Network Security – CSCI_6541_80

Namana Y Tarikere – G21372717

Homework Assignment – 4

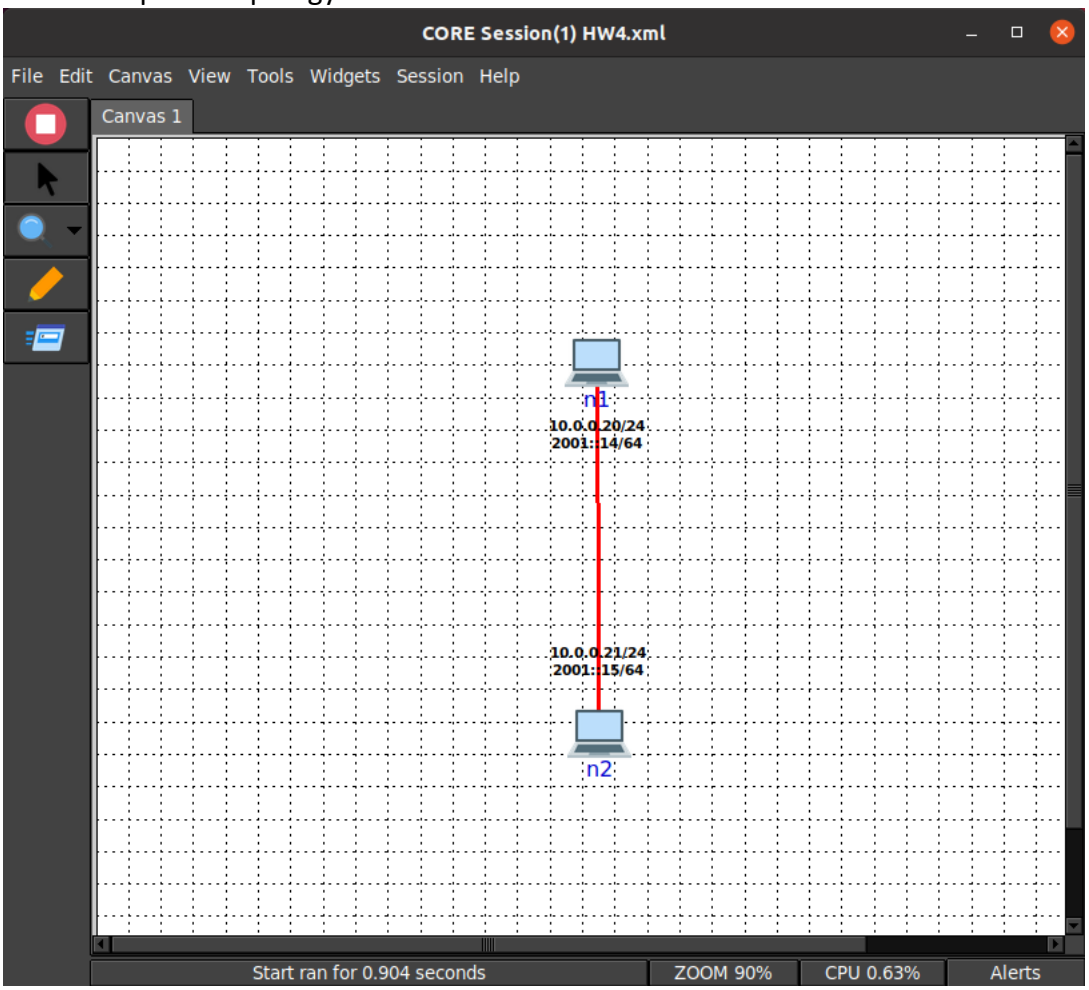
TLS Assignment

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

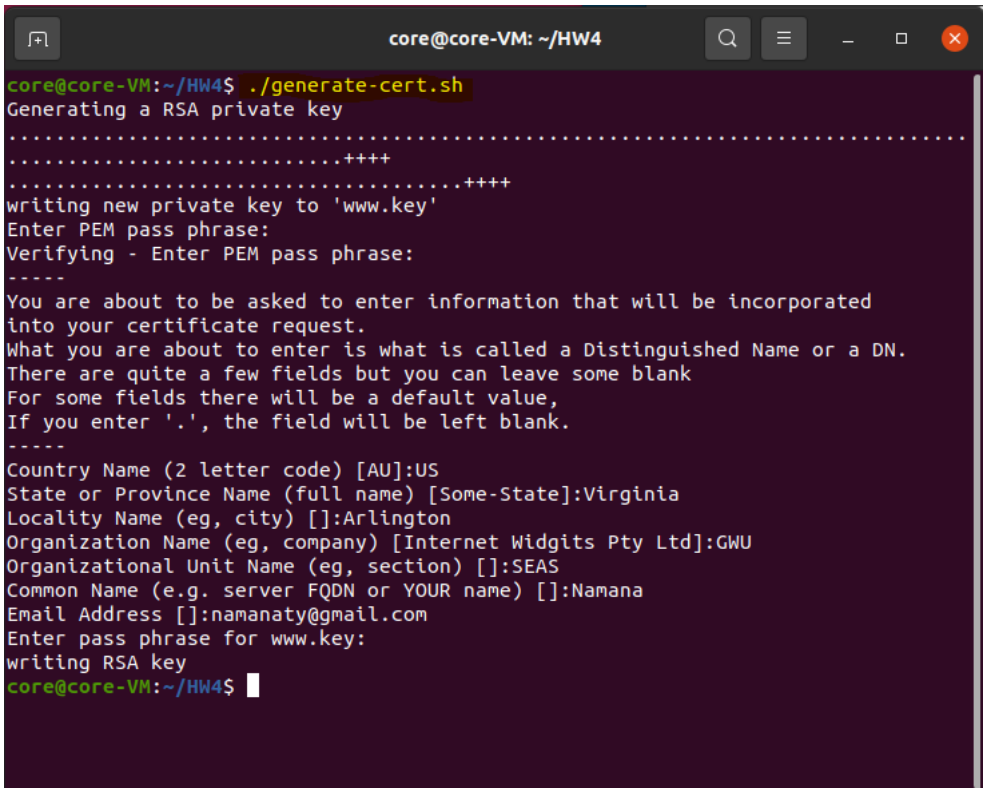
Part 1: TLS (10 points)

Created the required topology as shown below in CORE:

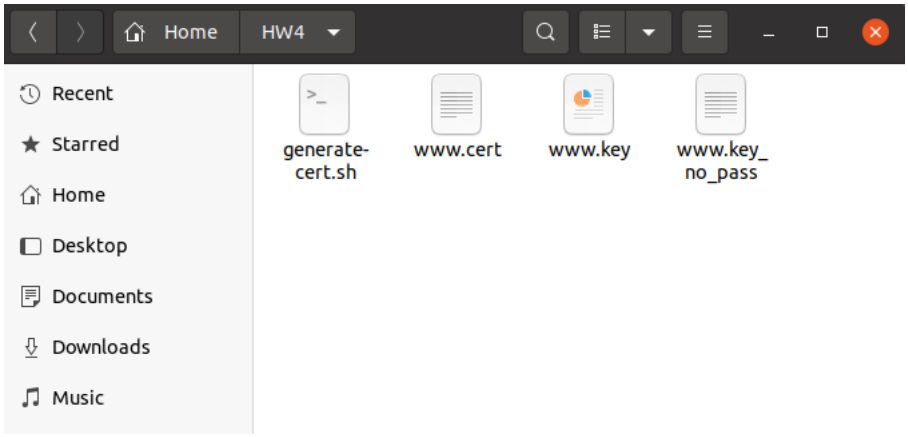


1. **Use openssl s_client and s_server to create the following configurations**
   1.1. Use the script provided to generate X.509 certs.

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:



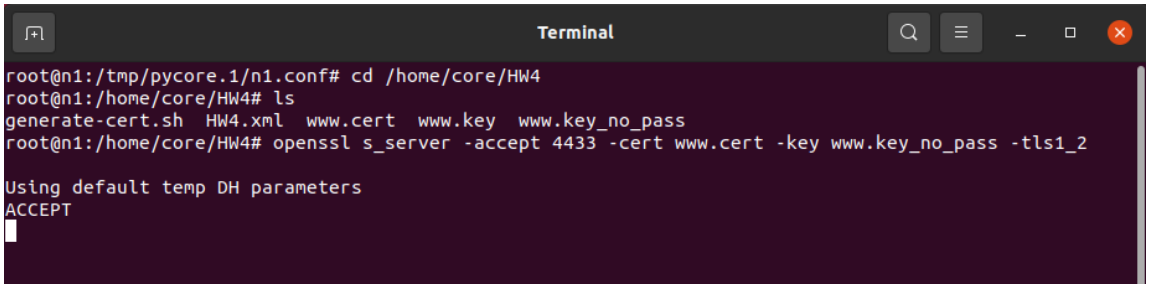
The script generated 3 files – a certificate and 2 private keys as shown below:



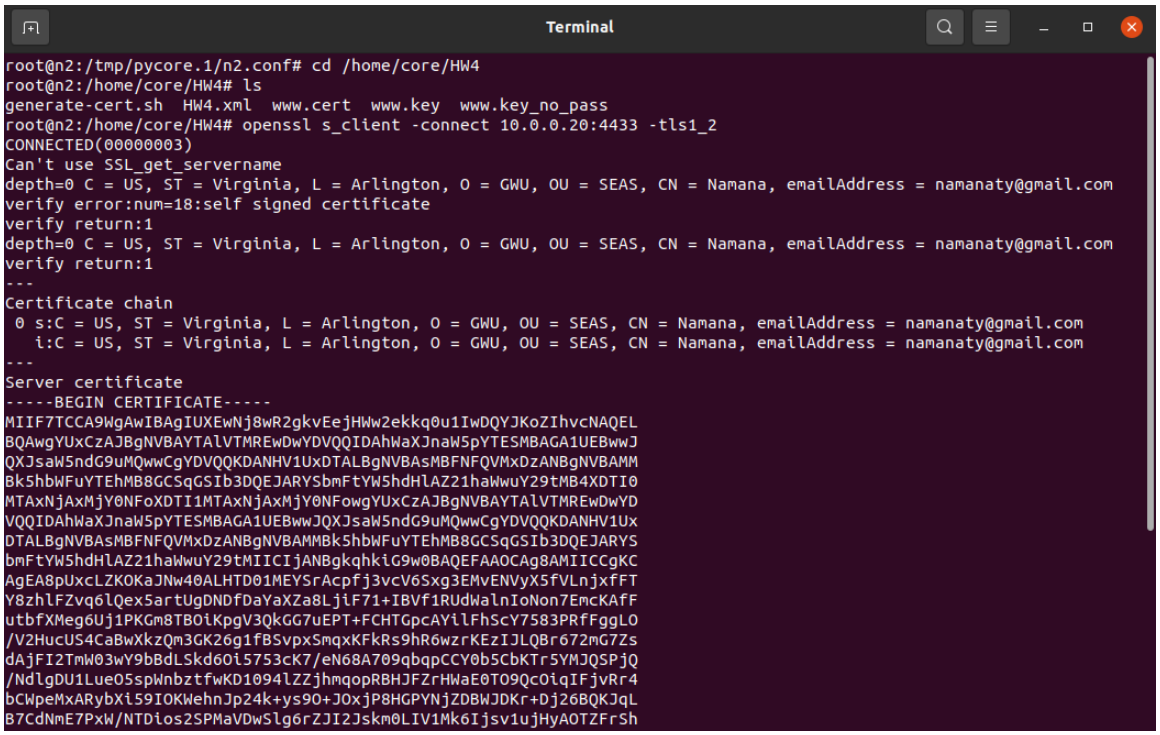
1.2. (1pts) Run a TLS1.2 server and a TLS1.2 client. Show the command you used on both and a screenshot of the TLS full handshake in Wireshark

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**,**

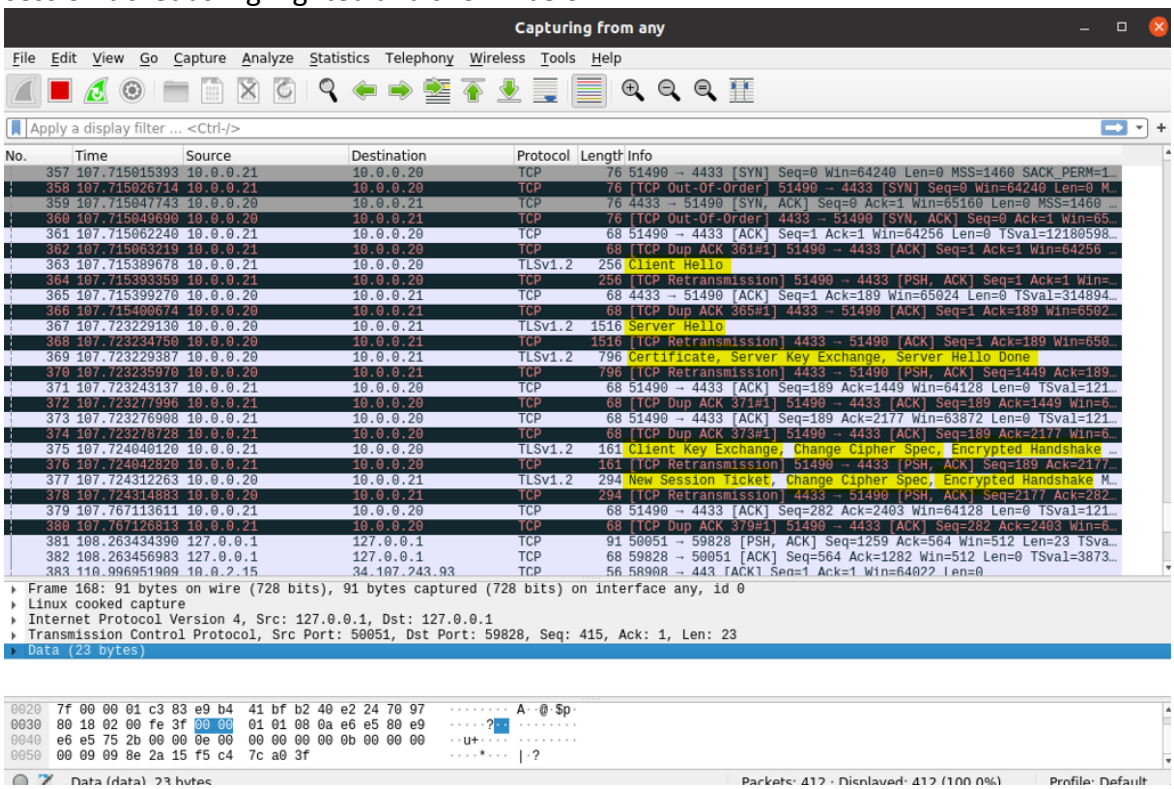
openssl s_server: starts the server using openssl
-accept 4433: listening on port 4433
-cert www.cert**:** setting up www.cert as the certificate used for connection
-key www.key_no_pass: using www.key_no_pass as the key for the connection
- tls1_2: the type of SSL/TLS version to set up the connection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



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:

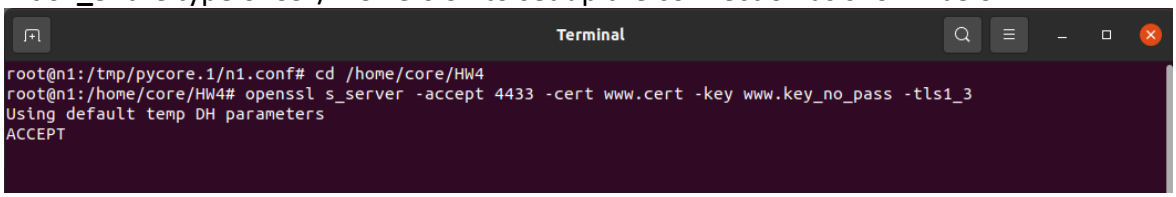


In Wireshark, we can see the full TLS Handshake. Initially, the client starts the connection by sending a Client Hello. In response, the server sends a Server Hello along with its Certificate, Server Key Exchange, and a Server Hello Done message. Lastly, the client sends the Client Key Exchange, Change 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:



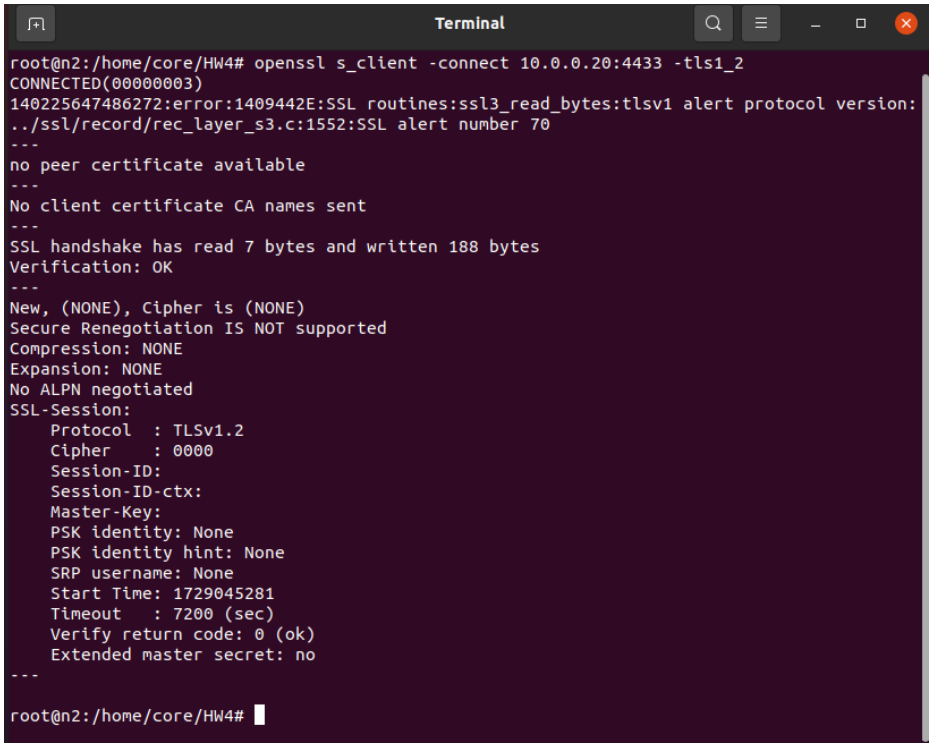
1.3. (1pts) Run a TLS1.3 server and a TLS1.2 client. Describe what happened. What Alert message type and what Level are generated?

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:

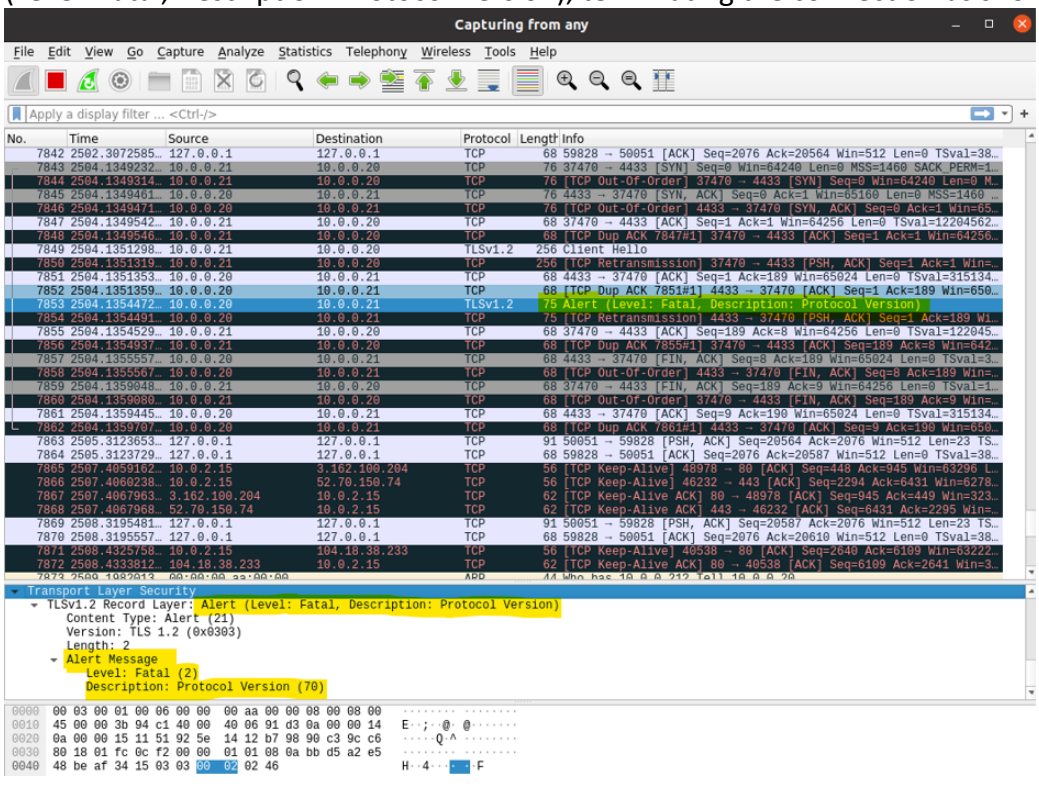


Next, run TLS1.2 Client on n2 node using the command **openssl s_client -connect 10.0.0.20:4433 -tls1_2**. Here,
openssl s_client: starting SSL/TLS in the client using OpenSSL

-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:



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:

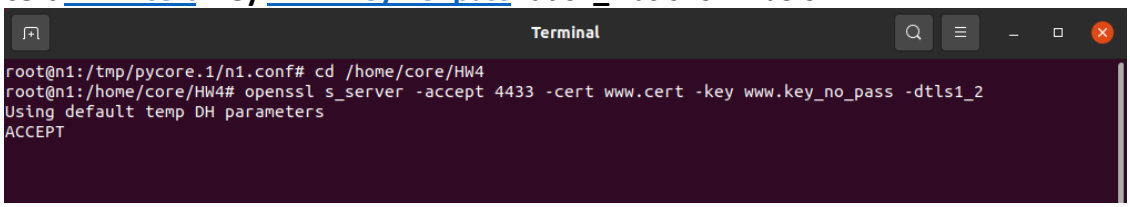


The client and server attempt to initiate an SSL handshake, but it fails because the server is using TLS 1.3, while the client is requesting a connection via TLS 1.2Since TLS 1.3 is not backward compatible and each protocol has its own distinct connection establishment process, different protocol versions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

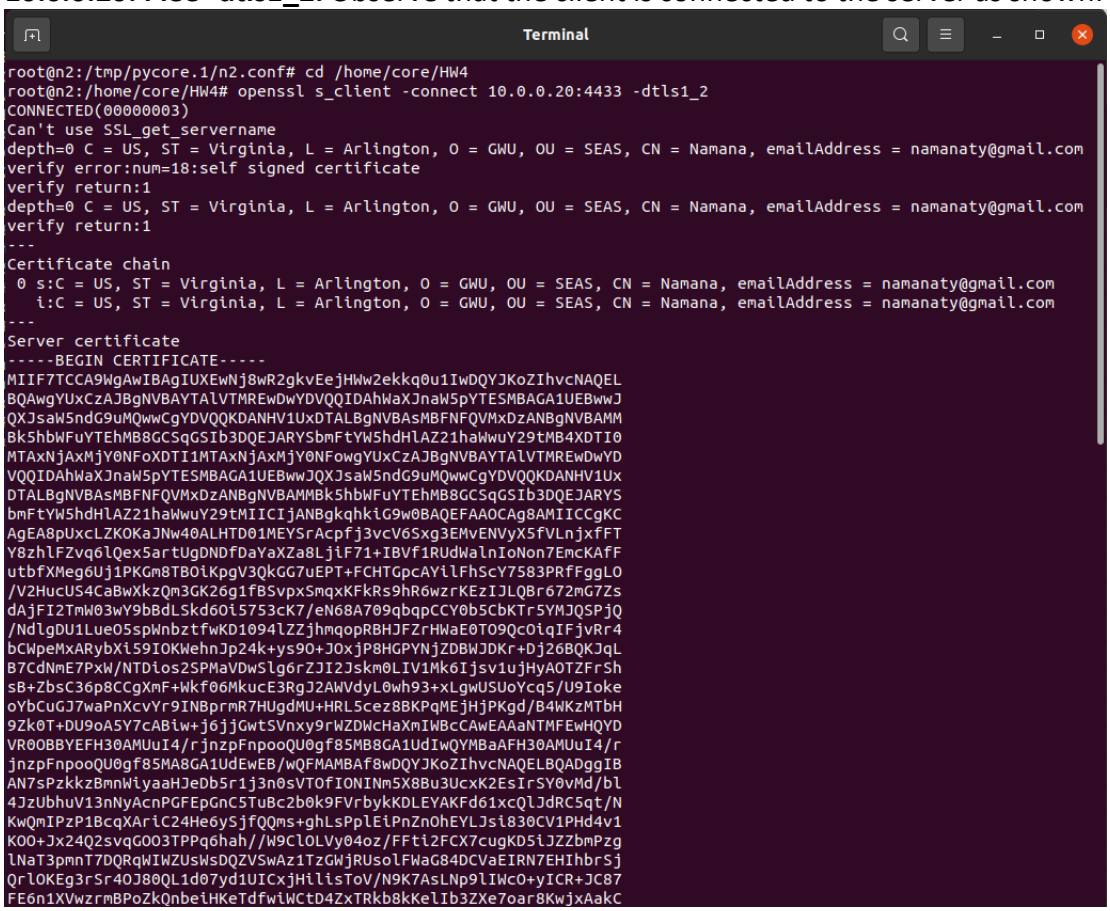
In Wireshark, the alert message shows the Level as "**Fatal**" with the Description "**Protocol Version**," indicating that the server does not support the protocol version the client is using. Additionally, in the client's terminal, we see **alert number 70**, which indicates an incompatible protocol.

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.

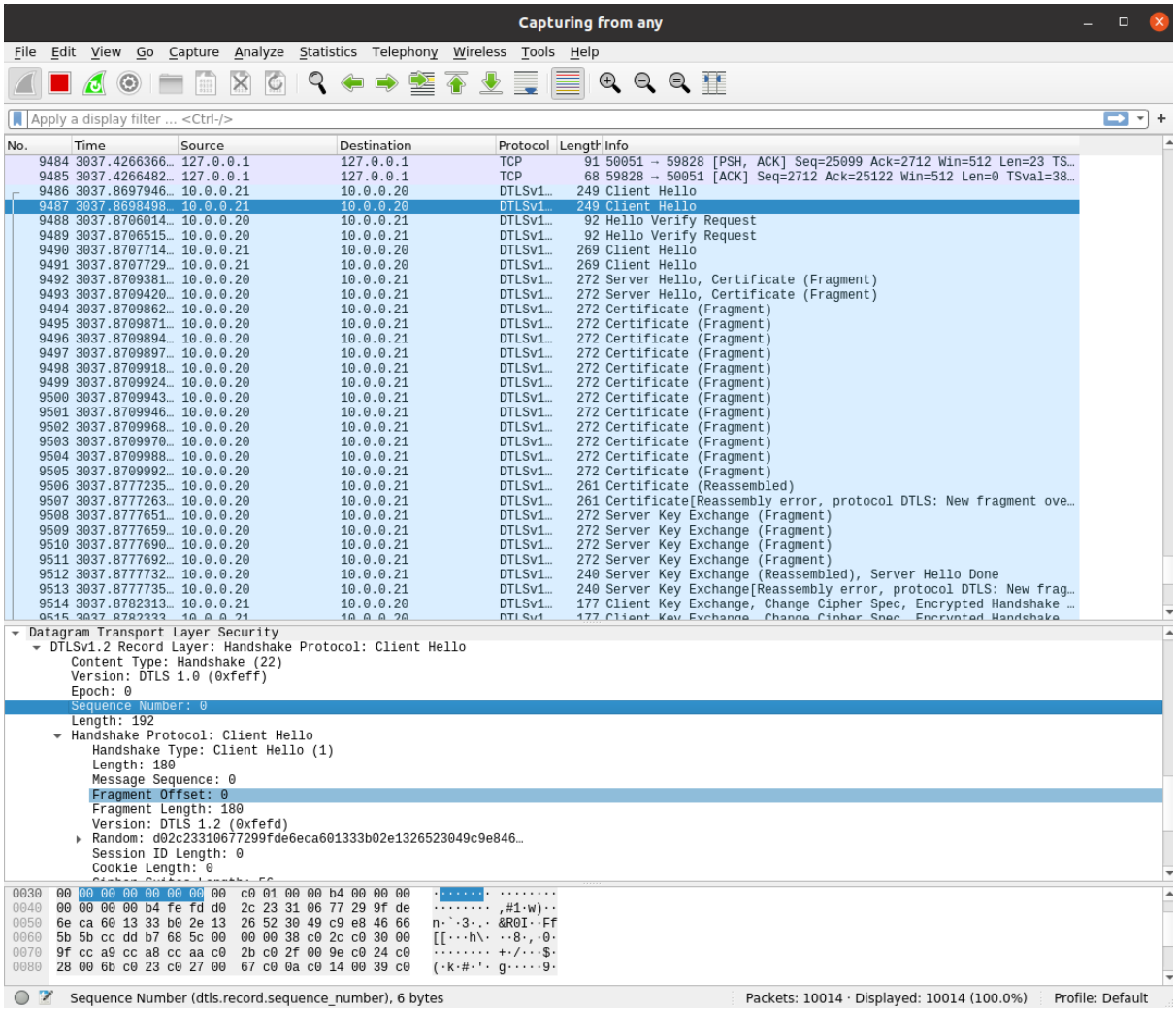
Run DTLS1.2 Server on node 1 using the command **openssl s_server -accept 4433 -cert www.cert -key www.key.no_pass -dtls1_2** as shown below:



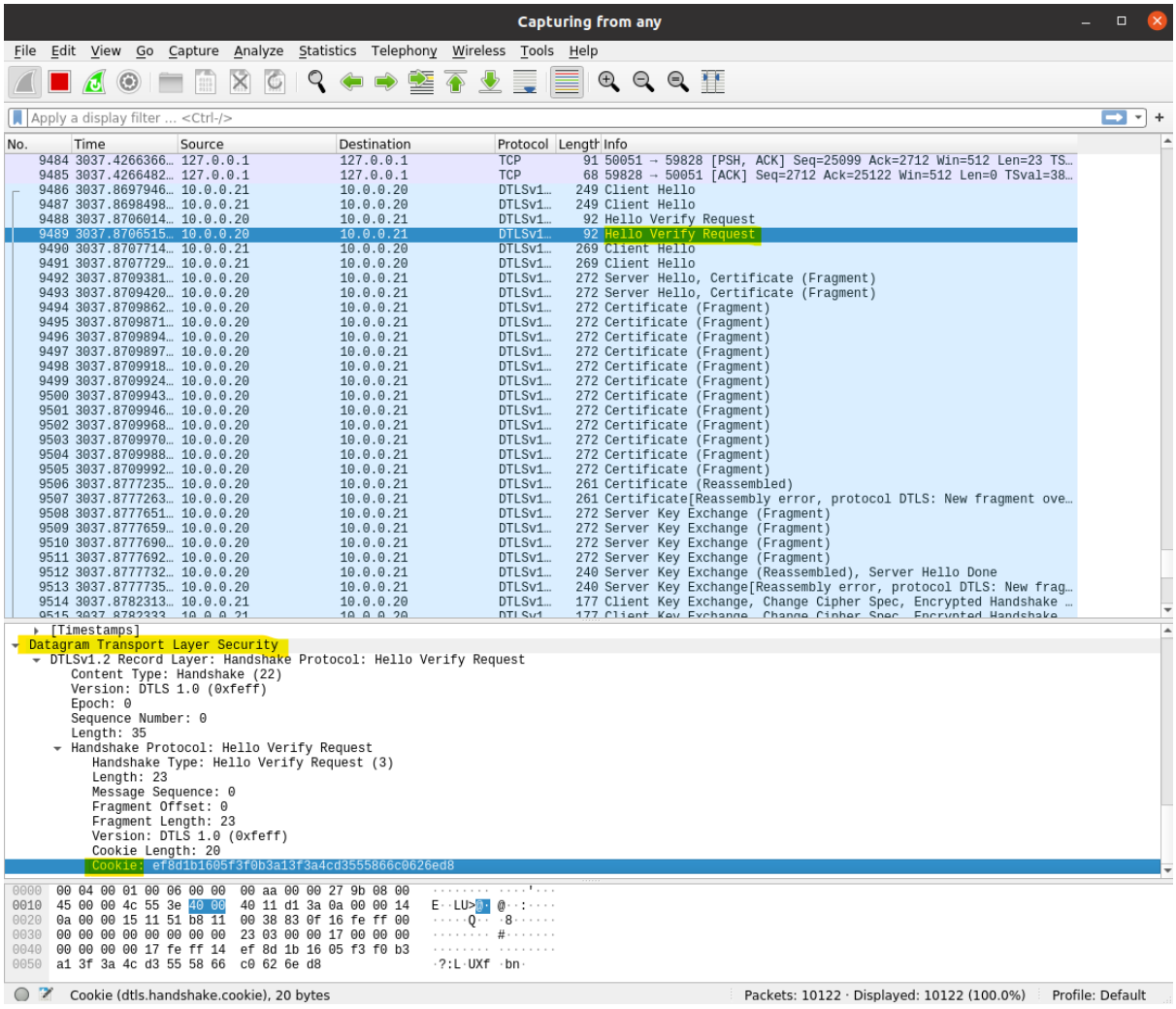
Run the DTLS1.2 Client on node 2 using the command **openssl s_client -connect 10.0.0.20:4433 -dtls1_2.** Observe that the client is connected to the server as shown:



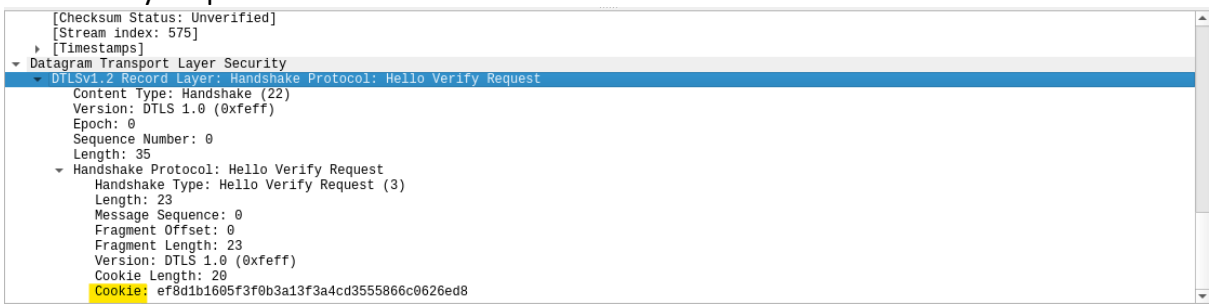
In Wireshark, you can see a full DTLS Handshake process. First, the server sends a Hello Verify Request. The client then initiates the connection by sending a Client Hello. In response, the server sends a Server Hello, along with Certificate Fragments, the Server Key Exchange, and a Server Hello Done 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:



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

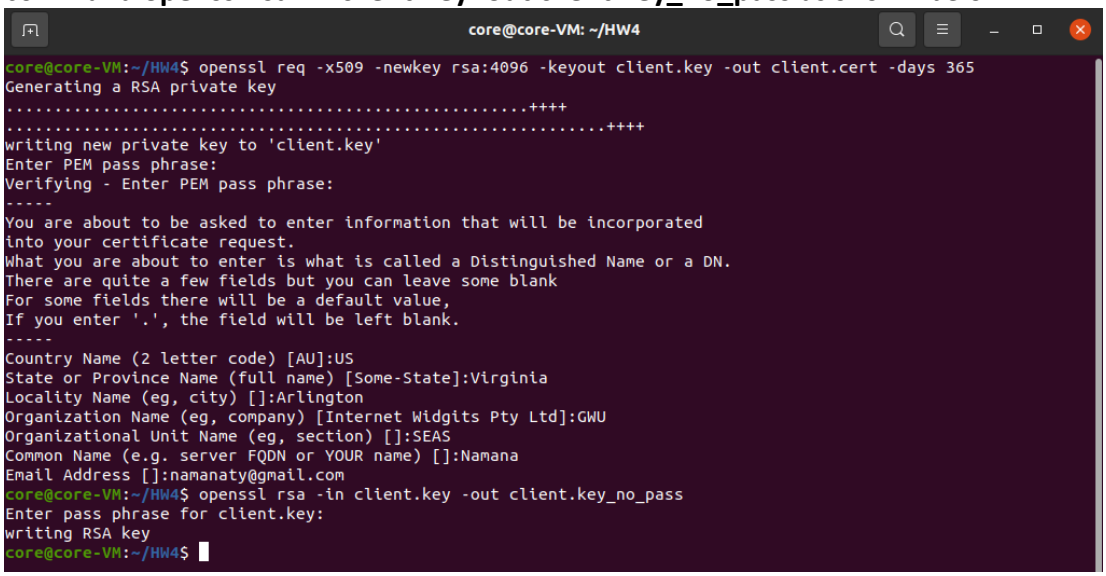


Hello Verify Request with Cookie Value:

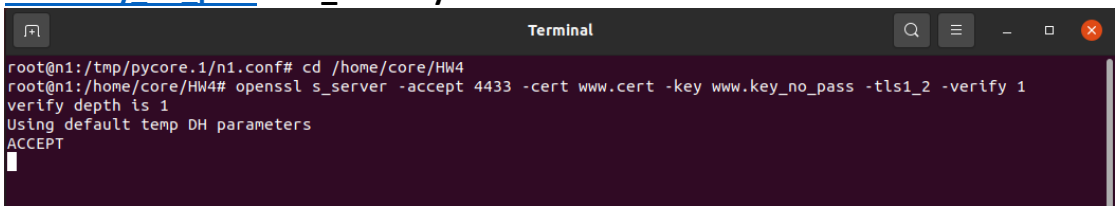


1.5. (2pts) Run a TLS1.2 server and a TLS1.2 client. Configure the server to request a certificate from the client. Configure the client with a cert and a private key.

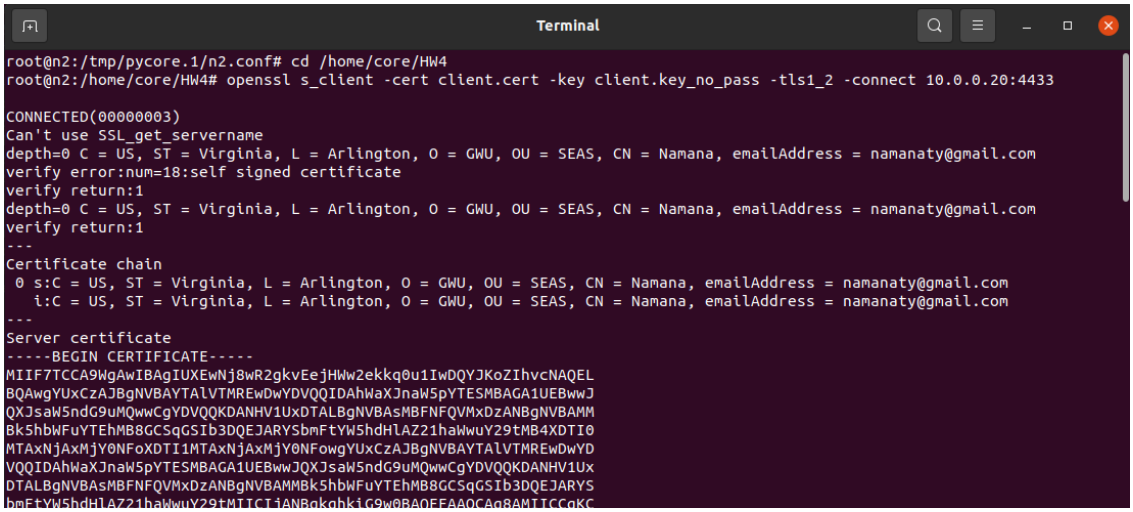
First let us generate the certificate using the command **openssl req -x509 -newkey rsa:4096 client.key -out client.cert -days365** and private key for the client using the command **openssl rsa -in client.key -out client.key_no_pass** as shown below:



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:

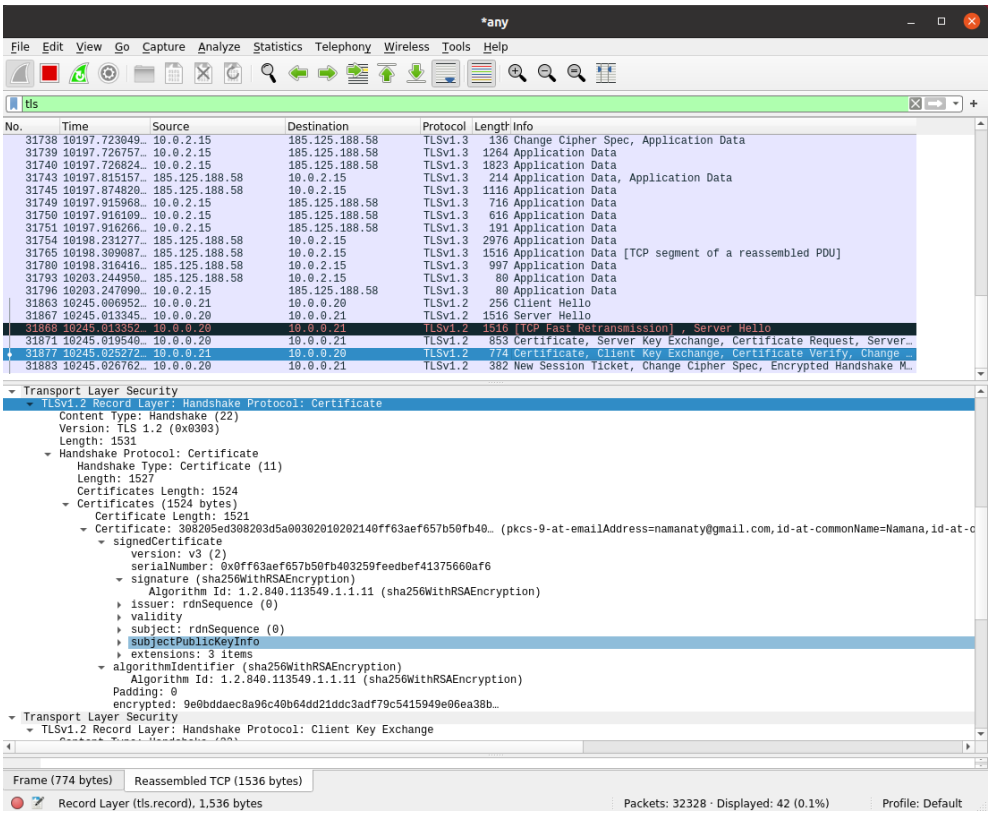


Next, run the TLS1.2 Client with its certificate and private key using the command **openssl s_client -cert client.cert -key client.key_no_pass -tls1_2 -connect 10.0.0.20:4433**. You can see that the client is connected to the server as shown:

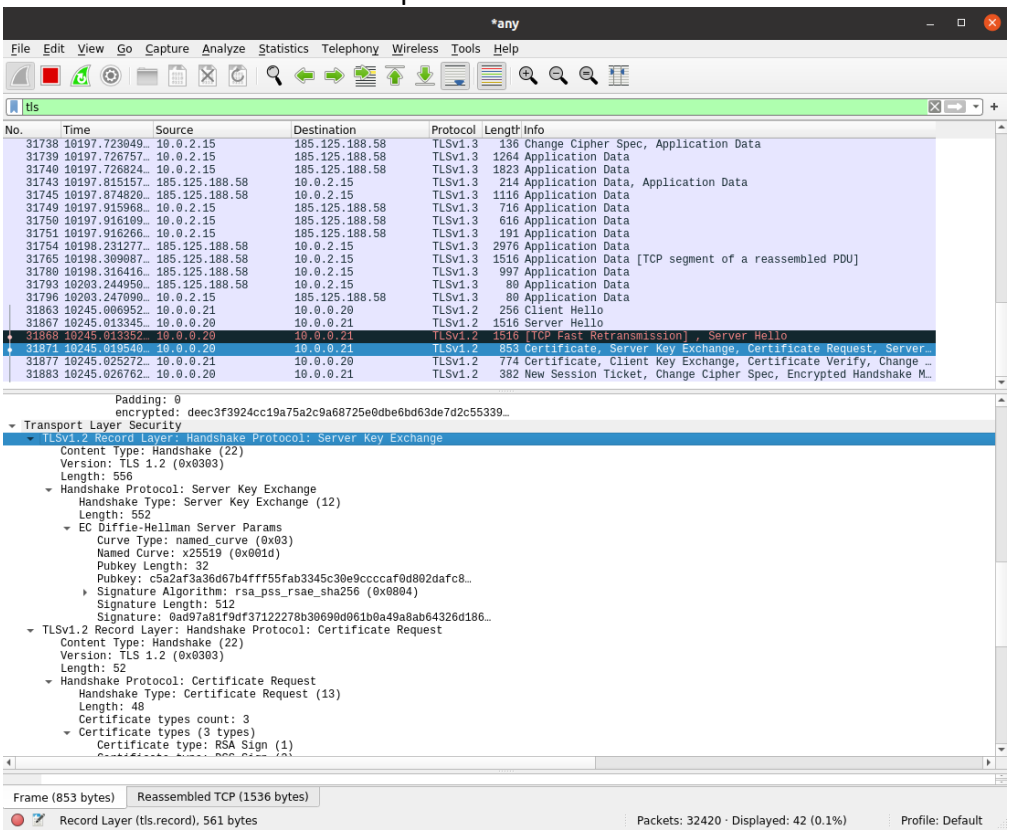


1.5.1. Show a screenshot of the certificate request from server, certificate from client, and certificate verify from the client.

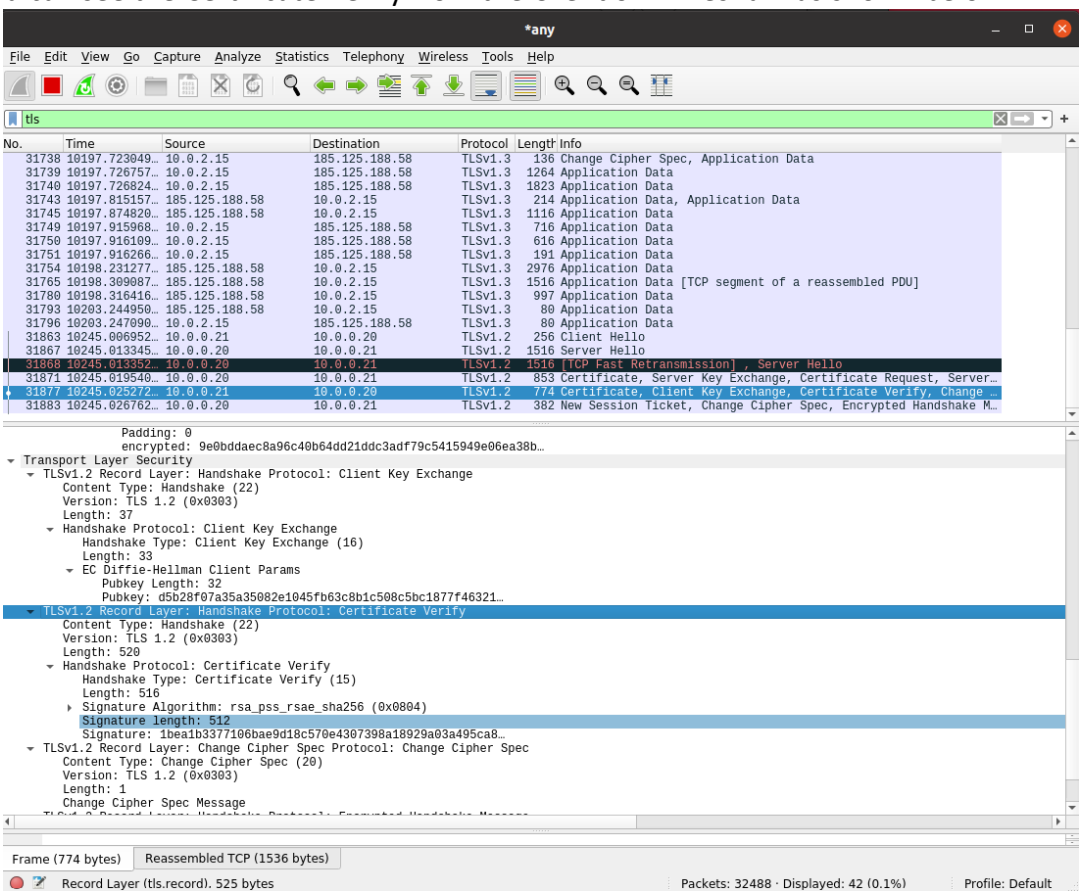
You can see the Certificate from the Client on Wireshark as shown below:



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:

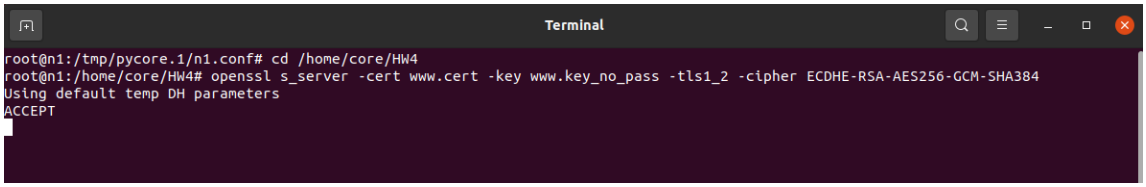


1.6. Run a TLS1.2 server configured to use the cipher: ECDHE-RSA-AES256-GCM-SHA384 and a TLS1.2 client configured to use the cipher: DHE-RSA-AES256-GCM-SHA384.

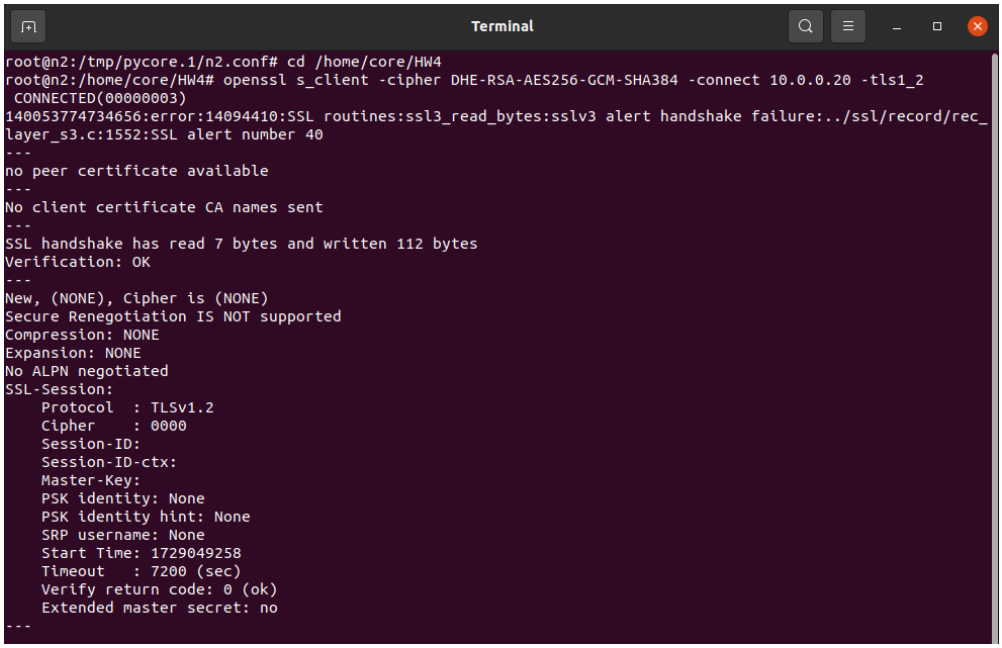
1.6.1. Hint: look at the output of openssl ciphers

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:

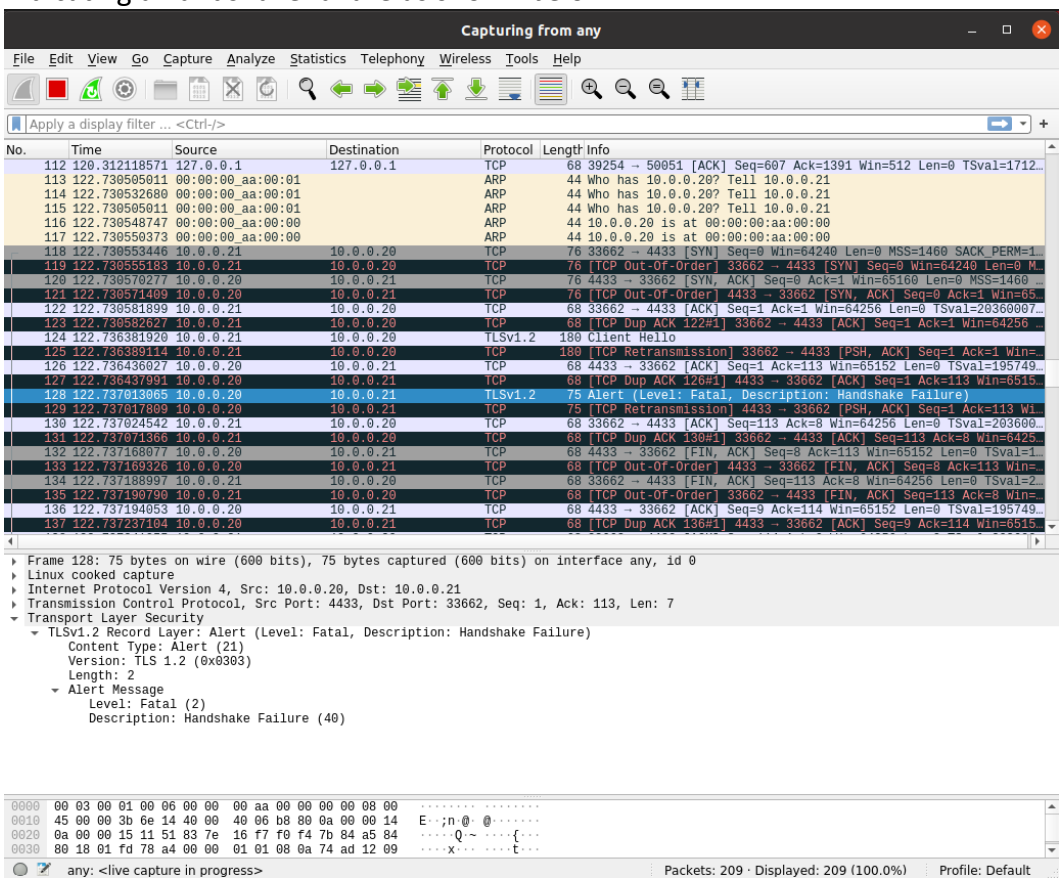


Next, run the TLS1.2 Client on node 2 using the command **openssl s_client -cipher DHE-RSA-AES256-GCM-SHA384 -connect 10.0.0.20 -tls1_2** to use the required cipher. You can see that the client is unable to connect to the server as shown below:



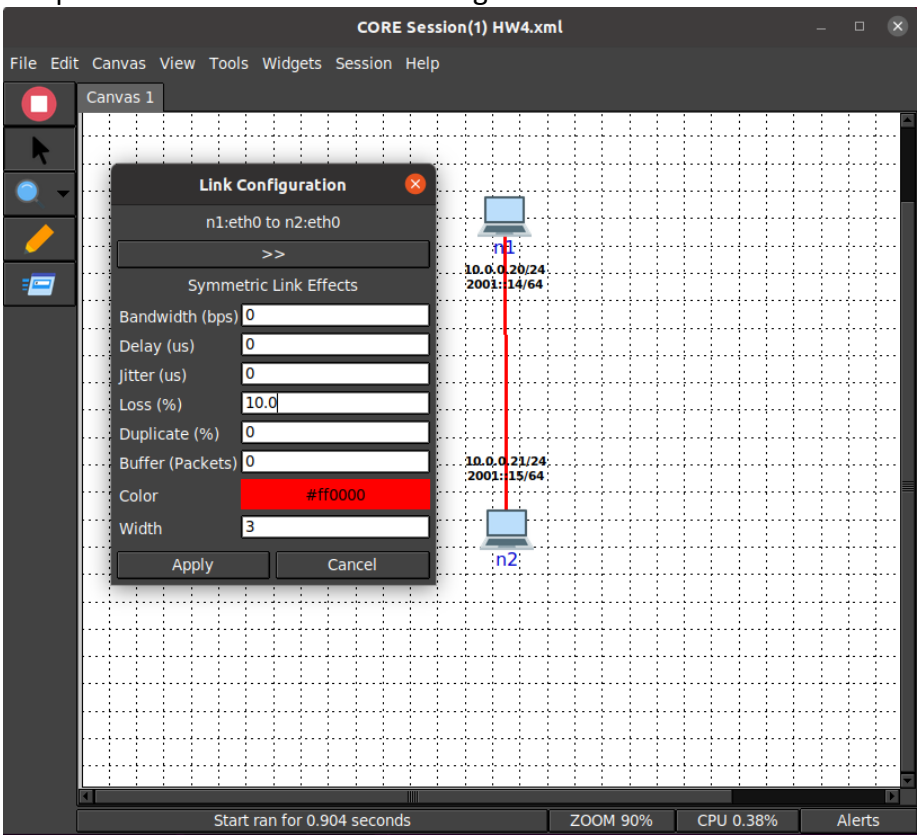
1.6.3. (1pt) Describe what happened. What Alert message type and what Level are generated?

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:

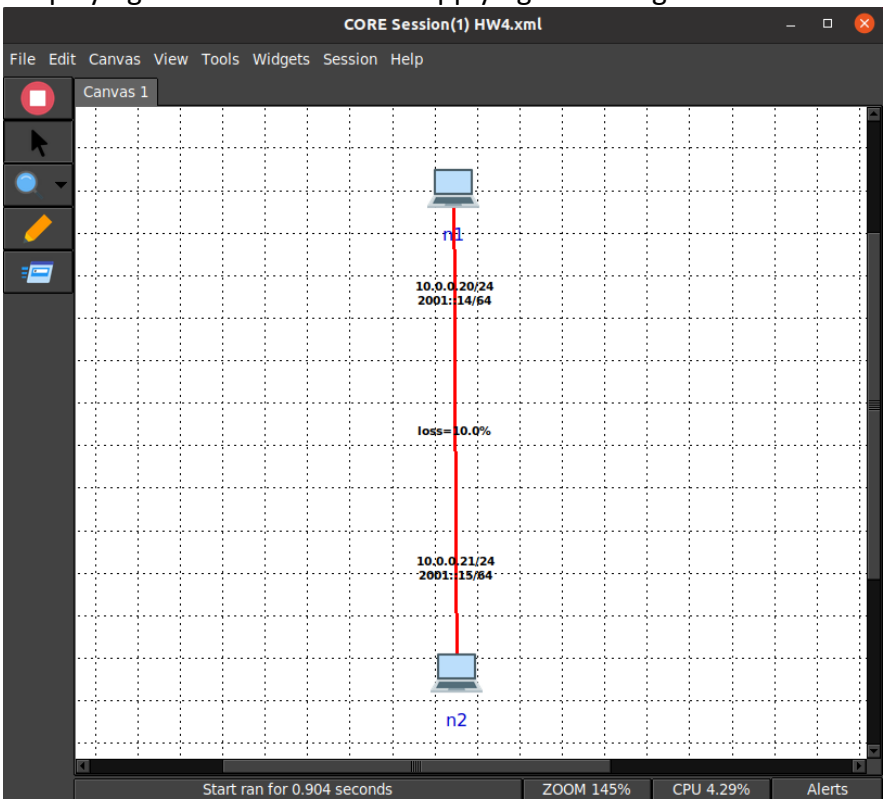


1.7. (1.5pts) Repeat 1.2 (do not show screenshot of handshake) but add a 10% packet loss to the link between the two nodes beforehand.

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



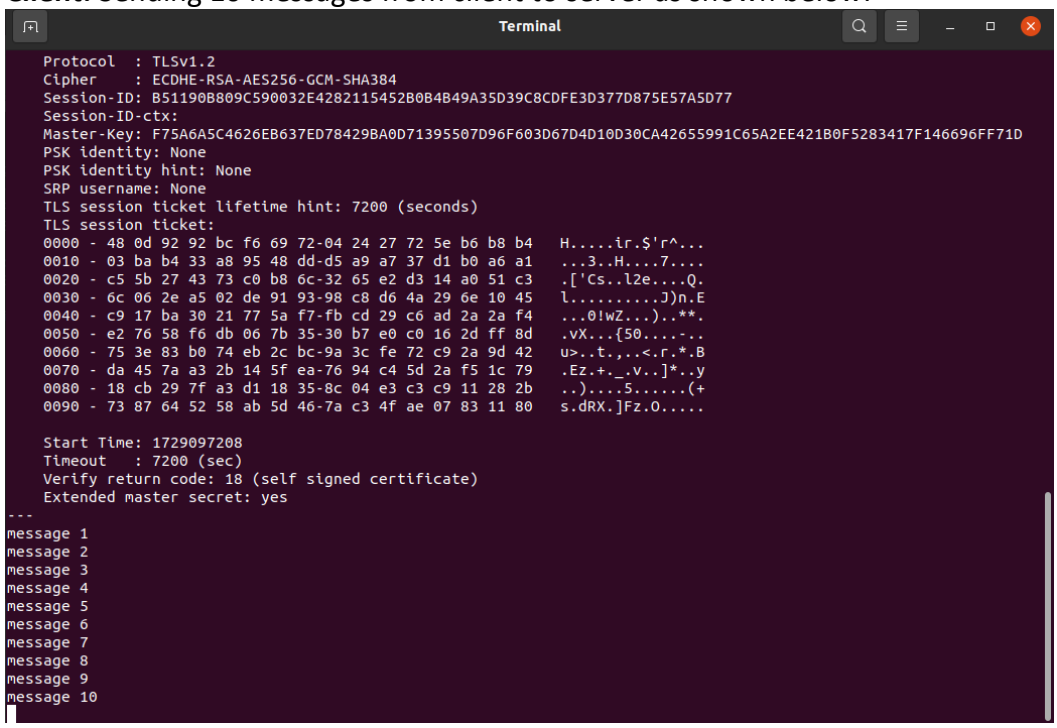
Core-GUI displaying the loss=10.0% after applying the changes as shown below:



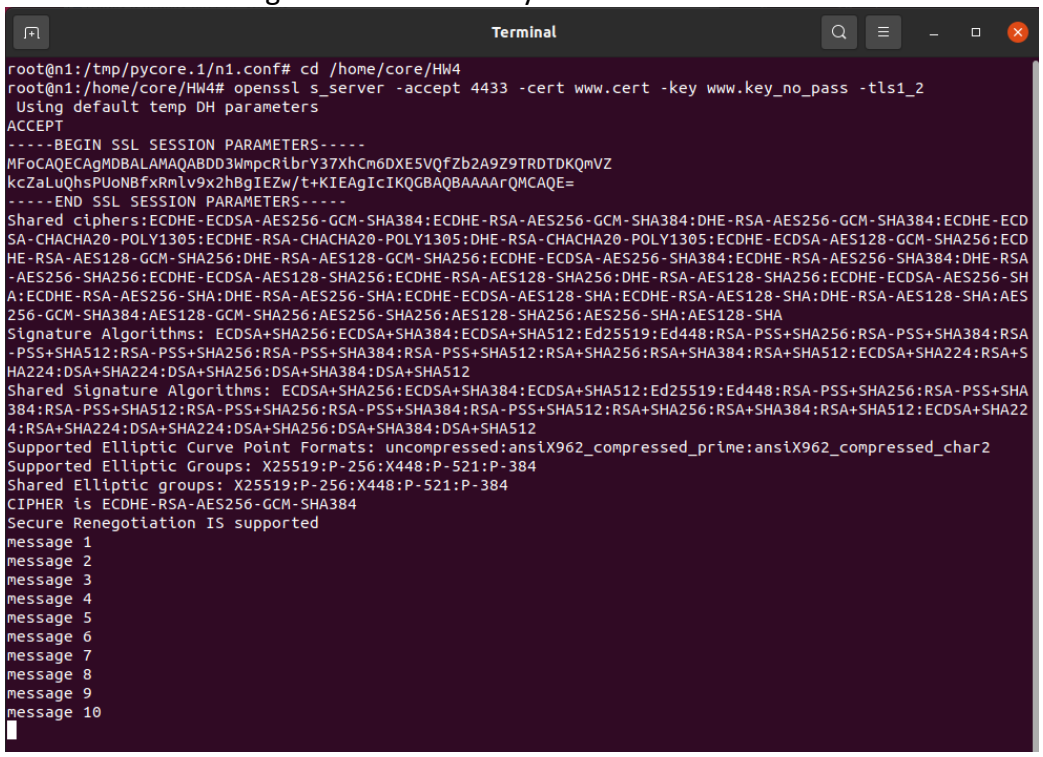
Repeat the steps from section 1.2 to configure the TLS 1.2 server and client.

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.

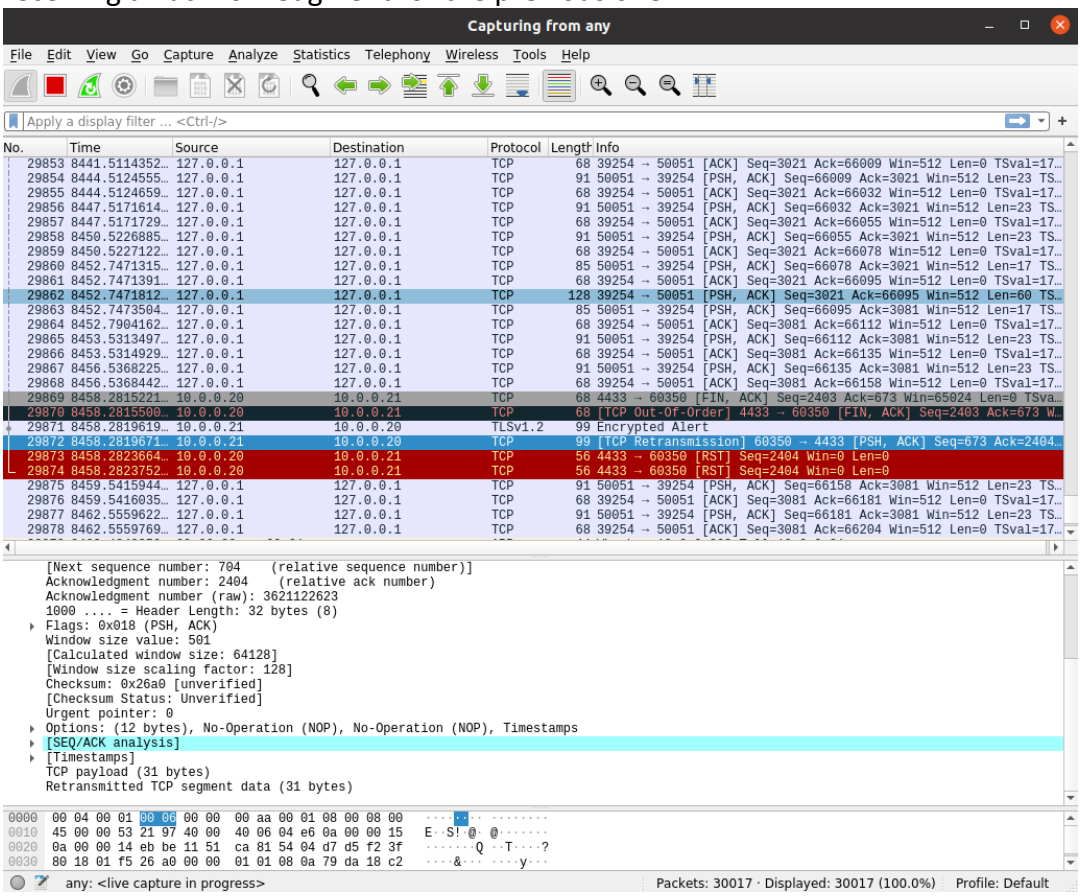
Client: Sending 10 messages from client to server as shown below:



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



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.

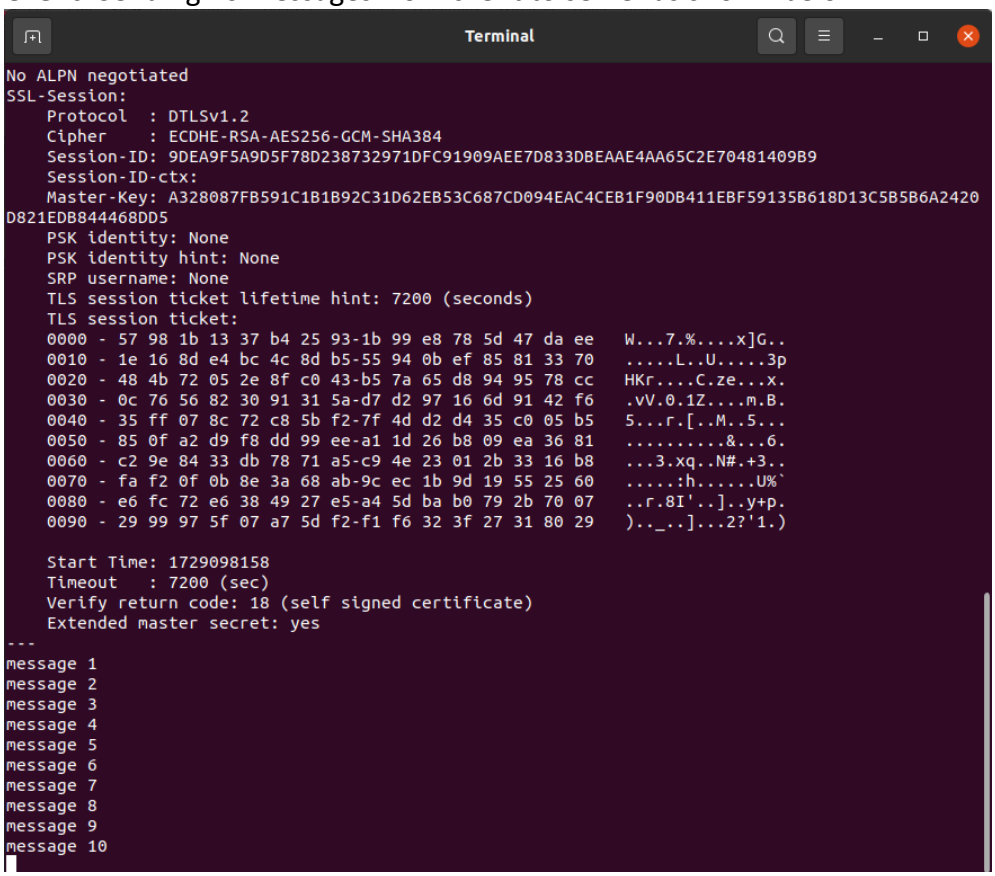


1.8. (1.5pts) Repeat 1.4 (do not show screenshot of handshake) but add a 10% packet loss to the link between the two nodes beforehand.

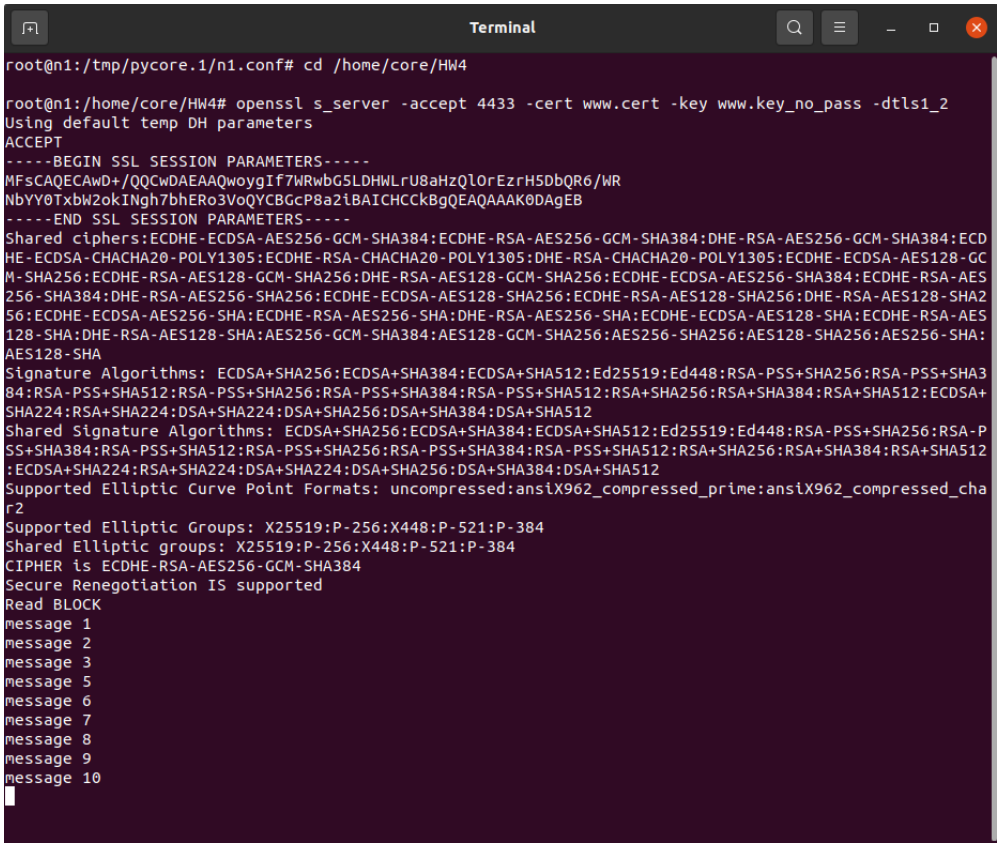
1.8.1. Send 10 messages from client to server: “message 1“, “message 2”, …etc. Show a screenshot of the server terminal after 2 minutes of sending the last message showing the list of messages received.

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:



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



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: