

Network Security – CSCI_6541_80

Namana Y Tarikere – G21372717

Homework Assignment – 4

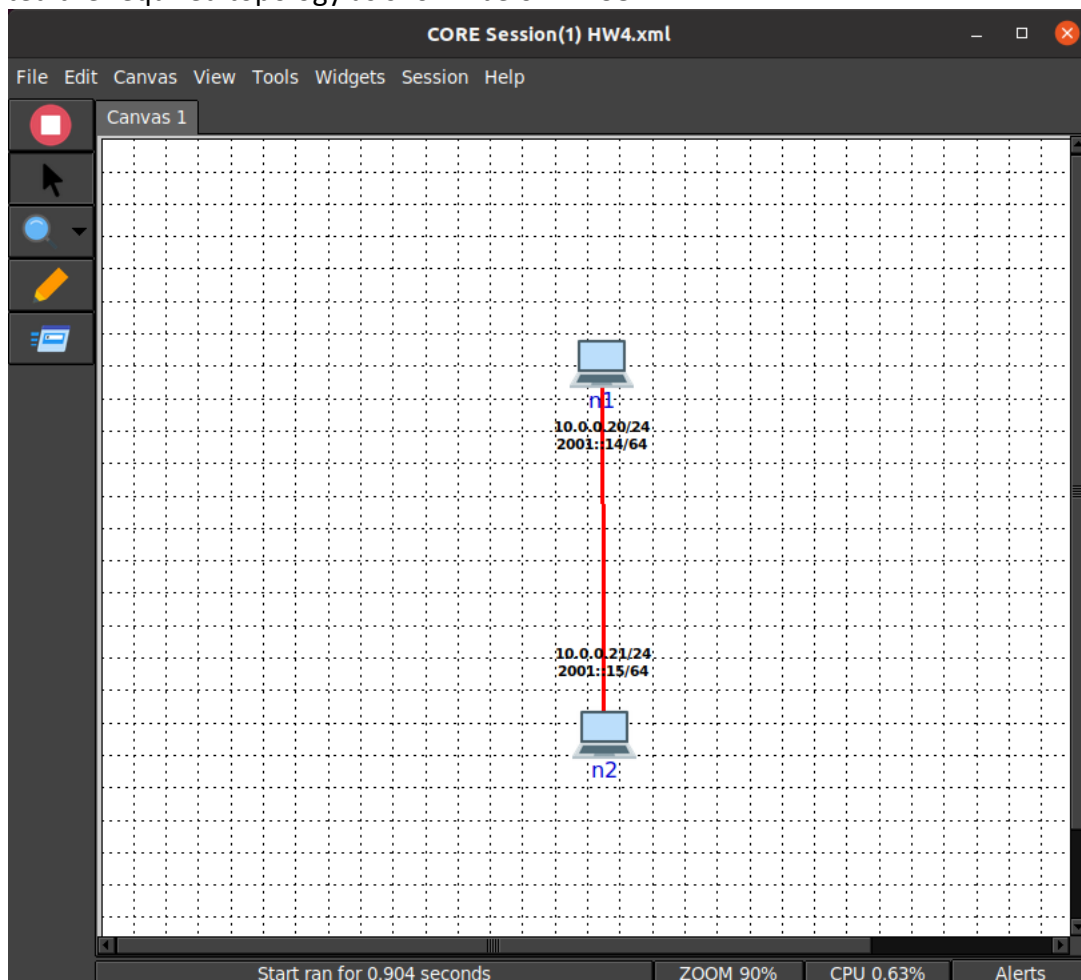
TLS Assignment

References:

- https://www.openssl.org/docs/man1.0.2/man1/openssl-s_client.html
- https://www.openssl.org/docs/man1.0.2/man1/openssl-s_server.html

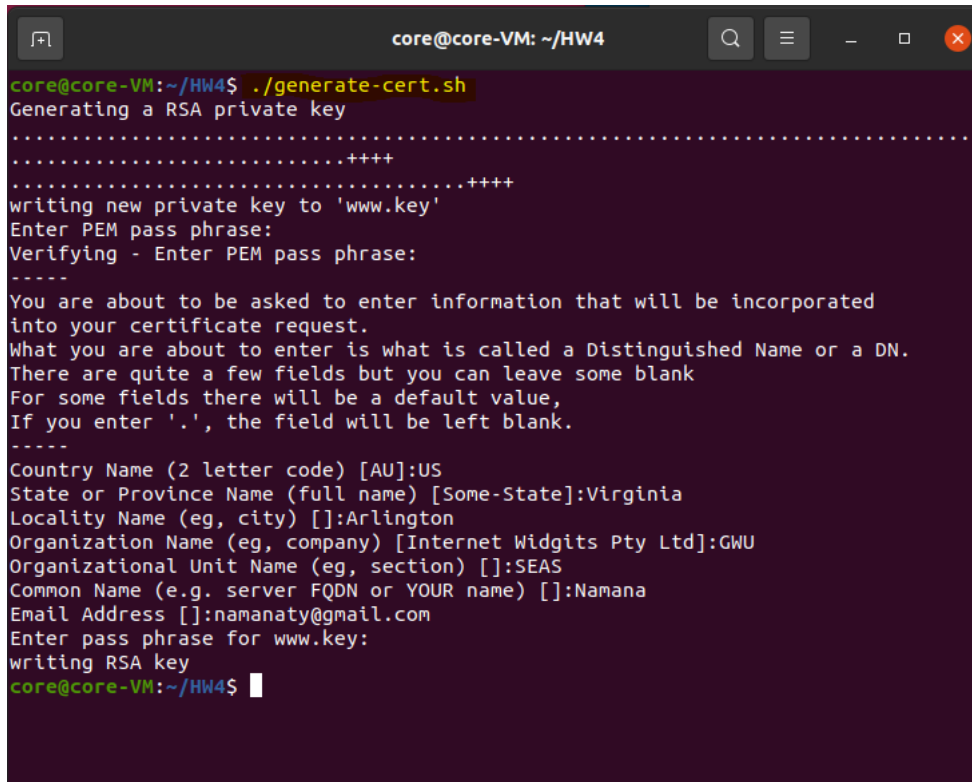
Part 1: TLS (10 points)

Created the required topology as shown below in CORE:



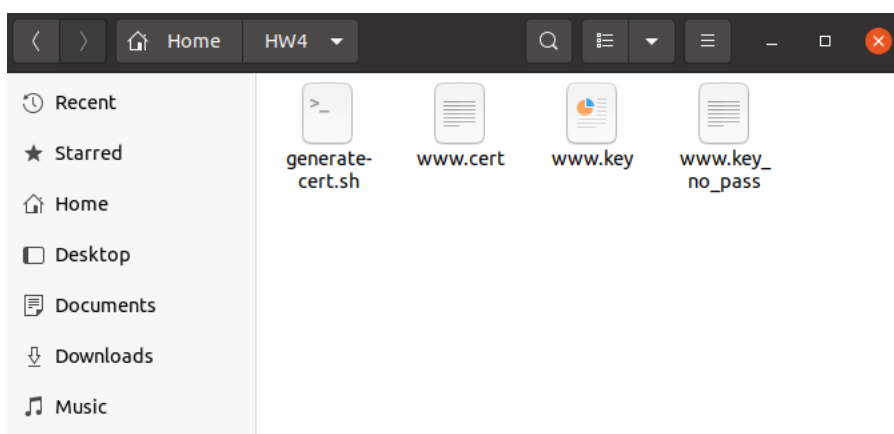
1. Use openssl s_client and s_server to create the following configurations
 - 1.1. Use the script provided to generate X.509 certs.

Download the generate-cert.sh.zip file and unzip the file in HW4 folder. Next, run the unzipped generate-cert.sh script using the command **./generate-cert.sh** and enter the required details to generate the certificates as shown:



```
core@core-VM: ~/HW4
core@core-VM:~/HW4$ ./generate-cert.sh
Generating a RSA private key
.....++++
.....++++
writing new private key to 'www.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Virginia
Locality Name (eg, city) []:Arlington
Organization Name (eg, company) [Internet Widgits Pty Ltd]:GWU
Organizational Unit Name (eg, section) []:SEAS
Common Name (e.g. server FQDN or YOUR name) []:Namana
Email Address []:namanaty@gmail.com
Enter pass phrase for www.key:
writing RSA key
core@core-VM:~/HW4$
```

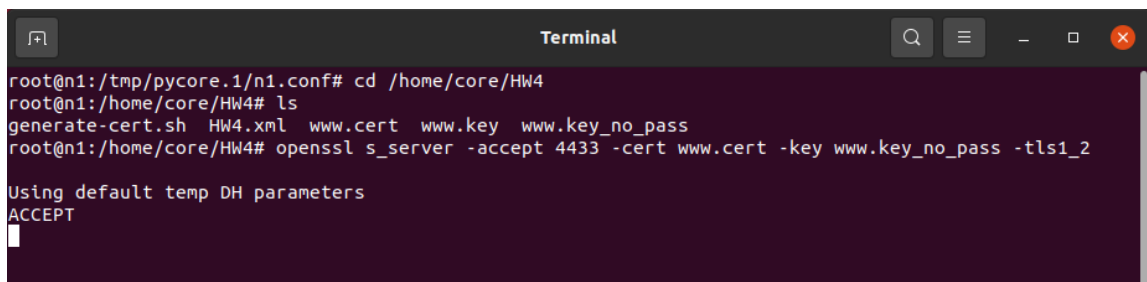
The script generated 3 files – a certificate and 2 private keys as shown below:



1.2. (1pts) Run a TLS1.2 server and a TLS1.2 client. Show the command you used on both and a screenshot of the TLS full handshake in Wireshark

Launch Wireshark using the command **sudo wireshark**. Next, run TLS1.2 Server on n1 node using the command **openssl s_server -accept 4433 -cert [www.cert](#) -key [www.key_no_pass](#) -tls1_2**. Here,

- openssl s_server**: starts the server using openssl
- accept 4433**: listening on port 4433
- cert [www.cert](#)**: setting up [www.cert](#) as the certificate used for connection
- key [www.key_no_pass](#)**: using [www.key_no_pass](#) as the key for the connection
- tls1_2**: the type of SSL/TLS version to set up the connection as shown below:



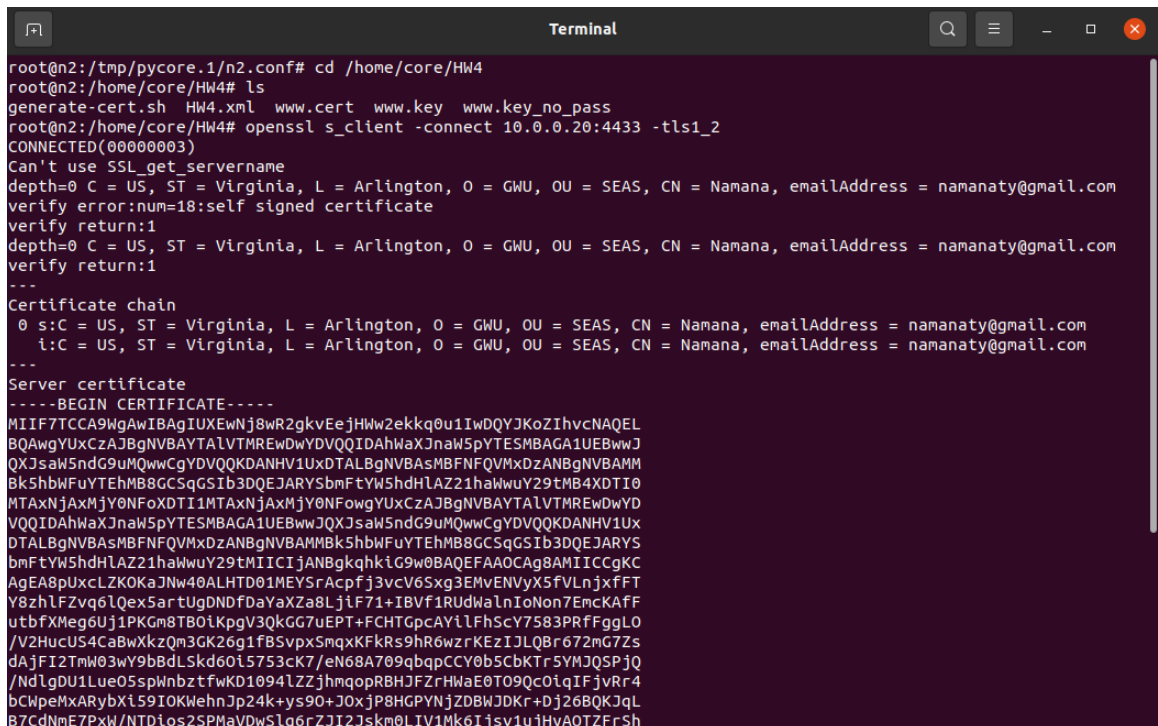
```
root@n1:/tmp/pycore.1/n1.conf# cd /home/core/HW4
root@n1:/home/core/HW4# ls
generate-cert.sh HW4.xml www.cert www.key www.key_no_pass
root@n1:/home/core/HW4# openssl s_server -accept 4433 -cert www.cert -key www.key_no_pass -tls1_2

Using default temp DH parameters
ACCEPT
```

Next, run TLS1.2 Client on n2 node using the command **openssl s_client -connect 10.0.0.20:4433 -tls1_2**. Here,

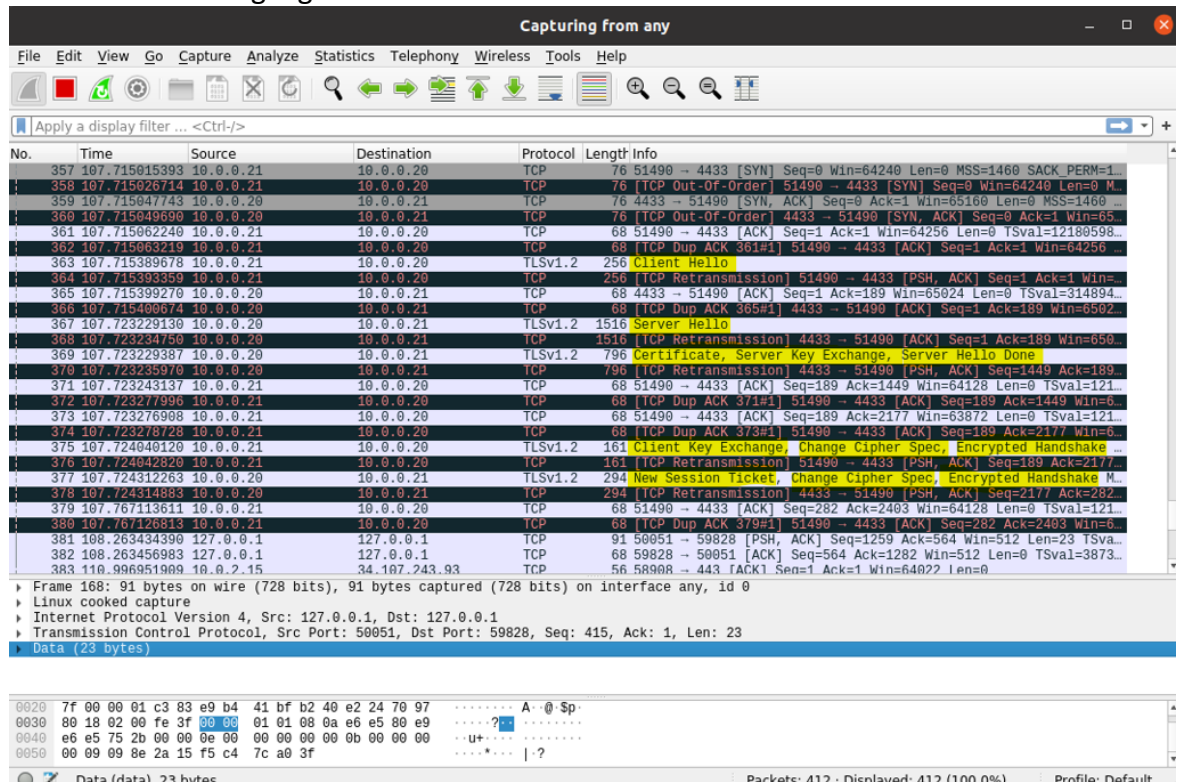
- openssl s_client**: starting SSL/TLS in the client using OpenSSL
- connect 10.0.0.20:4433**: indicates the client to initiate a connection to port 4433 of the IP 10.0.0.20
- tls1_2**: the type of SSL/TLS version to set up the connection.

You can see that the client is connected to the server as shown below:



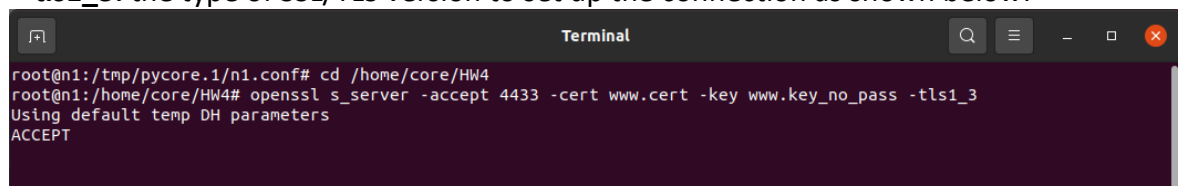
```
root@n2:/tmp/pycore.1/n2.conf# cd /home/core/HW4
root@n2:/home/core/HW4# ls
generate-cert.sh HW4.xml www.cert www.key www.key_no_pass
root@n2:/home/core/HW4# openssl s_client -connect 10.0.0.20:4433 -tls1_2
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
 1 i:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIF7TCCA9WgAwIBAgIUxEWnJ8wR2gkvEejHwW2ekq0u1IwDQYJKoZIhvcNAQEL
BQAwgYUxCzAJBgNVBAYTALVTRREwDwYDVQQIDAhW4XJnaW5pYTESMBAGA1UEBwwJ
QXJsaW5ndG9uMQwwCgYDVQQKDANHV1UxDTALBgNVBASMBFNQVW5kZANBgNVBAMM
Bk5hbWVufyTEhMB8GCSqGSIb3DQEJARYSbmFtYXN0eHwW2ekq0u1IwDQYJKoZI
MTAxNjAxMjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0NjY0
VQQAIDAhW4XJnaW5pYTESMBAGA1UEBwwJQXJsaW5ndG9uMQwwCgYDVQQKDANHV1Ux
DTALBgNVBASMBFNQVW5kZANBgNVBAMM Bk5hbWVufyTEhMB8GCSqGSIb3DQEJARYS
bmFtYXN0eHwW2ekq0u1IwDQYJKoZIhvcNAQELBQAwgYUxCzAJBgNVBAYTALVTRREw
AgEASpUxCLZK0KaJNw40ALHTD01MEYSrAcpfj3vcV6Sxg3EMvENVyX5fVLnJxFTF
Y8zhlfZvq6lQex5artUgDNDfDaYaXZa8Ljlf71+IBVf1RudWalnIoNon7EmcKAff
utbfXMEg6Uj1PKGm8TBOiKpgV3QkGG7uEPT+FCHTGpcAYlFhScY7583PRfFggL0
/V2HucUS4CaBwXkzQm3GK26g1fBSvpxSmqXKfKRs9hR6wzrKEzIJLQBr672mG7Zs
dAjfI2Tmd03W9yBdLSkd60i5753cK7/eN68A709qbqCCY0b5CbKTr5VMJQSPJQ
/NdLgDU1Lue05SpWnbztfwKD1094lZZjhmqopRBHJFZrHwAE0T09C0iQIFjvRr4
bcWpeMxARYbXiS9IOkWehnJp24k+ys90+J0xjP8HGPYNjZDBWJDKr+Dj26BQKJqL
B7CDNmE7PxW/NTDios2SPMaVdW5Lg6rZJi2Jskm0LIV1Mk6Ij5v1ujHyAOTZFrSh
```

In Wireshark, we can see the full TLS Handshake. Initially, the client starts the connection by sending a Client Hello. In response, the server sends a Server Hello along with its Certificate, Server Key Exchange, and a Server Hello Done message. Lastly, the client sends the Client Key Exchange, Change Cipher Specification, and Encrypted Handshake Message. If the client successfully verifies the handshake, it issues a new session ticket and sends a Change Cipher Specification and Encrypted Handshake Message, thereby establishing a connection with the server using the session ticket as highlighted and shown below:



1.3. (1pts) Run a TLS1.3 server and a TLS1.2 client. Describe what happened. What Alert message type and what Level are generated?

Run TLS1.3 Server on n1 node using the command **openssl s_server -accept 4433 -cert [www.cert](#) -key [www.key](#) no_pass -tls1_3**. Here, **openssl s_server**: starts the server using openssl
-accept 4433: listening on port 4433
-cert [www.cert](#): setting up [www.cert](#) as the certificate used for connection
-key [www.key](#) no_pass: using [www.key](#) no_pass as the key for the connection
-tls1_3: the type of SSL/TLS version to set up the connection as shown below:



Next, run TLS1.2 Client on n2 node using the command **openssl s_client -connect 10.0.0.20:4433 -tls1_2**. Here, **openssl s_client**: starting SSL/TLS in the client using OpenSSL

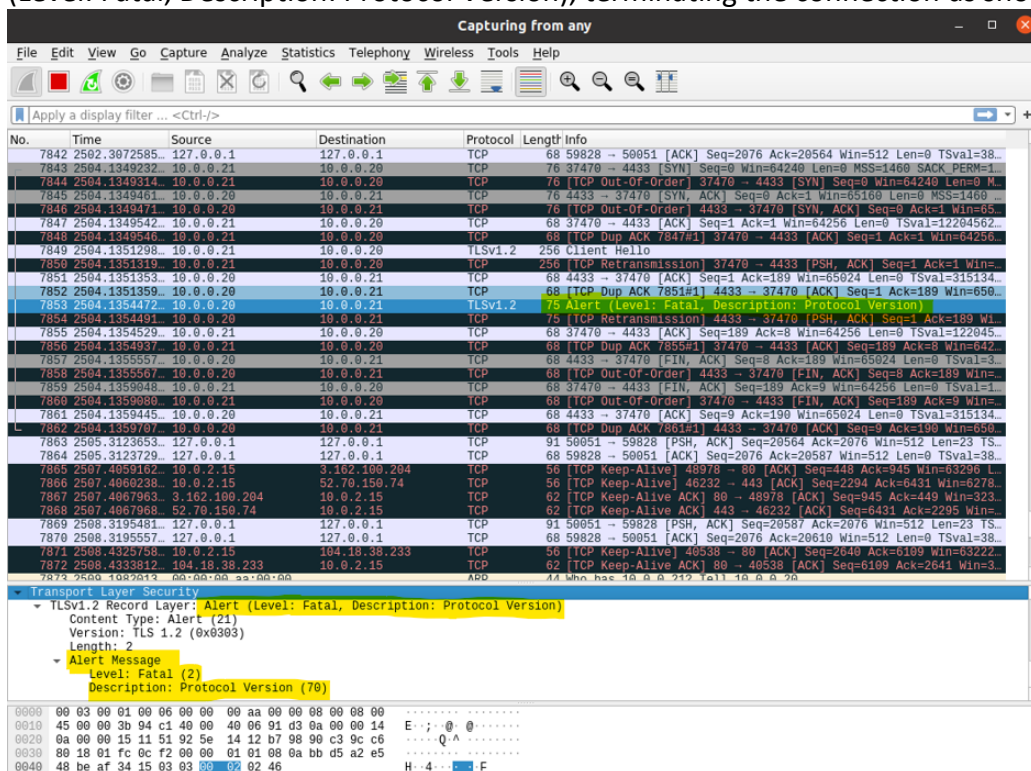
-connect 10.0.0.20:4433: indicates the client to connect to port 4433 of IP 10.0.0.20

-tls1_2: the type of SSL/TLS version to set up the connection.

You can see that the client is unable to connect to the server as shown below:

```
Terminal
root@n2:/home/core/HW4# openssl s_client -connect 10.0.0.20:4433 -tls1_2
CONNECTED(00000003)
140225647486272:error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version:
../ssl/record/rec_layer_s3.c:1552:SSL alert number 70
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 188 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1.2
    Cipher : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1729045281
    Timeout : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
---
root@n2:/home/core/HW4#
```

In Wireshark, we can see an SSL Handshake Attempt which represents the network packets exchanged between the server and client during their attempt to establish a connection ultimately failed. The process begins with the client initiating the connection by sending a Client Hello. In response, the server sends an Alert message (Level: Fatal, Description: Protocol Version), terminating the connection as shown:

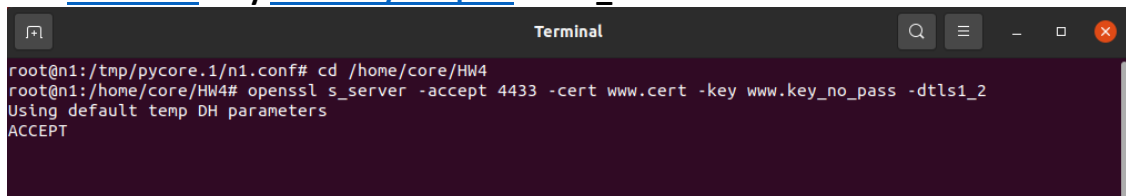


The client and server attempt to initiate an SSL handshake, but it fails because the server is using TLS 1.3, while the client is requesting a connection via TLS 1.2. Since TLS 1.3 is not backward compatible and each protocol has its own distinct connection establishment process, different protocol versions cannot successfully establish a connection. As a result, the handshake fails.

In Wireshark, the alert message shows the Level as **"Fatal"** with the Description **"Protocol Version,"** indicating that the server does not support the protocol version the client is using. Additionally, in the client's terminal, we see **alert number 70**, which indicates an incompatible protocol.

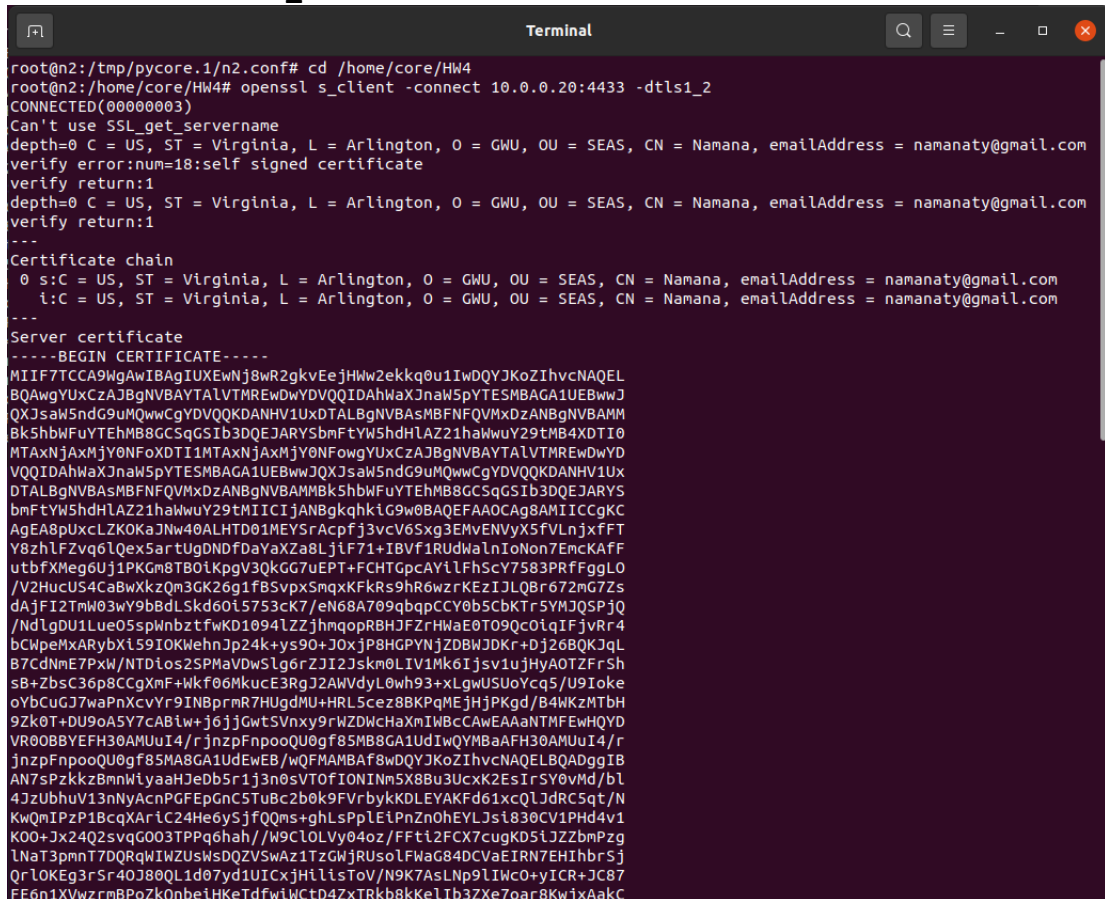
1.4. (1pts) Run a DTLS1.2 server and a DTLS1.2 client. Show a screenshot of the Hello Verify Request message with the cookie value.

Run DTLS1.2 Server on node 1 using the command `openssl s_server -accept 4433 -cert www.cert -key www.key.no_pass -dtls1_2` as shown below:



```
root@n1:/tmp/pycore.1/n1.conf# cd /home/core/HW4
root@n1:/home/core/HW4# openssl s_server -accept 4433 -cert www.cert -key www.key.no_pass -dtls1_2
Using default temp DH parameters
ACCEPT
```

Run the DTLS1.2 Client on node 2 using the command `openssl s_client -connect 10.0.0.20:4433 -dtls1_2`. Observe that the client is connected to the server as shown:



```
root@n2:/tmp/pycore.1/n2.conf# cd /home/core/HW4
root@n2:/home/core/HW4# openssl s_client -connect 10.0.0.20:4433 -dtls1_2
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify return:1
...
Certificate chain
 0 s:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
 1 i:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
...
Server certificate
-----BEGIN CERTIFICATE-----
MIIF7TCCA9WgAwIBAgIUXEwNj8wR2gkvEejHwW2ekkkQ0u1IwDQYJKoZIhvcNAQEL
BQAwgUxuCzAJBgNVBAYTALVMTREwDwYDVQQIDAhWaxJnaW5pYTESMBAGA1UEBwwJ
QXJsaW5ndG9uMQwwCgYDVQQKDANHV1UxDTALBgNVBAsMBFNFQVMxZDZANBgNVBAMM
Bk5hbWVufUyTEhMB8GCSqGSIb3DQEJARYSbmFtYV5hdHlAZ21haWwY29tMB4XDTE0
MTAxNjAxMjY0NFoXDTI1MTAxNjY0NFowYUxuCzAJBgNVBAYTALVMTREwDwYDV
QQIDAhWaxJnaW5pYTESMBAGA1UEBwwJQXJsaW5ndG9uMQwwCgYDVQQKDANHV1Ux
DTALBgNVBAsMBFNFQVMxZDZANBgNVBAMMBk5hbWVufUyTEhMB8GCSqGSIb3DQEJARYS
bmFtYV5hdHlAZ21haWwY29tMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKC
AgEA8pUxCLZKOKaJNw40ALHTD01MEYSrAcPfj3vcV6Sxg3EMvENVyX5fVLNjxfFT
Y8zhlfZvq6lQex5ar tUgDNDfDaYaXZa8Lj1f71+IBVF1RUdWalnIoNon7EmcKAff
utbFXMeg6Uj1PKGm8TBOlkpgV3QkGG7uEPT+FCHTGpcAYlLFhScY7583PRFfggLO
/V2HucUS4CaBwXkzQm3GK26g1fBSvpxSmqxKfKs9hR6wzrKEZIJLQB672mG7Zs
dAJfI2TmW03wY9bBdLSkd60i5753ck7/eN68A709qbqCCY0b5CbKTr5YMJQSPjQ
/NdIlgDU1Lue05spWnbztfwKD1094LZZjhmopRBHJFZrHwAE0T09Qc0iQfjvRr4
bcWpeMxARyBXI59IOKWehnJp24k+ys90+J0xJP8HGPYNjZDBWJDKr+Dj26BQKJqL
B7CdNmE7PwX/NTDios2SPMAVDwSLg6rZJI2Jskm0LIV1Mk6Ij5v1ujHyA0TZFrSh
sB+ZbsC36p8CgXmf+Wkf06MkucE3RgJ2AWVdYL0wh93+xlGwUSUoYcq5/U9Ioke
oYbCuGJ7waPnXcvYr9INBprmr7HUGdMU+HRL5cez8BKpQMEjHjPKgd/B4WkZMTbH
9Zk0T+DU9oASy7CABiw+j6jjCwtSVnxY9rGWDWChXmIBcCAWEAAaNTMFEWHQYD
VR00BBYEFH30AMUUI4/rjnzpFnpooQU0gf85MB8GA1UdIwQYMBaAFH30AMUUI4/r
jnzpFnpooQU0gf85MA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQAdggIB
AN7sPzkkZBmnWlyaaHJebD5r1j3n0sVTOFIONIm5X8Bu3UcxK2EsIrSY0vMd/bl
4JzUbhuV13NnyAcnPGFEpGnCS5tUbc2b0k9FVrbykKDLVAKFd61xcQlJdRCSqt/N
KwQmIPzP1BcqXArIC24He6y5jfQ0ms+ghLSpPleIPnZn0hEVLJst830CV1PHd4v1
K00+Jx24Q2svqG003TPPq6ah//W9CLOLVy04oz/FFtI2FCX7CugKD5IJZbMpzg
lNaT3pmt77DQRqWIWZUsWsDQZVSwAz1TzGwjRUsolFWaG84DCVaEIRN7EIHhbrSj
QrLOKEg3rSr40380QL1d07yd1UICxjhLlIsToV/N9K7AsLNp9LIwc0+yICR+JC87
FE6n1XVwzrmBPozKQnbeLHKeTdfwiWctD4ZxTRkb8KkELib3Zxe7oar8WkjxAakC
```

In Wireshark, you can see a full DTLS Handshake process. First, the server sends a Hello Verify Request. The client then initiates the connection by sending a Client Hello. In response, the server sends a Server Hello, along with Certificate Fragments, the Server Key Exchange, and a Server Hello Done message. The client follows by sending the Client Key Exchange, Change Cipher Specification, and Encrypted Handshake Message. Once the handshake is verified, the client issues a new session ticket, followed by another Change Cipher Specification and Encrypted Handshake Message, establishing the connection using the session ticket as shown below:

The image shows a Wireshark packet capture window titled "Capturing from any". The packet list pane displays a series of packets. The selected packet is packet 9487, a DTLSv1 Client Hello from 10.0.0.21 to 10.0.0.20. The packet details pane shows the structure of the DTLSv1.2 Record Layer: Handshake Protocol: Client Hello. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
9484	3037.4266366...	127.0.0.1	127.0.0.1	TCP	91	59051 → 59828 [PSH, ACK] Seq=25099 Ack=2712 Win=512 Len=23 TS...
9485	3037.4266482...	127.0.0.1	127.0.0.1	TCP	68	59828 → 59051 [ACK] Seq=2712 Ack=25122 Win=512 Len=0 TSval=38...
9486	3037.8697946...	10.0.0.21	10.0.0.20	DTLSv1..	249	Client Hello
9487	3037.8698498...	10.0.0.21	10.0.0.20	DTLSv1..	249	Client Hello
9488	3037.8706014...	10.0.0.21	10.0.0.21	DTLSv1..	92	Hello Verify Request
9489	3037.8706515...	10.0.0.20	10.0.0.21	DTLSv1..	92	Hello Verify Request
9490	3037.8707714...	10.0.0.21	10.0.0.20	DTLSv1..	269	Client Hello
9491	3037.8707729...	10.0.0.21	10.0.0.20	DTLSv1..	269	Client Hello
9492	3037.8709381...	10.0.0.21	10.0.0.21	DTLSv1..	272	Server Hello, Certificate (Fragment)
9493	3037.8709420...	10.0.0.20	10.0.0.21	DTLSv1..	272	Server Hello, Certificate (Fragment)
9494	3037.8709862...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9495	3037.8709871...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9496	3037.8709884...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9497	3037.8709897...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9498	3037.8709918...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9499	3037.8709924...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9500	3037.8709943...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9501	3037.8709946...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9502	3037.8709968...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9503	3037.8709970...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9504	3037.8709988...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9505	3037.8709992...	10.0.0.20	10.0.0.21	DTLSv1..	272	Certificate (Fragment)
9506	3037.8777235...	10.0.0.20	10.0.0.21	DTLSv1..	261	Certificate (Reassembled)
9507	3037.8777263...	10.0.0.20	10.0.0.21	DTLSv1..	261	Certificate[Reassembly error, protocol DTLS: New fragment ove...
9508	3037.8777651...	10.0.0.20	10.0.0.21	DTLSv1..	272	Server Key Exchange (Fragment)
9509	3037.8777659...	10.0.0.20	10.0.0.21	DTLSv1..	272	Server Key Exchange (Fragment)
9510	3037.8777690...	10.0.0.20	10.0.0.21	DTLSv1..	272	Server Key Exchange (Fragment)
9511	3037.8777692...	10.0.0.20	10.0.0.21	DTLSv1..	272	Server Key Exchange (Fragment)
9512	3037.8777732...	10.0.0.20	10.0.0.21	DTLSv1..	240	Server Key Exchange (Reassembled), Server Hello Done
9513	3037.8777735...	10.0.0.20	10.0.0.21	DTLSv1..	240	Server Key Exchange[Reassembly error, protocol DTLS: New frag...
9514	3037.8782313...	10.0.0.21	10.0.0.20	DTLSv1..	177	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...
9515	3037.8782322...	10.0.0.21	10.0.0.20	DTLSv1..	177	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...

Datagram Transport Layer Security

- DTLSv1.2 Record Layer: Handshake Protocol: Client Hello
 - Content Type: Handshake (22)
 - Version: DTLS 1.0 (0xfeff)
 - Epoch: 0
 - Sequence Number: 0
 - Length: 192
 - Handshake Protocol: Client Hello
 - Handshake Type: Client Hello (1)
 - Length: 180
 - Message Sequence: 0
 - Fragment Offset: 0
 - Fragment Length: 180
 - Version: DTLS 1.2 (0xfefd)
 - Random: d02c23310677299fde6eca601333b02e1326523049c9e846...
 - Session ID Length: 0
 - Cookie Length: 0

Sequence Number (dtls.record.sequence_number), 6 bytes

Packets: 10014 · Displayed: 10014 (100.0%) Profile: Default

In Wireshark, you can see the Hello Verify Request Message with the Cookie Value as highlighted and shown in the below screenshot:

The screenshot shows the Wireshark interface with a packet capture of a TLS handshake. The packet list on the left shows a 'Hello Verify Request' packet (No. 9489) selected. The packet details pane on the right shows the 'Datagram Transport Layer Security' section expanded, with the 'Cookie' field highlighted. The packet bytes pane at the bottom shows the raw data of the cookie.

No.	Time	Source	Destination	Protocol	Length	Info
9484	3037.4266366...	127.0.0.1	127.0.0.1	TCP	91	50051 → 59828 [PSH, ACK] Seq=25099 Ack=2712 Win=512 Len=23 TS...
9485	3037.4266482...	127.0.0.1	127.0.0.1	TCP	68	59828 → 50051 [ACK] Seq=2712 Ack=25122 Win=512 Len=0 TSval=38...
9486	3037.8697946...	10.0.0.21	10.0.0.20	DTLSv1...	249	Client Hello
9487	3037.8698498...	10.0.0.21	10.0.0.20	DTLSv1...	249	Client Hello
9488	3037.8706014...	10.0.0.20	10.0.0.21	DTLSv1...	92	Hello Verify Request
9489	3037.8706515...	10.0.0.21	10.0.0.20	DTLSv1...	92	Hello Verify Request
9490	3037.8707714...	10.0.0.21	10.0.0.20	DTLSv1...	269	Client Hello
9491	3037.8707729...	10.0.0.21	10.0.0.20	DTLSv1...	269	Client Hello
9492	3037.8709381...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Hello, Certificate (Fragment)
9493	3037.8709420...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Hello, Certificate (Fragment)
9494	3037.8709862...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9495	3037.8709871...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9496	3037.8709894...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9497	3037.8709897...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9498	3037.8709918...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9499	3037.8709924...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9500	3037.8709943...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9501	3037.8709946...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9502	3037.8709968...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9503	3037.8709970...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9504	3037.8709988...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9505	3037.8709992...	10.0.0.20	10.0.0.21	DTLSv1...	272	Certificate (Fragment)
9506	3037.8777235...	10.0.0.20	10.0.0.21	DTLSv1...	261	Certificate (Reassembled)
9507	3037.8777263...	10.0.0.20	10.0.0.21	DTLSv1...	261	Certificate (Reassembly error, protocol DTLS: New fragment ove...
9508	3037.8777651...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Key Exchange (Fragment)
9509	3037.8777659...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Key Exchange (Fragment)
9510	3037.8777690...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Key Exchange (Fragment)
9511	3037.8777692...	10.0.0.20	10.0.0.21	DTLSv1...	272	Server Key Exchange (Fragment)
9512	3037.8777732...	10.0.0.20	10.0.0.21	DTLSv1...	240	Server Key Exchange (Reassembled), Server Hello Done
9513	3037.8777735...	10.0.0.20	10.0.0.21	DTLSv1...	240	Server Key Exchange (Reassembly error, protocol DTLS: New frag...
9514	3037.8782313...	10.0.0.21	10.0.0.20	DTLSv1...	177	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...
9515	3037.8782322...	10.0.0.21	10.0.0.20	DTLSv1...	177	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...

Cookie: ef8d1b1605f3f0b3a13f3a4cd3555866c0626ed8

Hello Verify Request with Cookie Value:

The screenshot shows the details pane of the 'Hello Verify Request' packet. The 'Cookie' field is highlighted, showing the value 'ef8d1b1605f3f0b3a13f3a4cd3555866c0626ed8'.

Content Type: Handshake (22)
Version: DTLS 1.0 (0xfeff)
Epoch: 0
Sequence Number: 0
Length: 35
Handshake Protocol: Hello Verify Request
Handshake Type: Hello Verify Request (3)
Length: 23
Message Sequence: 0
Fragment Offset: 0
Fragment Length: 23
Version: DTLS 1.0 (0xfeff)
Cookie Length: 20
Cookie: ef8d1b1605f3f0b3a13f3a4cd3555866c0626ed8

- 1.5. (2pts) Run a TLS1.2 server and a TLS1.2 client. Configure the server to request a certificate from the client. Configure the client with a cert and a private key.

First let us generate the certificate using the command **openssl req -x509 -newkey rsa:4096 client.key -out client.cert -days365** and private key for the client using the command **openssl rsa -in client.key -out client.key_no_pass** as shown below:

```
core@core-VM: ~/HW4
core@core-VM:~/HW4$ openssl req -x509 -newkey rsa:4096 -keyout client.key -out client.cert -days 365
Generating a RSA private key
.....++++
writing new private key to 'client.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Virginia
Locality Name (eg, city) []:Arlington
Organization Name (eg, company) [Internet Widgits Pty Ltd]:GWU
Organizational Unit Name (eg, section) []:SEAS
Common Name (e.g. server FQDN or YOUR name) []:Namana
Email Address []:namanaty@gmail.com
core@core-VM:~/HW4$ openssl rsa -in client.key -out client.key_no_pass
Enter pass phrase for client.key:
writing RSA key
core@core-VM:~/HW4$
```

Next, run the TLS1.2 server on node 1 and set it up to request the client certificate using the command `openssl s_server -accept 4433 -cert www.cert -key www.key no pass -tls1_2 -verify 1` as shown below:

```
root@n1:/tmp/pycore.1/n1.conf# cd /home/core/HW4
root@n1:/home/core/HW4# openssl s_server -accept 4433 -cert www.cert -key www.key_no_pass -tls1_2 -verify 1
verify depth is 1
Using default temp DH parameters
ACCEPT
```

Next, run the TLS1.2 Client with its certificate and private key using the command **openssl s_client -cert client.cert -key client.key_no_pass -tls1_2 -connect 10.0.0.20:4433**. You can see that the client is connected to the server as shown:

```

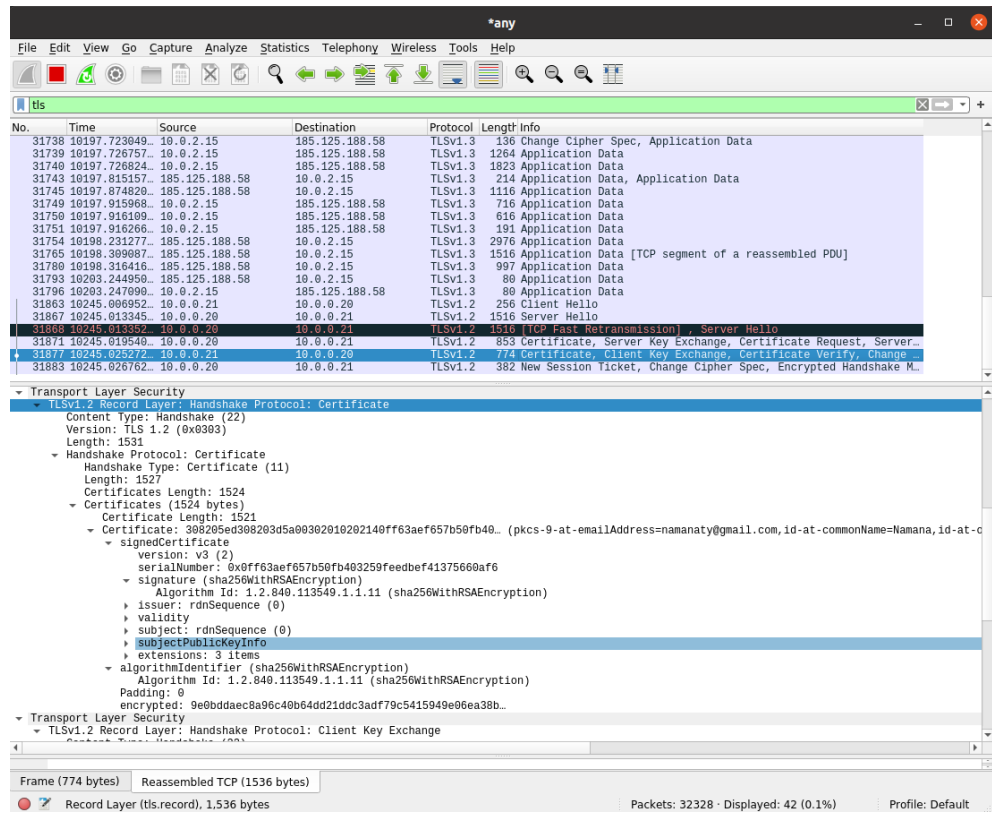
root@n2:/tmp/pycore.1/n2.conf# cd /home/core/HW4
root@n2:/home/core/HW4# openssl s_client -cert client.cert -key client.key_no_pass -tls1_2 -connect 10.0.0.20:4433

CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
  i:C = US, ST = Virginia, L = Arlington, O = GWU, OU = SEAS, CN = Namana, emailAddress = namanaty@gmail.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIF7TCCAG9wAgIBAgIUxEwNj8wR2gkvEejHwW2ekkg0u1IwDQYJKoZIhvcNAQEL
BQAwgYXUCZAJBGNVBAYTALVTMRwDwYDVQQIDAhHwAkJnaW5pYTESMBAGA1UEBwwJ
QXJ3aW5ndG9uMwQwYXUCYDVQKQDANHV1UxDTALBgNVBAsMBFNFQVMxZDZANBgNVBAMM
Bk5hbWVfYUeHMB8GCSqGSIb3DQEJARYSbmFtYU5hdHlAZ21haWwY29tMB4XD0TI0
MTAxNjA5MjY0N0F0XDTI1MTAxNjA5MjY0N0FwYXUCZAJBGNVBAYTALVTMRwDwYD
QYKIDAhHwAkJnaW5pYTESMBAGA1UEBwwJQXJ3aW5ndG9uMwQwYXUCYDVQKQDANHV1Ux
DTALBgNVBAsMBFNFQVMxZDZANBgNVBAMMBk5hbWVfYUeHMB8GCSqGSIb3DQEJARYS
bmFtYU5hdHlAZ21haWwY29tMB4XD0TI0MTAxNjA5MjY0N0F0XDTI1MTAxNjA5MjY0
N0FwYXUCZAJBGNVBAYTALVTMRwDwYDVQQIDAhHwAkJnaW5pYTESMBAGA1UEBwwJ
QXJ3aW5ndG9uMwQwYXUCYDVQKQDANHV1UxDTALBgNVBAsMBFNFQVMxZDZANBgNVBAMM
Bk5hbWVfYUeHMB8GCSqGSIb3DQEJARYSbmFtYU5hdHlAZ21haWwY29tMB4XD0TI0
MTAxNjA5MjY0N0F0XDTI1MTAxNjA5MjY0N0FwYXUCZAJBGNVBAYTALVTMRwDwYDV

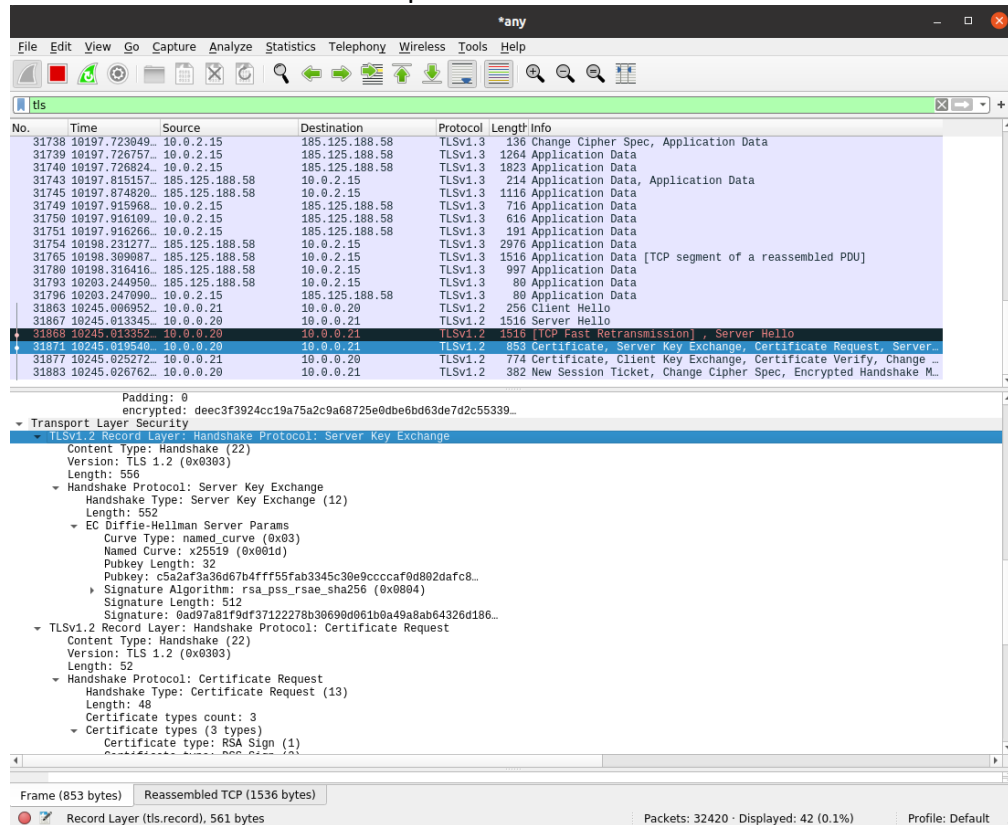
```

1.5.1. Show a screenshot of the certificate request from server, certificate from client, and certificate verify from the client.

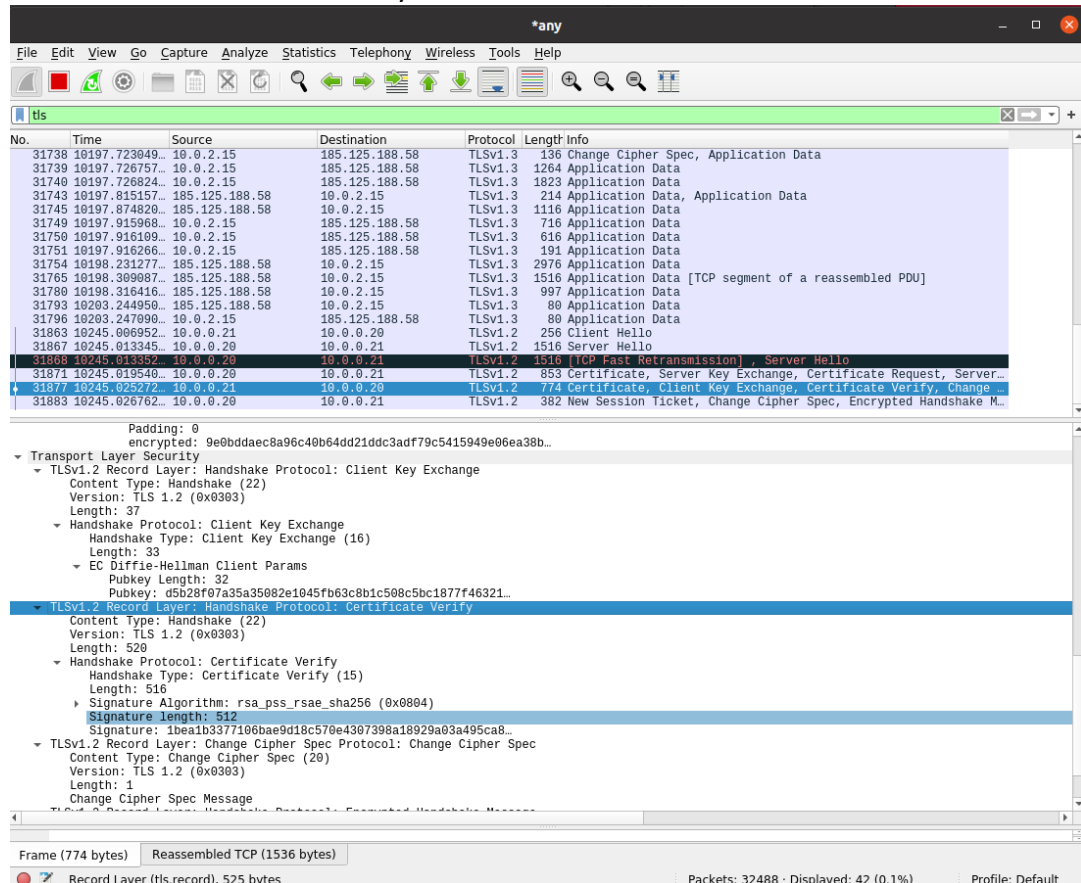
You can see the Certificate from the Client on Wireshark as shown below:



You can see the Certificate Request from the Server in Wireshark as shown:



You can see the Certificate Verify from the Client on Wireshark as shown below:

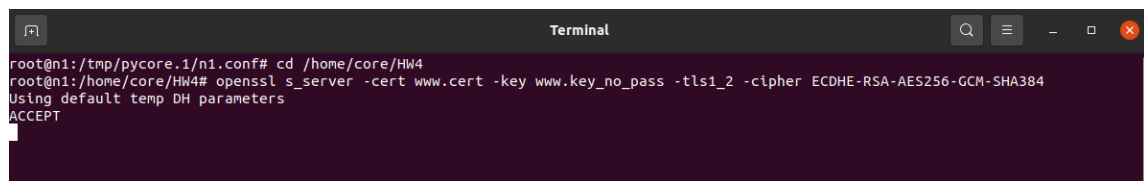


1.6. Run a TLS1.2 server configured to use the cipher: ECDHE-RSA-AES256-GCM-SHA384 and a TLS1.2 client configured to use the cipher: DHE-RSA-AES256-GCM-SHA384.

1.6.1. Hint: look at the output of openssl ciphers

1.6.2. (1pt) Show the command you used

First, run the TLS1.2 Server on node 1 using the command **openssl s_server -cert www.cert -key www.key_no_pass -tls1_2 -cipher ECDHE-RSA-AES256-GCM-SHA384** to use the required cipher as shown below:



Next, run the TLS1.2 Client on node 2 using the command **openssl s_client -cipher DHE-RSA-AES256-GCM-SHA384 -connect 10.0.0.20 -tls1_2** to use the required cipher. You can see that the client is unable to connect to the server as shown below:

```
Terminal
root@n2:/tmp/pycore.1/n2.conf# cd /home/core/HW4
root@n2:/home/core/HW4# openssl s_client -cipher DHE-RSA-AES256-GCM-SHA384 -connect 10.0.0.20 -tls1_2
CONNECTED(00000003)
140053774734656:error:14094410:SSL routines:ssl3_read_bytes:ssl3 alert handshake failure:../ssl/record/rec_
layer_s3.c:1552:SSL alert number 40
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 112 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1.2
    Cipher   : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1729049258
    Timeout : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
---
```

1.6.3. (1pt) Describe what happened. What Alert message type and what Level are generated?

The client cannot connect to the server because they do not share a common cipher. Due to the mismatch in ciphers, a secure connection cannot be established. In Wireshark, the alert message shows the Level as **"Fatal"** with the Description **"Handshake Failure."** Additionally, the client's terminal displays alert number as 40, indicating a handshake failure as shown below:

The Wireshark packet capture shows a TLS handshake failure. The packet list displays the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
112	120.312118571	127.0.0.1	127.0.0.1	TCP	68	39254 → 50051 [ACK] Seq=607 Ack=1391 Win=512 Len=0 TSval=1712.
113	122.730505011	00:00:00:aa:00:01	10.0.0.20	ARP	44	Who has 10.0.0.20? Tell 10.0.0.21
114	122.730532680	00:00:00:aa:00:01	10.0.0.20	ARP	44	Who has 10.0.0.20? Tell 10.0.0.21
115	122.730505011	00:00:00:aa:00:01	10.0.0.20	ARP	44	Who has 10.0.0.20? Tell 10.0.0.21
116	122.730548747	00:00:00:aa:00:00	10.0.0.20	ARP	44	10.0.0.20 is at 00:00:00:aa:00:00
117	122.730550373	00:00:00:aa:00:00	10.0.0.20	ARP	44	10.0.0.20 is at 00:00:00:aa:00:00
118	122.730553446	10.0.0.21	10.0.0.20	TCP	76	33662 → 4433 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
119	122.730555183	10.0.0.21	10.0.0.20	TCP	76	[TCP Out-Of-Order] 33662 → 4433 [SYN] Seq=0 Win=64240 Len=0 M.
120	122.730570277	10.0.0.20	10.0.0.21	TCP	76	4433 → 33662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460
121	122.730571409	10.0.0.20	10.0.0.21	TCP	76	[TCP Out-Of-Order] 4433 → 33662 [SYN, ACK] Seq=0 Ack=1 Win=65
122	122.73051899	10.0.0.21	10.0.0.20	TCP	68	33662 → 4433 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=20360007.
123	122.730502627	10.0.0.21	10.0.0.20	TCP	68	[TCP Dup ACK 122#1] 33662 → 4433 [ACK] Seq=1 Ack=1 Win=64256
124	122.736381920	10.0.0.21	10.0.0.20	TLSv1.2	180	Client Hello
125	122.736389114	10.0.0.21	10.0.0.20	TCP	180	[TCP Retransmission] 33662 → 4433 [PSH, ACK] Seq=1 Ack=1 Win=
126	122.736436027	10.0.0.20	10.0.0.21	TCP	68	4433 → 33662 [ACK] Seq=1 Ack=113 Win=65152 Len=0 TSval=195749.
127	122.736437991	10.0.0.20	10.0.0.21	TCP	68	[TCP Dup ACK 126#1] 4433 → 33662 [ACK] Seq=1 Ack=113 Win=6515
128	122.737013065	10.0.0.20	10.0.0.21	TLSv1.2	75	Alert (Level: Fatal, Description: Handshake Failure)
129	122.737017809	10.0.0.20	10.0.0.21	TCP	75	[TCP Retransmission] 4433 → 33662 [PSH, ACK] Seq=1 Ack=113 W1
130	122.737024542	10.0.0.21	10.0.0.20	TCP	68	33662 → 4433 [ACK] Seq=113 Ack=8 Win=64256 Len=0 TSval=203600.
131	122.737071606	10.0.0.21	10.0.0.20	TCP	68	[TCP Dup ACK 130#1] 33662 → 4433 [ACK] Seq=113 Ack=8 Win=6425
132	122.737103077	10.0.0.20	10.0.0.21	TCP	68	4433 → 33662 [FIN, ACK] Seq=8 Ack=113 Win=65152 Len=0 TSval=1
133	122.737169326	10.0.0.20	10.0.0.21	TCP	68	[TCP Out-Of-Order] 4433 → 33662 [FIN, ACK] Seq=8 Ack=113 Win=
134	122.737188997	10.0.0.21	10.0.0.20	TCP	68	33662 → 4433 [FIN, ACK] Seq=113 Ack=8 Win=64256 Len=0 TSval=2
135	122.737190790	10.0.0.21	10.0.0.20	TCP	68	[TCP Out-Of-Order] 33662 → 4433 [FIN, ACK] Seq=113 Ack=8 Win=
136	122.737194053	10.0.0.20	10.0.0.21	TCP	68	4433 → 33662 [ACK] Seq=9 Ack=114 Win=65152 Len=0 TSval=195749.
137	122.737237194	10.0.0.20	10.0.0.21	TCP	68	[TCP Dup ACK 136#1] 4433 → 33662 [ACK] Seq=9 Ack=114 Win=6515

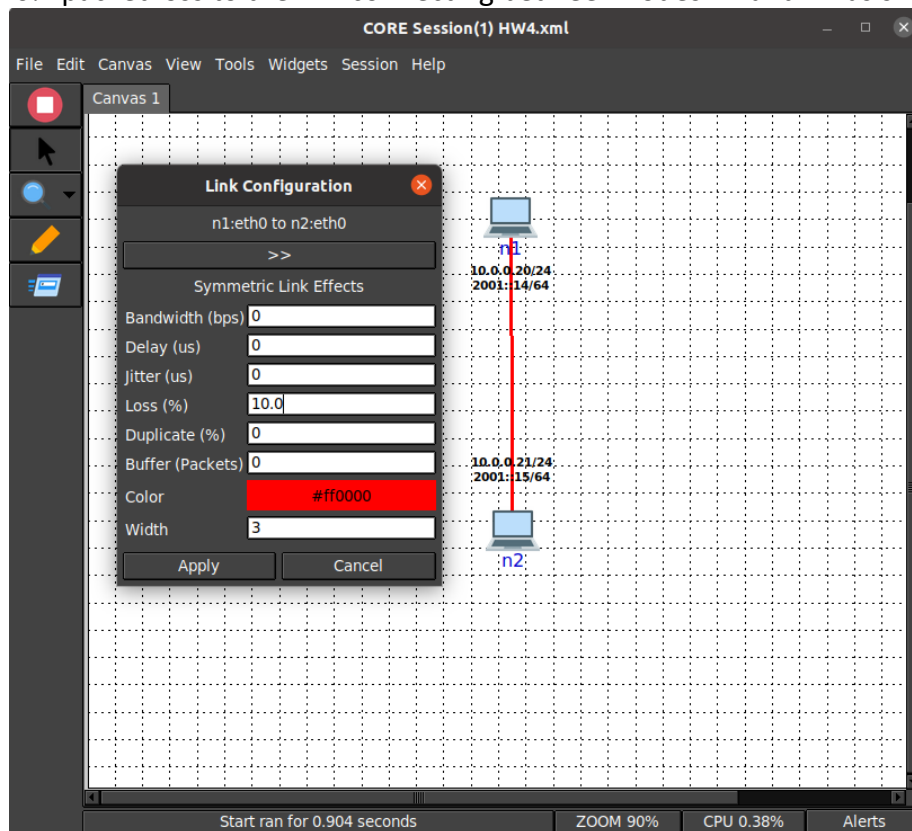
Frame 128: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface any, id 0

- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.0.20, Dst: 10.0.0.21
- Transmission Control Protocol, Src Port: 4433, Dst Port: 33662, Seq: 1, Ack: 113, Len: 7
- Transport Layer Security
 - TLSv1.2 Record Layer: Alert (Level: Fatal, Description: Handshake Failure)
 - Content Type: Alert (21)
 - Version: TLS 1.2 (0x0303)
 - Length: 2
 - Alert Message
 - Level: Fatal (2)
 - Description: Handshake Failure (40)

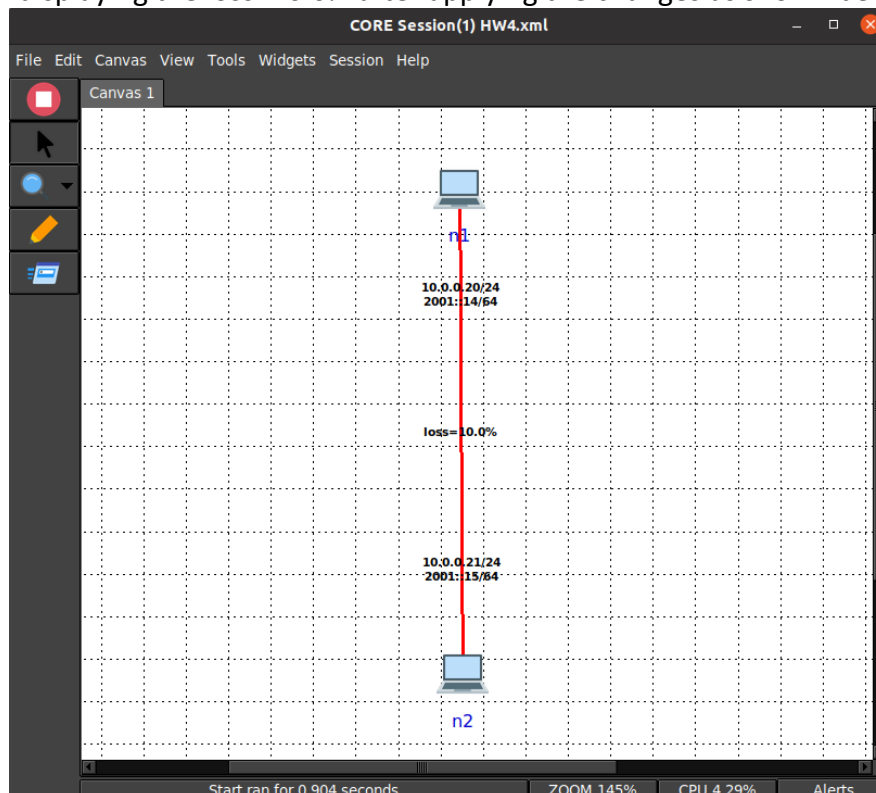
any: <live capture in progress> Packets: 209 · Displayed: 209 (100.0%) Profile: Default

- 1.7. (1.5pts) Repeat 1.2 (do not show screenshot of handshake) but add a 10% packet loss to the link between the two nodes beforehand.

Add a 10% packet loss to the link connecting between nodes n1 and n2 as shown:



Core-GUI displaying the loss=10.0% after applying the changes as shown below:



Repeat the steps from section 1.2 to configure the TLS 1.2 server and client.

1.7.1. Send 10 messages from client to server: “message 1”, “message 2”, ...etc. Show a screenshot of the server terminal after 2 minutes of sending the last message showing the list of messages received.

Client: Sending 10 messages from client to server as shown below:

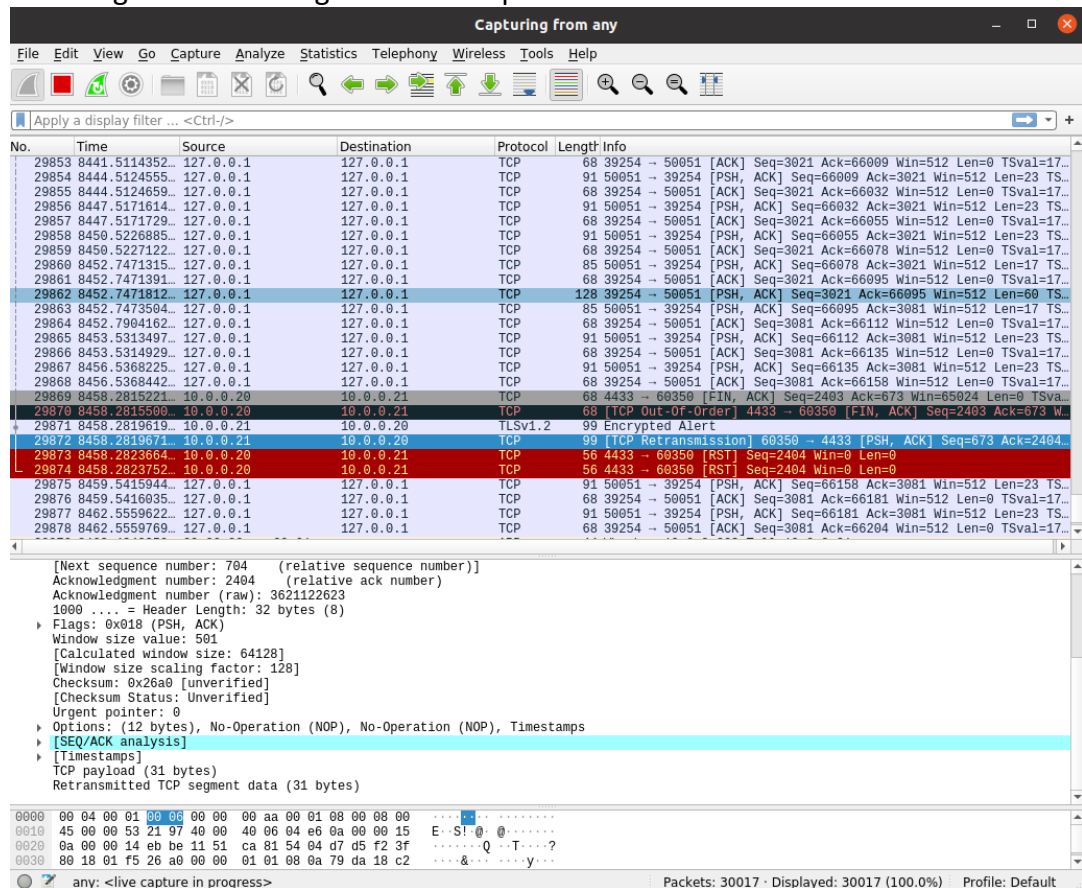
```
Terminal
Protocol : TLSv1.2
Cipher : ECDHE-RSA-AES256-GCM-SHA384
Session-ID: B51190B809C590032E4282115452B0B4B49A35D39C8CDFE3D377D875E57A5D77
Session-ID-ctx:
Master-Key: F75A6A5C4626EB637ED78429BA0D71395507D96F603D67D4D10D30CA42655991C65A2EE421B0F5283417F146696FF71D
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 7200 (seconds)
TLS session ticket:
0000 - 48 0d 92 92 bc f6 69 72-04 24 27 72 5e b6 b8 b4 H....ir.$'r^...
0010 - 03 ba b4 33 a8 95 48 dd-d5 a9 a7 37 d1 b0 a6 a1 ...3..H...7....
0020 - c5 5b 27 43 73 c0 b8 6c-32 65 e2 d3 14 a0 51 c3 .['Cs..l2e....Q.
0030 - 6c 06 2e a5 02 de 91 93-98 c8 d6 4a 29 6e 10 45 l.....J)n.E
0040 - c9 17 ba 30 21 77 5a f7-fb cd 29 c6 ad 2a 2a f4 ...0!wZ....)*.
0050 - e2 76 58 f6 db 06 7b 35-30 b7 e0 c0 16 2d ff 8d .vX...[S0.....
0060 - 75 3e 83 b0 74 eb 2c bc-9a 3c fe 72 c9 2a 9d 42 u>..t,...<.r*.B
0070 - da 45 7a a3 2b 14 5f ea-76 94 c4 5d 2a f5 1c 79 .Ez.+...v...]*..y
0080 - 18 cb 29 7f a3 d1 18 35-8c 04 e3 c3 c9 11 28 2b ..)....5.....(+
0090 - 73 87 64 52 58 ab 5d 46-7a c3 4f ae 07 83 11 80 s.dRX.]Fz.O....

Start Time: 1729097208
Timeout : 7200 (sec)
Verify return code: 18 (self signed certificate)
Extended master secret: yes
---
message 1
message 2
message 3
message 4
message 5
message 6
message 7
message 8
message 9
message 10
```

Server: All 10 messages were received by the Server from Client as shown below:

```
Terminal
root@n1:/tmp/pycore.1/n1.conf# cd /home/core/HW4
root@n1:/home/core/HW4# openssl s_server -accept 4433 -cert www.cert -key www.key_no_pass -tls1_2
Using default temp DH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MfoCAQCACgMDBALAMABDD3WmpcRtbrY37Xhcm6DXE5VQfZb2A9Z9TRDTDKQmVZ
kcZaLuQhsPUoNBfxRmIv9x2hBgIEZw/t+KIEAgIcIKQGBAQBAARQMCAQE=
-----END SSL SESSION PARAMETERS-----
Shared ciphers: ECDHE-ECDSA-AES256-GCM-SHA384: ECDHE-RSA-AES256-GCM-SHA384: DHE-RSA-AES256-GCM-SHA384: ECDHE-ECDSA-CHACHA20-POLY1305: ECDHE-RSA-CHACHA20-POLY1305: DHE-RSA-CHACHA20-POLY1305: ECDHE-ECDSA-AES128-GCM-SHA256: ECDHE-RSA-AES128-GCM-SHA256: DHE-RSA-AES128-GCM-SHA256: ECDHE-ECDSA-AES256-SHA384: ECDHE-RSA-AES256-SHA384: DHE-RSA-AES256-SHA256: ECDHE-ECDSA-AES128-SHA256: ECDHE-RSA-AES128-SHA256: DHE-RSA-AES128-SHA256: ECDHE-ECDSA-AES256-SHA: ECDHE-RSA-AES256-SHA: DHE-RSA-AES256-SHA: ECDHE-ECDSA-AES128-SHA: ECDHE-RSA-AES128-SHA: DHE-RSA-AES128-SHA: AES256-GCM-SHA384: AES128-GCM-SHA256: AES256-SHA256: AES128-SHA256: AES256-SHA: AES128-SHA
Signature Algorithms: ECDSA+SHA256: ECDSA+SHA384: ECDSA+SHA512: Ed25519: Ed448: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA+SHA256: RSA+SHA384: RSA+SHA512: ECDSA+SHA224: RSA+SHA224: DSA+SHA224: DSA+SHA256: DSA+SHA384: DSA+SHA512
Shared Signature Algorithms: ECDSA+SHA256: ECDSA+SHA384: ECDSA+SHA512: Ed25519: Ed448: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA+SHA256: RSA+SHA384: RSA+SHA512: ECDSA+SHA224: RSA+SHA224: DSA+SHA224: DSA+SHA256: DSA+SHA384: DSA+SHA512
Supported Elliptic Curve Point Formats: uncompressed: ansiX962_compressed_prime: ansiX962_compressed_char2
Supported Elliptic Groups: X25519: P-256: X448: P-521: P-384
Shared Elliptic groups: X25519: P-256: X448: P-521: P-384
CIPHER is ECDHE-RSA-AES256-GCM-SHA384
Secure Renegotiation IS supported
message 1
message 2
message 3
message 4
message 5
message 6
message 7
message 8
message 9
message 10
```

Wireshark shows the packets being transferred between Server and client during the transactions. Packet loss does not affect the messages, as the server successfully receives all messages from the client. TLS uses TCP, which provides ordered data delivery through its sequence acknowledgment. TCP ensures that messages are received in the correct order, as each message is sent only after receiving an acknowledgment for the previous one.



1.8. (1.5pts) Repeat 1.4 (do not show screenshot of handshake) but add a 10% packet loss to the link between the two nodes beforehand.

1.8.1. Send 10 messages from client to server: “message 1”, “message 2”, ...etc. Show a screenshot of the server terminal after 2 minutes of sending the last message showing the list of messages received.

Packet Loss of 10% is already added to the link connecting between the nodes n1 and n2 as part of section 1.7. Repeat the steps from section 1.4 to configure the DTLS 1.2 server and client.

Client: Sending 10 messages from client to server as shown below:

```
Terminal
No ALPN negotiated
SSL-Session:
  Protocol : DTLSv1.2
  Cipher : ECDHE-RSA-AES256-GCM-SHA384
  Session-ID: 9DEA9F5A9D5F78D238732971DFC91909AEE7D833DBEAAE4AA65C2E70481409B9
  Session-ID-ctx:
  Master-Key: A328087FB591C1B1B92C31D62EB53C687CD094EAC4CEB1F90DB411EBF59135B618D13C5B5B6A2420
D821EDB8444680D5
  PSK identity: None
  PSK identity hint: None
  SRP username: None
  TLS session ticket lifetime hint: 7200 (seconds)
  TLS session ticket:
0000 - 57 98 1b 13 37 b4 25 93-1b 99 e8 78 5d 47 da ee W...7.%....x]G..
0010 - 1e 16 8d e4 bc 4c 8d b5-55 94 0b ef 85 81 33 70 .....L..U.....3p
0020 - 48 4b 72 05 2e 8f c0 43-b5 7a 65 d8 94 95 78 cc HKr....C.ze...x.
0030 - 0c 76 56 82 30 91 31 5a-d7 d2 97 16 6d 91 42 f6 .vV.0.1Z....m.B.
0040 - 35 ff 07 8c 72 c8 5b f2-7f 4d d2 d4 35 c0 05 b5 5...r.[...M..S...
0050 - 85 0f a2 d9 f8 dd 99 ee-a1 1d 26 b8 09 ea 36 81 .....&....6.
0060 - c2 9e 84 33 db 78 71 a5-c9 4e 23 01 2b 33 16 b8 ...3.xq..N#.+3..
0070 - fa f2 0f 0b 8e 3a 68 ab-9c ec 1b 9d 19 55 25 60 .....:h.....U%'
0080 - e6 fc 72 e6 38 49 27 e5-a4 5d ba b0 79 2b 70 07 ..r.8I'...]..y+p.
0090 - 29 99 97 5f 07 a7 5d f2-f1 f6 32 3f 27 31 80 29 )......]...2?'1.)

  Start Time: 1729098158
  Timeout : 7200 (sec)
  Verify return code: 18 (self signed certificate)
  Extended master secret: yes
---
message 1
message 2
message 3
message 4
message 5
message 6
message 7
message 8
message 9
message 9
message 10
```

Server: Here you can observe that one message (message 4) is lost in the transmission between client and server as shown below:

```
Terminal
root@n1:/tmp/pycore.1/n1.conf# cd /home/core/HW4

root@n1:/home/core/HW4# openssl s_server -accept 4433 -cert www.cert -key www.key_no_pass -dtls1_2
Using default temp DH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MfSCAQECawD+/QQCwDAEAAQwoygIf7WRwbG5LDHWLrU8aHzQlOrEzrH5DbQR6/WR
NBYY0TxbW2okIngh7bHERo3VoQYCBGcP8a2iBAICHcCkBgQEAAAK0DAgEB
-----END SSL SESSION PARAMETERS-----
Shared ciphers: ECDHE-ECDSA-AES256-GCM-SHA384: ECDHE-RSA-AES256-GCM-SHA384: DHE-RSA-AES256-GCM-SHA384: ECD
HE-ECDSA-CHACHA20-POLY1305: ECDHE-RSA-CHACHA20-POLY1305: DHE-RSA-CHACHA20-POLY1305: ECDHE-ECDSA-AES128-GC
M-SHA256: ECDHE-RSA-AES128-GCM-SHA256: DHE-RSA-AES128-GCM-SHA256: ECDHE-ECDSA-AES256-SHA384: ECDHE-RSA-AES
256-SHA384: DHE-RSA-AES256-SHA256: ECDHE-ECDSA-AES128-SHA256: ECDHE-RSA-AES128-SHA256: DHE-RSA-AES128-SHA2
56: ECDHE-ECDSA-AES256-SHA: ECDHE-RSA-AES256-SHA: DHE-RSA-AES256-SHA: ECDHE-ECDSA-AES128-SHA: ECDHE-RSA-AES
128-SHA: DHE-RSA-AES128-SHA: AES256-GCM-SHA384: AES128-GCM-SHA256: AES256-SHA256: AES128-SHA256: AES256-SHA:
AES128-SHA
Signature Algorithms: ECDSA+SHA256: ECDSA+SHA384: ECDSA+SHA512: Ed25519: Ed448: RSA-PSS+SHA256: RSA-PSS+SHA3
84: RSA-PSS+SHA512: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA+SHA256: RSA+SHA384: RSA+SHA512: ECDSA+
SHA224: RSA+SHA224: DSA+SHA224: DSA+SHA256: DSA+SHA384: DSA+SHA512
Shared Signature Algorithms: ECDSA+SHA256: ECDSA+SHA384: ECDSA+SHA512: Ed25519: Ed448: RSA-PSS+SHA256: RSA-P
SS+SHA384: RSA-PSS+SHA512: RSA-PSS+SHA256: RSA-PSS+SHA384: RSA-PSS+SHA512: RSA+SHA256: RSA+SHA384: RSA+SHA512
: ECDSA+SHA224: RSA+SHA224: DSA+SHA224: DSA+SHA256: DSA+SHA384: DSA+SHA512
Supported Elliptic Curve Point Formats: uncompressed: ansiX962_compressed_prime: ansiX962_compressed_cha
r2
Supported Elliptic Groups: X25519:P-256: X448:P-521: P-384
Shared Elliptic groups: X25519:P-256: X448:P-521: P-384
CIPHER is ECDHE-RSA-AES256-GCM-SHA384
Secure Renegotiation IS supported
Read BLOCK
message 1
message 2
message 3
message 5
message 6
message 7
message 8
message 9
message 10
```

This issue occurs only in DTLS, not in TLS, because DTLS operates over UDP, where packet loss can occur during transmission. In contrast, TLS uses TCP, which ensures reliable delivery of every packet.

In the screenshot above, you can see that message 4 is missing from the server terminal. This is due to DTLS using UDP for communication, which lacks the ability to track and confirm receipt of all messages and does not guarantee message sequencing. On the other hand, TCP ensures that messages are delivered in order through its sequence acknowledgment mechanism. With TCP, a message is sent only after receiving acknowledgment for the previous one.

The Wireshark displays the packet exchange between the server and client as shown below:

