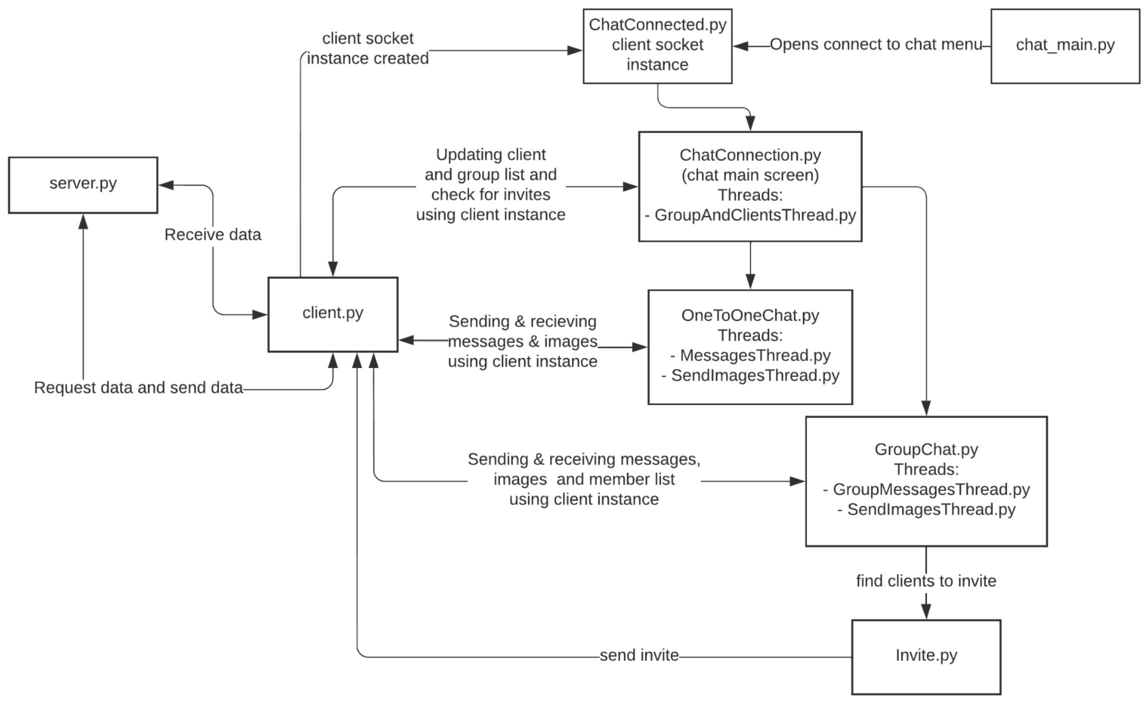
1. Show a system diagram of your chatting program, and explain each sub-system or module.



ChatConnected.py client socket instance
chat_main.py
Opens connect to chat menu
client socket instance created
server.py
ChatConnection.py (chat main screen) Threads: - GroupAndClientsThread.py
Updating client and group list and check for invites using client instance
Receive data
client.py
OneToOneChat.py Threads: - MessagesThread.py - SendImagesThread.py
Sending & recieving messages & images using client instance
Request data and send data
GroupChat.py Threads: - GroupMessagesThread.py - SendImagesThread.py
Sending & receiving messages, images and member list using client instance
find clients to invite
Invite.py
send invite

Server.py and client.py handle the backend manipulation of sending and receiving data to the respective places. In order to send a message to a particular client, the message needs to be sent from the client to the server and then the message is sent from the server to the receiving client.

All subsystems, except for the server and chat_main.py, use the same instance of the client socket to send data to the server and receive data from the server. This instance is first created in the ChatConnected.py when the user inputs the IP address, port number and nickname.

In order to know when a new client has joined the server, when a new group chat has been created or when there is a new member to the group chat, the GroupAndClientsThread requests data from the server using the client. The server then sends back the information required and so the list can be updated. This allows the GUI to have real-time information on the clients who are on the server, the group names and group members. The invite messages sent by the server are also checked in this thread. The recipient client of the invite will receive a notification asking if they want to join the group chat.

To allow for messaging between two clients in OneToOneChat, the MessagesThread checks if the data received from the server is a message or if the file name of the image that was sent has been received. While in the OneToOneChat, we use the client to send messages that were inputted by the user. The server will send the message to both themselves and the recipient in the OneToOneChat so both clients can see the message. In order for the clients to see images being sent to them, the SendImagesThread is used to send images from the client socket to the server. The server will then download the image to the /ChatProgram/

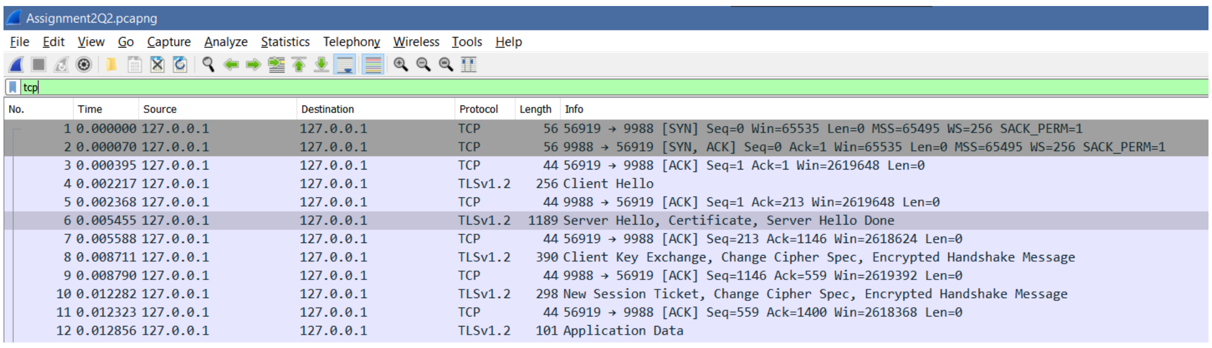
parent folder. Also the file name will be sent to the clients in the OneToOneChat which will be handled by MessagesThread.

Similarly messaging between two or more clients in GroupChat uses the GroupMessagesThread which has the same functionality. However, GroupMessagesThread also checks if the member list of the group chat that is sent by the server has been updated. This will allow users to see the member list update on the GUI when clients join their group chat for the first time. SendImagesThread is also used in GroupChat and has the same functionality as previously mentioned.

When trying to invite clients to a group chat, the member list of the group chat is checked against the list of all clients on the server. If there is any client who is not a member of the group chat, they are able to be invited. An invite is sent using the client socket to the server where the server will send the invite message to the respective recipient.

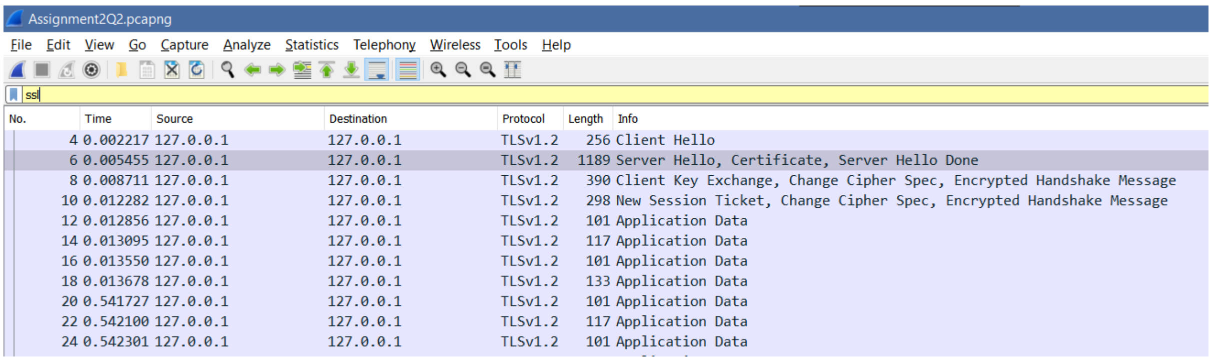
2. Describe the encryption algorithm of your system, and show evidence using wireshark if the encryption works correctly

I'm using SSL/TLSv1.2 with the "AES 128 CBC SHA" cipher key in my chat program. The wireshark file is provided in the /ChatProgram/Quiz Answer Sheet folder.
SSL allows for server and client authentication, negotiation on crypto algorithms and establishing keys.


Assignment2Q2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp
No.  Time     Source    Destination Protocol Length Info
 1 0.000000 127.0.0.1 127.0.0.1 TCP       56 56919 → 9988 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
 2 0.000070 127.0.0.1 127.0.0.1 TCP       56 9988 → 56919 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
 3 0.000395 127.0.0.1 127.0.0.1 TCP       44 56919 → 9988 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 4 0.002217 127.0.0.1 127.0.0.1 TLSv1.2  256 Client Hello
 5 0.002368 127.0.0.1 127.0.0.1 TCP       44 9988 → 56919 [ACK] Seq=1 Ack=213 Win=2619648 Len=0
 6 0.005455 127.0.0.1 127.0.0.1 TLSv1.2 1189 Server Hello, Certificate, Server Hello Done
 7 0.005588 127.0.0.1 127.0.0.1 TCP       44 56919 → 9988 [ACK] Seq=213 Ack=1146 Win=2618624 Len=0
 8 0.008711 127.0.0.1 127.0.0.1 TLSv1.2  390 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
 9 0.008790 127.0.0.1 127.0.0.1 TCP       44 9988 → 56919 [ACK] Seq=1146 Ack=559 Win=2619392 Len=0
10 0.012282 127.0.0.1 127.0.0.1 TLSv1.2  298 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
11 0.012323 127.0.0.1 127.0.0.1 TCP       44 56919 → 9988 [ACK] Seq=559 Ack=1400 Win=2618368 Len=0
12 0.012856 127.0.0.1 127.0.0.1 TLSv1.2  101 Application Data
```

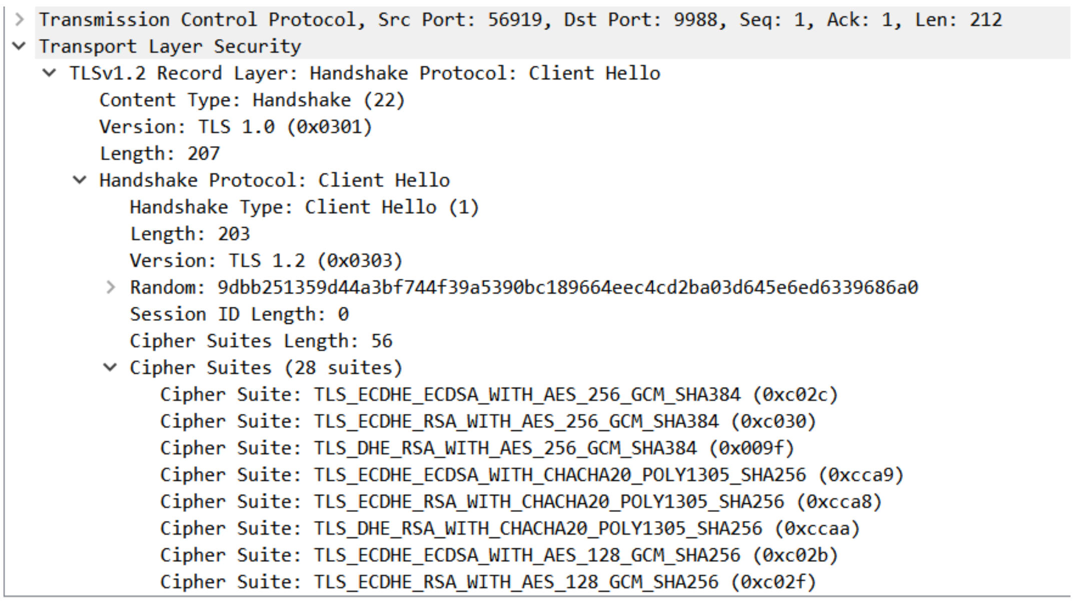
In the screenshot above, the connection starts off with a TCP handshake, SYN, SYN-ACK and ACK. Right after this, we have the TLS handshake which allows client and server to authenticate each other.


Assignment2Q2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ssl
No.  Time     Source    Destination Protocol Length Info
 4 0.002217 127.0.0.1 127.0.0.1 TLSv1.2  256 Client Hello
 6 0.005455 127.0.0.1 127.0.0.1 TLSv1.2 1189 Server Hello, Certificate, Server Hello Done
 8 0.008711 127.0.0.1 127.0.0.1 TLSv1.2  390 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10 0.012282 127.0.0.1 127.0.0.1 TLSv1.2  298 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
12 0.012856 127.0.0.1 127.0.0.1 TLSv1.2  101 Application Data
14 0.013095 127.0.0.1 127.0.0.1 TLSv1.2  117 Application Data
16 0.013550 127.0.0.1 127.0.0.1 TLSv1.2  101 Application Data
18 0.013678 127.0.0.1 127.0.0.1 TLSv1.2  133 Application Data
20 0.541727 127.0.0.1 127.0.0.1 TLSv1.2  101 Application Data
22 0.542100 127.0.0.1 127.0.0.1 TLSv1.2  117 Application Data
24 0.542301 127.0.0.1 127.0.0.1 TLSv1.2  101 Application Data
```

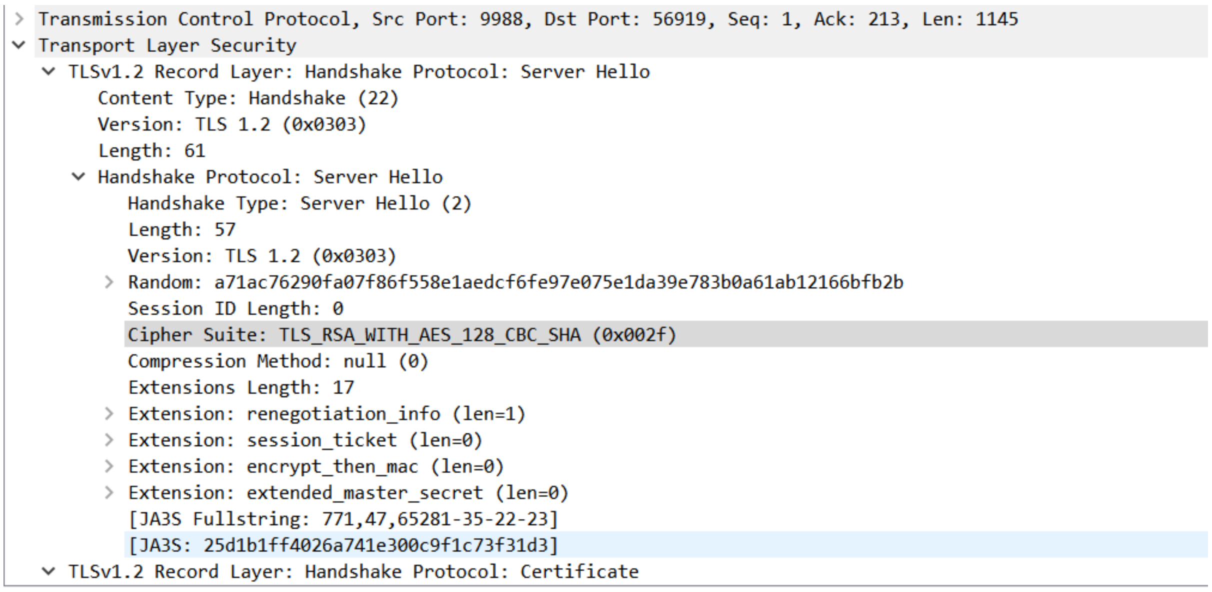
The above screenshot shows TLS handshake occurs as follows: Client Hello then Server

Hello, Certificate, Server Done then Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message and finally New Session Ticket, Change Cipher Spec, Encrypted Handshake message.



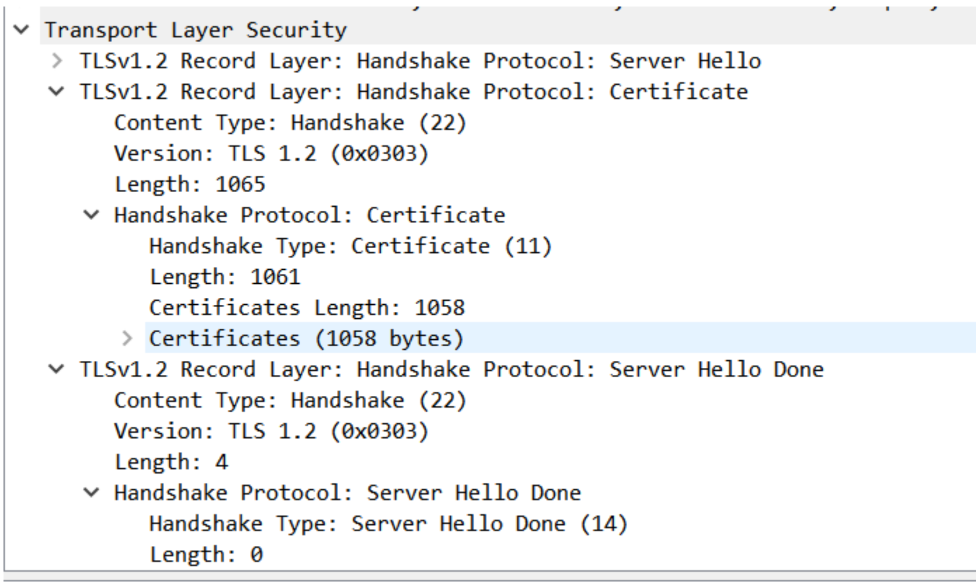
```
> Transmission Control Protocol, Src Port: 56919, Dst Port: 9988, Seq: 1, Ack: 1, Len: 212
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 207
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 203
        Version: TLS 1.2 (0x0303)
      > Random: 9dbb251359d44a3bf744f39a5390bc189664eec4cd2ba03d645e6ed6339686a0
        Session ID Length: 0
        Cipher Suites Length: 56
      v Cipher Suites (28 suites)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
          Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
```

In the TLS handshake, the client and then server will each send a “hello” message to each other. The client hello message will include which TLS version and cipher suites the client can use. In the screenshot above, there are 28 cipher suites that the client can use and only one TLS version, TLS version 1.2.



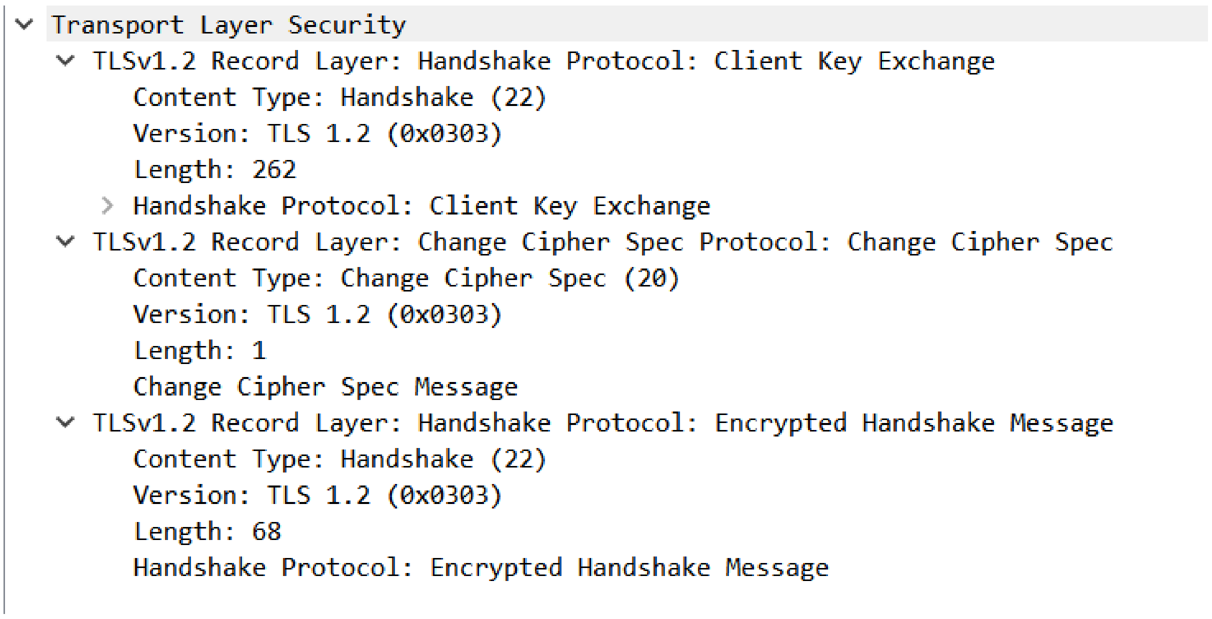
```
> Transmission Control Protocol, Src Port: 9988, Dst Port: 56919, Seq: 1, Ack: 213, Len: 1145
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 61
    v Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 57
        Version: TLS 1.2 (0x0303)
      > Random: a71ac76290fa07f86f558e1aedcf6fe97e075e1da39e783b0a61ab12166bfb2b
        Session ID Length: 0
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Compression Method: null (0)
        Extensions Length: 17
      > Extension: renegotiation_info (len=1)
      > Extension: session_ticket (len=0)
      > Extension: encrypt_then_mac (len=0)
      > Extension: extended_master_secret (len=0)
        [JA3S Fullstring: 771,47,65281-35-22-23]
        [JA3S: 25d1b1ff4026a741e300c9f1c73f31d3]
  v TLSv1.2 Record Layer: Handshake Protocol: Certificate
```

The server hello message will contain the server’s SSL certificate along with the chosen cipher suite. This will be indicated by a Server Hello, Certificate, Server Done protocol. In my case, the server chose a cipher suite TLS_RSA_WITH_AES_128_CBC_SHA and the TLS version 1.2 as seen in the server hello screenshot above. In the screenshot below, the handshake server’s SSL certificate is being sent.



```
v Transport Layer Security
   > TLSv1.2 Record Layer: Handshake Protocol: Server Hello
   v TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 1065
      v Handshake Protocol: Certificate
           Handshake Type: Certificate (11)
           Length: 1061
           Certificates Length: 1058
         > Certificates (1058 bytes)
   v TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 4
      v Handshake Protocol: Server Hello Done
           Handshake Type: Server Hello Done (14)
           Length: 0
```

The client verifies the server’s SSL certificate with a certificate authority. This allows the client to ensure the identity of the server.



```
v Transport Layer Security
   v TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 262
      > Handshake Protocol: Client Key Exchange
   v TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
        Content Type: Change Cipher Spec (20)
        Version: TLS 1.2 (0x0303)
        Length: 1
        Change Cipher Spec Message
   v TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 68
        Handshake Protocol: Encrypted Handshake Message
```

A client key exchange message is sent from the client which creates the pre-master secret. This pre-master secret is then shared with the server. However, before sending it to the server, the client encrypts it using the server public key extracted from the server’s certificate. The change cipher spec protocol tells the server that any further messages sent will be encrypted with the negotiated keys. The encrypted handshake message is the client sending a “Finished” message to the server using the encryption.
Hence, the TLS handshake is complete and any further communication will use those secret keys.