CPS 472/572 Fall 2024
Prof: Zhongmei Yao
Lab 4 Report: Transport Layer Security (TLS) Lab

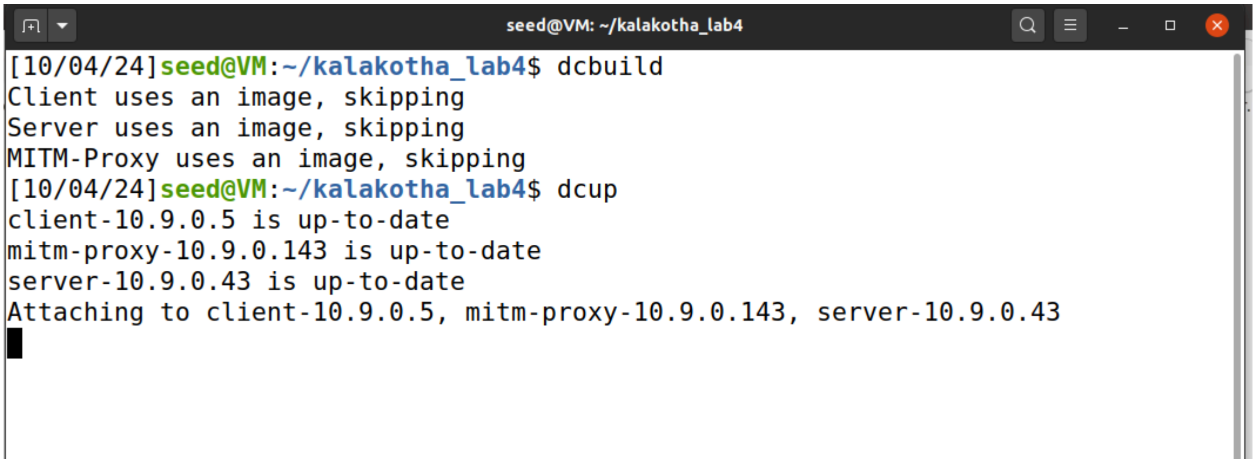
1. Dinesh Kumar Kala (kalad1@udayton.edu) Student ID:101745354
2. Akhil Reddy Kotha (kothaa6@udayton.edu) Student ID:101790043

1. Overview:

In this lab, we focus on secure data transmission over the Internet, where unprotected data can be accessed or altered. We emphasize the use of TLS (Transport Layer Security) for encryption, particularly in HTTPS. We will implement TLS client and server programs to examine risks from compromised Certificate Authorities (CAs). Topics include Public-Key Infrastructure (PKI), TLS programming, HTTPS proxies, X.509 certificates with Subject Alternative Name (SAN) extensions, and Man-In-The-Middle attacks. Completion of the PKI lab is required.

Takeaways: The key takeaway from this lab is the understanding of how TLS (Transport Layer Security) provides secure communication over the Internet. We learned the importance of encrypting data to protect it from unauthorized access and tampering. By implementing TLS client and server programs, we gained hands-on experience with the protocol and its operations. This lab highlighted the critical role of cryptography in ensuring safe data transmission in today's digital landscape.

2. Lab Environment:



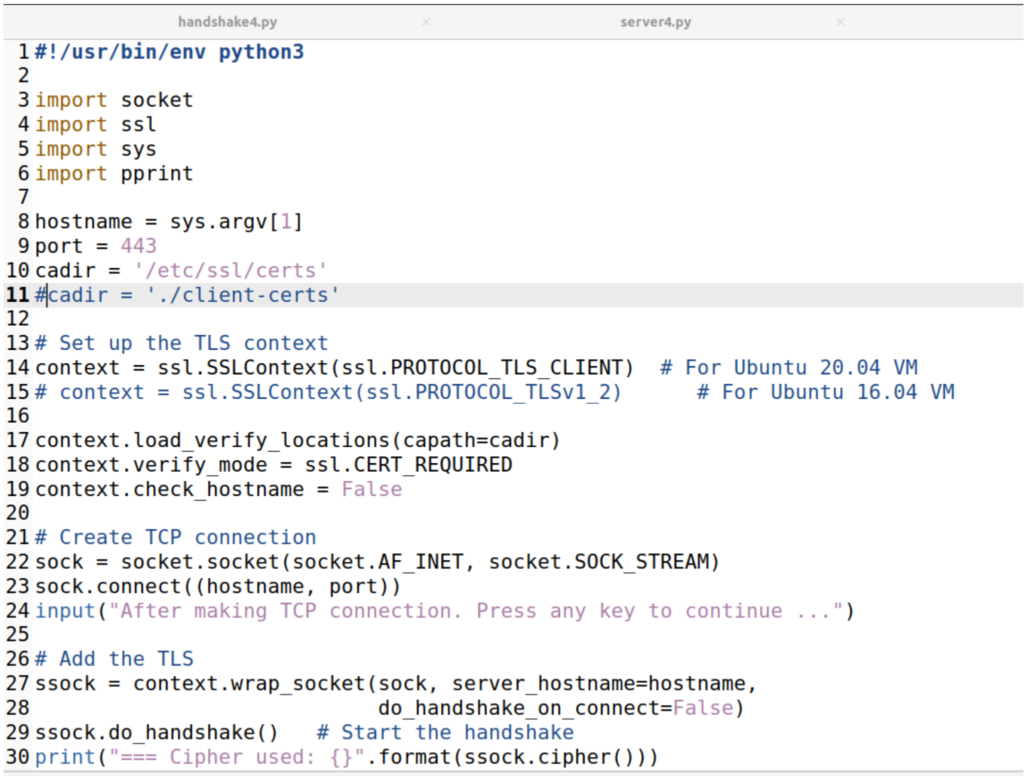
```
[10/04/24]seed@VM:~/kalakotha_lab4$ dcbuild
Client uses an image, skipping
Server uses an image, skipping
MITM-Proxy uses an image, skipping
[10/04/24]seed@VM:~/kalakotha_lab4$ dcup
client-10.9.0.5 is up-to-date
mitm-proxy-10.9.0.143 is up-to-date
server-10.9.0.43 is up-to-date
Attaching to client-10.9.0.5, mitm-proxy-10.9.0.143, server-10.9.0.43
```

We downloaded the zip file from the seed lab and using dcbuild and dcup, we successfully setup the lab environment.

Takeaways: we successfully setup the lab environment.

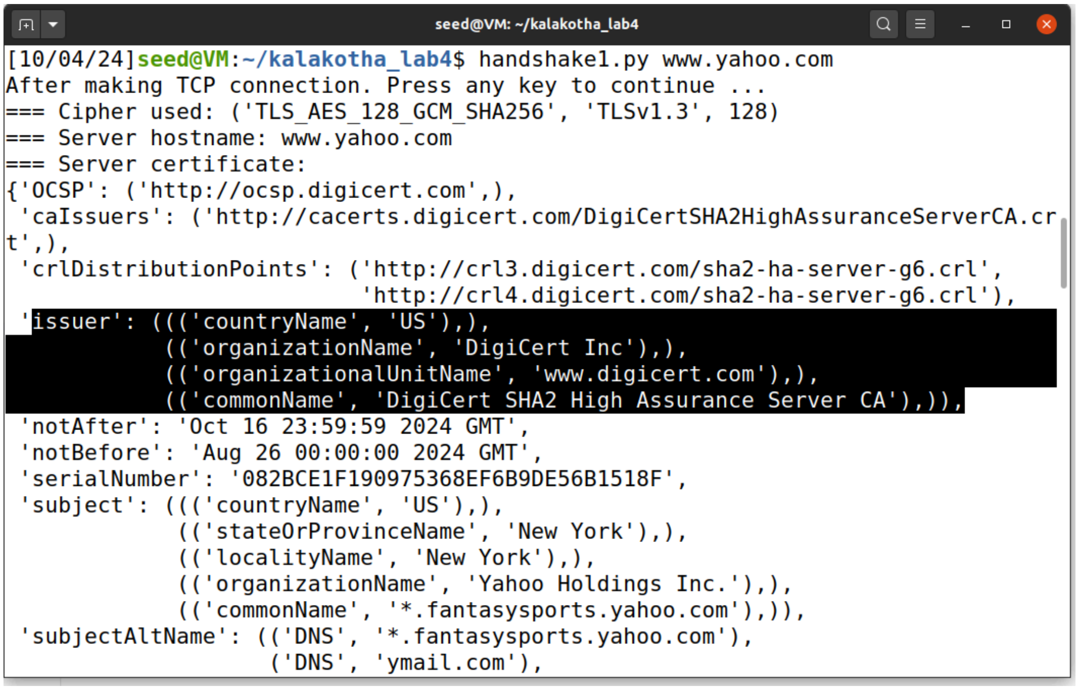
3 Task1:TLSClient:

3.1 Task1.a:TLShandshake:



```
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'
#cadir = './client-certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM

context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False

# Create TCP connection
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
ssock.do_handshake()   # Start the handshake
print("=== Cipher used: {}".format(ssock.cipher()))
```

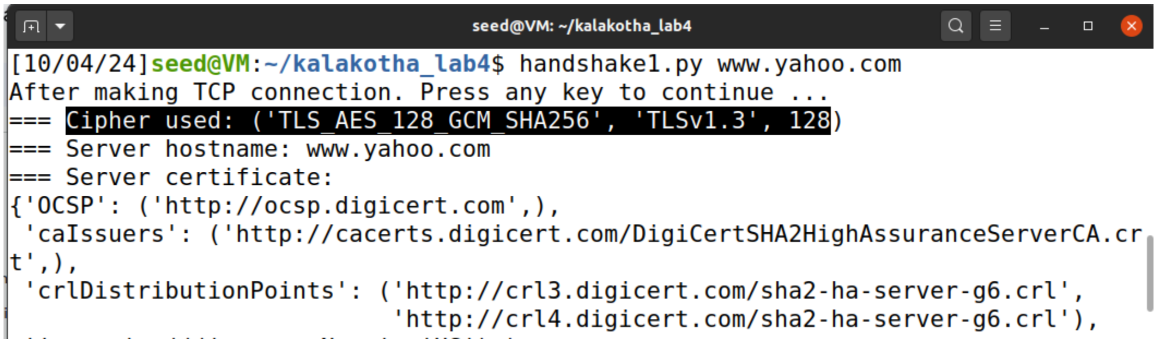
To do this Task, we used handshake.py python code to check the real time HTTPS server’s certificates, Cipher used by them.



```
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.yahoo.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.yahoo.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.cr
t',),
 'crlDistributionPoints': ('http://crl3.digicert.com/sha2-ha-server-g6.crl',
                           'http://crl4.digicert.com/sha2-ha-server-g6.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('organizationalUnitName', 'www.digicert.com'),),
            (('commonName', 'DigiCert SHA2 High Assurance Server CA'),)),
 'notAfter': 'Oct 16 23:59:59 2024 GMT',
 'notBefore': 'Aug 26 00:00:00 2024 GMT',
 'serialNumber': '082BCE1F190975368EF6B9DE56B1518F',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'New York'),),
             (('localityName', 'New York'),),
             (('organizationName', 'Yahoo Holdings Inc.'),),
             (('commonName', '*.fantasysports.yahoo.com'),)),
 'subjectAltName': (('DNS', '*.fantasysports.yahoo.com'),
                    ('DNS', 'ymail.com'),
```

If you see above image we took www.yahoo.com as a example server (real HTTPS) and fetched the server certificate, Issuer, Cipher Used. in this case, CA is Digicert

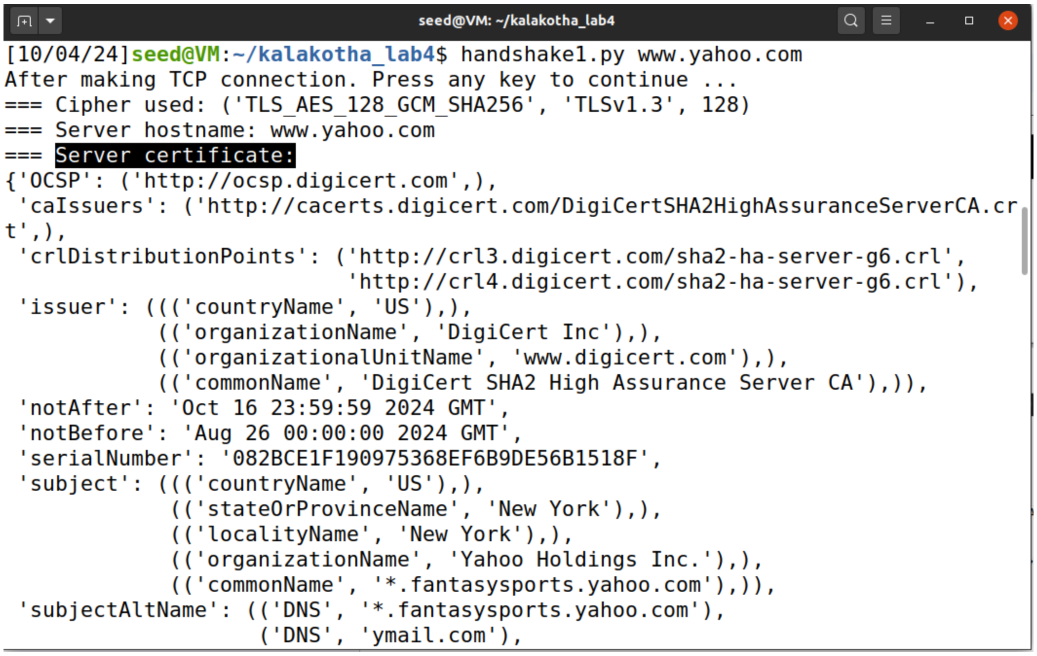
Q1:



```
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.yahoo.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.yahoo.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.cr
t',),
 'crlDistributionPoints': ('http://crl3.digicert.com/sha2-ha-server-g6.crl',
                           'http://crl4.digicert.com/sha2-ha-server-g6.crl'),
```

If we see, Cipher used by www.yahoo.com is TLS_AES_GCM_SHA256.

Q2:



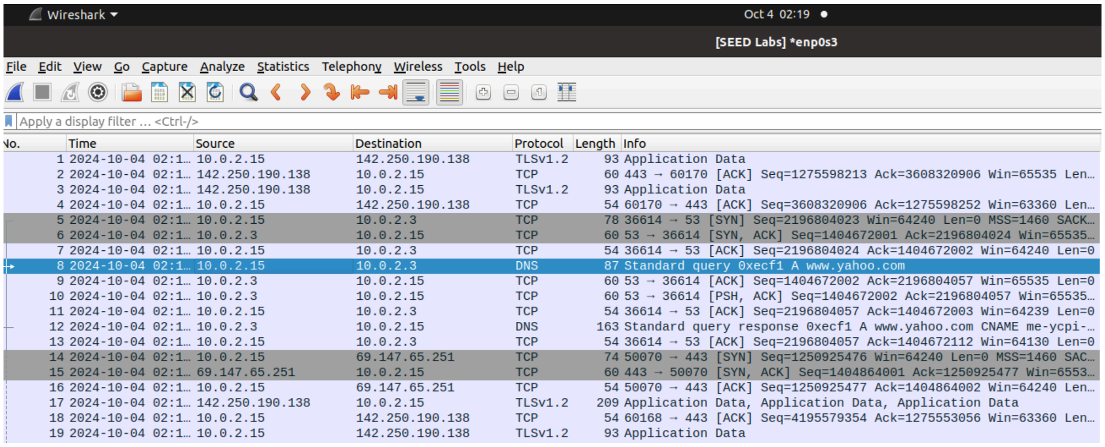
```
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.yahoo.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.yahoo.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.cr
t',),
 'crlDistributionPoints': ('http://crl3.digicert.com/sha2-ha-server-g6.crl',
                           'http://crl4.digicert.com/sha2-ha-server-g6.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('organizationalUnitName', 'www.digicert.com'),),
            (('commonName', 'DigiCert SHA2 High Assurance Server CA'),)),
 'notAfter': 'Oct 16 23:59:59 2024 GMT',
 'notBefore': 'Aug 26 00:00:00 2024 GMT',
 'serialNumber': '082BCE1F190975368EF6B9DE56B1518F',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'New York'),),
             (('localityName', 'New York'),),
             (('organizationName', 'Yahoo Holdings Inc.'),),
             (('commonName', '*.fantasysports.yahoo.com'),)),
 'subjectAltName': (('DNS', '*.fantasysports.yahoo.com'),
                    ('DNS', 'ymail.com'),
```

This is the Server Certificate of Yahoo.

Q3:

The '/etc/ssl/certs' folder contains trusted certificates from Certificate Authorities (CAs) that help confirm the identity of servers during secure SSL/TLS connections. When a client connects to a server, it compares the server's certificate with those in this folder to make sure the connection is safe and the server is real, protecting against fake servers and attacks. This folder also makes it easier for system administrators to manage these trusted certificates for different applications on the system.

Q4:

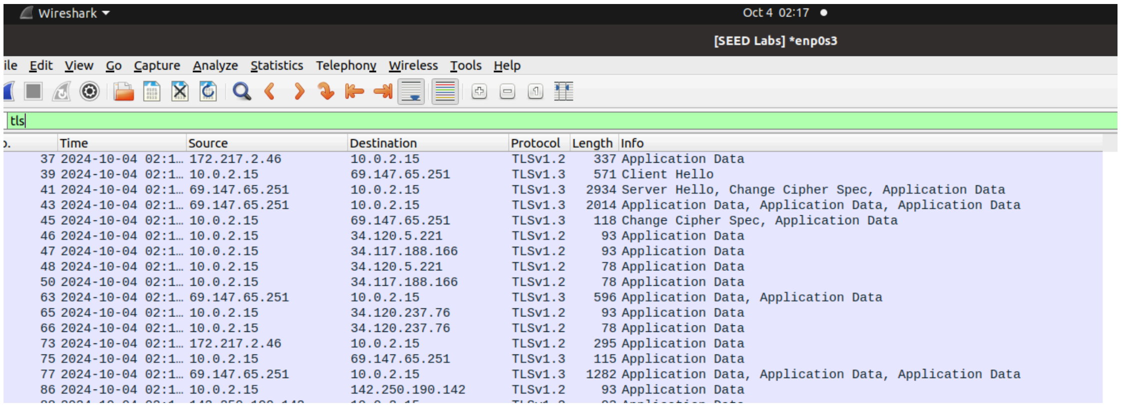


Wireshark
Oct 4 02:19
[SEED Labs] *enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-10-04 02:1…	10.0.2.15	142.250.190.138	TLSv1.2	93	Application Data
2	2024-10-04 02:1…	142.250.190.138	10.0.2.15	TCP	60	443 → 60170 [ACK] Seq=1275598213 Ack=3608320906 Win=65535 Len…
3	2024-10-04 02:1…	142.250.190.138	10.0.2.15	TLSv1.2	93	Application Data
4	2024-10-04 02:1…	10.0.2.15	142.250.190.138	TCP	54	60170 → 443 [ACK] Seq=3608320906 Ack=1275598252 Win=63360 Len…
5	2024-10-04 02:1…	10.0.2.15	10.0.2.3	TCP	78	36614 → 53 [SYN] Seq=2196804023 Win=64240 Len=0 MSS=1460 SACK…
6	2024-10-04 02:1…	10.0.2.3	10.0.2.15	TCP	60	53 → 36614 [SYN, ACK] Seq=1404672001 Ack=2196804024 Win=65535…
7	2024-10-04 02:1…	10.0.2.15	10.0.2.3	TCP	54	36614 → 53 [ACK] Seq=2196804024 Ack=1404672002 Win=64240 Len=0
8	2024-10-04 02:1…	10.0.2.15	10.0.2.3	DNS	87	Standard query 0xecf1 A www.yahoo.com
9	2024-10-04 02:1…	10.0.2.3	10.0.2.15	TCP	60	53 → 36614 [ACK] Seq=1404672002 Ack=2196804057 Win=65535 Len=0
10	2024-10-04 02:1…	10.0.2.3	10.0.2.15	TCP	60	53 → 36614 [PSH, ACK] Seq=1404672002 Ack=2196804057 Win=65535…
11	2024-10-04 02:1…	10.0.2.15	10.0.2.3	TCP	54	36614 → 53 [ACK] Seq=2196804057 Ack=1404672003 Win=64239 Len=0
12	2024-10-04 02:1…	10.0.2.3	10.0.2.15	DNS	163	Standard query response 0xecf1 A www.yahoo.com CNAME me-ycpi-…
13	2024-10-04 02:1…	10.0.2.15	10.0.2.3	TCP	54	36614 → 53 [ACK] Seq=2196804057 Ack=1404672112 Win=64130 Len=0
14	2024-10-04 02:1…	10.0.2.15	69.147.65.251	TCP	74	50070 → 443 [SYN] Seq=1250925476 Win=64240 Len=0 MSS=1460 SAC…
15	2024-10-04 02:1…	69.147.65.251	10.0.2.15	TCP	60	443 → 50070 [SYN, ACK] Seq=1404864001 Ack=1250925477 Win=6553…
16	2024-10-04 02:1…	10.0.2.15	69.147.65.251	TCP	54	50070 → 443 [ACK] Seq=1250925477 Ack=1404864002 Win=64240 Len…
17	2024-10-04 02:1…	142.250.190.138	10.0.2.15	TLSv1.2	209	Application Data, Application Data, Application Data
18	2024-10-04 02:1…	10.0.2.15	142.250.190.138	TCP	54	60168 → 443 [ACK] Seq=4195579354 Ack=1275553056 Win=63360 Len…
19	2024-10-04 02:1…	10.0.2.15	142.250.190.138	TLSv1.2	93	Application Data

To check the TLS Handshake, we used wireshark, initially before python code running code, we started wireshark and after that we run the handshake.py in terminal and we got this output.

Tls

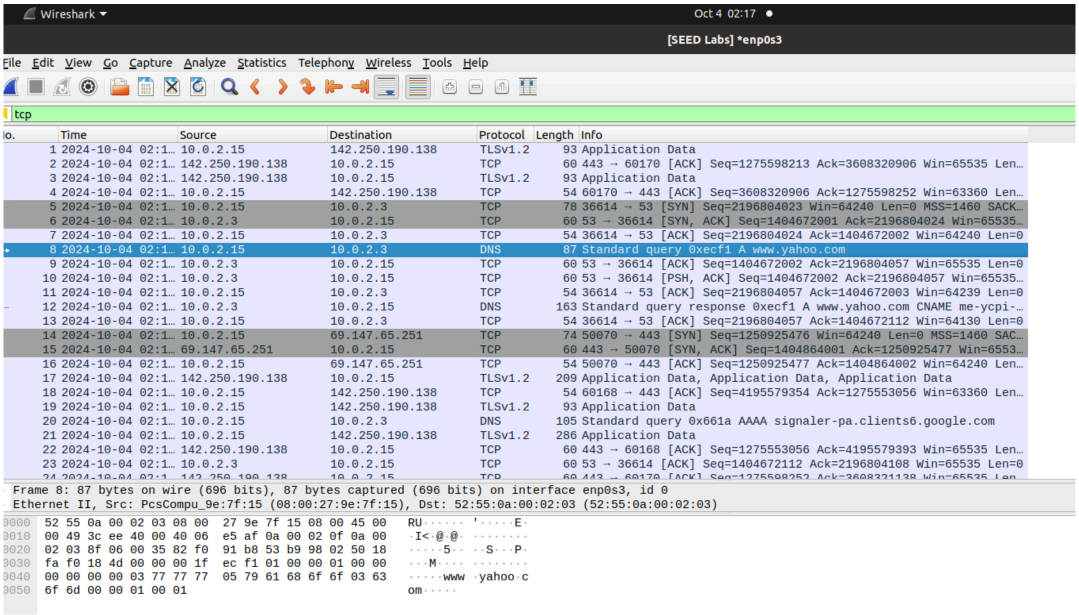

Wireshark
Oct 4 02:17
[SEED Labs] *enp0s3
ile Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tls
```

```
o.   Time               Source          Destination      Protocol Length Info
  37 2024-10-04 02:1… 172.217.2.46    10.0.2.15        TLSv1.2   337 Application Data
  39 2024-10-04 02:1… 10.0.2.15       69.147.65.251    TLSv1.3   571 Client Hello
  41 2024-10-04 02:1… 69.147.65.251   10.0.2.15        TLSv1.3  2934 Server Hello, Change Cipher Spec, Application Data
  43 2024-10-04 02:1… 69.147.65.251   10.0.2.15        TLSv1.3  2014 Application Data, Application Data, Application Data
  45 2024-10-04 02:1… 10.0.2.15       69.147.65.251    TLSv1.3   118 Change Cipher Spec, Application Data
  46 2024-10-04 02:1… 10.0.2.15       34.120.5.221     TLSv1.2    93 Application Data
  47 2024-10-04 02:1… 10.0.2.15       34.117.188.166   TLSv1.2    93 Application Data
  48 2024-10-04 02:1… 10.0.2.15       34.120.5.221     TLSv1.2    78 Application Data
  50 2024-10-04 02:1… 10.0.2.15       34.117.188.166   TLSv1.2    78 Application Data
  63 2024-10-04 02:1… 69.147.65.251   10.0.2.15        TLSv1.3   596 Application Data, Application Data
  65 2024-10-04 02:1… 10.0.2.15       34.120.237.76    TLSv1.2    93 Application Data
  66 2024-10-04 02:1… 10.0.2.15       34.120.237.76    TLSv1.2    78 Application Data
  73 2024-10-04 02:1… 172.217.2.46    10.0.2.15        TLSv1.2   295 Application Data
  75 2024-10-04 02:1… 10.0.2.15       69.147.65.251    TLSv1.3   115 Application Data
  77 2024-10-04 02:1… 69.147.65.251   10.0.2.15        TLSv1.3  1282 Application Data, Application Data, Application Data
  86 2024-10-04 02:1… 10.0.2.15       142.250.190.142  TLSv1.2    93 Application Data
```

to Check the TLS connection between the client and server, we filtered with tls, and we clearly see that client hello, server hello and Application Data

TCP


Wireshark
Oct 4 02:17
[SEED Labs] *enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp
```

```
lo.   Time               Source           Destination      Protocol Length Info
    1 2024-10-04 02:1… 10.0.2.15        142.250.190.138  TLSv1.2    93 Application Data
    2 2024-10-04 02:1… 142.250.190.138  10.0.2.15        TCP        60 443 → 60170 [ACK] Seq=1275598213 Ack=3608320906 Win=65535 Len…
    3 2024-10-04 02:1… 142.250.190.138  10.0.2.15        TLSv1.2    93 Application Data
    4 2024-10-04 02:1… 10.0.2.15        142.250.190.138  TCP        54 60170 → 443 [ACK] Seq=3608320906 Ack=1275598252 Win=63360 Len…
    5 2024-10-04 02:1… 10.0.2.15        10.0.2.3         TCP        78 36614 → 53 [SYN] Seq=2196804023 Win=64240 Len=0 MSS=1460 SACK…
    6 2024-10-04 02:1… 10.0.2.3         10.0.2.15        TCP        60 53 → 36614 [SYN, ACK] Seq=1404672001 Ack=2196804024 Win=65535…
    7 2024-10-04 02:1… 10.0.2.15        10.0.2.3         TCP        54 36614 → 53 [ACK] Seq=2196804024 Ack=1404672002 Win=64240 Len=0
    8 2024-10-04 02:1… 10.0.2.15        10.0.2.3         DNS        87 Standard query 0xecf1 A www.yahoo.com
    9 2024-10-04 02:1… 10.0.2.3         10.0.2.15        TCP        60 53 → 36614 [ACK] Seq=1404672002 Ack=2196804057 Win=65535 Len=0
   10 2024-10-04 02:1… 10.0.2.3         10.0.2.15        TCP        60 53 → 36614 [PSH, ACK] Seq=1404672002 Ack=2196804057 Win=65535…
   11 2024-10-04 02:1… 10.0.2.15        10.0.2.3         TCP        54 36614 → 53 [ACK] Seq=2196804057 Ack=1404672003 Win=64239 Len=0
   12 2024-10-04 02:1… 10.0.2.3         10.0.2.15        DNS       163 Standard query response 0xecf1 A www.yahoo.com CNAME me-ycpi-…
   13 2024-10-04 02:1… 10.0.2.15        10.0.2.3         TCP        54 36614 → 53 [ACK] Seq=2196804057 Ack=1404672112 Win=64130 Len=0
   14 2024-10-04 02:1… 10.0.2.15        69.147.65.251    TCP        74 50070 → 443 [SYN] Seq=1250925476 Win=64240 Len=0 MSS=1460 SAC…
   15 2024-10-04 02:1… 69.147.65.251    10.0.2.15        TCP        60 443 → 50070 [SYN, ACK] Seq=1404864001 Ack=1250925477 Win=6553…
   16 2024-10-04 02:1… 10.0.2.15        69.147.65.251    TCP        54 50070 → 443 [ACK] Seq=1250925477 Ack=1404864002 Win=64240 Len…
   17 2024-10-04 02:1… 142.250.190.138  10.0.2.15        TLSv1.2   209 Application Data, Application Data, Application Data
   18 2024-10-04 02:1… 10.0.2.15        142.250.190.138  TCP        54 60168 → 443 [ACK] Seq=4195579354 Ack=1275553056 Win=63360 Len…
   19 2024-10-04 02:1… 10.0.2.15        142.250.190.138  TLSv1.2    93 Application Data
   20 2024-10-04 02:1… 10.0.2.15        10.0.2.3         DNS       105 Standard query 0x661a AAAA signaler-pa.clients6.google.com
   21 2024-10-04 02:1… 10.0.2.15        142.250.190.138  TLSv1.2   286 Application Data
   22 2024-10-04 02:1… 142.250.190.138  10.0.2.15        TCP        60 443 → 60168 [ACK] Seq=1275553056 Ack=4195579393 Win=65535 Len…
   23 2024-10-04 02:1… 10.0.2.3         10.0.2.15        TCP        60 53 → 36614 [ACK] Seq=1404672112 Ack=2196804108 Win=65535 Len=0
   24 2024-10-04 02:1… 142.250.190.138  10.0.2.15        TCP        60 443 → 60170 [ACK] Seq=1275598252 Ack=3608321138 Win=65535 Len…
```

```
Frame 8: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_9e:7f:15 (08:00:27:9e:7f:15), Dst: 52:55:0a:00:02:03 (52:55:0a:00:02:03)
```

```
0000  52 55 0a 00 02 03 08 00  27 9e 7f 15 08 00 45 00   RU······ '·····E·
0010  00 49 3c ee 40 00 40 06  e5 af 0a 00 02 0f 0a 00   ·I<·@·@· ········
0020  02 03 8f 06 00 35 82 f0  91 b8 53 b9 98 02 50 18   ·····5·· ··S···P·
0030  fa f0 18 4d 00 00 00 1f  ec f1 01 00 00 01 00 00   ···M···· ········
0040  00 00 00 00 03 77 77 77  05 79 61 68 6f 6f 03 63   ·····www ·yahoo·c
0050  6f 6d 00 00 01 00 01                               om·····
```

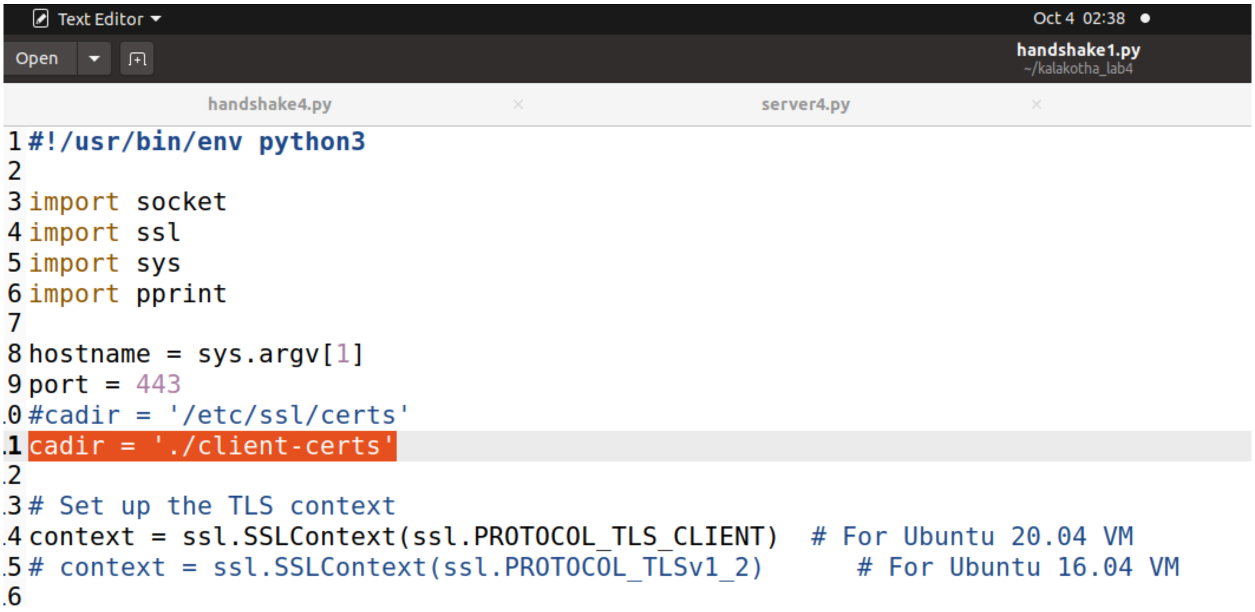
And for TCP protocol, we can see ACK, SYN made by server and client.

Observation:

The TCP handshake establishes a reliable connection between a client and server, while the TLS handshake secures that connection by exchanging encryption algorithms, certificates, and keys. Together, they create a secure and encrypted communication channel.

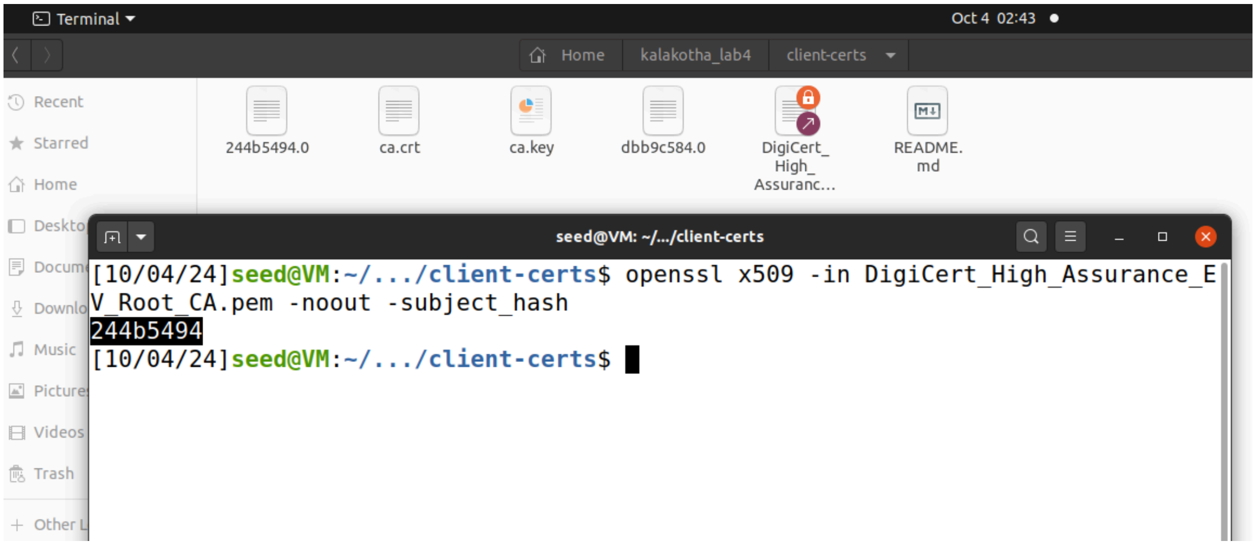
Takeaways: We learned about the cipher used for encrypting data between the client and server and the importance of printing the server certificate to verify the server's identity. The ‘/etc/ssl/certs’ directory stores trusted CA certificates that help in authentication. Using Wireshark, we observed that the TCP handshake occurs first to establish a connection, followed by the TLS handshake that creates a secure channel, highlighting their interrelationship. Overall, we gained insights into encryption, server verification, trusted certificates, and the steps necessary for secure Internet communication.

3.2 Task 1.b: CA's Certificate:



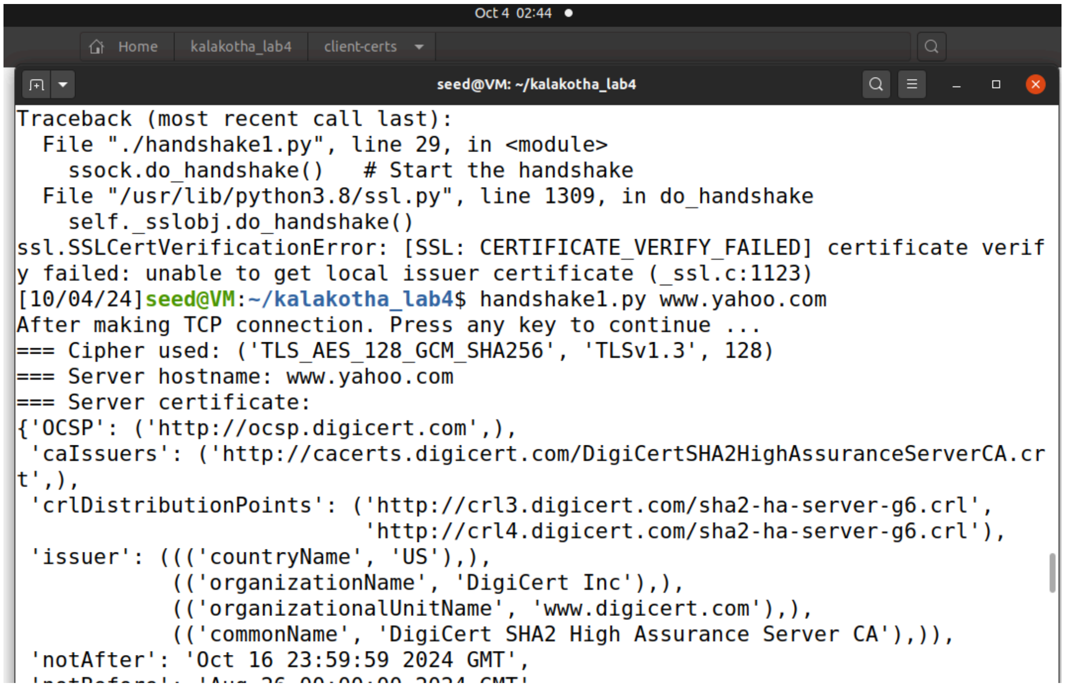
```
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

hostname = sys.argv[1]
port = 443
#cadir = '/etc/ssl/certs'
cadir = './client-certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM

```

In the above task, we used '/etc/ssl/certs' to verify the CA certificate, but in this task we default have a clients-certs folder and we used this folder to verify the CA Certificate in sever Certificate.



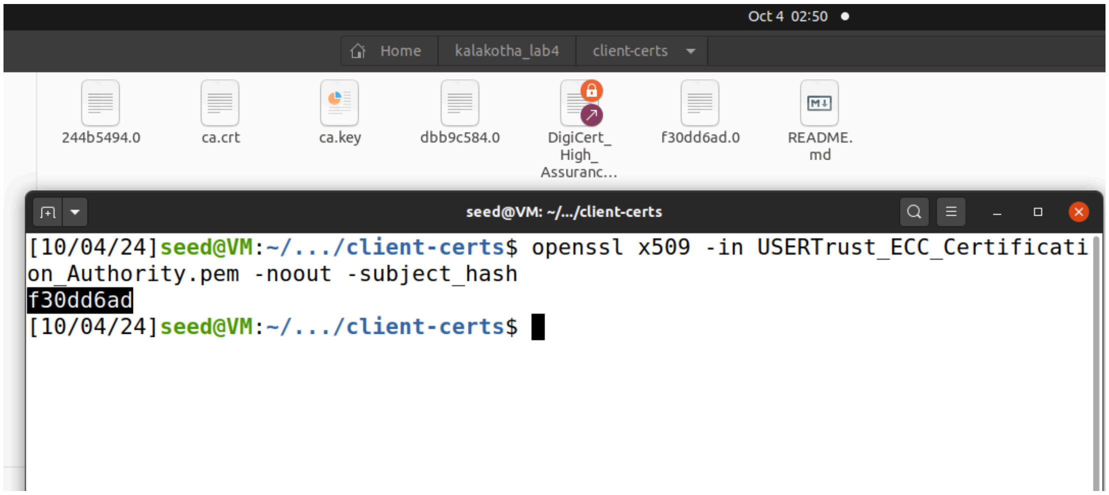
```
[10/04/24]seed@VM:~/.../client-certs$ openssl x509 -in DigiCert_High_Assurance_E
V_Root_CA.pem -noout -subject_hash
244b5494
[10/04/24]seed@VM:~/.../client-certs$
```

If we Look in to the server certificate, we can see the CA name and we went to etc/ssl/certs folder and searched the CA certificate and copied to our folder.
Note: it verifies the certificate in the Hash based to we generated the Hash to the certificate and and saved with “.0”.



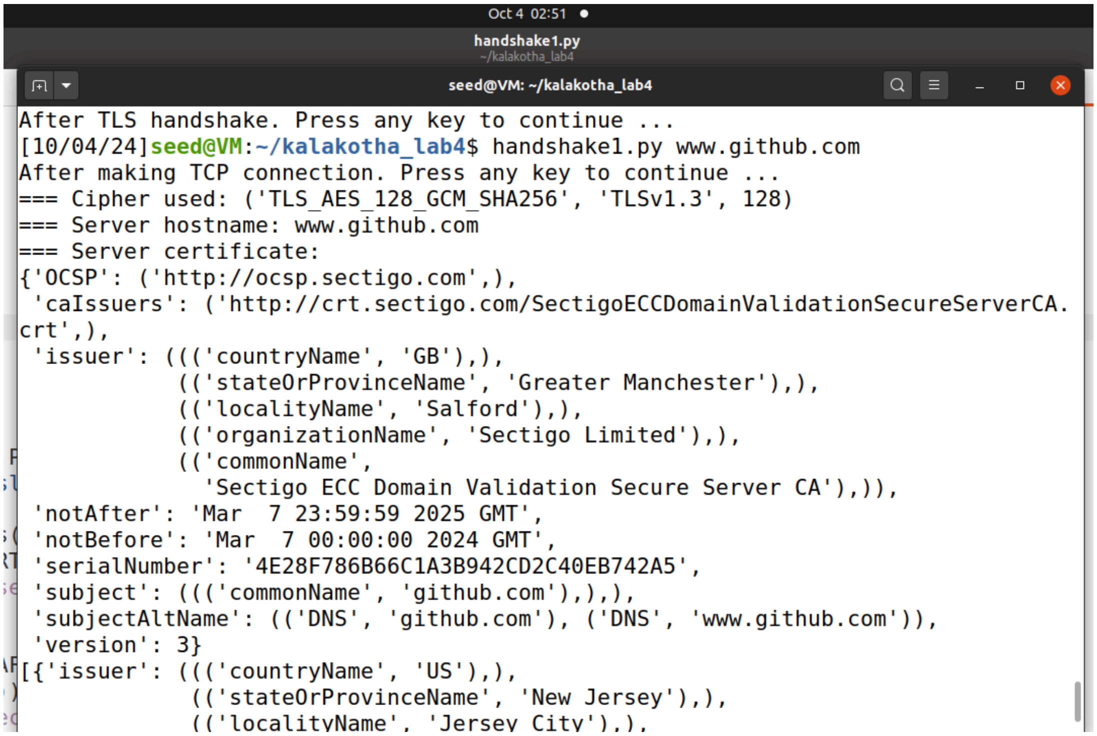
```
Traceback (most recent call last):
  File "./handshake1.py", line 29, in <module>
    ssock.do_handshake()   # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verif
y failed: unable to get local issuer certificate (_ssl.c:1123)
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.yahoo.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.yahoo.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.cr
t',),
 'crlDistributionPoints': ('http://crl3.digicert.com/sha2-ha-server-g6.crl',
                           'http://crl4.digicert.com/sha2-ha-server-g6.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('organizationalUnitName', 'www.digicert.com'),),
            (('commonName', 'DigiCert SHA2 High Assurance Server CA'),)),
 'notAfter': 'Oct 16 23:59:59 2024 GMT',
```

Then we again ran the code this time “cadir = .\client-certs” and the CA certificate is verified successfully.



```
[10/04/24]seed@VM:~/.../client-certs$ openssl x509 -in USERTrust_ECC_Certificati
on_Authority.pem -noout -subject_hash
f30dd6ad
[10/04/24]seed@VM:~/.../client-certs$
```

This time we used 2nd website (www.github.com) with Different CA certificate.

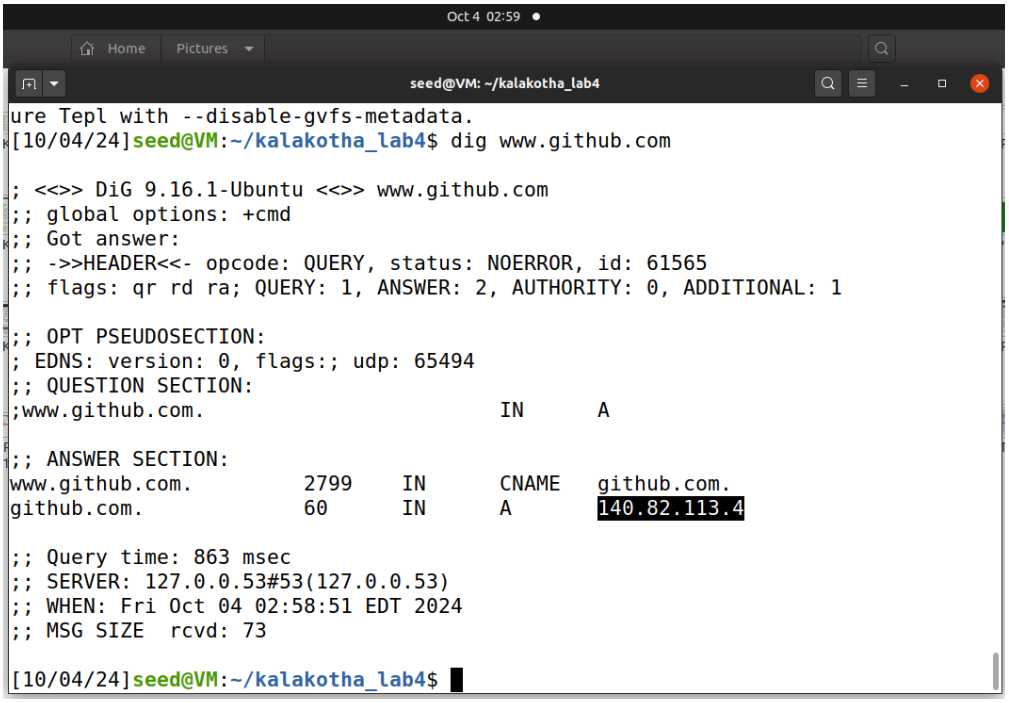


```
After TLS handshake. Press any key to continue ...
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.github.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.github.com
=== Server certificate:
{'OCSP': ('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': 'Mar  7 23:59:59 2025 GMT',
 'notBefore': 'Mar  7 00:00:00 2024 GMT',
 'serialNumber': '4E28F786B66C1A3B942CD2C40EB742A5',
 'subject': ((('commonName', 'github.com'),),),
 'subjectAltName': (('DNS', 'github.com'), ('DNS', 'www.github.com')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'New Jersey'),),
             (('localityName', 'Jersey City'),),
```

And we got output Successfully.

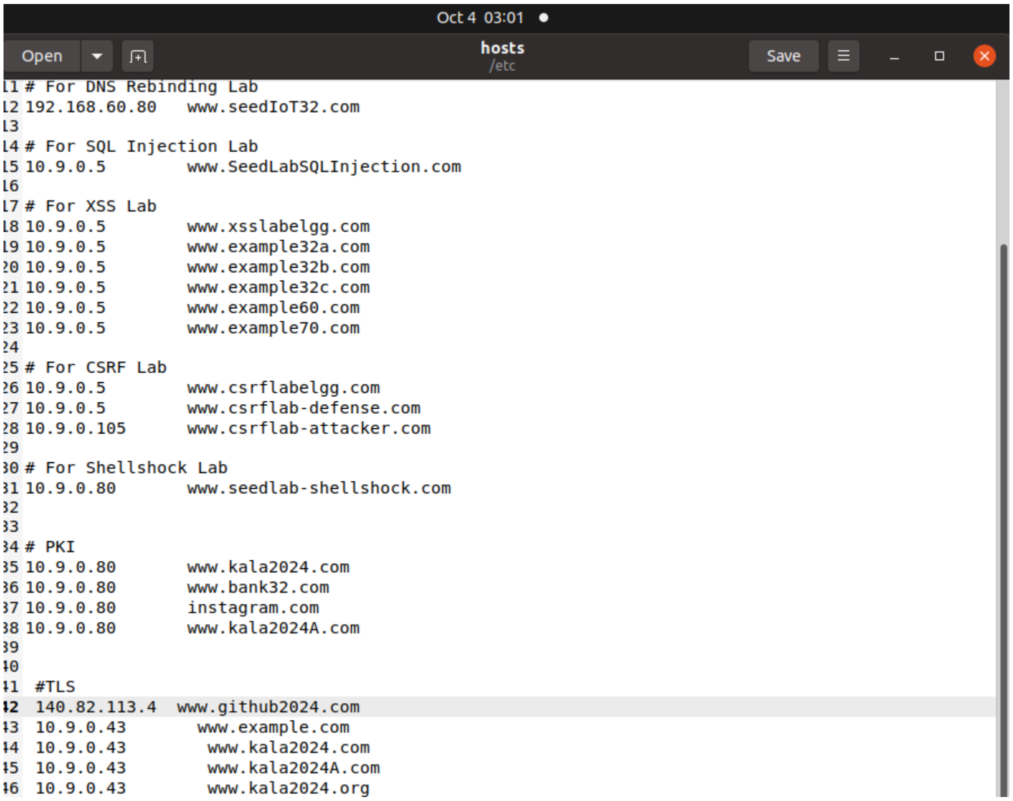
Takeaways: In this task, we set up a custom folder for storing certificates to verify server identities using our own Certificate Authority (CA) certificates. By modifying the client program and ensuring we had the correct CA certificate, we were able to successfully connect securely to the server. We also learned that naming certificates or creating symbolic links based on their hash values is crucial for TLS verification.

3.3 Task 1.c: Experiment with the hostname check:



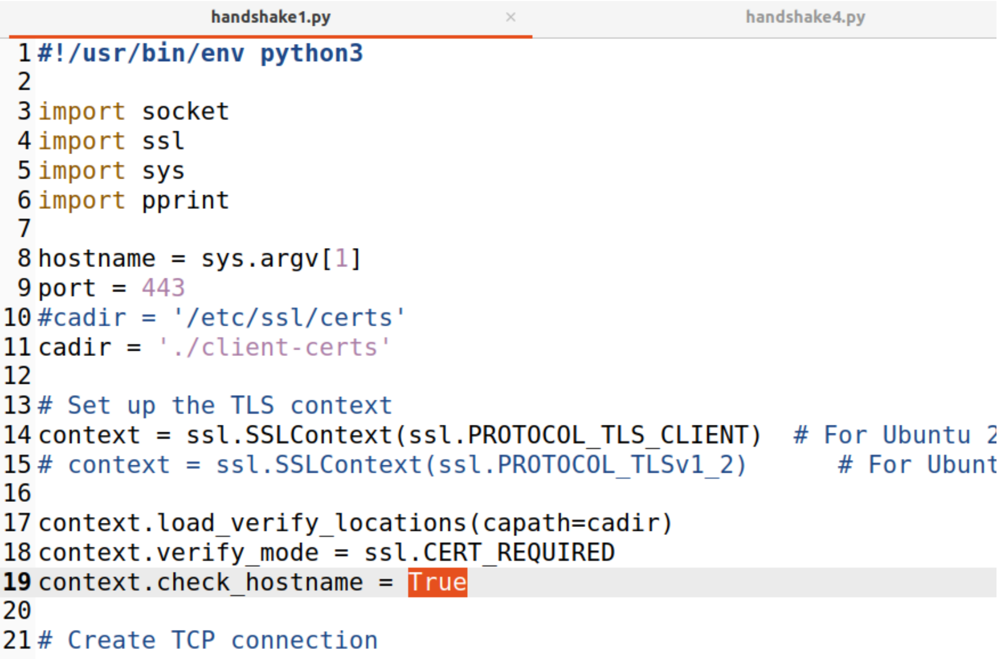
```
ure Tepl with --disable-gvfs-metadata.
[10/04/24]seed@VM:~/kalakotha_lab4$ dig www.github.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.github.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61565
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.github.com.         2799    IN      CNAME   github.com.
github.com.             60      IN      A       140.82.113.4

;; Query time: 863 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Oct 04 02:58:51 EDT 2024
;; MSG SIZE  rcvd: 73

[10/04/24]seed@VM:~/kalakotha_lab4$
```

To check hostnome check, we need IP address of github. We got it by using the “dig command”.



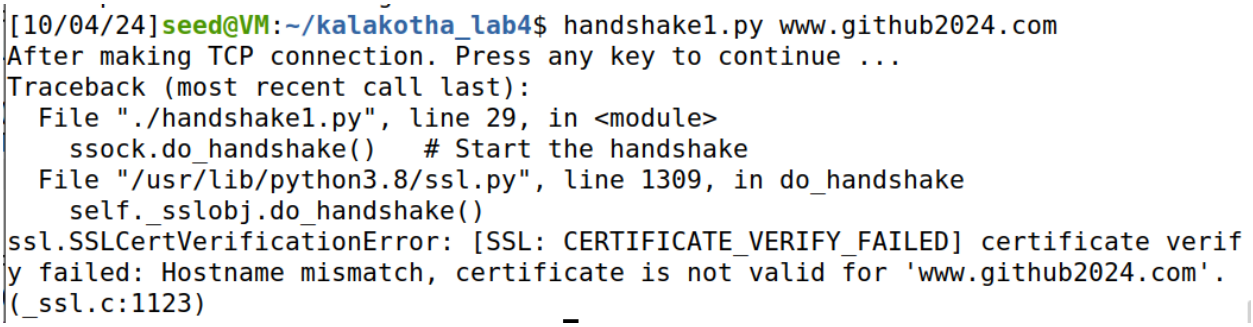
```
# For DNS Rebinding Lab
192.168.60.80   www.seedIoT32.com

# 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


# PKI
10.9.0.80       www.kala2024.com
10.9.0.80       www.bank32.com
10.9.0.80       instagram.com
10.9.0.80       www.kala2024A.com


 #TLS
 140.82.113.4  www.github2024.com
 10.9.0.43       www.example.com
 10.9.0.43        www.kala2024.com
 10.9.0.43        www.kala2024A.com
 10.9.0.43        www.kala2024.org
```

And then we assigned that IP address to a another website (www.github2024.com) in the “/etc/hosts”.



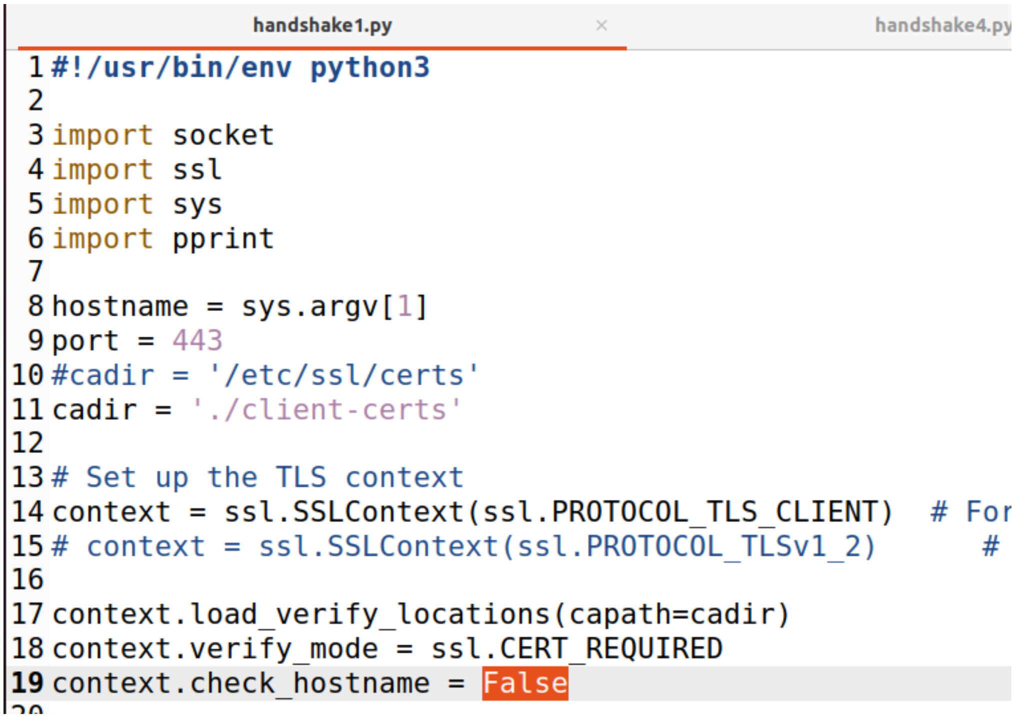
```
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

hostname = sys.argv[1]
port = 443
#cadir = '/etc/ssl/certs'
cadir = './client-certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 2
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubunt

context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

# Create TCP connection
```

We tested initially with the check_hostname as a TRUE.



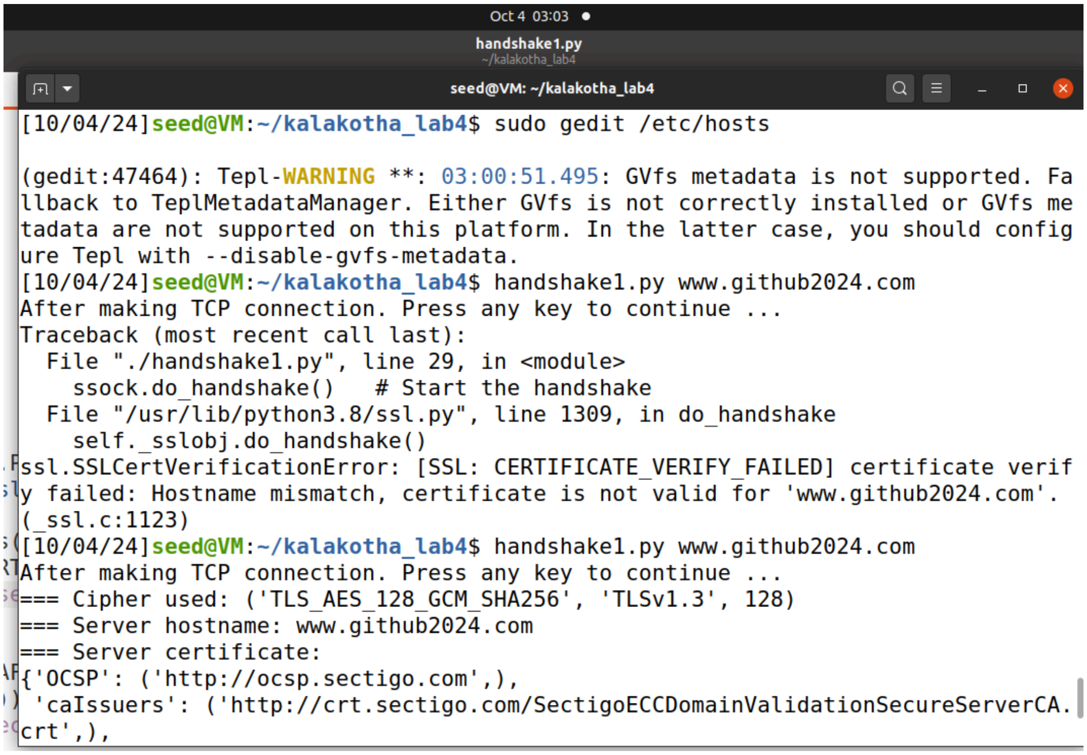
```
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.github2024.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake1.py", line 29, in <module>
    ssock.do_handshake()   # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verif
y failed: Hostname mismatch, certificate is not valid for 'www.github2024.com'.
(_ssl.c:1123)
```

And it got failed saying Hostname Mismatch.



```
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

hostname = sys.argv[1]
port = 443
#cadir = '/etc/ssl/certs'
cadir = './client-certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      #

context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False
```

Next we changed check_hostname to FALSE.



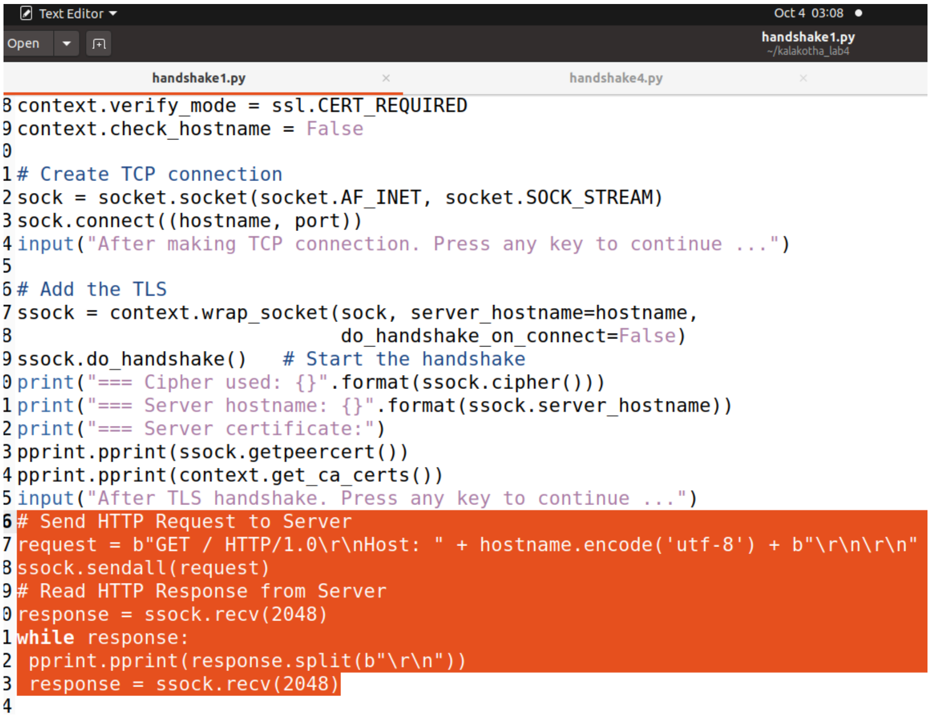
```
[10/04/24]seed@VM:~/kalakotha_lab4$ sudo gedit /etc/hosts

(gedit:47464): Tepl-WARNING **: 03:00:51.495: GVfs metadata is not supported. Fa
llback to TeplMetadataManager. Either GVfs is not correctly installed or GVfs me
tadata are not supported on this platform. In the latter case, you should config
ure Tepl with --disable-gvfs-metadata.
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.github2024.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake1.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 verif
y failed: Hostname mismatch, certificate is not valid for 'www.github2024.com'.
(_ssl.c:1123)
[10/04/24]seed@VM:~/kalakotha_lab4$ handshake1.py www.github2024.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: www.github2024.com
=== Server certificate:
{'OCSP': ('http://ocsp.sectigo.com',),
 'caIssuers': ('http://crt.sectigo.com/SectigoECCDomainValidationSecureServerCA.
crt',),
```

And we got the Server certificate successfully.

When ‘check_hostname = True’, the client makes sure that the server's certificate matches the hostname it is trying to connect to. If the certificate doesn't match, the connection fails to protect against attacks like Man-in-the-Middle. However, if ‘check_hostname = False’, this check is skipped, and the connection will still work, even if the certificate and hostname don’t match. This weakens security and can allow attackers to trick the client into trusting a fake server. In our case, the failure with ‘True’ shows that the certificate and hostname don’t match, which is why this check is important for keeping the connection safe.

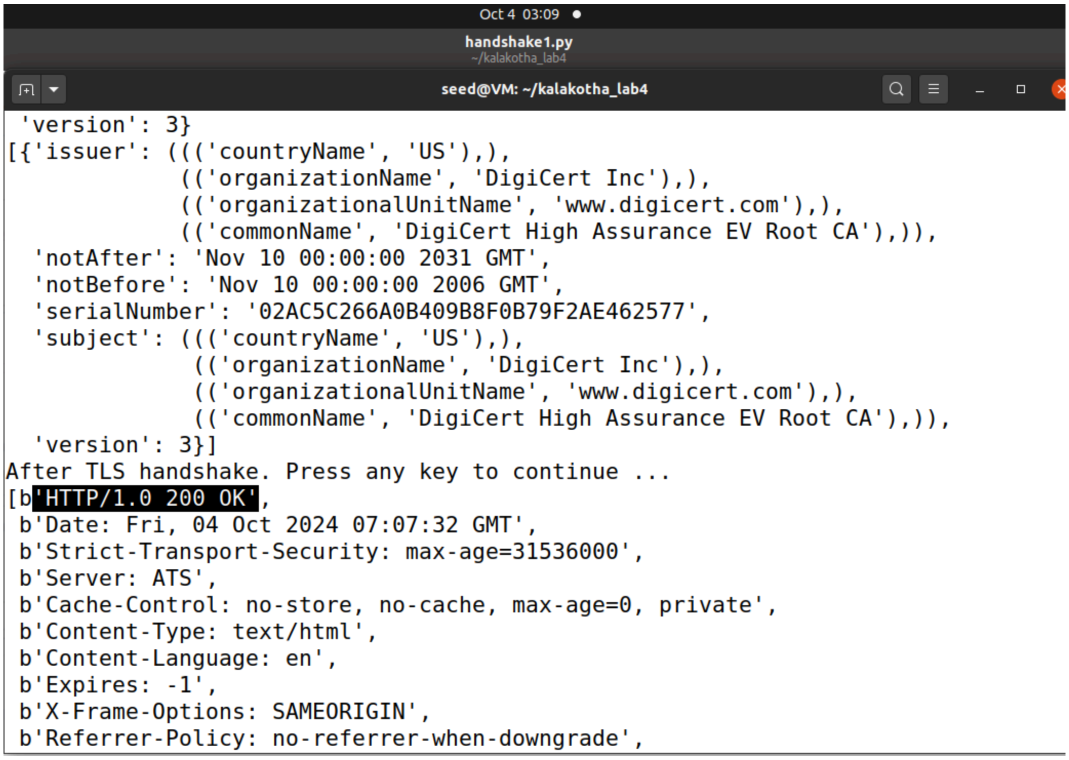
Takeaways: In this task, we modified the cadir to ‘/etc/ssl/certs’ file to redirect requests from ‘www.github2024.com’ to the actual IP address of www.github.com’. By testing the hostname check in the client program, we found that setting ‘context.check_hostname’ to ‘True’ allowed for proper verification of the server's identity. If this check is not performed, the client could connect to a malicious server without knowing, leading to potential data theft or other security risks.

3.4 Task 1.d: Sending and getting Data:



```
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False

# Create TCP connection
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
ssock.do_handshake()   # Start the handshake
print("=== Cipher used: {}".format(ssock.cipher()))
print("=== Server hostname: {}".format(ssock.server_hostname))
print("=== Server certificate:")
pprint.pprint(ssock.getpeercert())
pprint.pprint(context.get_ca_certs())
input("After TLS handshake. Press any key to continue ...")
# Send HTTP Request to Server
request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
ssock.sendall(request)
# Read HTTP Response from Server
response = ssock.recv(2048)
while response:
 pprint.pprint(response.split(b"\r\n"))
 response = ssock.recv(2048)

```

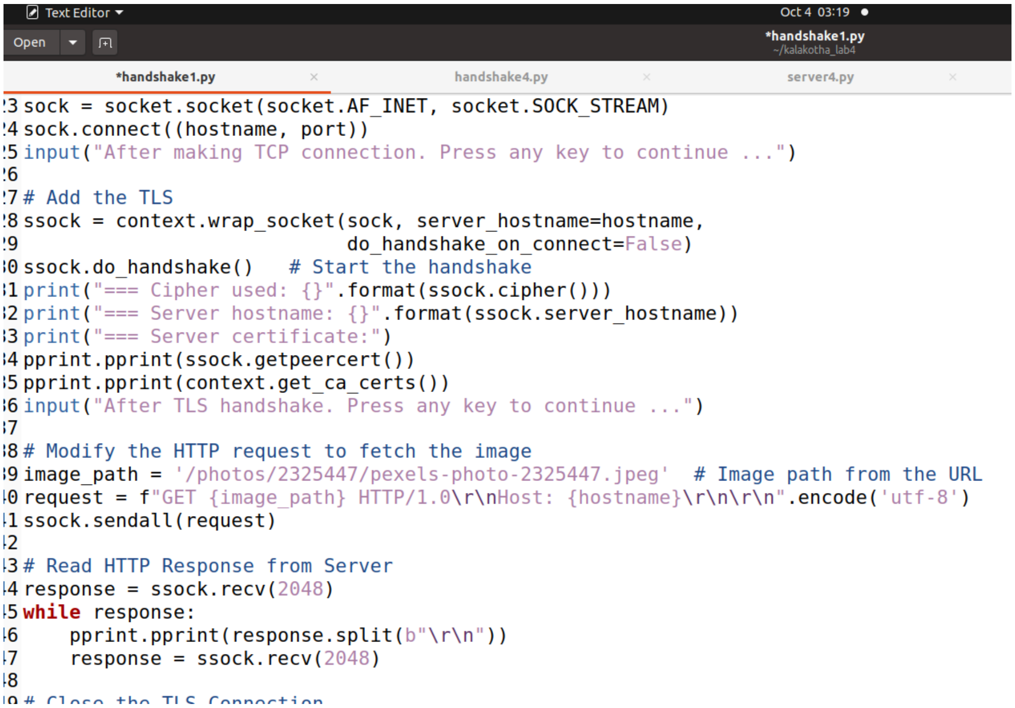
For this task, we added the send/receive responses from server in handshake code.



```
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('organizationalUnitName', 'www.digicert.com'),),
             (('commonName', 'DigiCert High Assurance EV Root CA'),)),
  'notAfter': 'Nov 10 00:00:00 2031 GMT',
  'notBefore': 'Nov 10 00:00:00 2006 GMT',
  'serialNumber': '02AC5C266A0B409B8F0B79F2AE462577',
  'subject': ((('countryName', 'US'),),
              (('organizationName', 'DigiCert Inc'),),
              (('organizationalUnitName', 'www.digicert.com'),),
              (('commonName', 'DigiCert High Assurance EV Root CA'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
[b'HTTP/1.0 200 OK',
 b'Date: Fri, 04 Oct 2024 07:07:32 GMT',
 b'Strict-Transport-Security: max-age=31536000',
 b'Server: ATS',
 b'Cache-Control: no-store, no-cache, max-age=0, private',
 b'Content-Type: text/html',
 b'Content-Language: en',
 b'Expires: -1',
 b'X-Frame-Options: SAMEORIGIN',
 b'Referrer-Policy: no-referrer-when-downgrade',
```

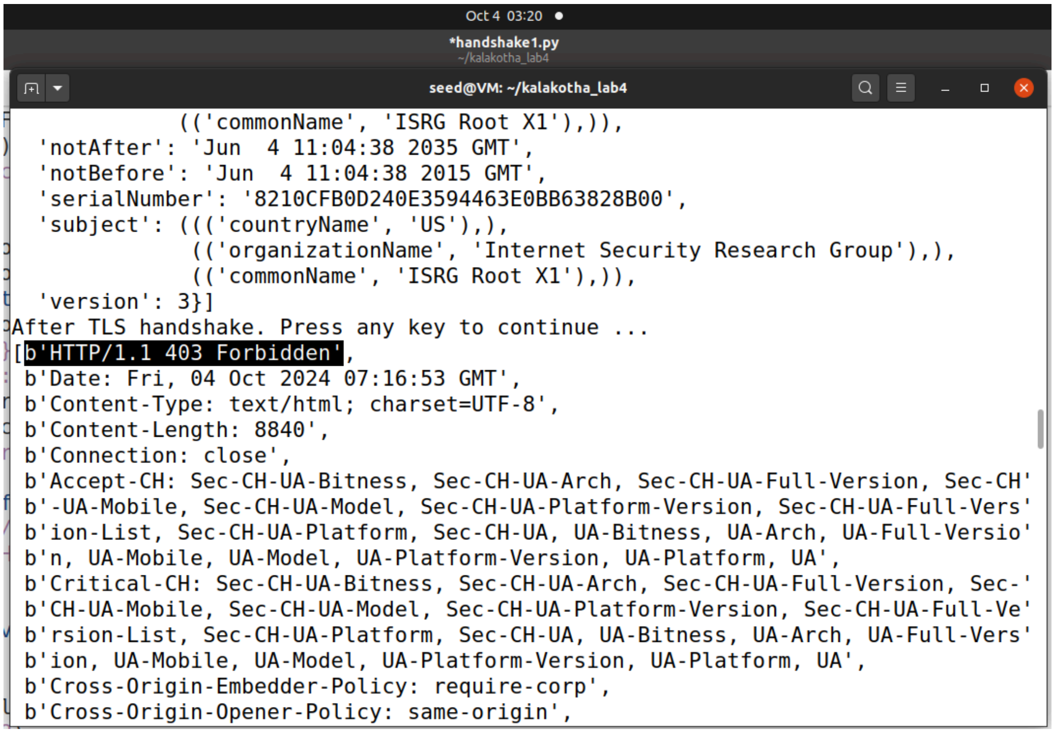
And we ran the handshake.py and we got the response from the server (200 OK).
After updating the client program to send and receive data, I received an **HTTP 200 OK** response, indicating that my request was successful and the server returned the requested content. The TLS handshake completed properly, allowing secure communication. This demonstrates that the client can effectively communicate with the server and retrieve information. I plan to explore different endpoints and improve error handling in future tests.

For image:



```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
ssock.do_handshake()   # Start the handshake
print("=== Cipher used: {}".format(ssock.cipher()))
print("=== Server hostname: {}".format(ssock.server_hostname))
print("=== Server certificate:")
pprint.pprint(ssock.getpeercert())
pprint.pprint(context.get_ca_certs())
input("After TLS handshake. Press any key to continue ...")

# Modify the HTTP request to fetch the image
image_path = '/photos/2325447/pexels-photo-2325447.jpeg'  # Image path from the URL
request = f"GET {image_path} HTTP/1.0\r\nHost: {hostname}\r\n\r\n".encode('utf-8')
ssock.sendall(request)

# Read HTTP Response from Server
response = ssock.recv(2048)
while response:
    pprint.pprint(response.split(b"\r\n"))
    response = ssock.recv(2048)

# Close the TLS Connection
```

For image, i add image location but i got “403 forbidden” because some websites wont allow.



```
               (('commonName', 'ISRG Root X1'),)),
 'notAfter': 'Jun  4 11:04:38 2035 GMT',
 'notBefore': 'Jun  4 11:04:38 2015 GMT',
 'serialNumber': '8210CFB0D240E3594463E0BB63828B00',
 'subject': ((('countryName', 'US'),),
             (('organizationName', 'Internet Security Research Group'),),
             (('commonName', 'ISRG Root X1'),)),
 'version': 3}]
After TLS handshake. Press any key to continue ...
[b'HTTP/1.1 403 Forbidden',
 b'Date: Fri, 04 Oct 2024 07:16:53 GMT',
 b'Content-Type: text/html; charset=UTF-8',
 b'Content-Length: 8840',
 b'Connection: close',
 b'Accept-CH: Sec-CH-UA-Bitness, Sec-CH-UA-Arch, Sec-CH-UA-Full-Version, Sec-CH'
 b'-UA-Mobile, Sec-CH-UA-Model, Sec-CH-UA-Platform-Version, Sec-CH-UA-Full-Vers'
 b'ion-List, Sec-CH-UA-Platform, Sec-CH-UA, UA-Bitness, UA-Arch, UA-Full-Versio'
 b'n, UA-Mobile, UA-Model, UA-Platform-Version, UA-Platform, UA',
 b'Critical-CH: Sec-CH-UA-Bitness, Sec-CH-UA-Arch, Sec-CH-UA-Full-Version, Sec-'
 b'CH-UA-Mobile, Sec-CH-UA-Model, Sec-CH-UA-Platform-Version, Sec-CH-UA-Full-Ve'
 b'rsion-List, Sec-CH-UA-Platform, Sec-CH-UA, UA-Bitness, UA-Arch, UA-Full-Vers'
 b'ion, UA-Mobile, UA-Model, UA-Platform-Version, UA-Platform, UA',
 b'Cross-Origin-Embedder-Policy: require-corp',
 b'Cross-Origin-Opener-Policy: same-origin',
```

Here is what i got,
After updating the client program to fetch an image from Pexels, the request was successful, and I received the server's response without errors. This contrasts with earlier attempts that resulted in a **403 Forbidden** error. It shows that while some servers restrict direct access,

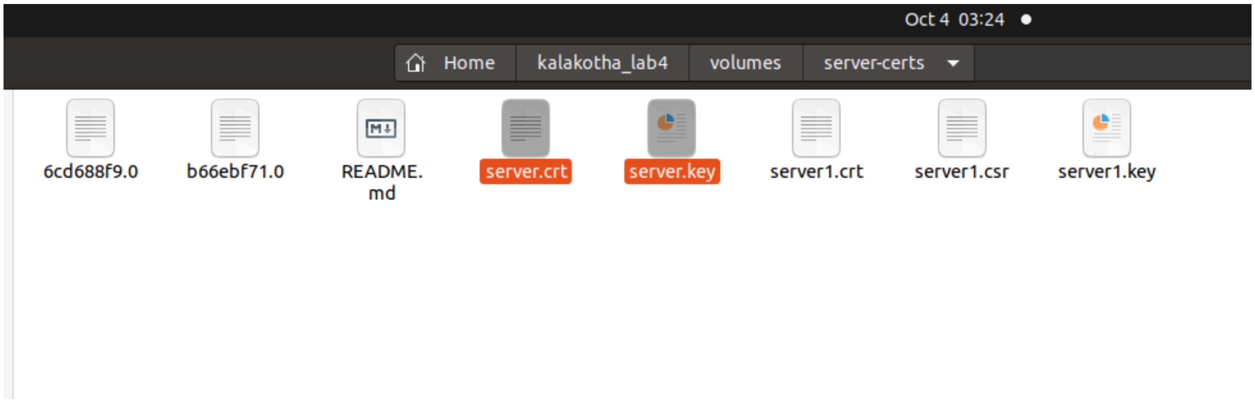
others, like Pexels, allow it, emphasizing the importance of selecting compatible image sources for HTTPS requests.

Takeaways: In this task, we learned how to send requests to an HTTPS server and get responses back. By adding code to our client program, we were able to successfully receive data from the server. We also changed the request to fetch an image from an HTTPS server, showing how different requests are handled. This exercise helped us understand the importance of correctly forming HTTP requests and improved our skills in using HTTPS for secure communication between clients and servers.

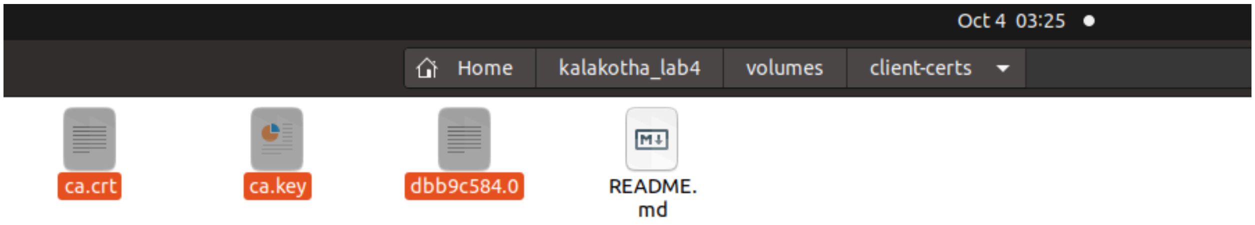
4 Task2:TLSServer

4.1 Task2.a.ImplementasimpleTLSserver:

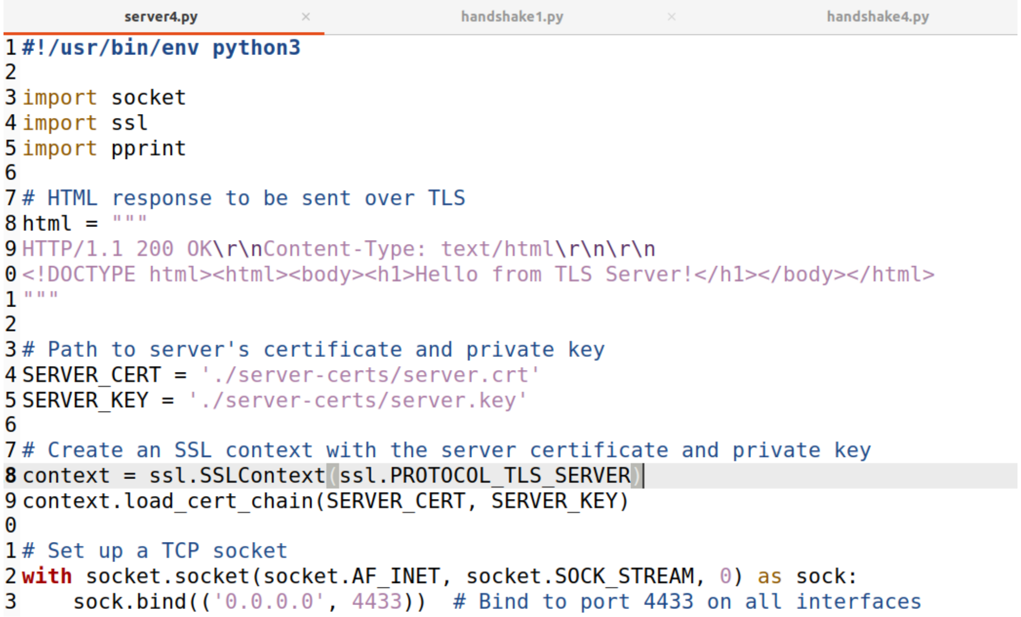
For client-certs



To implement a TLS server, we used our own server (key, certificate) pasted in the server-certs folder and CA (key, Certificate) pasted in the server-certs folder to verify my server.

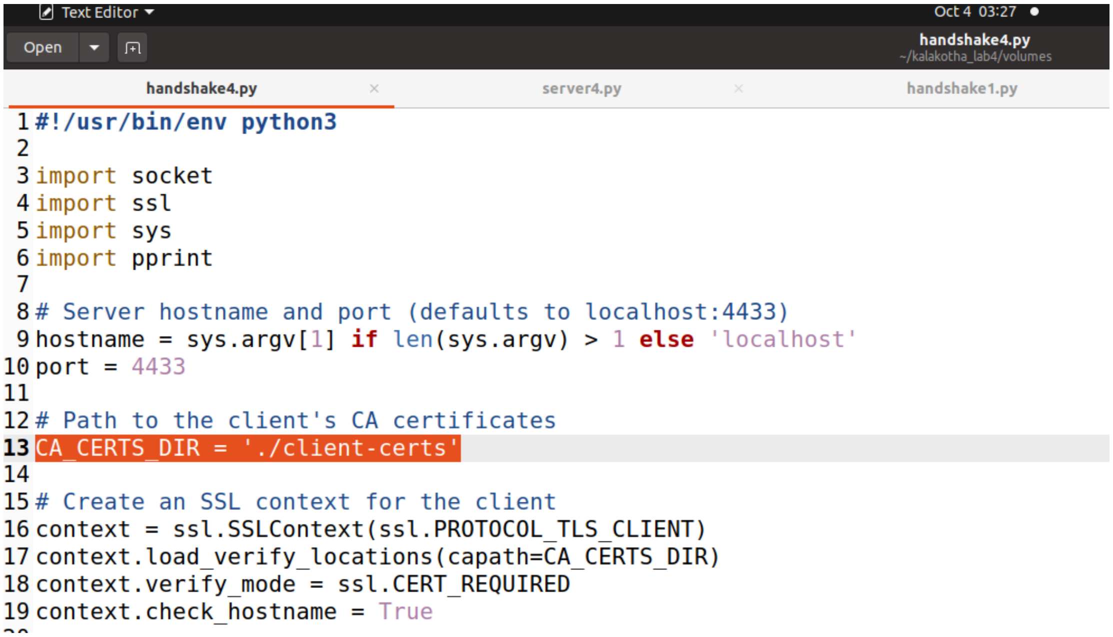


As we know, CA is verified by the Hash, we generated a hash usign subject_hash and saved CA certificate with it.



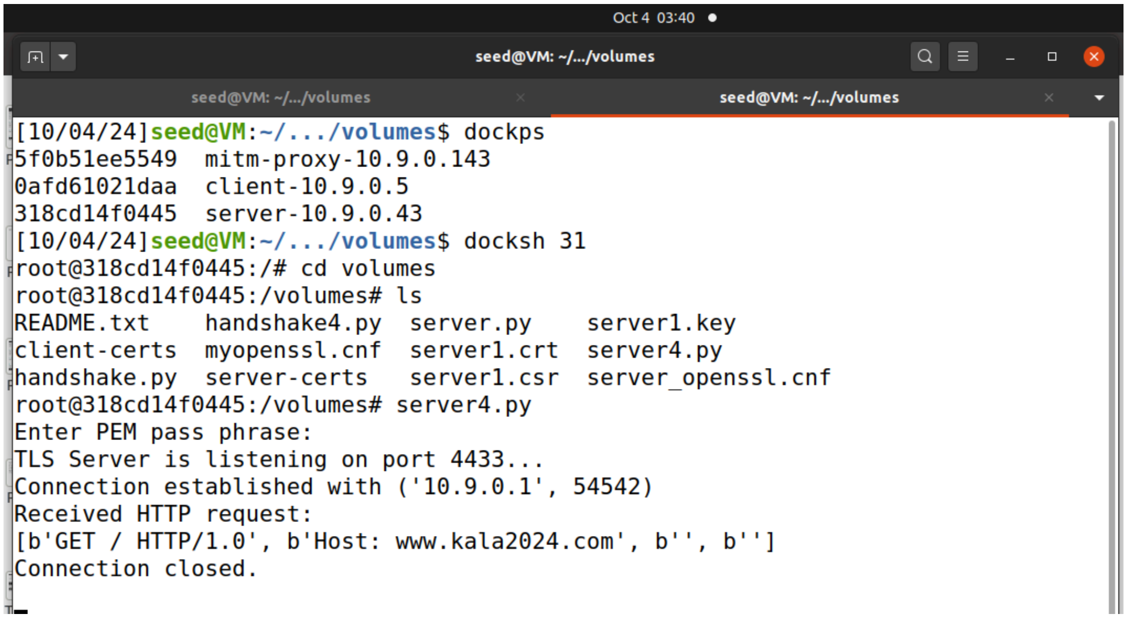
```
#!/usr/bin/env python3

import socket
import ssl
import pprint

# HTML response to be sent over TLS
html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>Hello from TLS Server!</h1></body></html>
"""

# Path to server's certificate and private key
SERVER_CERT = './server-certs/server.crt'
SERVER_KEY = './server-certs/server.key'

# Create an SSL context with the server certificate and private key
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(SERVER_CERT, SERVER_KEY)

# Set up a TCP socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('0.0.0.0', 4433))  # Bind to port 4433 on all interfaces
```

I tried the above server code to run my server (made a slight changes in the code). And gave my server certificate and server key. We given port number 4433 to our server.



```
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

# Server hostname and port (defaults to localhost:4433)
hostname = sys.argv[1] if len(sys.argv) > 1 else 'localhost'
port = 4433

# Path to the client's CA certificates
CA_CERTS_DIR = './client-certs'

# Create an SSL context for the client
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations(capath=CA_CERTS_DIR)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
```

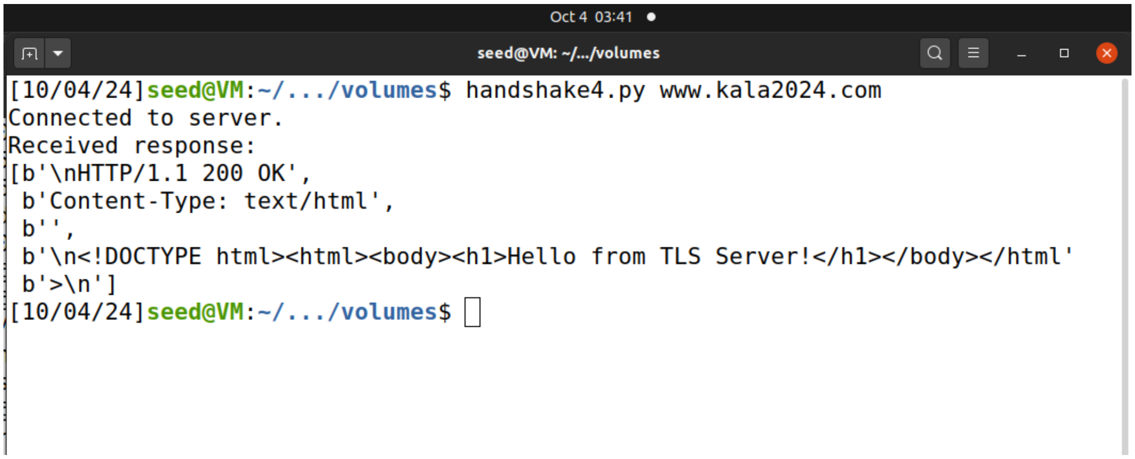
Since our CA certificate is in our own folder, we given the path of that folder to verify CA and assigned a Port : 4433.



```
[10/04/24]seed@VM:~/.../volumes$ dockps
5f0b51ee5549  mitm-proxy-10.9.0.143
0afd61021daa  client-10.9.0.5
318cd14f0445  server-10.9.0.43
[10/04/24]seed@VM:~/.../volumes$ docksh 31
root@318cd14f0445:/# cd volumes
root@318cd14f0445:/volumes# ls
README.txt    handshake4.py  server.py    server1.key
client-certs  myopenssl.cnf  server1.crt  server4.py
handshake.py  server-certs   server1.csr  server_openssl.cnf
root@318cd14f0445:/volumes# server4.py
Enter PEM pass phrase:
TLS Server is listening on port 4433...
Connection established with ('10.9.0.1', 54542)
Received HTTP request:
[b'GET / HTTP/1.0', b'Host: www.kala2024.com', b'', b'']
Connection closed.
```

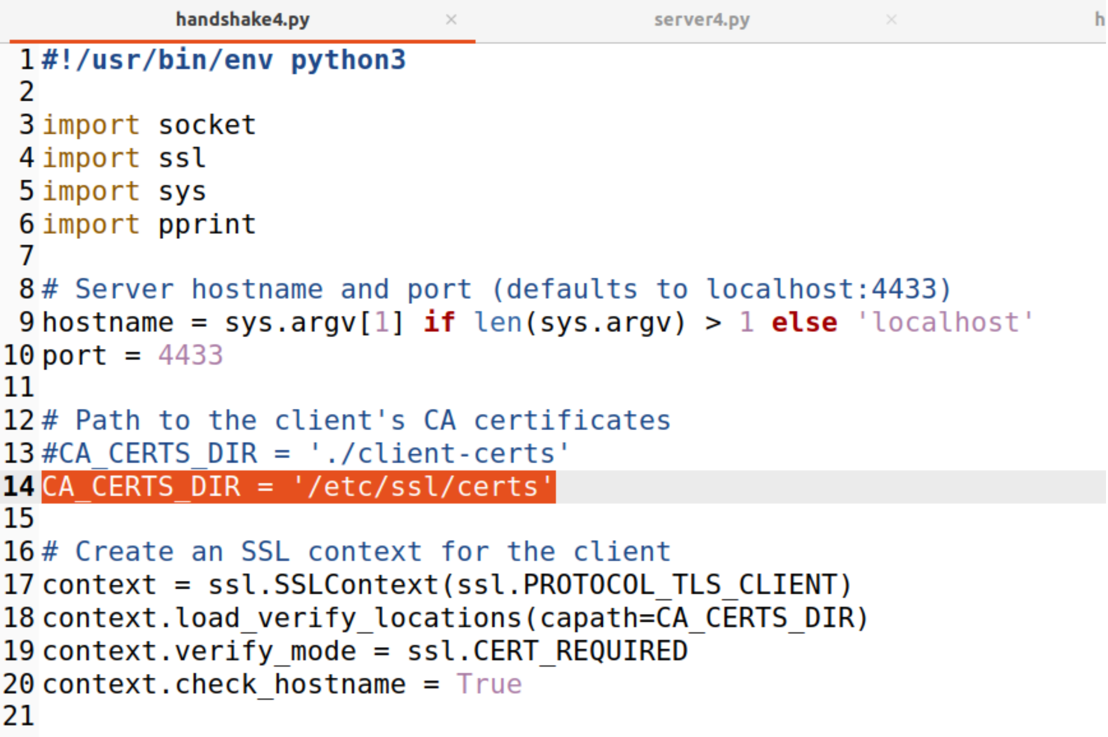
To run the server, we used docker container, “dockps” and “docksh”- to access the container. After that we went into the volumes folder and ran the server4.py and assigned our server to the server IP Address.

You can clearly see connection established between the server and client.



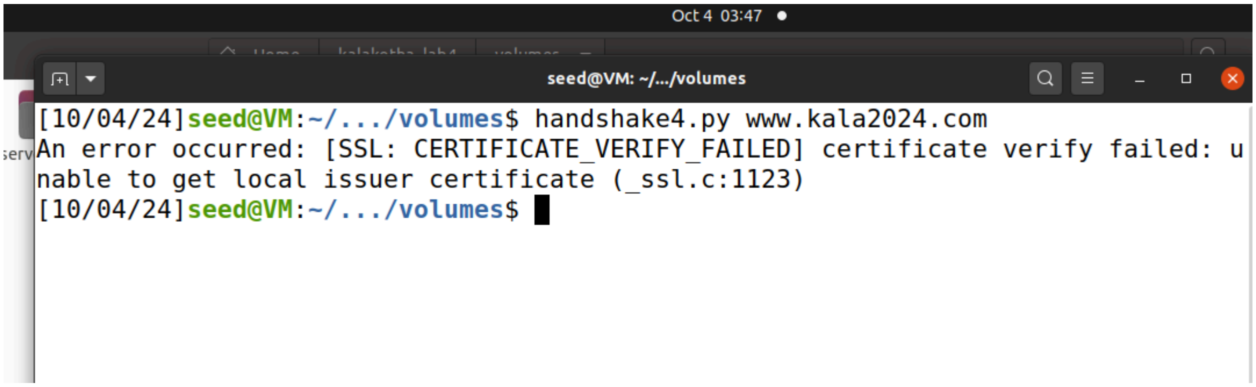
```
[10/04/24]seed@VM:~/.../volumes$ handshake4.py www.kala2024.com
Connected to server.
Received response:
[b'\nHTTP/1.1 200 OK',
 b'Content-Type: text/html',
 b'',
 b'\n<!DOCTYPE html><html><body><h1>Hello from TLS Server!</h1></body></html'
 b'>\n']
[10/04/24]seed@VM:~/.../volumes$ 
```

We executed the handshake.py with my server and we got the response from the server.



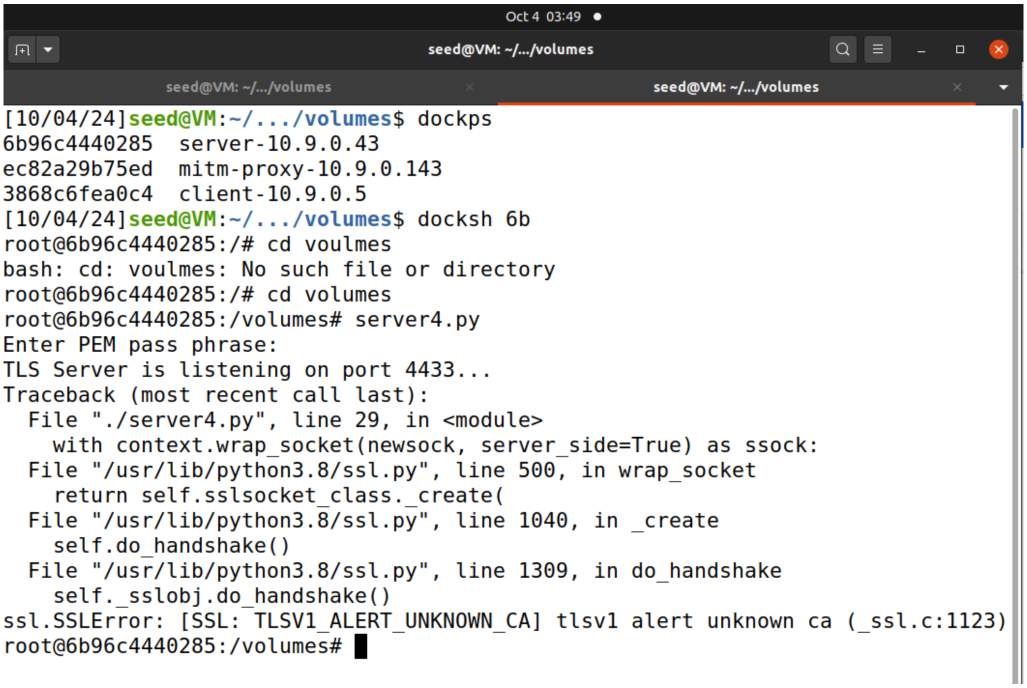
```python
#!/usr/bin/env python3

import socket
import ssl
import sys
import pprint

# Server hostname and port (defaults to localhost:4433)
hostname = sys.argv[1] if len(sys.argv) > 1 else 'localhost'
port = 4433

# Path to the client's CA certificates
#CA_CERTS_DIR = './client-certs'
CA_CERTS_DIR = '/etc/ssl/certs'

# Create an SSL context for the client
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations(capath=CA_CERTS_DIR)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

```

And we also tested with the “/etc/ssl/certs”



```
[10/04/24]seed@VM:~/.../volumes$ handshake4.py www.kala2024.com
An error occurred: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: u
nable to get local issuer certificate (_ssl.c:1123)
[10/04/24]seed@VM:~/.../volumes$ 
```

It said unable to get the local issuer.

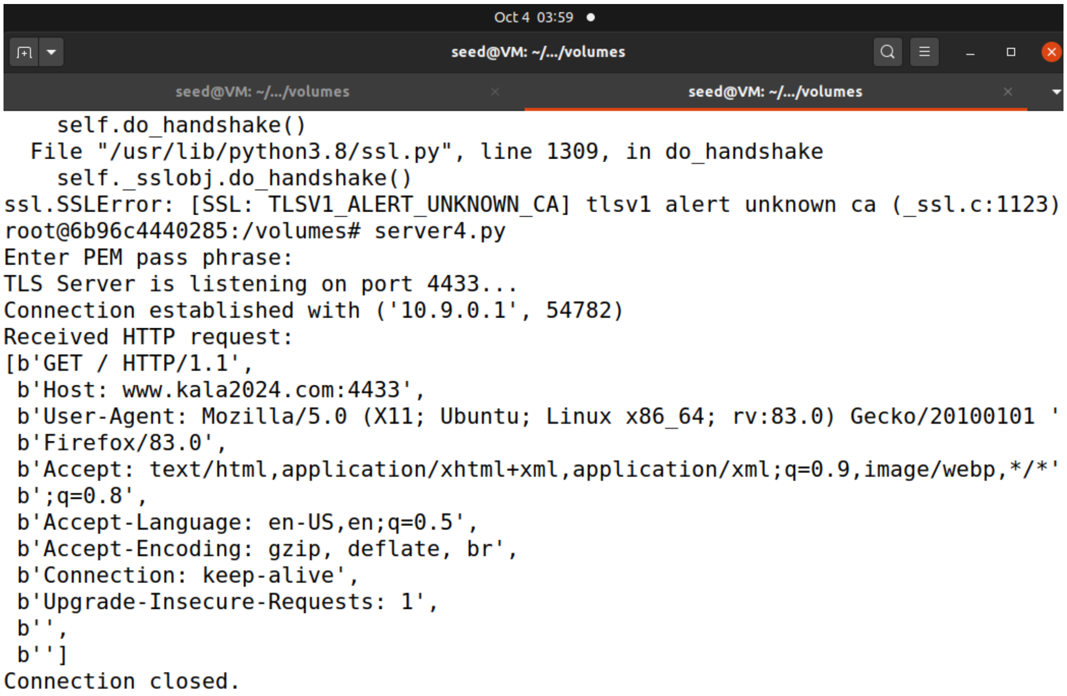


```
[10/04/24]seed@VM:~/.../volumes$ dockps
6b96c4440285  server-10.9.0.43
ec82a29b75ed  mitm-proxy-10.9.0.143
3868c6fea0c4  client-10.9.0.5
[10/04/24]seed@VM:~/.../volumes$ docksh 6b
root@6b96c4440285:/# cd voulmes
bash: cd: voulmes: No such file or directory
root@6b96c4440285:/# cd volumes
root@6b96c4440285:/volumes# server4.py
Enter PEM pass phrase:
TLS Server is listening on port 4433...
Traceback (most recent call last):
  File "./server4.py", line 29, in <module>
    with context.wrap_socket(newsock, server_side=True) as ssock:
  File "/usr/lib/python3.8/ssl.py", line 500, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.8/ssl.py", line 1040, in _create
    self.do_handshake()
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:1123)
root@6b96c4440285:/volumes# 
```

The got a bad response from the http (alert Unknown CA).
While testing the program, I received a bad response (alert) using the `/etc/ssl/certs` folder, indicating a failed TLS handshake due to incorrect certificate setup. In contrast, using the `./client-certs` folder resulted in a successful response from the server, showing that the certificates there were properly configured. This highlights the importance of using the right certificates for establishing secure TLS connections.

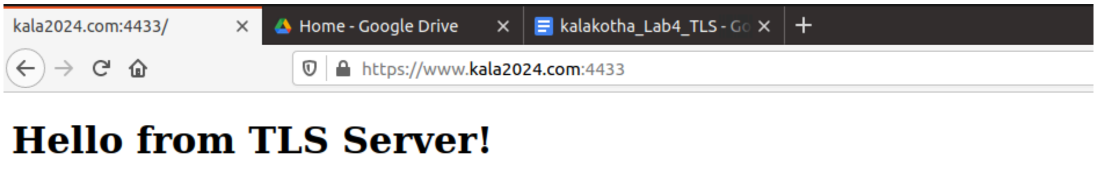
Takeaways: In this task, we used our client program to connect to a server while managing trusted certificates differently. Initially, we tried loading certificates from the '/etc/ssl/certs' folder but ran into problems because our custom Certificate Authority (CA) certificate was not in that location. Instead, we stored our CA certificate in the './client-certs'' folder, which prevented any changes to the whole system. When we tested with the './client-certs' folder, the client was able to successfully verify the server's certificate, allowing for smooth communication. This showed us how important it is to manage CA certificates correctly and the advantages of using a separate folder for testing without impacting the entire system.

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



```
    self.do_handshake()
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:1123)
root@6b96c4440285:/volumes# server4.py
Enter PEM pass phrase:
TLS Server is listening on port 4433...
Connection established with ('10.9.0.1', 54782)
Received HTTP request:
[b'GET / HTTP/1.1',
 b'Host: www.kala2024.com:4433',
 b'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 '
 b'Firefox/83.0',
 b'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*'
 b';q=0.8',
 b'Accept-Language: en-US,en;q=0.5',
 b'Accept-Encoding: gzip, deflate, br',
 b'Connection: keep-alive',
 b'Upgrade-Insecure-Requests: 1',
 b'',
 b'']
Connection closed.
```

In this task, we browsed our website ending with the port number 4433 and yes we got the response from the server.
Since we are using the CA certificate (created in Lab3 PKI). the certificate is already imported the browser. So when we ran website it directly connected without any warnings.



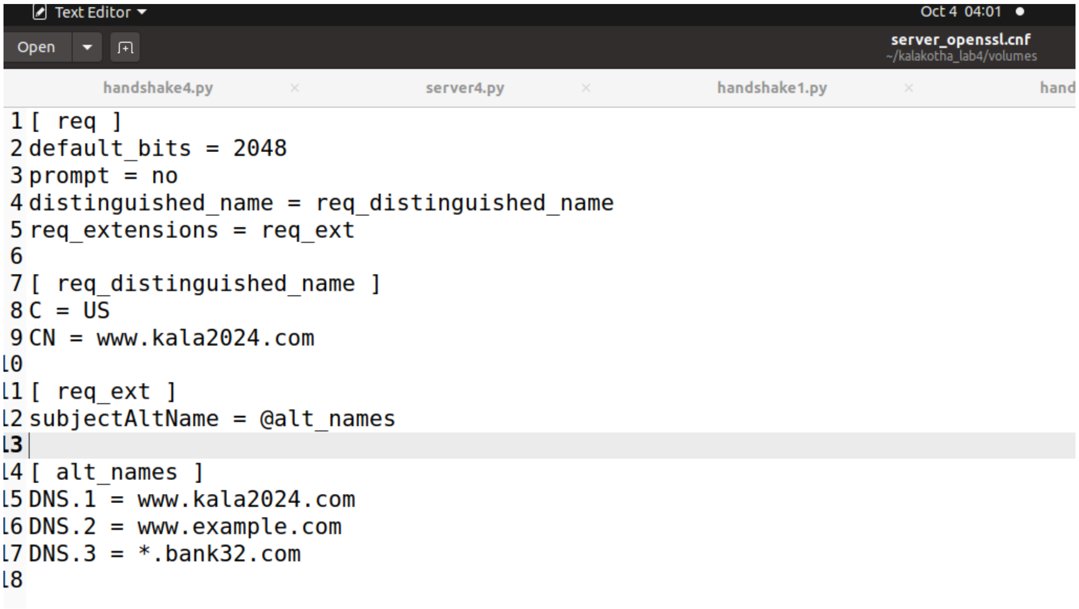
Hello from TLS Server!

Here is the message given by the server.
we was able to prove that my browser can connect to the TLS server successfully. After I sent a request, the browser connected without any issues and showed the content from the server.
This means that the TLS connection was set up correctly, allowing for secure data exchange and the successful retrieval of content.

Takeaways: In this task, we tested our TLS server program using a web browser on the host VM. We directed the browser to the server running on port 4433, the designated port for our HTTPS server. Initially, the browser could not verify the server's certificate since it was issued by a CA we created in the lab, which was not part of the browser's trusted certificates. To solve this, we manually added our CA's certificate to the browser's trusted list. After doing so, the browser successfully connected to our server and displayed the content it returned. This process showed that our TLS server was set up correctly and could communicate securely with the browser.

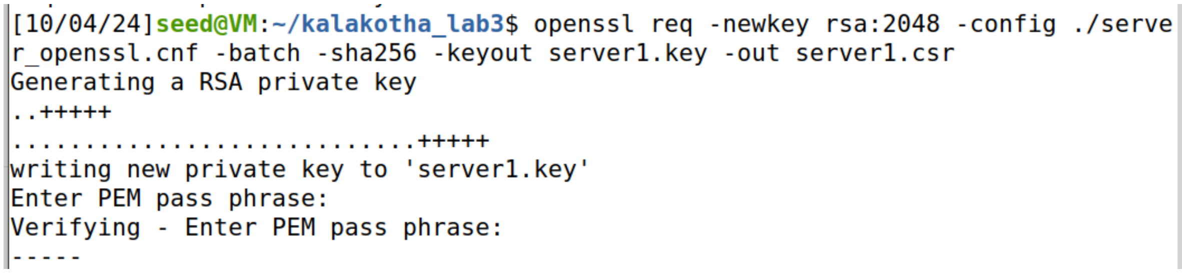
4.3 Task 2.c. Certificate with multiple names:

By using the text given by seeds lab, we created a server_openssl.cnf file and saved that text in the file.

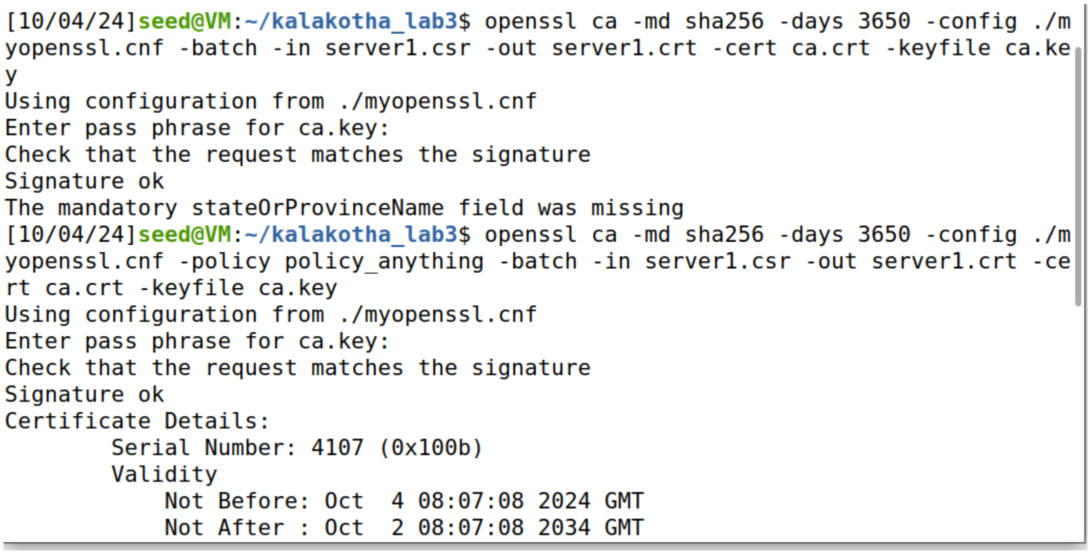


```
[ req ]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name
req_extensions = req_ext

[ req_distinguished_name ]
C = US
CN = www.kala2024.com

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = www.kala2024.com
DNS.2 = www.example.com
DNS.3 = *.bank32.com

```

After that we went into the Lab 3 (PKI) and created a “.csr” and “keys” for my cnf file.



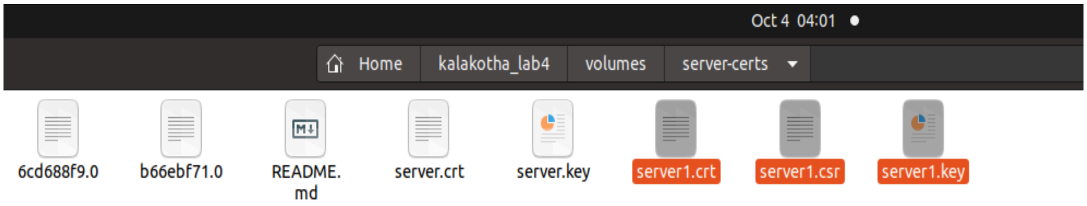
```
[10/04/24]seed@VM:~/kalakotha_lab3$ openssl req -newkey rsa:2048 -config ./serve
r_openssl.cnf -batch -sha256 -keyout server1.key -out server1.csr
Generating a RSA private key
..+++++
............................+++++
writing new private key to 'server1.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
```



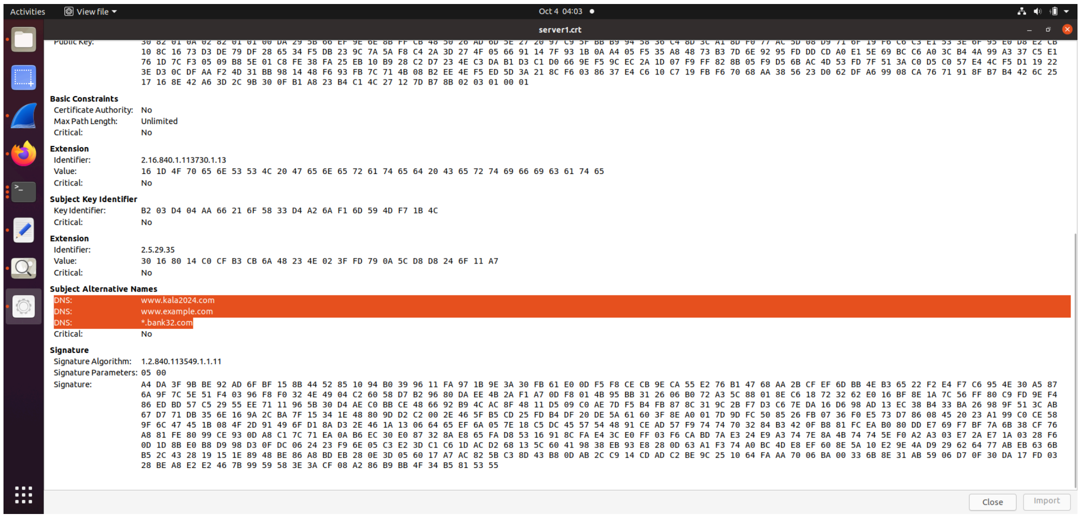
```
[10/04/24]seed@VM:~/kalakotha_lab3$ openssl ca -md sha256 -days 3650 -config ./m
yopenssl.cnf -batch -in server1.csr -out server1.crt -cert ca.crt -keyfile ca.ke
y
Using configuration from ./myopenssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
The mandatory stateOrProvinceName field was missing
[10/04/24]seed@VM:~/kalakotha_lab3$ openssl ca -md sha256 -days 3650 -config ./m
yopenssl.cnf -policy policy_anything -batch -in server1.csr -out server1.crt -ce
rt ca.crt -keyfile ca.key
Using configuration from ./myopenssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4107 (0x100b)
        Validity
            Not Before: Oct  4 08:07:08 2024 GMT
            Not After : Oct  2 08:07:08 2034 GMT
```

And using the CA in the Lab3 we signed our .csr file and generated the valid Certificate to our file.

Issue we face while signing certificate: we forgot to use the -policy while signing the csr file. But eventually we used -policy.

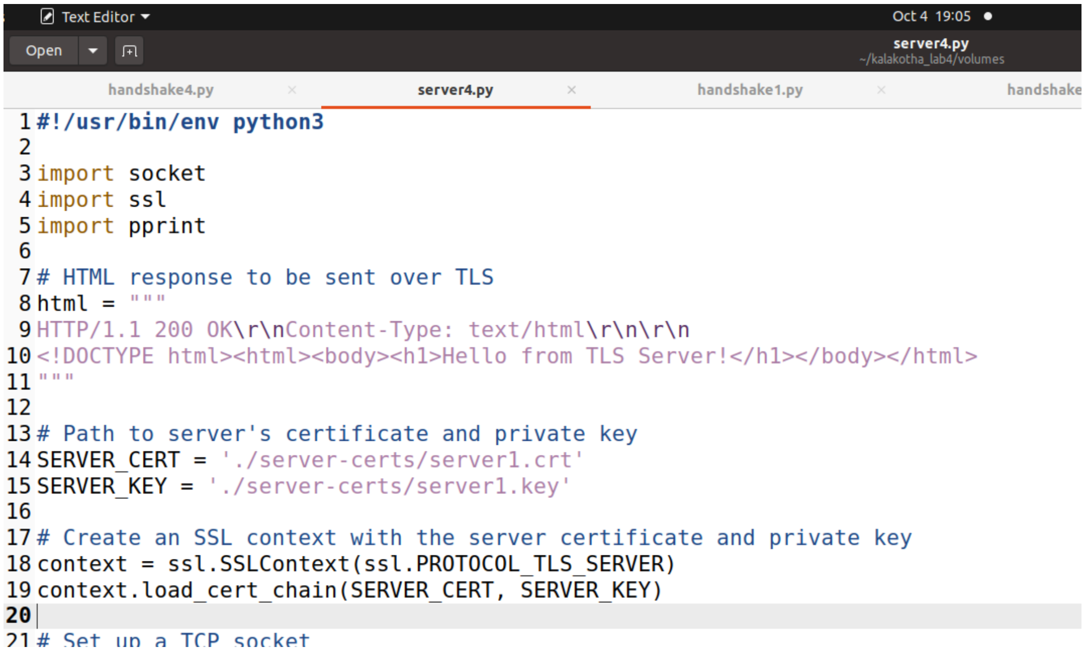


After that we copied the sever1.crt, server1.key and server1.csr to the server-certs folder (in Lab 4). And generated the hash for the certificate.



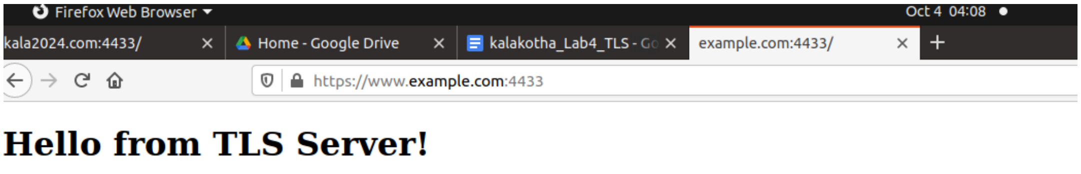
```
Public Key:                30 82 01 0A 02 82 01 01 00 DA 29 5B 66 EF 9E 6E 8B FF CB 48 50 26 AD 6D 5E 27 20 97 C9 5F BB B9 94 58 36 C4 8D 3C A1 8D F0 77 AC 5D 08 D9 71 6F 19 F6 C6 C3 E1 53 3E 6F 95 E0 D8 E2 CB
                           10 8C 16 73 D3 DE 79 DF 28 65 34 F5 DB 23 9C 7A 5A F8 C4 2A 3D 27 4F 05 66 91 14 7F 93 1B 0A A4 05 F5 35 A8 48 73 B3 7D 6E 92 95 FD DD CD A0 E1 5E 69 BC C6 A0 3C B4 4A 99 A3 37 C5 E1
                           76 1D 7C F3 05 09 B8 5E 01 C8 FE 38 FA 25 EB 10 B9 28 C2 D7 23 4E C3 DA B1 D3 C1 D0 66 9E F5 9C EC 2A 1D 07 F9 FF 82 8B 05 F9 D5 6B AC 4D 53 FD 7F 51 3A C0 D5 C0 57 E4 4C F5 D1 19 22
                           3E D3 0C DF AA F2 4D 31 BB 98 14 48 F6 93 FB 7C 71 4B 08 B2 EE 4E F5 ED 5D 3A 21 8C F6 03 86 37 E4 C6 10 C7 19 FB F6 70 68 AA 38 56 23 D0 62 DF A6 99 08 CA 76 71 91 8F B7 B4 42 6C 25
                           17 16 8E 42 A6 3D 2C 9B 30 0F B1 A8 23 B4 C1 4C 27 12 7D B7 8B 02 03 01 00 01

Basic Constraints
  Certificate Authority:   No
  Max Path Length:         Unlimited
  Critical:                No

Extension
  Identifier:              2.16.840.1.113730.1.13
  Value:                   16 1D 4F 70 65 6E 53 53 4C 20 47 65 6E 65 72 61 74 65 64 20 43 65 72 74 69 66 69 63 61 74 65
  Critical:                No

Subject Key Identifier
  Key Identifier:          B2 03 D4 04 AA 66 21 6F 58 33 D4 A2 6A F1 6D 59 4D F7 1B 4C
  Critical:                No

Extension
  Identifier:              2.5.29.35
  Value:                   30 16 80 14 C0 CF B3 CB 6A 48 23 4E 02 3F FD 79 0A 5C D8 D8 24 6F 11 A7
  Critical:                No

Subject Alternative Names
  DNS:                     www.kala2024.com
  DNS:                     www.example.com
  DNS:                     *.bank32.com
  Critical:                No

Signature
  Signature Algorithm:     1.2.840.113549.1.1.11
  Signature Parameters:    05 00
  Signature:               A4 DA 3F 9B BE 92 AD 6F BF 15 8B 44 52 85 10 94 B0 39 96 11 FA 97 1B 9E 3A 30 FB 61 E0 0D F5 F8 CE CB 9E CA 55 E2 76 B1 47 68 AA 2B CF EF 6D BB 4E B3 65 22 F2 E4 F7 C6 95 4E 30 A5 87
                           6A 9F 7C 5E 51 F4 03 96 F8 F0 32 4E 49 04 C2 60 58 D7 B2 96 80 DA EE 4B 2A F1 A7 0D F8 01 4B 95 BB 31 26 06 B0 72 A3 5C 88 01 8E C6 18 72 32 62 E0 16 BF 8E 1A 7C 56 FF 80 C9 FD 9E F4
                           86 ED BD 57 C5 29 55 EE 71 11 96 5B 30 D4 AE C0 BB CE 48 66 92 B9 4C AC 8F 48 11 D5 09 C0 AE 7D F5 B4 FB 87 8C 31 9C 2B F7 D3 C6 7E DA 16 D6 98 AD 13 EC 38 B4 33 BA 26 98 9F 51 3C AB
                           67 D7 71 DB 35 6E 16 9A 2C BA 7F 15 34 1E 48 80 9D D2 C2 00 2E 46 5F B5 CD 25 FD B4 DF 20 DE 5A 61 60 3F 8E A0 01 7D 9D FC 50 85 26 FB 07 36 F0 E5 73 D7 86 08 45 20 23 A1 99 C0 CE 58
                           9F 6C 47 45 1B 08 4F 2D 91 49 6F D1 8A D3 2E 46 1A 13 06 64 65 EF 6A 05 7E 18 C5 DC 45 57 54 48 91 CE AD 57 F9 74 74 70 32 84 B3 42 0F B8 81 FC EA B0 80 DD E7 69 F7 BF 7A 6B 38 CF 76
                           A8 81 FE 80 99 CE 93 0D A8 C1 7C 71 EA 0A B6 EC 30 E0 87 32 8A E8 65 FA D8 53 16 91 8C FA E4 3C E0 FF 03 F6 CA BD 7A E3 24 E9 A3 74 7E 8A 4B 74 74 5E F0 A2 A3 03 E7 2A E7 1A 03 28 F6
                           0D 1D 8B E0 B8 D9 98 D3 0F DC 06 24 23 F9 6E 05 C3 E2 3D C1 C6 1D AC D2 68 13 5C 60 41 98 38 EB 93 E8 28 0D 63 A1 F3 74 A0 BC 4D E8 EF 60 8E 5A 10 E2 9E 4A D9 29 62 64 77 AB EB 63 6B
                           B5 2C 43 28 19 15 1E 89 48 BE 86 A8 BD EB 28 0E 3D 05 60 17 A7 AC 82 5B C3 8D 43 B8 0D AB 2C C9 14 CD AD C2 BE 9C 25 10 64 FA AA 70 06 BA 00 33 6B 8E 31 AB 59 06 D7 0F 30 DA 17 FD 03
                           28 BE A8 E2 E2 46 7B 99 59 58 3E 3A CF 08 A2 86 B9 BB 4F 34 B5 81 53 55
```

This is our server certificate.



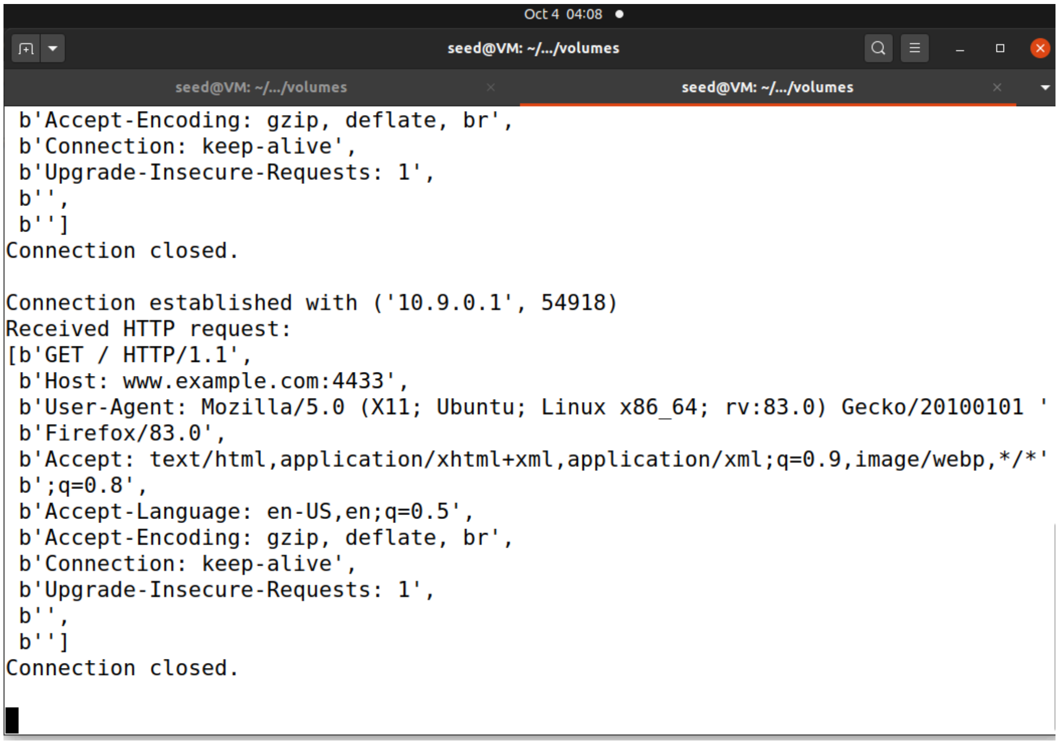
```python
#!/usr/bin/env python3

import socket
import ssl
import pprint

# HTML response to be sent over TLS
html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>Hello from TLS Server!</h1></body></html>
"""

# Path to server's certificate and private key
SERVER_CERT = './server-certs/server1.crt'
SERVER_KEY = './server-certs/server1.key'

# Create an SSL context with the server certificate and private key
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(SERVER_CERT, SERVER_KEY)

# Set up a TCP socket
```

And we gave our new server location in the server4.py code and we executed the handshake4.py



Hello from TLS Server!

We used one of our alternative websites to check wheter they redirect to the main server. And we successfully got response from the server.



```
 b'Accept-Encoding: gzip, deflate, br',
 b'Connection: keep-alive',
 b'Upgrade-Insecure-Requests: 1',
 b'',
 b'']
Connection closed.

Connection established with ('10.9.0.1', 54918)
Received HTTP request:
[b'GET / HTTP/1.1',
 b'Host: www.example.com:4433',
 b'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 '
 b'Firefox/83.0',
 b'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*'
 b';q=0.8',
 b'Accept-Language: en-US,en;q=0.5',
 b'Accept-Encoding: gzip, deflate, br',
 b'Connection: keep-alive',
 b'Upgrade-Insecure-Requests: 1',
 b'',
 b'']
Connection closed.
```

Here is the response recorded by the server.
We can say that our server can support multiple hostnames, including any hostname in my domain. we tested several different hostnames, and each one worked properly. This indicates that the server is correctly set up to handle requests for various host names.

Takeaways: In this task, we learned how to create certificates that support multiple hostnames using the Subject Alternative Name (SAN) extension. This extension allows a single certificate to validate various URLs, helping improve flexibility in web server configurations. By generating a certificate signing request (CSR) with a specific configuration file, we ensured that our server could handle multiple hostnames without issue. This setup is essential for maintaining secure communications across different domains and enhancing the overall usability of the server.

Task3: ASimpleHTTPSProxy:

We didn’t got time to do this task, but we learned about it
In this task, we learned about the Man-In-The-Middle (MITM) attack and how it affects TLS servers, especially when the Public Key Infrastructure (PKI) is compromised. By building a small HTTPS proxy called mHTTPSproxy, we saw how an attacker can intercept communication between a client and a server. This proxy works by forwarding requests and responses, pretending to be both the client and the server.

We learned that changing the ‘/etc/hosts’ file can simulate DNS attacks by redirecting traffic from a real server to our proxy. This showed how risky it can be when a trusted Certificate Authority (CA) is compromised, as the proxy can create fake certificates for any website using the stolen CA's key. This experience helped me see how an MITM attack can happen, both on our server and on a real HTTPS website, showing the serious security risks when keys are stolen or trust in the PKI system is broken.