Network Systems and Security

Assignment 5: Transport Layer security

Report Part 1

Course: SIL765 (Network Systems and Security)
Submission Date: 20 April 2025

Author: Anand Sharma
Entry no. : 2024JCS2049

Contents

1 Introduction

This report documents the implementation and analysis of a Python-based TLS client that performs TLS handshakes, certificate verification, hostname checking, and data exchange over HTTPS. Experiments ran on **Kali Linux** with **Python 3.13** and **Wireshark**.

2 Environment Setup

- **OS:** Kali Linux (WSL2)
- **Python:** 3.13
- **Wireshark:** 4.x
- **Script:** `tls_client.py` (functions: `ssl_context`, `task1`–`task4`)
- Run the command to run as : python tls_client.py task1

3 Task 1: TLS Handshake

3.1 Objective

Perform a TCP connection, manual TLS handshake, print server certificate and cipher, and capture handshakes in Wireshark.

3.2 Execution Details

- Before Running the program, start Wireshark to capture the TCP handshake.
- Run the client :

```
$ python3 tls_client.py home.iitd.ac.in task1
```

- Press a key to initiate the TLS handshake; Wireshark will now show `ClientHello`, `ServerHello`.

3.3 Observations

- Cipher suite: (`TLS_AES_128_GCM_SHA256` TLSv1.3, 256). This indicates the encryption algorithm, protocol version, and key size used in the session. Server Certificate and the cipher certificate used is marked by red box

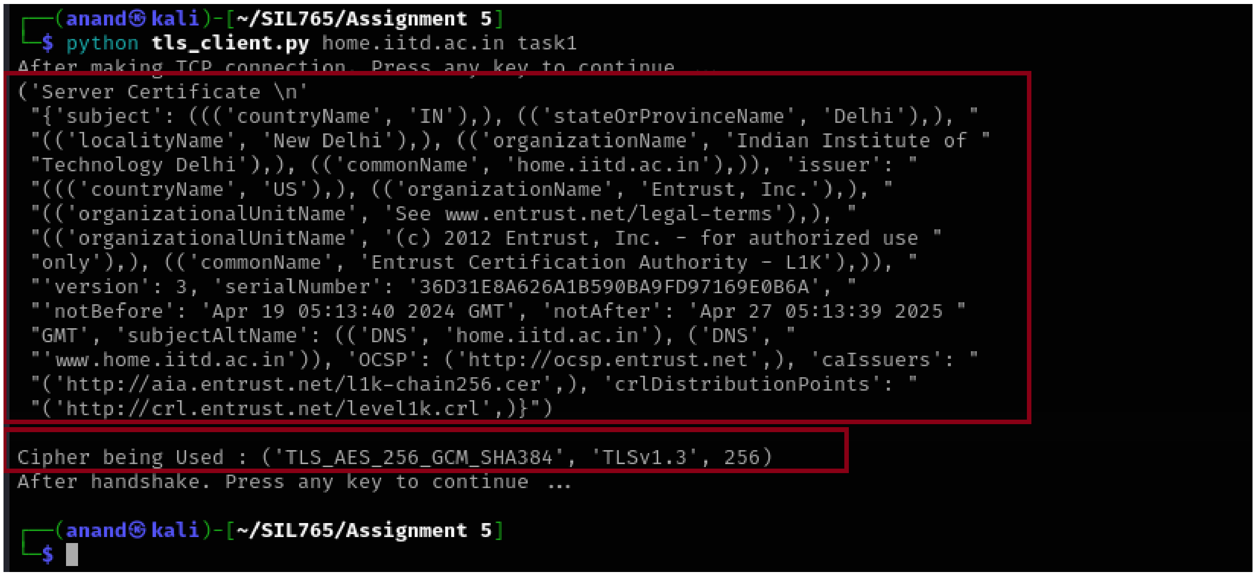
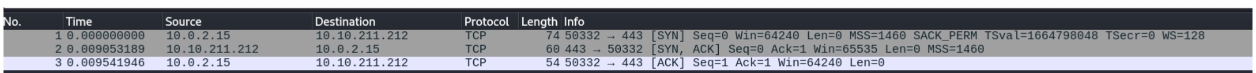


Figure 1: Server Certificate of home.iitd.ac.in

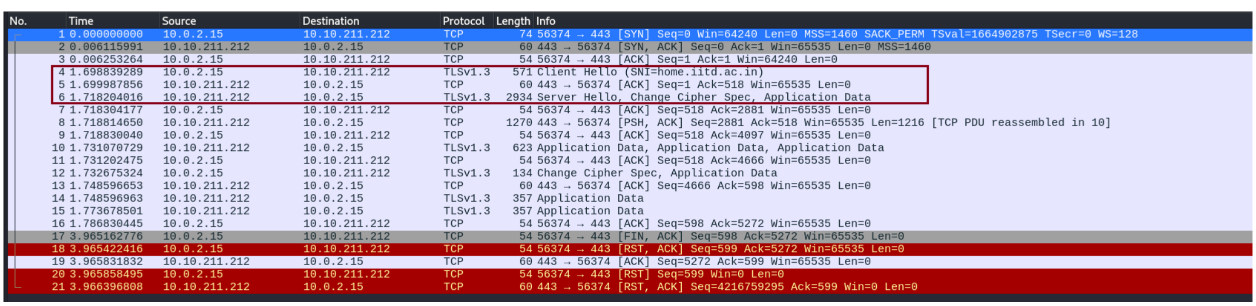
- TCP handshake: SYN, SYN-ACK, ACK.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	10.10.211.212	TCP	74	50332 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1664798048 TSecr=0 WS=128
2	0.009053189	10.10.211.212	10.0.2.15	TCP	60	443 → 50332 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.009541946	10.0.2.15	10.10.211.212	TCP	54	50332 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0

Figure 2: TCP handshake (SYN, SYN ACK, ACK).

- TLS handshake: ClientHello, ServerHello, Certificate, Finished.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	10.10.211.212	TCP	74	56374 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1664902875 TSecr=0 WS=128
2	0.006115991	10.10.211.212	10.0.2.15	TCP	60	443 → 56374 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.006253264	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	1.698839289	10.0.2.15	10.10.211.212	TLSv1.3	571	Client Hello (SNI=home.iitd.ac.in)
5	1.699987856	10.10.211.212	10.0.2.15	TCP	60	443 → 56374 [ACK] Seq=1 Ack=518 Win=65535 Len=0
6	1.718204016	10.10.211.212	10.0.2.15	TLSv1.3	2934	Server Hello, Change Cipher Spec, Application Data
7	1.718304177	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [ACK] Seq=518 Ack=2881 Win=65535 Len=0
8	1.718814650	10.10.211.212	10.0.2.15	TCP	1270	443 → 56374 [PSH, ACK] Seq=2881 Ack=518 Win=65535 Len=1216 [TCP PDU reassembled in 10]
9	1.718830040	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [ACK] Seq=518 Ack=4097 Win=65535 Len=0
10	1.731070729	10.10.211.212	10.0.2.15	TLSv1.3	623	Application Data, Application Data, Application Data
11	1.731202475	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [ACK] Seq=518 Ack=4666 Win=65535 Len=0
12	1.732675324	10.0.2.15	10.10.211.212	TLSv1.3	134	Change Cipher Spec, Application Data
13	1.748596653	10.10.211.212	10.0.2.15	TCP	60	443 → 56374 [ACK] Seq=4666 Ack=598 Win=65535 Len=0
14	1.748596963	10.10.211.212	10.0.2.15	TLSv1.3	357	Application Data
15	1.773678501	10.10.211.212	10.0.2.15	TLSv1.3	357	Application Data
16	1.786830445	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [ACK] Seq=598 Ack=5272 Win=65535 Len=0
17	3.965162776	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [FIN, ACK] Seq=598 Ack=5272 Win=65535 Len=0
18	3.965422416	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [RST, ACK] Seq=599 Ack=5272 Win=65535 Len=0
19	3.965831832	10.10.211.212	10.0.2.15	TCP	60	443 → 56374 [ACK] Seq=5272 Ack=599 Win=65535 Len=0
20	3.965858495	10.0.2.15	10.10.211.212	TCP	54	56374 → 443 [RST] Seq=599 Win=0 Len=0
21	3.966396808	10.10.211.212	10.0.2.15	TCP	60	443 → 56374 [RST, ACK] Seq=4216759295 Ack=599 Win=0 Len=0

Figure 3: TLS handshake (ClientHello, ServerHello).

3.4 Role of /etc/ssl/certs

The directory `/etc/ssl/certs` contains a collection of hashed root CA certificates in PEM format. It plays a critical role in TLS verification:

- It serves as the default trust store for many Linux-based systems and is used to verify the authenticity of server certificates during a TLS handshake.
- When the Python `ssl` module calls `load_verify_locations(capath)`, it walks through this directory to match the issuer of the server's certificate against one of the trusted root CAs.
- Without access to this directory or equivalent trusted roots, the TLS client cannot verify the server's certificate chain and will raise an `SSLCertVerificationError`.

4 Task 2: CA's Directory

4.1 Objective

Demonstrate failure with empty `./certs`, then fix by copying system certificates.

4.2 Execution Details

1. First, no `certs/` folder existed. We ran `task2` directly using:

```
$ python3 tls_client.py home.iitd.ac.in task2
```

2. The script detected the missing folder and automatically created it, then copied all certificates using:

```
$ cp -r /etc/ssl/certs ./certs
```

3. After copying, the TLS context was built using the custom path `./certs`, and the handshake completed successfully.

4.3 Observations

- Output after just making the "certs" directory.

```
┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ python tls_client.py home.iitd.ac.in
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "/home/anand/SIL765/Assignment 5/tls_client.py", line 22, in <module>
    ssock.do_handshake() # start the handshake
    ^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.12/ssl.py", line 1319, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self-signed cer
tificate in certificate chain (_ssl.c:1011)

┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ █
```

Figure 4: fas

- After copying the certficates

```
┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ cp -r /etc/ssl/certs ./certs

┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ python tls_client.py home.iitd.ac.in
After making TCP connection. Press any key to continue ...
{'OCSP': ('http://ocsp.entrust.net',),
 'caIssuers': ('http://aia.entrust.net/l1k-chain256.cer',),
 'crlDistributionPoints': ('http://crl.entrust.net/level1k.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Entrust, Inc.'),),
            (('organizationalUnitName', 'See www.entrust.net/legal-terms'),),
            (('organizationalUnitName',
              '(c) 2012 Entrust, Inc. - for authorized use only'),),
            (('commonName', 'Entrust Certification Authority - L1K'),)),
 'notAfter': 'Apr 27 05:13:39 2025 GMT',
 'notBefore': 'Apr 19 05:13:40 2024 GMT',
 'serialNumber': '36D31E8A626A1B590BA9FD97169E0B6A',
 'subject': ((('countryName', 'IN'),),
             (('stateOrProvinceName', 'Delhi'),),
             (('localityName', 'New Delhi'),),
             (('organizationName', 'Indian Institute of Technology Delhi'),),
             (('commonName', 'home.iitd.ac.in'),)),
 'subjectAltName': (('DNS', 'home.iitd.ac.in'), ('DNS', 'www.home.iitd.ac.in')),
 'version': 3}
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
After handshake. Press any key to continue ...

┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ █
```

Figure 5: fas

5 Task 3: Hostname

5.1 Objective

Show impact of `context.check_hostname` on handshake success.

5.2 Execution Details

1. Use the following command to resolve the IP address of `home.iitd.ac.in`:

```
$ dig +short home.iitd.ac.in
```

2. Edit the `/etc/hosts` file to map the resolved IP to a fake domain:

```
<resolved_ip>  anything.com
```

3. Run the script using:

```
$ python3 tls_client.py anything.com task3
```

   With `check_hostname=True` (default), the TLS handshake fails due to hostname mismatch.

4. Modify the code to set `context.check_hostname = False` after creating the context.

5. Re-run the same command:

```
$ python3 tls_client.py anything.com task3
```

   This time, the TLS handshake succeeds despite the mismatch between the hostname and the certificate's CN.

5.3 Observations

- Finding the IP address of "home.iitd.ac.in" using the dig command

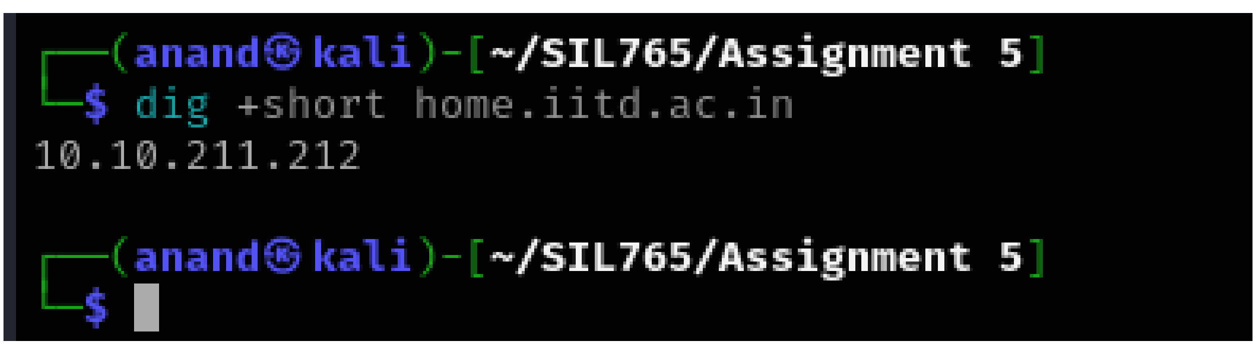


Figure 6: fas

- Output when the check.hostname was kept True

```
┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ python tls_client.py anything.com task3
Kept context.check_hostname = TRUE.
Change to FALSE run without hostname matching.

After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "/home/anand/SIL765/Assignment 5/tls_client.py", line 152, in <module>
    task3(hostname)
  File "/home/anand/SIL765/Assignment 5/tls_client.py", line 87, in task3
    ssock.do_handshake() # start the handshake
    ^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.12/ssl.py", line 1319, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: Hostname mismat
ch, certificate is not valid for 'anything.com'. (_ssl.c:1011)

┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$
```

Figure 7: context.check_hostname = True

- Output when the check.hostname was kept False

```
┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$ python tls_client.py anything.com task3
Kept context.check_hostname = TRUE.
Change to FALSE run without hostname matching.

After making TCP connection. Press any key to continue ...
{'OCSP': ('http://ocsp.entrust.net',),
 'caIssuers': ('http://aia.entrust.net/l1k-chain256.cer',),
 'crlDistributionPoints': ('http://crl.entrust.net/level1k.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Entrust, Inc.'),),
            (('organizationalUnitName', 'See www.entrust.net/legal-terms'),),
            (('organizationalUnitName',
              '(c) 2012 Entrust, Inc. - for authorized use only'),),
            (('commonName', 'Entrust Certification Authority - L1K'),)),
 'notAfter': 'Apr 27 05:13:39 2025 GMT',
 'notBefore': 'Apr 19 05:13:40 2024 GMT',
 'serialNumber': '36D31E8A626A1B590BA9FD97169E0B6A',
 'subject': ((('countryName', 'IN'),),
             (('stateOrProvinceName', 'Delhi'),),
             (('localityName', 'New Delhi'),),
             (('organizationName', 'Indian Institute of Technology Delhi'),),
             (('commonName', 'home.iitd.ac.in'),)),
 'subjectAltName': (('DNS', 'home.iitd.ac.in'), ('DNS', 'www.home.iitd.ac.in')),
 'version': 3}
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
After handshake. Press any key to continue ...

┌──(anand㉿kali)-[~/SIL765/Assignment 5]
└─$
```

Figure 8: context.check_hostname = False

6 Task 4: Communicating Data

6.1 Objective

Send HTTP GET over TLS to fetch HTML and an image.

6.2 Execution

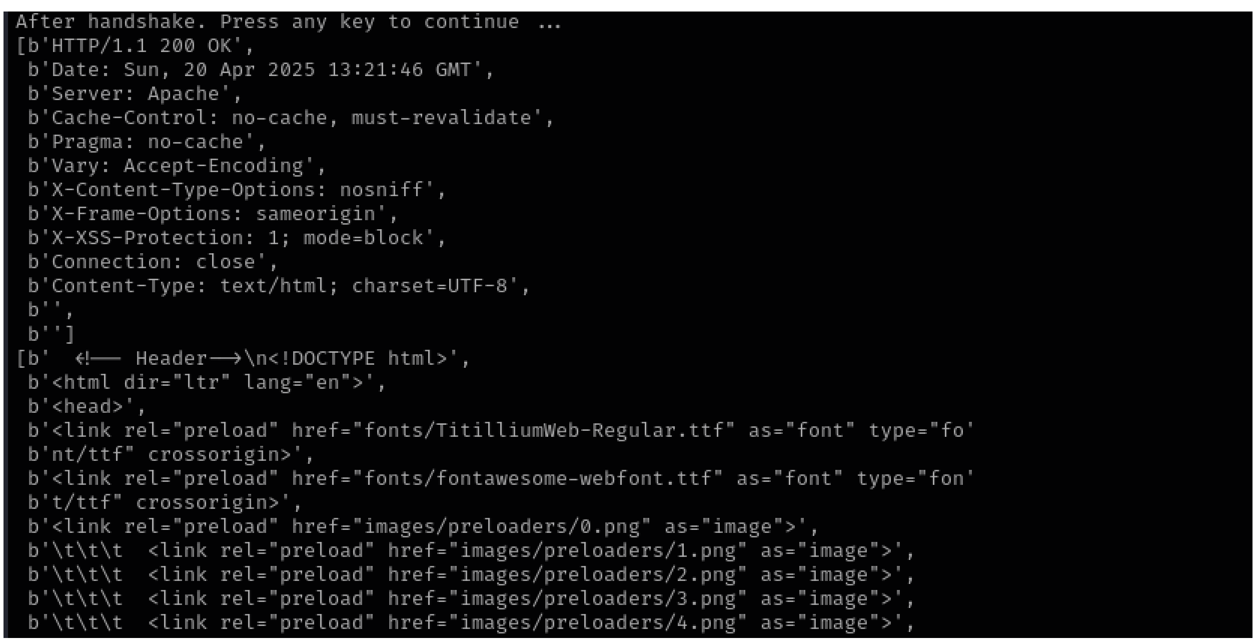
1. Run the following command to start Task 4:

   ```
   $ python3 tls_client.py home.iitd.ac.in task4
   ```

2. After the TCP and TLS handshake completes, the script first sends a raw HTTP GET request to fetch the homepage. The response is printed in the terminal.
3. Next, a second GET request is sent to download an image (`/images/logo-diamond.png`). The image is received as binary data.
4. The script strips the HTTP headers and saves the image content to a file named `downloaded.png`.
5. Open the saved image with an image viewer to verify its integrity:

6.3 Observations

- GET request



```
After handshake. Press any key to continue ...
[b'HTTP/1.1 200 OK',
 b'Date: Sun, 20 Apr 2025 13:21:46 GMT',
 b'Server: Apache',
 b'Cache-Control: no-cache, must-revalidate',
 b'Pragma: no-cache',
 b'Vary: Accept-Encoding',
 b'X-Content-Type-Options: nosniff',
 b'X-Frame-Options: sameorigin',
 b'X-XSS-Protection: 1; mode=block',
 b'Connection: close',
 b'Content-Type: text/html; charset=UTF-8',
 b'',
 b'']
[b'  <!-- Header-->\n<!DOCTYPE html>',
 b'<html dir="ltr" lang="en">',
 b'<head>',
 b'<link rel="preload" href="fonts/TitilliumWeb-Regular.ttf" as="font" type="fo'
 b'nt/ttf" crossorigin>',
 b'<link rel="preload" href="fonts/fontawesome-webfont.ttf" as="font" type="fon'
 b't/ttf" crossorigin>',
 b'<link rel="preload" href="images/preloaders/0.png" as="image">',
 b'\t\t\t  <link rel="preload" href="images/preloaders/1.png" as="image">',
 b'\t\t\t  <link rel="preload" href="images/preloaders/2.png" as="image">',
 b'\t\t\t  <link rel="preload" href="images/preloaders/3.png" as="image">',
 b'\t\t\t  <link rel="preload" href="images/preloaders/4.png" as="image">',
```

Figure 9: Raw HTTP GET request of the Homepage of home.iitd.ac.in

- Image Download from home.iitd.ac.in website

Figure 10: Image Download from home.iitd.ac.in website

7 Security and Efficiency of TLS Protocol

7.1 Security

The TLS (Transport Layer Security) protocol is designed to provide end-to-end encrypted communication over an insecure network. Our implementation and experiments demonstrate the following key security properties:

- **Confidentiality:** All data exchanged between client and server is encrypted using the negotiated cipher suite (e.g., `TLS_AES_128_GCM_SHA256`). This ensures that an eavesdropper cannot view or tamper with the communication.
- **Authentication:** During the TLS handshake, the server presents a digital certificate issued by a trusted Certificate Authority (CA). This certificate is verified using the CA store in `/etc/ssl/certs` or a custom directory.
- **Integrity:** TLS uses AEAD (Authenticated Encryption with Associated Data) ciphers which provide both encryption and integrity protection. This ensures that any modification to the ciphertext by a man-in-the-middle will be detected.
- **Hostname Verification:** Enabled by default using `context.check_hostname = True`. This ensures that the certificate matches the intended domain and prevents redirection or impersonation attacks.
- **Resistance to MITM:** The use of trusted certificates, strict hostname checks, and encrypted handshakes collectively protect against man-in-the-middle attacks.

7.2 Efficiency

TLS 1.3 brings notable performance enhancements over previous versions, and our client benefits from these improvements:

- **Reduced Handshake Latency:** TLS 1.3 requires only 1-RTT (Round Trip Time) for the full handshake compared to 2-RTT in TLS 1.2, making connections faster and more responsive.
- **Optimized Cipher Suites:** TLS 1.3 removes legacy ciphers and uses only modern, efficient algorithms like AES-GCM and ChaCha20-Poly1305.
- **Session Resumption (not shown in this report):** TLS 1.3 supports session tickets for quick resumption without repeating the full handshake.
- **Resource Usage:** The Python implementation is lightweight and efficient, suitable for use even in low-resource environments. The use of non-blocking reads and memory-efficient buffer handling helps maintain good performance.
- **Layered Design:** By sending raw HTTP requests over the established TLS session, we demonstrate how TLS cleanly separates secure transport from the application layer (HTTP).

In conclusion, TLS 1.3 as implemented in our client offers both robust security and modern performance optimizations, making it ideal for secure communications on the internet.