.25	TCP	76	54638 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
31	6.895293032	192.168.64.5	142.250.206.164	TCP	68	55954 → 443 [FIN, ACK] Seq=598 Ack=4107 Win=64128
32	6.900613404	142.250.206.164	192.168.64.5	TCP	68	443 → 55954 [FIN, ACK] Seq=4107 Ack=599 Win=268806
33	6.900685570	192.168.64.5	142.250.206.164	TCP	68	55954 → 443 [ACK] Seq=599 Ack=4108 Win=64128 Len=0
34	7.093063721	20.42.73.25	192.168.64.5	TCP	76	443 → 54638 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
35	7.093183137	192.168.64.5	20.42.73.25	TCP	68	54638 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
36	7.093799095	192.168.64.5	20.42.73.25	TLSv1.3	585	Client Hello (SNI=mobile.events.data.microsoft.com
39	7.202730932	fddf:6397:33db:7526…	2620:2d:4000:1::22	TCP	96	45084 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 SA
40	7.299588528	20.42.73.25	192.168.64.5	TLSv1.3	167	Hello Retry Request, Change Cipher Spec
41	7.299735278	192.168.64.5	20.42.73.25	TCP	68	54638 → 443 [ACK] Seq=518 Ack=100 Win=64256 Len=0
42	7.300629777	192.168.64.5	20.42.73.25	TLSv1.3	505	Change Cipher Spec, Client Hello (SNI=mobile.event
43	7.360893858	2620:2d:4000:1::22	fddf:6397:33db:7526…	TCP	96	80 → 45084 [SYN, ACK] Seq=0 Ack=1 Win=64260 Len=0
44	7.360961899	fddf:6397:33db:7526…	2620:2d:4000:1::22	TCP	88	45084 → 80 [ACK] Seq=1 Ack=1 Win=64896 Len=0 TSval
45	7.361363899	fddf:6397:33db:7526…	2620:2d:4000:1::22	HTTP	176	GET / HTTP/1.1
46	7.520149283	2620:2d:4000:1::22	fddf:6397:33db:7526…	HTTP	277	HTTP/1.1 204 No Content
47	7.520149824	2620:2d:4000:1::22	fddf:6397:33db:7526…	TCP	88	80 → 45084 [FIN, ACK] Seq=190 Ack=89 Win=65536 Len
48	7.520221449	fddf:6397:33db:7526…	2620:2d:4000:1::22	TCP	88	45084 → 80 [ACK] Seq=89 Ack=190 Win=64768 Len=0 TS
49	7.520400157	fddf:6397:33db:7526…	2620:2d:4000:1::22	TCP	88	45084 → 80 [FIN, ACK] Seq=89 Ack=191 Win=64768 Len
50	7.527317486	20.42.73.25	192.168.64.5	TLSv1.3	4603	Server Hello, Application Data
51	7.527617152	192.168.64.5	20.42.73.25	TCP	68	54638 → 443 [ACK] Seq=955 Ack=4635 Win=64128 Len=0
52	7.530789275	192.168.64.5	20.42.73.25	TLSv1.3	592	Application Data
53	7.530792733	192.168.64.5	20.42.73.25	TCP	592	54638 → 443 [PSH, ACK] Seq=1479 Ack=4635 Win=64128
54	7.530806025	192.168.64.5	20.42.73.25	TCP	592	54638 → 443 [ACK] Seq=2003 Ack=4635 Win=64128 Len=
55	7.530806775	192.168.64.5	20.42.73.25	TLSv1.3	316	Application Data
56	7.678448417	2620:2d:4000:1::22	fddf:6397:33db:7526…	TCP	88	80 → 45084 [ACK] Seq=191 Ack=90 Win=65536 Len=0 TS
57	7.737371832	20.42.73.25	192.168.64.5	TCP	68	443 → 54638 [ACK] Seq=4635 Ack=2527 Win=4194560 Le
58	7.737372332	20.42.73.25	192.168.64.5	TCP	68	443 → 54638 [ACK] Seq=4635 Ack=2775 Win=4194304 Le
59	7.737372373	20.42.73.25	192.168.64.5	TLSv1.3	171	Application Data
60	7.737995540	20.42.73.25	192.168.64.5	TLSv1.3	532	Application Data
61	7.738039873	192.168.64.5	20.42.73.25	TCP	68	54638 → 443 [ACK] Seq=2775 Ack=5203 Win=64128 Len=
62	7.739220955	192.168.64.5	20.42.73.25	TLSv1.3	92	Application Data
63	7.739318789	192.168.64.5	20.42.73.25	TCP	68	54638 → 443 [FIN, ACK] Seq=2799 Ack=5203 Win=64128
64	7.986640857	20.42.73.25	192.168.64.5	TCP	68	443 → 54638 [ACK] Seq=5203 Ack=2800 Win=4194304 Le

Figure 4: Wireshark capture showing TCP handshake and TLS handshake

Task 2: CA's Certificate

1. **Description of error when using empty `./certs` folder:**

Role of CA Certificates

TLS uses a bundle of trusted Certificate Authority (CA) certificates to verify that a server's certificate is signed by a trusted authority. Normally, these certificates are stored in the system directory (e.g., `/etc/ssl/certs` on Ubuntu). When a client connects to a server using TLS, it uses these CA certificates as the basis for trusting the server's certificate.

Why an Empty Certificate Folder Fails

In the TLS client code, the line:

```
cadir = '/etc/ssl/certs'
```

points to the directory containing the trusted CA certificates. Changing this to a local folder (e.g., `./certs`) tells the client to load certificates from that folder. If

the folder is empty, there are no trusted CA certificates available. Consequently, the client cannot verify the server's certificate, causing the TLS handshake to fail with an error like:

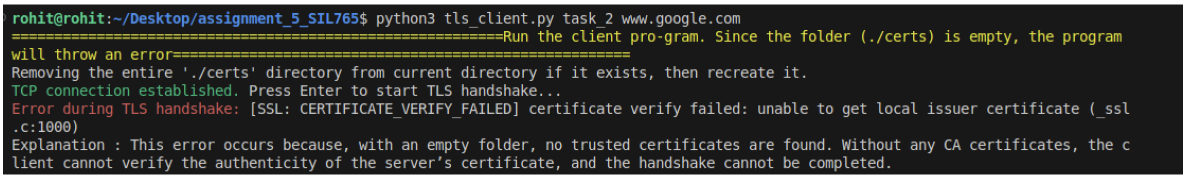


Figure 5: Error when using empty ./certs folder

This error signifies that the certificate chain could not be built and verified.

2. **Steps taken to resolve the issue:**
   There are two common ways to populate your local `./certs` folder with valid CA certificates:

   - **Option A: Copying from the System CA Directory**
     (a) **Copy CA Certificates:**

```
cp /etc/ssl/certs/* ./certs/
```

     (b) **Note:** This may copy many files you don't need. Some systems provide a combined bundle (e.g. `ca-certificates.crt`), but many applications expect individual certificate files.

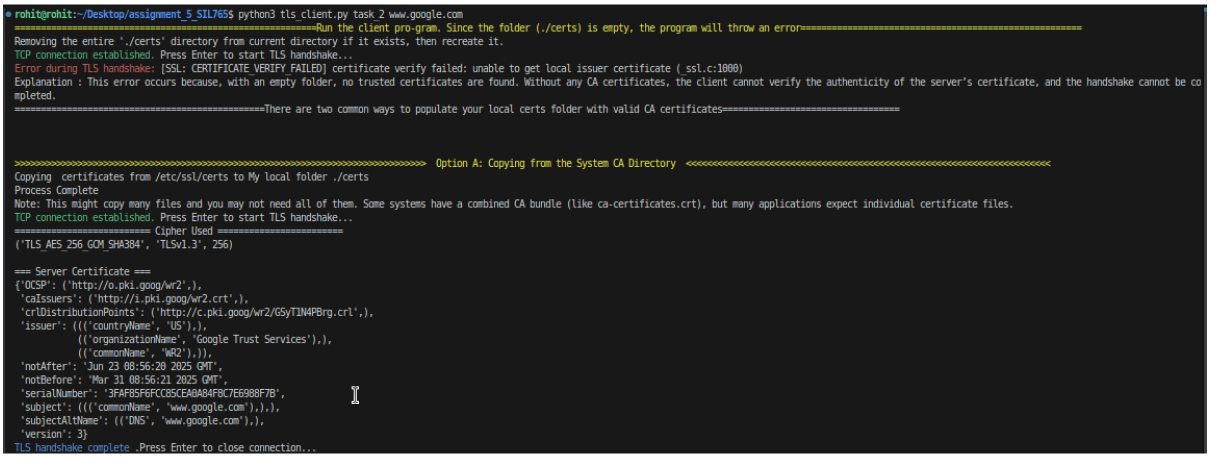


Figure 6: Option A: Copying from the System CA Directory

   - **Option B: Downloading a Trusted CA Bundle**
     (a) **Download a CA Bundle File:**

```
wget -O ./certs/cacert.pem https://curl.se/ca/cacert.
    pem
```

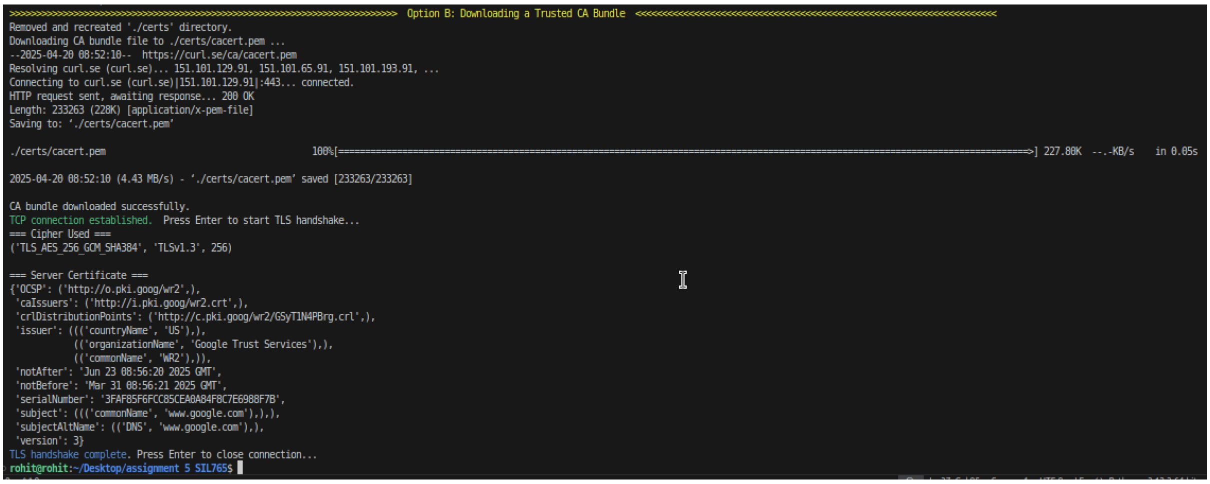


Figure 7: Option B: Downloading a Trusted CA Bundle

(b) **Note:** Depending on your SSL context, you may need either individual files or a single bundle.

3. **Observations after resolution:**

- With an empty folder, the handshake fails due to the inability to validate the server's certificate.
- With a populated folder, the handshake succeeds and certificate validation proceeds correctly.

Task 3: Hostname Check

1. IP address of server (output of `dig`):

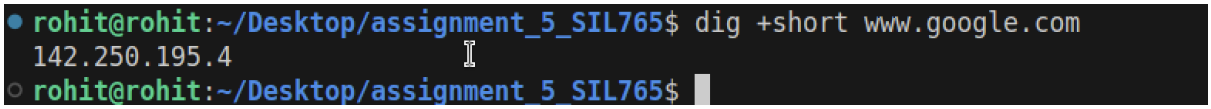


Figure 8: IP address of server (output of dig)

2. Contents added to /etc/hosts:

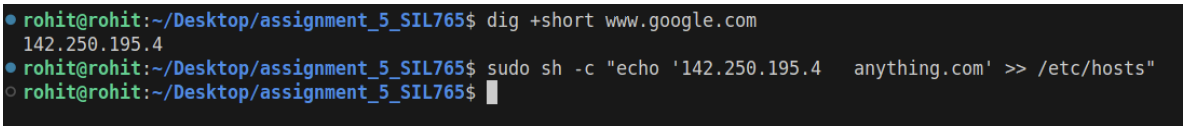


Figure 9: Contents added to /etc/hosts

3. Observations with `context.check_hostname = False`:

   - The handshake now succeeds (cipher and cert print out), because although the cert is for www.google.com, you've disabled the check that enforces matching names.

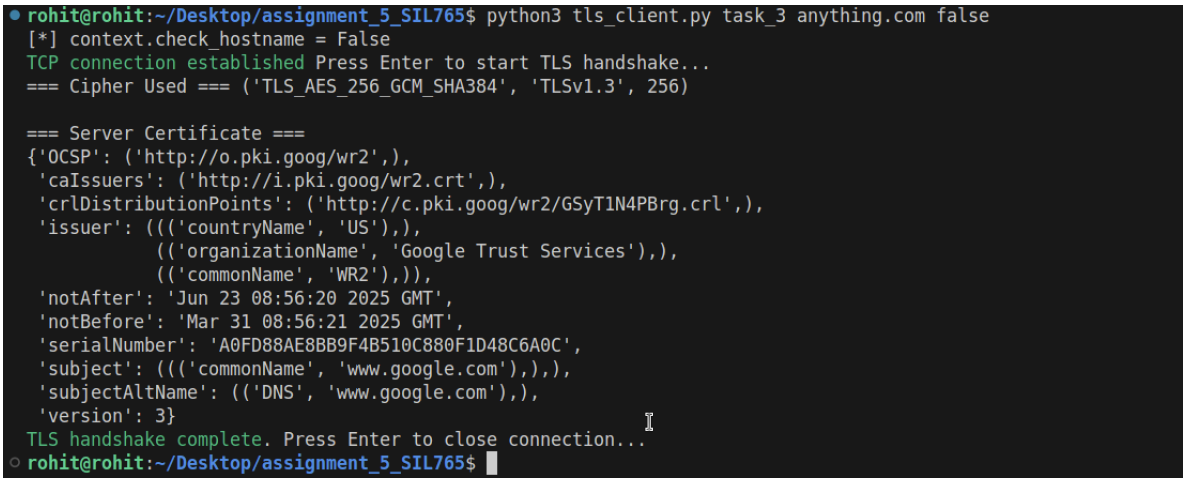


Figure 10: check hostname = false

.

4. Observations with `context.check_hostname = True`:

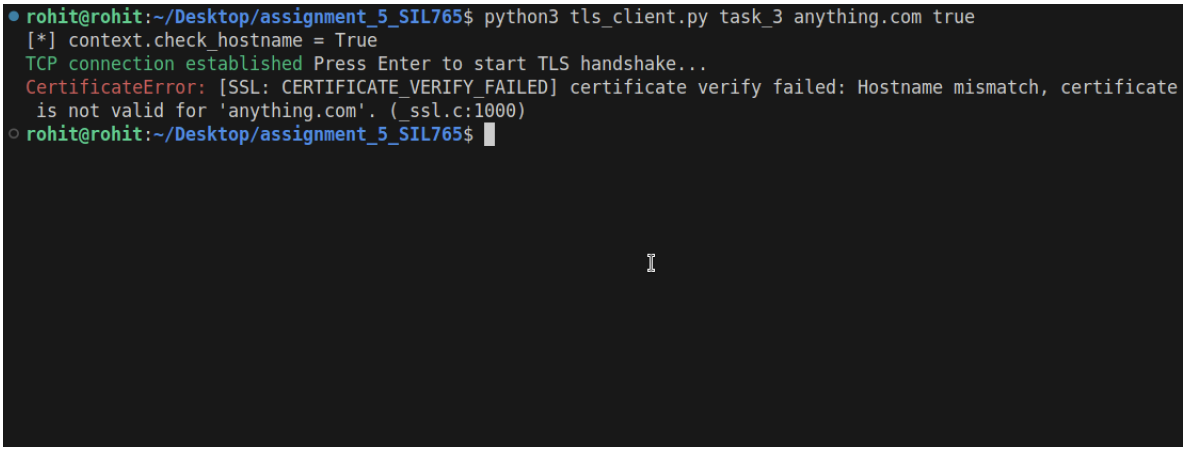


Figure 11: context.check_hostname = True

- The TLS handshake will fail with an error like:

5. Discussion on importance and security consequences:

Observations and Security Implications

Setting	Outcome	Security Consequen
`check_hostname = True`	Handshake **fails** with hostname mismatch	Protects against MIT cert is for the host you
`check_hostname = False`	Handshake **succeeds** despite mismatch	× Vulnerable to MITM with valid cert for *a* can impersonate

Key takeaway: Even if a certificate is valid (signed by a trusted CA), it must also be valid for the specific hostname you're connecting to. Disabling hostname checking opens the door to trivial man-in-the-middle attacks.

- **Identity Binding:** Hostname checking ensures the server's certificate is issued for the exact DNS name you intended, binding the CA trust chain to a specific endpoint.
- **MITM Protection:** Without it, an attacker with any valid certificate can impersonate your target host, intercepting or tampering with traffic undetected.
- **Complements CA Verification:** CA validation alone confirms a certificate is signed by a trusted authority, but hostname checking confirms it was signed for the correct service.
- **Production Best Practice:** Disabling hostname checks nullifies the security guarantees of TLS; always enable `verify_mode=CERT_REQUIRED` *and* `check_hostname=True` in real deployments.

Task 4: Communicating Data

1. Modified code for sending/receiving HTTP data:

```python
# Build and send HTTP request
req = f"GET {resource} HTTP/1.0\r\nHost: {hostname}\r\n\r\n"
ssock.sendall(req.encode('utf-8'))

# Read and pretty-print response
data = ssock.recv(2048)
while data:
    pprint.pprint(data.split(b"\r\n"))
    data = ssock.recv(2048)
```

2. Observation of HTTP response:
   The server responds first with a status line, for example:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1256
```

   A blank line (˚) separates the headers from the HTML body, which is streamed and printed line-by-line until the connection closes.

3. Modified request for fetching an image:

```python
# Request an image resource
resource = "/images/branding/googlelogo/2x/
   googlelogo_color_272x92dp.png"
req = f"GET {resource} HTTP/1.0\r\nHost: {hostname}\r\n\r\n"
ssock.sendall(req.encode('utf-8'))
```

```
rohit@rohit:~/Desktop/assignment_5_SIL765$ ls
certs  tls_client.py  wget-log
rohit@rohit:~/Desktop/assignment_5_SIL765$  python3 tls_client.py task_4 www.google.com
/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png
TCP connected. Press Enter to start TLS handshake...
[+] TLS handshake complete
[>] Sent HTTP GET /images/branding/googlelogo/2x/googlelogo_color_272x92dp.png
[+] Image saved as googlelogo_color_272x92dp.png
rohit@rohit:~/Desktop/assignment_5_SIL765$ ls
certs  googlelogo_color_272x92dp.png  tls_client.py  wget-log
```

Figure 12: Screenshot of the fetched image data response