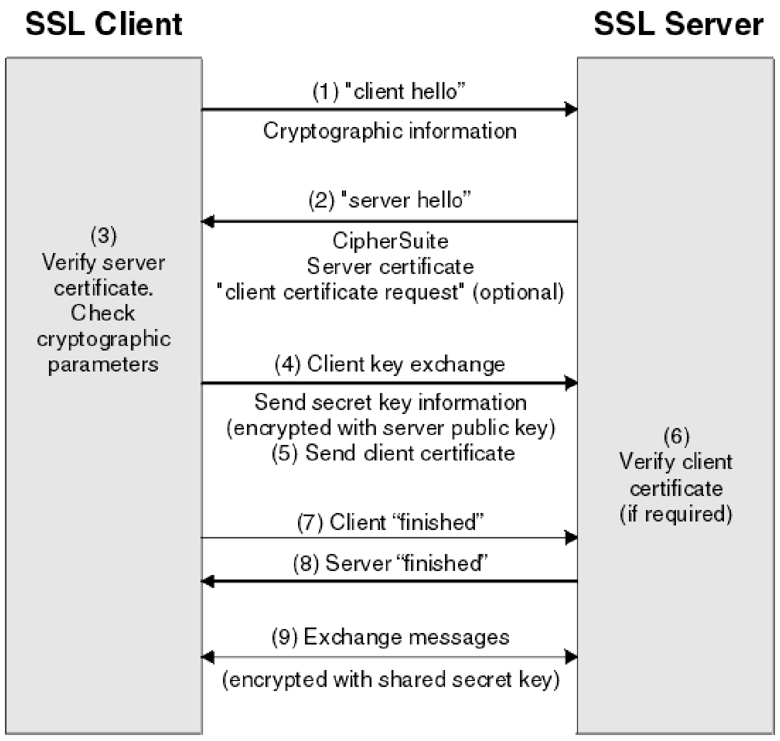
CSE 554 Networks and Systems Security II
Assignment 4

Name: Harsh Kumar **Roll No:** 2019043

Assumptions:

- Each message sent by the client is within the buffer size, i.e. 1024 bytes.

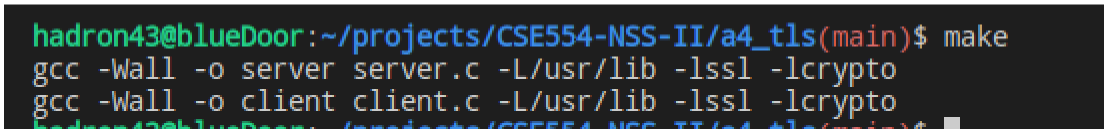
System Design:



SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

After basic TCP three way handshake, client sends a "client hello" message with the cryptographic information. The server replies with the server certificate, and chooses a cipher for further communication. The client verifies the server certificate. Client comes up with a pre master secret from the information gathered till now. After this key exchanges happen, and the master secret key is derived. Now, the client and server use the master secret to derive a session key, which is used for symmetric encryption. SSL handshake is now complete, and SSL session has started. Now, the client and server can exchange encrypted messages, using this session key.

Compiling the program:



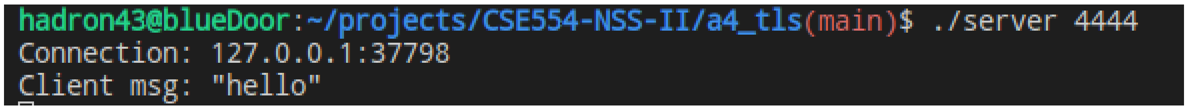
```
hadron43@blueDoor:~/projects/CSE554-NSS-II/a4_tls(main)$ make
gcc -Wall -o server server.c -L/usr/lib -lssl -lcrypto
gcc -Wall -o client client.c -L/usr/lib -lssl -lcrypto
```

The server program uses the server certificate saved in *server-cert.pem* file, and the private key is stored in *server-key.pem*. This is signed by *ca-certificate.pem*.

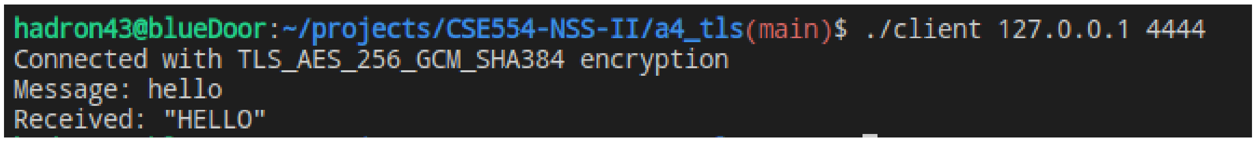
The client program uses the root CA certificate stored in *ca-cert.pem*, and the private key is stored in *ca-key.pem*. This is used to verify the server certificate in the TLS connection.

Note: Due to the updated version of openssl library, it is using TLSv1.3 instead of TLSv1.2 as mentioned in the problem statement.

Running the program:

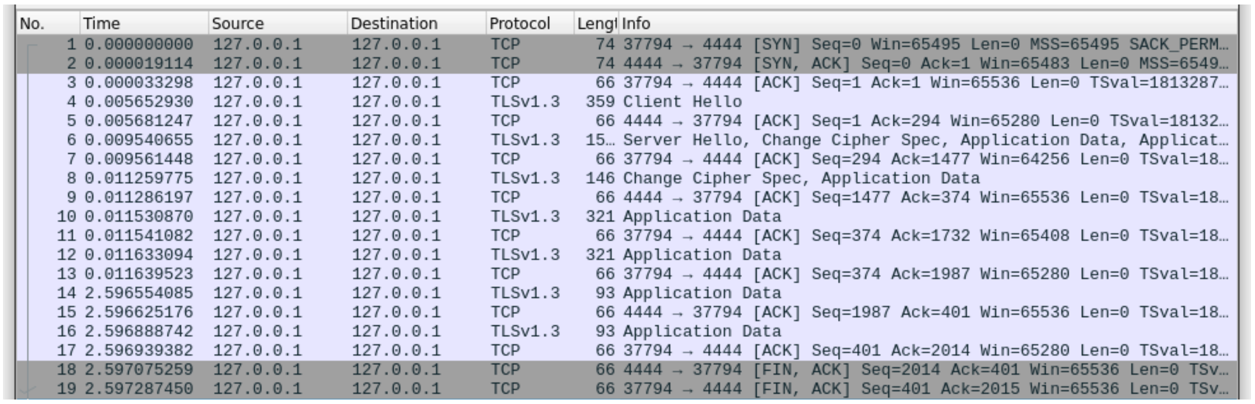


```
hadron43@blueDoor:~/projects/CSE554-NSS-II/a4_tls(main)$ ./server 4444
Connection: 127.0.0.1:37798
Client msg: "hello"
```



```
hadron43@blueDoor:~/projects/CSE554-NSS-II/a4_tls(main)$ ./client 127.0.0.1 4444
Connected with TLS_AES_256_GCM_SHA384 encryption
Message: hello
Received: "HELLO"
```

Wireshark output:



```
No.  Time         Source     Destination  Protocol  Lengt Info
   1 0.000000000  127.0.0.1  127.0.0.1    TCP         74 37794 → 4444 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM…
   2 0.000019114  127.0.0.1  127.0.0.1    TCP         74 4444 → 37794 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549…
   3 0.000033298  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1813287…
   4 0.005652930  127.0.0.1  127.0.0.1    TLSv1.3    359 Client Hello
   5 0.005681247  127.0.0.1  127.0.0.1    TCP         66 4444 → 37794 [ACK] Seq=1 Ack=294 Win=65280 Len=0 TSval=18132…
   6 0.009540655  127.0.0.1  127.0.0.1    TLSv1.3    15… Server Hello, Change Cipher Spec, Application Data, Applicat…
   7 0.009561448  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [ACK] Seq=294 Ack=1477 Win=64256 Len=0 TSval=18…
   8 0.011259775  127.0.0.1  127.0.0.1    TLSv1.3    146 Change Cipher Spec, Application Data
   9 0.011286197  127.0.0.1  127.0.0.1    TCP         66 4444 → 37794 [ACK] Seq=1477 Ack=374 Win=65536 Len=0 TSval=18…
  10 0.011530870  127.0.0.1  127.0.0.1    TLSv1.3    321 Application Data
  11 0.011541082  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [ACK] Seq=374 Ack=1732 Win=65408 Len=0 TSval=18…
  12 0.011633094  127.0.0.1  127.0.0.1    TLSv1.3    321 Application Data
  13 0.011639523  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [ACK] Seq=374 Ack=1987 Win=65280 Len=0 TSval=18…
  14 2.596554085  127.0.0.1  127.0.0.1    TLSv1.3     93 Application Data
  15 2.596625176  127.0.0.1  127.0.0.1    TCP         66 4444 → 37794 [ACK] Seq=1987 Ack=401 Win=65536 Len=0 TSval=18…
  16 2.596888742  127.0.0.1  127.0.0.1    TLSv1.3     93 Application Data
  17 2.596939382  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [ACK] Seq=401 Ack=2014 Win=65280 Len=0 TSval=18…
  18 2.597075259  127.0.0.1  127.0.0.1    TCP         66 4444 → 37794 [FIN, ACK] Seq=2014 Ack=401 Win=65536 Len=0 TSv…
  19 2.597287450  127.0.0.1  127.0.0.1    TCP         66 37794 → 4444 [FIN, ACK] Seq=401 Ack=2015 Win=65536 Len=0 TSv…
```

1: Client initiates a TCP connection with the server by sending a SYN packet.
2: Server replies with a SYN ACK.
3: Client replies with ACK and TCP connection is established.
4: Client sends a “Client Hello” message, asking for a TLS connection.
5: Server receives the “Client Hello” message, and sends ACK.
6: Server sends a “Server Hello” along with its cipher spec and other information.
7: Client receives the “Server Hello” message, and sends ACK to the server.
8-17: packets are encrypted, so we cannot see contents. Here, key exchanges happen and application data is transferred.
18: Connection teardown request by client.
19: Connection teardown complete.

We can see that packets 8-17 were encrypted and not readable.

References:

- Simplest Codings: Secure Server Client using OpenSSL in C
- Simple TLS Server - OpenSSLWiki