PROJECT 4: Transport Layer Security (TLS) Lab

Objective: The objective of this project is to help students understand how the TLS works and how to use TLS in programming.

Task 1: TLS Client:

Task 1.a: TLS handshake:

```
[04/17/23]seed@VM:~/HW4$ handshake.py www.google.com
After making TCP connection. Press any key to continue ...
*********Cipher Used***********
TLS_AES_256_GCM_SHA384
******Server certificate*******
{'OCSP': ('http://ocsp.pki.goog/gts1c3',),
 'caIssuers': ('http://pki.goog/repo/certs/gts1c3.der',),
 'crlDistributionPoints': ('http://crls.pki.goog/gts1c3/fVJxbV-Ktmk.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services LLC'),),
            (('commonName', 'GTS CA 1C3'),)),
 'notAfter': 'Jun 20 16:54:57 2023 GMT',
 'notBefore': 'Mar 28 16:54:58 2023 GMT',
 'serialNumber': '751A47665BB124F20A5F38180A2BEC77',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}
After handshake. Press any key to continue ...
[04/17/23]seed@VM:~/HW4$
```

• What is the cipher used between the client and the server?

- The cipher used between the client and the server is determined by the SSL/TLS negotiation during the handshake, and it is not explicitly specified in the code. However, we can print out the selected cipher by adding the following line after the do_handshake() function call.
- ***print(ssock.cipher()[0])***

```
*********Cipher Used***********
TLS_AES_256_GCM_SHA384
```

• Please print out the server certificate in the program.

- To print out the server certificate, we can add the following line after the do_handshake() call.
- ***pprint.pprint(ssock.getpeercert())***

```
******Server certificate*******
{'OCSP': ('http://ocsp.pki.goog/gts1c3',),
 'caIssuers': ('http://pki.goog/repo/certs/gts1c3.der',),
 'crlDistributionPoints': ('http://crls.pki.goog/gts1c3/fVJxbV-Ktmk.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services LLC'),),
            (('commonName', 'GTS CA 1C3'),)),
 'notAfter': 'Jun 20 16:54:57 2023 GMT',
 'notBefore': 'Mar 28 16:54:58 2023 GMT',
 'serialNumber': '751A47665BB124F20A5F38180A2BEC77',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}
After handshake. Press any key to continue ...
```

• Explain the purpose of **/etc/ssl/certs**.

- When a client, such as a web browser or the Python code in the example, connects to an SSL/TLS server, it needs to verify that the server's certificate is issued by a trusted CA and has not been tampered with. The client does this by checking the server's certificate chain against the public key certificates of trusted CAs that it has on its system. The directory **/etc/ssl/certs** is the default location for storing these trusted CA certificates.
- **cd /etc/ssl/certs**
- **ls**

```
ACCVRAIZ1.pem                                  QuoVadis_Root_CA_1_G3.pem
AC_RAIZ_FNMT-RCM.pem                           QuoVadis_Root_CA_2_G3.pem
Actalis_Authentication_Root_CA.pem             QuoVadis_Root_CA_2.pem
ad088e1d.0                                     QuoVadis_Root_CA_3_G3.pem
aee5f10d.0                                     QuoVadis_Root_CA_3.pem
AffirmTrust_Commercial.pem                     QuoVadis_Root_CA.pem
AffirmTrust_Networking.pem                     Secure_Global_CA.pem
AffirmTrust_Premium_ECC.pem                    SecureSign_RootCA11.pem
AffirmTrust_Premium.pem                        SecureTrust_CA.pem
Amazon_Root_CA_1.pem                           Security_Communication_RootCA2.pem
Amazon_Root_CA_2.pem                           Security_Communication_Root_CA.pem
Amazon_Root_CA_3.pem                           Sonera_Class_2_Root_CA.pem
Amazon_Root_CA_4.pem                           ssl-cert-snakeoil.pem
Atos_TrustedRoot_2011.pem                      SSL.com_EV_Root_Certification_Authority_ECC.pem
Autoridad_de_Certificacion_Firmaprofesional_CIF_A62634068.pem  SSL.com_EV_Root_Certification_Authority_RSA_R2.pem
```

-

• Use Wireshark to capture the network traffics during the execution of the program, and explain your observation. 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.

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	74	56784 → 443 [SYN] Seq=3920370834 Win=64240 Len=0 MSS=1460 SACK_PERM=
2	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TCP	60	443 → 56784 [SYN, ACK] Seq=499136001 Ack=3920370835 Win=65535 Len=0
3	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	54	56784 → 443 [ACK] Seq=3920370835 Ack=499136002 Win=64240 Len=0
4	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TLSv1.3	571	Client Hello
5	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TCP	60	443 → 56784 [ACK] Seq=499136002 Ack=3920371352 Win=65535 Len=0
6	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TLSv1.3	1304	Server Hello, Change Cipher Spec
7	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	54	56784 → 443 [ACK] Seq=3920371352 Ack=499137252 Win=63750 Len=0
8	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TLSv1.3	3097	Application Data
9	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	54	56784 → 443 [ACK] Seq=3920371352 Ack=499140295 Win=62780 Len=0
10	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TLSv1.3	134	Change Cipher Spec, Application Data
11	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TCP	60	443 → 56784 [ACK] Seq=499140295 Ack=3920371432 Win=65535 Len=0
12	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	54	56784 → 443 [FIN, ACK] Seq=3920371432 Ack=499140295 Win=62780 Len=0
13	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TCP	60	443 → 56784 [ACK] Seq=499140295 Ack=3920371433 Win=65535 Len=0
14	2023-04-27 19:5…	142.251.163.105	10.0.2.15	TCP	60	443 → 56784 [FIN, ACK] Seq=499140295 Ack=3920371433 Win=65535 Len=0
15	2023-04-27 19:5…	10.0.2.15	142.251.163.105	TCP	54	56784 → 443 [ACK] Seq=3920371433 Ack=499140296 Win=62780 Len=0

- A TCP SYN packet sent from the client to the server to initiate the TCP handshake, **line number 1.**
- A TCP SYN-ACK packet sent from the server to the client in response to the TCP SYN packet.

- A TCP ACK packet sent from the client to the server to acknowledge the TCP SYN-ACK packet.
- An SSL/TLS Client Hello message sent from the client to the server to initiate the SSL/TLS handshake.
- An SSL/TLS Server Hello message sent from the server to the client in response to the Client Hello message, **line number 4.**
- An SSL/TLS Certificate message sent from the server to the client containing the server's certificate.
- An SSL/TLS Server Hello message sent from the server to the client indicating the end of the Server Hello phase.
- An SSL/TLS Client Key Exchange message sent from the client to the server to exchange cryptographic keys.
- An SSL/TLS Change Cipher Spec message sent from the client to the server to indicate that subsequent messages will be encrypted using the negotiated cipher.
- An SSL/TLS Encrypted Handshake Message and Application Data message sent from the client to the server containing encrypted application data.
- An SSL/TLS Encrypted Handshake Message and Application Data message sent from the server to the client containing encrypted application data.
- An SSL/TLS Alert message sent from the client to the server indicating that the client has closed the connection.
- A TCP FIN-ACK packet sent from the client to the server to initiate the TCP connection termination.
- A TCP ACK packet sent from the server to the client to acknowledge the TCP FIN-ACK packet.
- The TCP handshake is a three-way process that establishes a reliable, connection-oriented communication channel between two endpoints before data can be exchanged. The TLS handshake, on the other hand, is a process that occurs on top of the TCP connection and establishes a secure, encrypted connection between two endpoints before any sensitive data is exchanged. The TLS handshake is dependent on the TCP handshake, as it requires an established and reliable TCP connection for the exchange of handshake messages.

Task 1.b: CA's Certificate:

Created /etc/ssl/certs/**client-certs** folder and updated the 'cadir' path in handshake.py file.

```
[05/01/23]seed@VM:~/HW4$ cat handshake.py
#!/usr/bin/env python3

import socket, ssl, sys, pprint
hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs/client-certs'

# Set up the TLS context
```

Then I tried running handshake.py with 'www.google.com' as parameter but I am getting error such as "unable to get local issuer certificate", as you can see below snapshot:

```
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake.py", line 25, 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:
)
[05/01/23]seed@VM:~/HW4$
```

In order to resolve this error, I need to first copy the corresponding CA's certificate into client-certs folder and this can be done by seeing the **issuer's common name** which starts from **GTS_*** in my case (refer server certificate image attached in Task 1.a).

After replicating **GTS_Root_R1.crt** in /etc/ssl/certs/client-certs folder, I used OPENSSL command to generate a hash value for that certificate, after which I created a symbolic link. Now, you can see our client program is able to talk to the server in the below snapshot.

```
[05/01/23]seed@VM:~/HW4$ handshake.py www.google.com
After making TCP connection. Press any key to continue ...
*********Cipher Used***********
TLS_AES_256_GCM_SHA384
******Server certificate*******
{'OCSP': ('http://ocsp.pki.goog/gts1c3',),
 'caIssuers': ('http://pki.goog/repo/certs/gts1c3.der',),
 'crlDistributionPoints': ('http://crls.pki.goog/gts1c3/QqFxbi9M48c.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services LLC'),),
            (('commonName', 'GTS CA 1C3'),)),
 'notAfter': 'Jun 26 08:25:06 2023 GMT',
 'notBefore': 'Apr  3 08:25:07 2023 GMT',
 'serialNumber': '029AA3D282DC117C0A1458D6EABCFF02',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}
After handshake. Press any key to continue ...
[05/01/23]seed@VM:~/HW4$
lrwxrwxrwx 1 root root   15 May  1 20:35 1001acf7.0 -> GTS_Root_R1.crt
lrwxrwxrwx 1 root root   15 May  1 20:35 626dceaf.0 -> GTS_Root_R2.crt
-rw-r--r-- 1 root root 1915 May  1 20:35 GTS_Root_R1.crt
-rw-r--r-- 1 root root 1915 May  1 20:34 GTS_Root_R2.crt
-rw-r--r-- 1 root root  769 May  1 20:31 GTS_Root_R3.crt
[05/01/23]seed@VM:.../client-certs$
```

Additional requirement:

Previously, it was done for www.google.com and now I am conducting the same task for www.github.com & www.linkedin.com

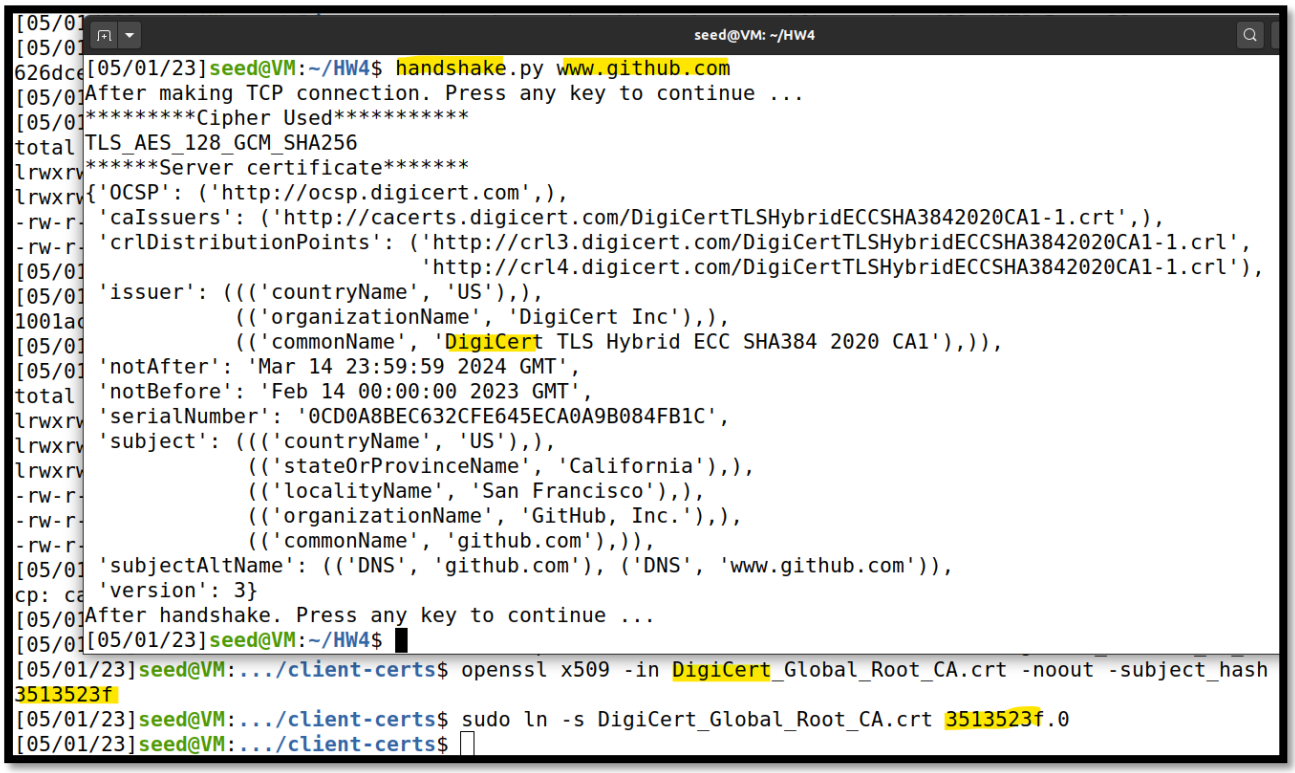
Github Server Certificate:

```
[05/01/23]seed@VM:~/HW4$ handshake.py www.github.com
After making TCP connection. Press any key to continue ...
*********Cipher Used**********
TLS_AES_128_GCM_SHA256
******Server certificate*******
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('commonName', 'DigiCert TLS Hybrid ECC SHA384 2020 CA1'),)),
 'notAfter': 'Mar 14 23:59:59 2024 GMT',
 'notBefore': 'Feb 14 00:00:00 2023 GMT',
 'serialNumber': '0CD0A8BEC632CFE645ECA0A9B084FB1C',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'San Francisco'),),
             (('organizationName', 'GitHub, Inc.'),),
             (('commonName', 'github.com'),)),
 'subjectAltName': (('DNS', 'github.com'), ('DNS', 'www.github.com')),
 'version': 3}
After handshake. Press any key to continue ...
[05/01/23]seed@VM:~/HW4$
```

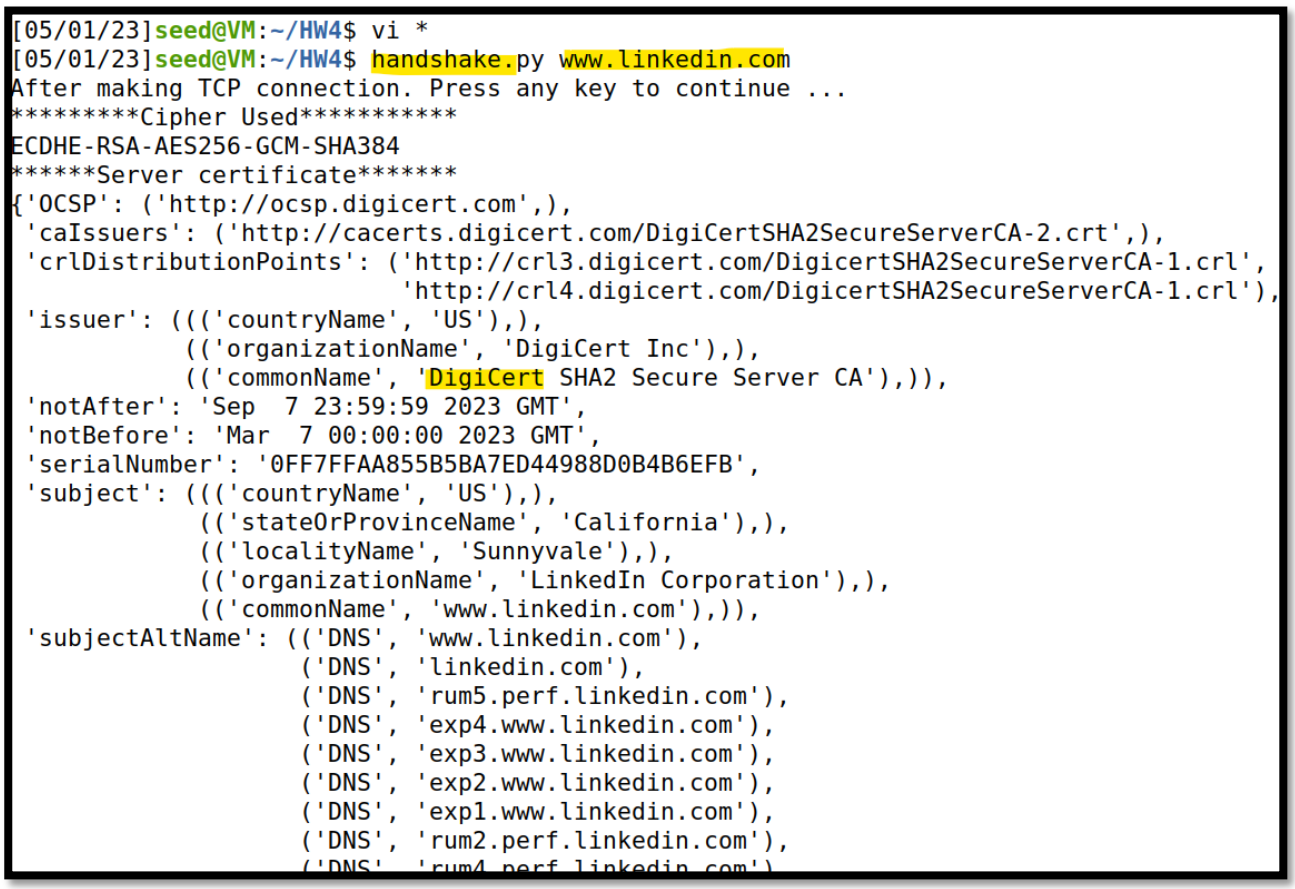
LinkedIn Server Certificate:

```
[05/01/23]seed@VM:~/HW4$ handshake.py www.linkedin.com
After making TCP connection. Press any key to continue ...
*********Cipher Used**********
ECDHE-RSA-AES256-GCM-SHA384
******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  7 23:59:59 2023 GMT',
 'notBefore': 'Mar  7 00:00:00 2023 GMT',
 'serialNumber': '0FF7FFAA855B5BA7ED44988D0B4B6EFB',
 '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')
```

Result:



As both Github and LinkedIn Issuer's name is common, so by using one DigiCert Global CA certificate, I was able to connect to both the domain servers.



Task 1.c: Experiment with the hostname check:

Step 1: Get the Ip address of www.linkedin.com using the **dig** command.

```
[05/01/23]seed@VM:~/HW4$ dig www.linkedin.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.linkedin.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48915
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.linkedin.com.              IN      A

;; ANSWER SECTION:
www.linkedin.com.       181     IN      CNAME   www-linkedin-com.l-0005.l-msedge.net.
www-linkedin-com.l-0005.l-msedge.net. 225 IN CNAME l-0005.l-msedge.net.
l-0005.l-msedge.net.    38      IN      A       13.107.42.14
```

Step 2: Modify /etc/hosts file by adding the above IP address **"**13.107.42.14**"** with www.linkedin2023.com

```
#For HW4
10.9.0.80       www.example2020.com
13.107.42.14    www.linkedin2023.com
[05/01/23]seed@VM:~$
```

Step 3: Modify "context.check_hostname" to **True** and **False** in handshake.py and observe the result.

When the check_hostname is **True**:

```
[05/01/23]seed@VM:~/HW4$ handshake.py www.linkedin2023.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake.py", line 25, 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:
w.linkedin2023.com'. (_ssl.c:1123)
[05/01/23]seed@VM:~/HW4$
```

When the check_hostname is **False**:

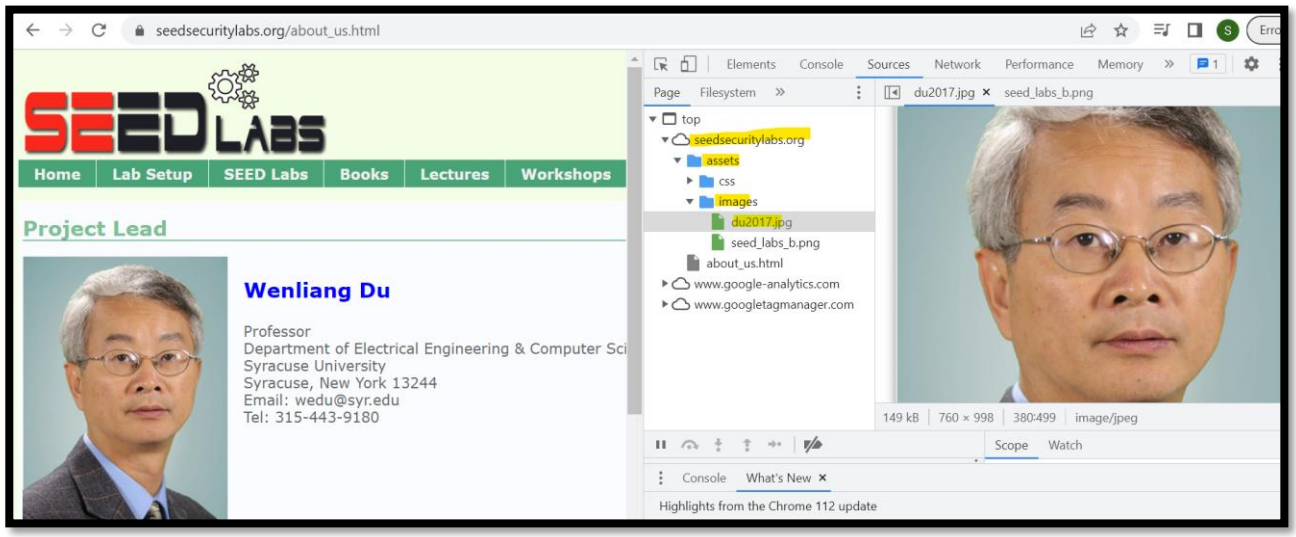
Observation: When the check_hostname is True, an error appears because the hostname I provided to establish the connection, is different from the one specified in the Server certificate. And if the check_hostname is set to False, it bypasses the hostname check and show the server certificate as you can see in the above snapshot.

Task 1.d: Sending and getting Data:

(1) In handshake.py file, I added the required HTTP Request and Response code. This code is an example of how to send an HTTP GET request to a server using low-level socket programming and receive the HTTP response. Here, I observed that I am getting the website's (which I passed as a parameter) HTML headers, body in chunks of up to 2048 bytes in response.
Output:

```
After handshake. Press any key to continue ...
[b'HTTP/1.1 200 OK',
 b'Cache-Control: no-cache, no-store',
 b'Pragma: no-cache',
 b'Content-Length: 116929',
 b'Content-Type: text/html; charset=utf-8',
 b'Expires: Thu, 01 Jan 1970 00:00:00 GMT',
 b'Set-Cookie: JSESSIONID=ajax:7684152035633533608; SameSite=None; Path=/; Doma'
 b'in=.www.linkedin.com; Secure',
 b'Set-Cookie: lang=v=2&lang=en-us; SameSite=None; Path=/; Domain=linkedin.com;'
 b' Secure',
 b'Set-Cookie: bcookie="v=2&61eb5841-0291-4f19-8f3f-8ca53ff0aa71"; domain=.link'
 b'edin.com; Path=/; Secure; Expires=Wed, 01-May-2024 02:27:40 GMT; SameSite=No'
 b'ne',
 b'Set-Cookie: bscookie="v=1&20230502022740efd8595c-80cf-42c0-8e0b-567143bc0dc1'
 b'AQFtVBmgn5IyHFl1IE69MTBS0IYEE6X-"; domain=.www.linkedin.com; Path=/; Secure;'
 b' Expires=Wed, 01-May-2024 02:27:40 GMT; HttpOnly; SameSite=None',
 b'Set-Cookie: lidc="b=VGST00:s=V:r=V:a=V:p=V:g=2992:u=1:x=1:i=1682994460:t=168'
 b'3080860:v=2:sig=AQGgXjM_fr-ZzfIXsLBBCdgqmunHWb-S"; Expires=Wed, 03 May 2023 '
 b'02:27:40 GMT; domain=.linkedin.com; Path=/; Secure',
 b"Content-Security-Policy: default-src 'none'; connect-src 'self' *.licdn.com "
 b'*.linkedin.com cdn.linkedin.oribi.io dpm.demdex.net/id lnkd.demdex.net blob:'
 b' accounts.google.com/gsi/ linkedin.sc.omtrdc.net/b/ss/ *.microsoft.com *.adn'
 b"xs.com; script-src 'report-sample' 'sha256-SSoodjUD3LGm2FfFCVHGqEb8D4UM3OOig"
 b"idT2UKDcYg=' 'sha256-cKTgdnmO6+hXd85a9wKg1effVfVzenUAtUCyOKY9bQE=' 'sha256-D"
 b"wtT8+ZZKpxH9pqZNAmJ3GdbLAh5SsYaXR3omTXPCns=' 'sha256-sV9jZa797T0QWBzcU/CNd4t"
 b"pBhTnh+TFdLnfjlitl28=' 'sha256-aa/Q8CRBDSqTQbCIyioPhZaz+G+dbPyu7BzsjInEmiU='"
 b" 'sha256-THuVhwbXPeTR0HszASqMOnIyxqEgvGyBwSPBKBF/iMc=' 'sha256-zTIusdVJJeXz9"
 b"+iox2a+pdDglzbpRpFVRzEwvW4AONk=' 'sha256-iC8MPqNLw0FDnsBf4DlSkFLNTwhkI85aoui"
```

(2) In this part, I am trying to fetch Professor Wenliang Du image from SEED Labs.



To do so, I need to pass seedsecuritylabs.org with the highlighted image path as shown in the below snapshot:

```
[05/01/23]seed@VM:~/HW4$ handshake.py seedsecuritylabs.org assets/images/du2017.jpg
After making TCP connection. Press any key to continue ...
*********Cipher Used**********
TLS_AES_256_GCM_SHA384
******Server certificate*******
{'OCSP': ('http://r3.o.lencr.org',),
 'caIssuers': ('http://r3.i.lencr.org/',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', "Let's Encrypt"),),
            (('commonName', 'R3'),)),
 'notAfter': 'Jul 26 20:21:26 2023 GMT',
 'notBefore': 'Apr 27 20:21:27 2023 GMT',
 'serialNumber': '03829E3A2467A03013FBA43949D5A91A24DF',
 'subject': ((('commonName', 'seedsecuritylabs.org'),),),
 'subjectAltName': (('DNS', 'seedsecuritylabs.org'),),
 'version': 3}
After handshake. Press any key to continue ...
[b'HTTP/1.1 200 OK',
 b'Connection: close',
 b'Content-Length: 7072',
 b'Server: GitHub.com',
 b'Content-Type: text/html; charset=utf-8',
 b'Last-Modified: Mon, 24 Apr 2023 19:01:31 GMT',
 b'Access-Control-Allow-Origin: *',
 b'ETag: "6446d20b-1ba0"',
```

You can see, I am able to fetch data in HTML following handshake.

Task 2: TLS Server:

Task 2.a: Implement a simple TLS Server:

To Implement a simple TLS Server, I copied my CA.crt to /client-certs folder and added server.py file. In Server.py and handshake.py, I used the port 4433. Since, I already had my www.shubham2023.com IP in the local /etc/hosts file as well as its CA certificate, so I used this to demonstrate this task.

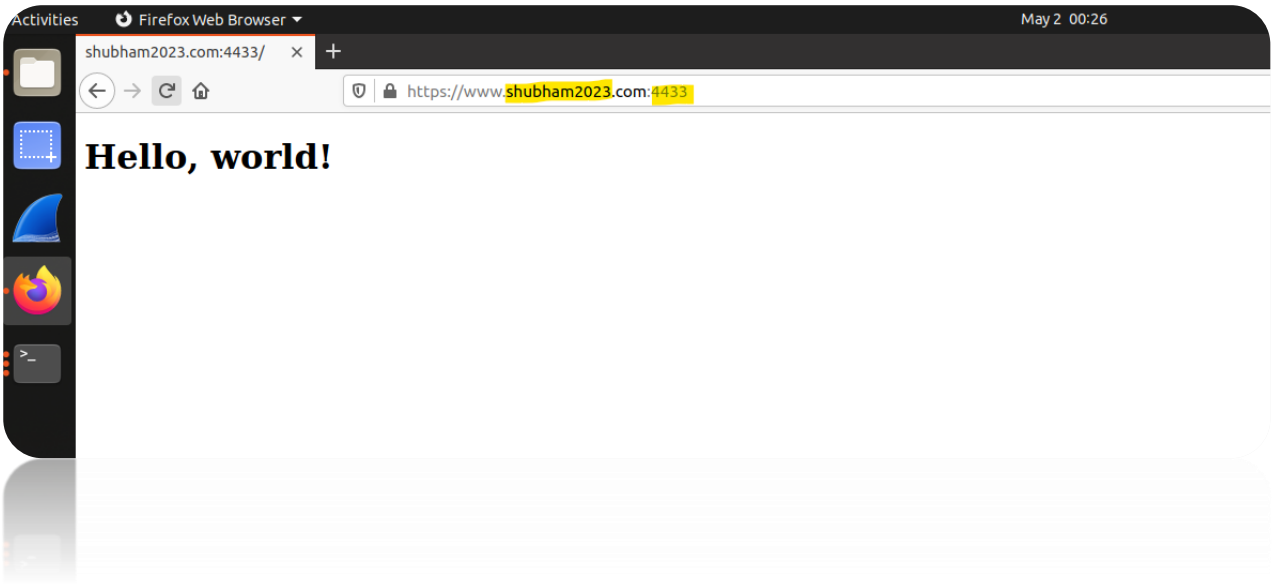
I opened two terminals, one for server (server.py) and another for client (handshake.py).

After running the server, I ran the client (www.shubham2023.com), keeping "/etc/ssl/certs/client-certs" as "cadir" path. The below snapshot shows a successful demonstration of the server and the client.

Once I change the “cadir” path in the client to “/etc/ssl/certs”. It will throw error as below because the required SSL Certificate is not present in that path. Hence, it throws error.

Task 2.b: Testing the server program using browsers:

In the last PKI assignment, I had already added CA certificate in the browser. Now I am going to test it. After running the server, I am able browse my website on the port 4433, on which the server is listening.



Task 2.c: Certificate with multiple names:

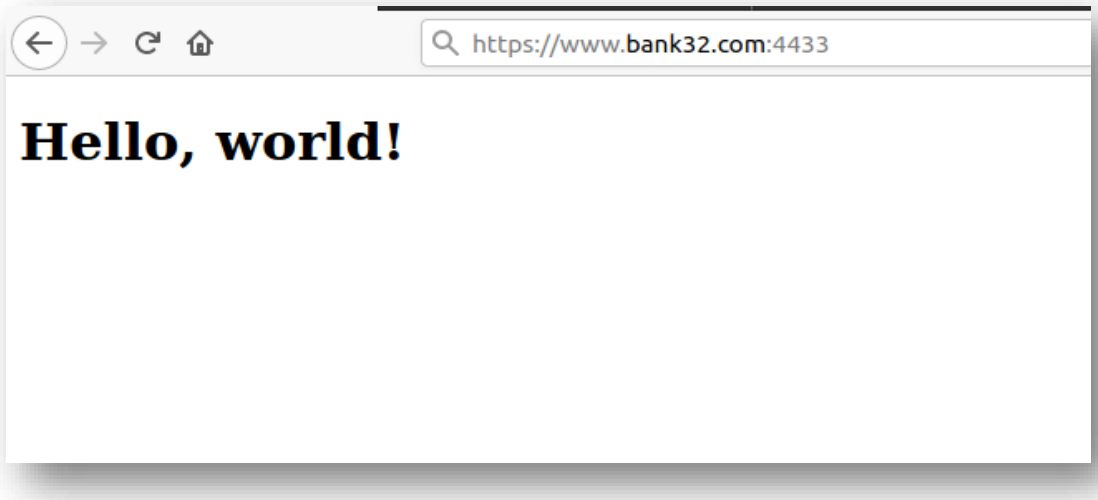
Created multiple names as you can see the highlighted in the below Subject Alternative Names.

```
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: May  2 04:52:10 2023 GMT
            Not After : Apr 29 04:52:10 2033 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Virginia
            organizationName          = GMU
            commonName                = www.shubham2023.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                E8:E1:48:C7:48:02:AE:37:D9:F7:80:50:F2:A2:B1:02:6B:69:EC:8A
            X509v3 Authority Key Identifier:
                keyid:EE:BC:AD:97:21:3A:E4:CE:91:16:F1:06:C3:FE:C0:67:05:CE:E4:0B

            X509v3 Subject Alternative Name:
                DNS:www.bank32.com, DNS:www.example.com, DNS:*.shubham2023.com
Certificate is to be certified until Apr 29 04:52:10 2033 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
[05/02/23]seed@VM:~/HW4$ server.py
Enter PEM pass phrase:
```

Below multiple browsers with different hostnames are pointing to the same shubham2023.com domain. This demonstrates that my server now supports multiple hostnames.



References:

[1] Crypto_TLS.pdf (seedsecuritylabs.org)
[2] https://docs.docker.com/