Assignment 3 : Secure chat using openssl and MITM attacks

TEAM:

G Vishal Siva Kumar	CS18BTECH11013
P V Asish	CS18BTECH11037
T Krishna Prashanth	CS18BTECH11045

PLAGIARISM STATEMENT

I certify that this assignment/report is our own work, based on our combined personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, tutorials, sample programs, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if any of us become aware of it.

Names: T. Krishna Prashanth, G. Vishal Siva Kumar, P.V.Asish
Date: 10-04-2021
Signature: T.K.P, G.V.S.K, P.V.A

TASK 1

Generating Root CA key and certificate

```
$ openssl x509 -in root.crt -text
```

Transferring files between ns08 and lxd containers is done using

```
$ lxc file push <source> <destination container>/<path>
$ lxc file pull <source container>/<path> <destination>
```

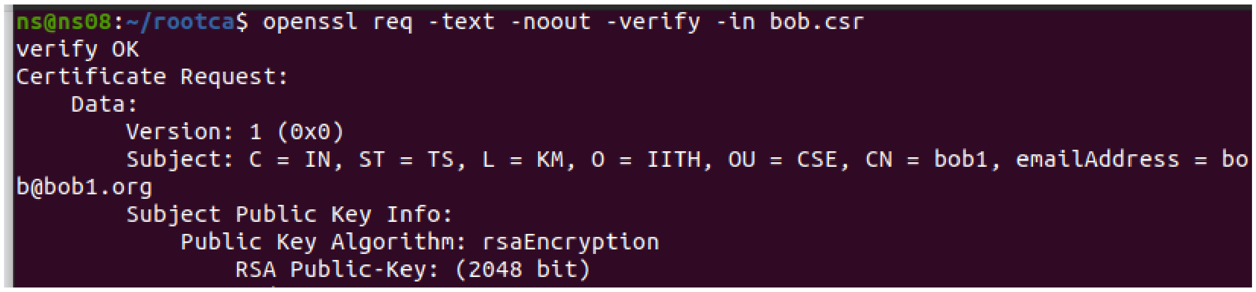
Generating key file and CSR for alice

```
     $ openssl req -newkey rsa:2048 -nodes -keyout alice-key.pem
-out alice.csr
```

Generating key file and CSR for bob

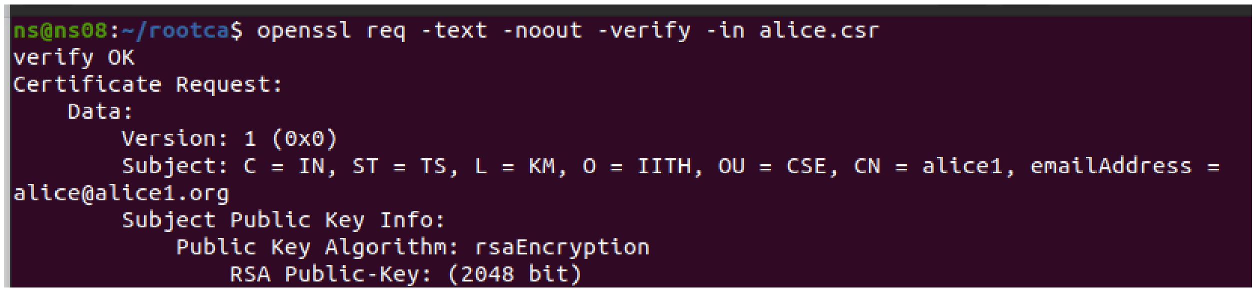
```
     $ openssl req -newkey rsa:2048 -nodes -keyout bob-key.pem
-out bob.csr
```

Verifying CSR of bob



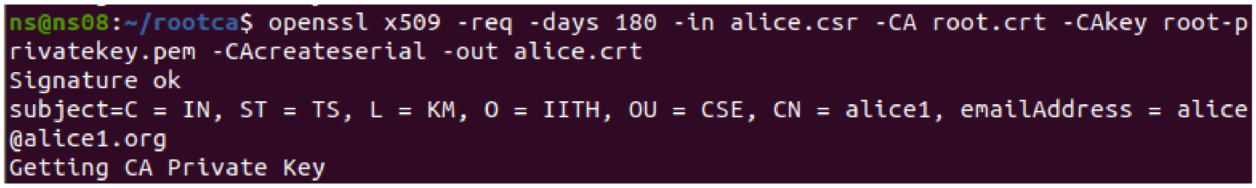
```
ns@ns08:~/rootca$ openssl req -text -noout -verify -in bob.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = bob1, emailAddress = bo
b@bob1.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
```

Verifying CSR of alice



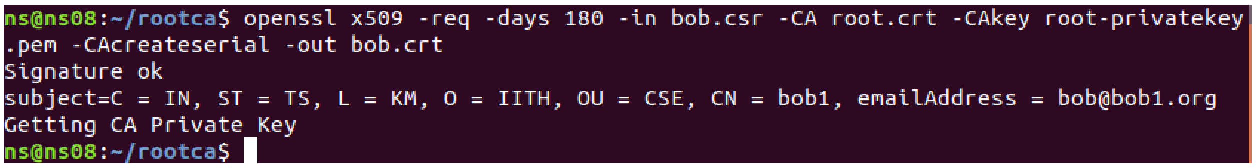
```
ns@ns08:~/rootca$ openssl req -text -noout -verify -in alice.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = alice1, emailAddress =
alice@alice1.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
```

Generating Certificate of alice



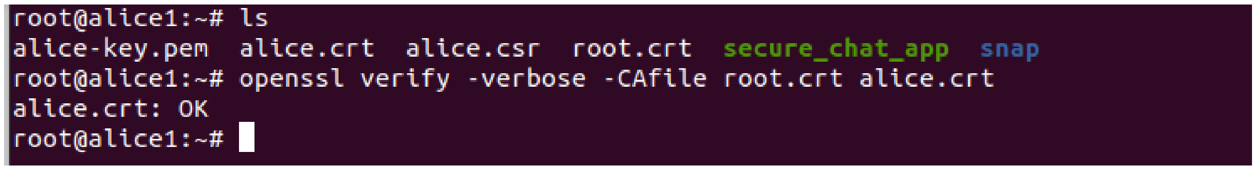
```
ns@ns08:~/rootca$ openssl x509 -req -days 180 -in alice.csr -CA root.crt -CAkey root-p
rivatekey.pem -CAcreateserial -out alice.crt
Signature ok
subject=C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = alice1, emailAddress = alice
@alice1.org
Getting CA Private Key
```

Generating Certificate of bob



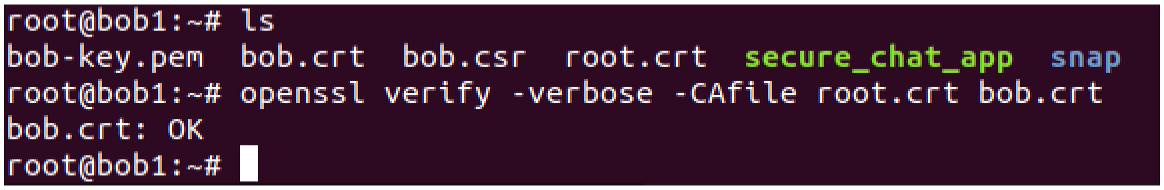
```
ns@ns08:~/rootca$ openssl x509 -req -days 180 -in bob.csr -CA root.crt -CAkey root-privatekey
.pem -CAcreateserial -out bob.crt
Signature ok
subject=C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = bob1, emailAddress = bob@bob1.org
Getting CA Private Key
ns@ns08:~/rootca$ 
```

Verifying Certificate of Alice:



```
root@alice1:~# ls
alice-key.pem  alice.crt  alice.csr  root.crt  secure_chat_app  snap
root@alice1:~# openssl verify -verbose -CAfile root.crt alice.crt
alice.crt: OK
root@alice1:~# 
```

Verifying Certificate of Bob:



```
root@bob1:~# ls
bob-key.pem  bob.crt  bob.csr  root.crt  secure_chat_app  snap
root@bob1:~# openssl verify -verbose -CAfile root.crt bob.crt
bob.crt: OK
root@bob1:~# 
```

We are assuming that rootCA has verified Bob and Alice in the offline phase if they are Bob or Alice in person which is generally followed in the real world.

TASK 2:

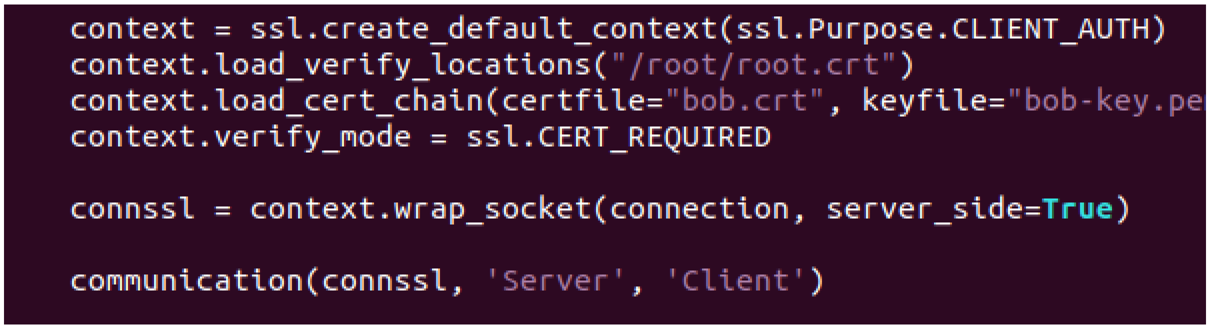
Secure Chat App:

Design:
We designed the chat app in accordance with the instructions given in the assignment document.
We did not define any chat headers. Messages are sent directly as binary encoded strings using TCP/TLS connections.
For both client and server, we created a new concurrent thread to receive messages from the other side of the connection and the main thread takes input from the standard input to send to the other side.

Flow:
Alice first sends a "chat_hello" message to Bob and Bob replies with "chat_reply". If Alice doesn't receive the "chat_reply" message, the session is terminated. Same is the case for Bob on "chat_hello" message. After this handshake, Alice sends a "chat_STARTLS" message to Bob. If Bob receives this message he sends "chat_STARTTLS_ACK" to Alice. Upon receiving this message, Alice sends the "chat_STARTTLS_ACK" again to Bob who is waiting for this message. This acts as a three way handshake for STARTTLS.

Underlying Structure:
Now the real TLS handshake starts in the “wrap_socket()” method.
Once Bob receives this ack message, He starts the TLS Session handshake and waits for Alice's TLS "client_hello". If in case Bob receives any message other than this, he doesn't start the TLS Session and communication happens over insecure connection.
Alice now initiates TLS handshake by sending "client_hello".

Verifications:
Alice also loads her own certificate for client verification.
By default, verification of client’s certificate is set to optional. So server requests for a certificate from client but when there’s no response from client it proceeds further to next steps.
But we made it compulsory for the client(Alice in this case) to load its certificate and the server made it compulsory for the client certificate verification.



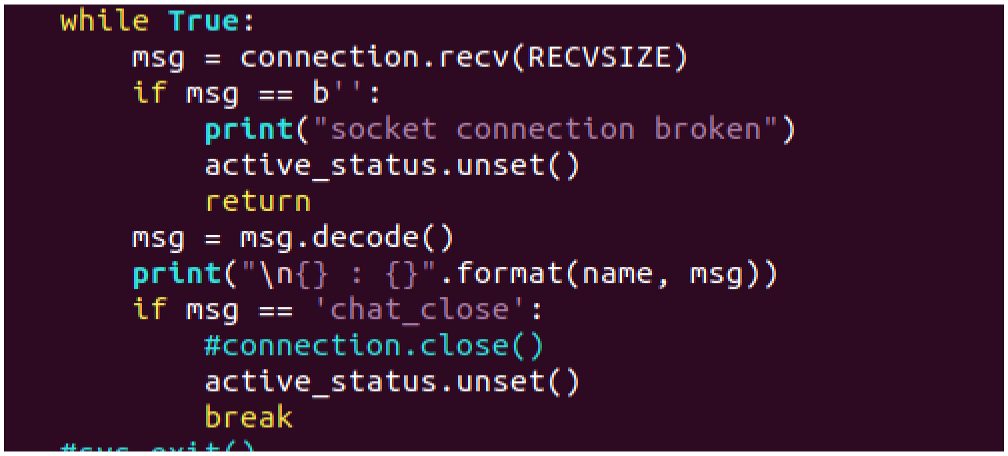
```
    context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_verify_locations("/root/root.crt")
    context.load_cert_chain(certfile="bob.crt", keyfile="bob-key.pe
    context.verify_mode = ssl.CERT_REQUIRED

    connssl = context.wrap_socket(connection, server_side=True)

    communication(connssl, 'Server', 'Client')
```

TLS 1.3 Handshake happens between alice and bob.
Although many cipher suites are supported by the client and server in our app, we put the default highest preference order for client and server. So, Alice and Bob use TLS 1.3 Ciphers always. In handshake messages like client hello and server hello, we can't observe the certificates because in TLS1.3 even handshake messages are encrypted using handshake keys.

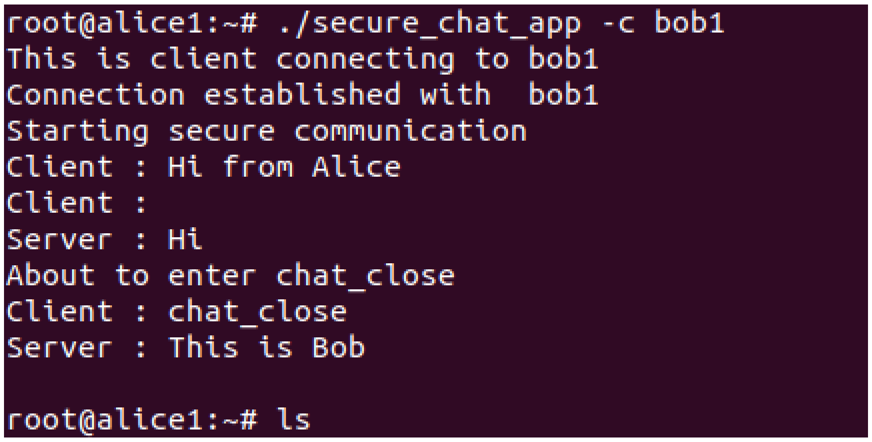
When either side sends the "chat_close" message to the other side, the chat ends gracefully. Even when one of the sides gives a keyboard interrupt (Ctrl+C), an empty byte string will be sent to the other side and the chat ends gracefully. We handled it using a flag called "active_status" as follows.



```
    while True:
        msg = connection.recv(RECVSIZE)
        if msg == b'':
            print("socket connection broken")
            active_status.unset()
            return
        msg = msg.decode()
        print("\n{} : {}".format(name, msg))
        if msg == 'chat_close':
            #connection.close()
            active_status.unset()
            break
    #sys.exit()
```

Working:

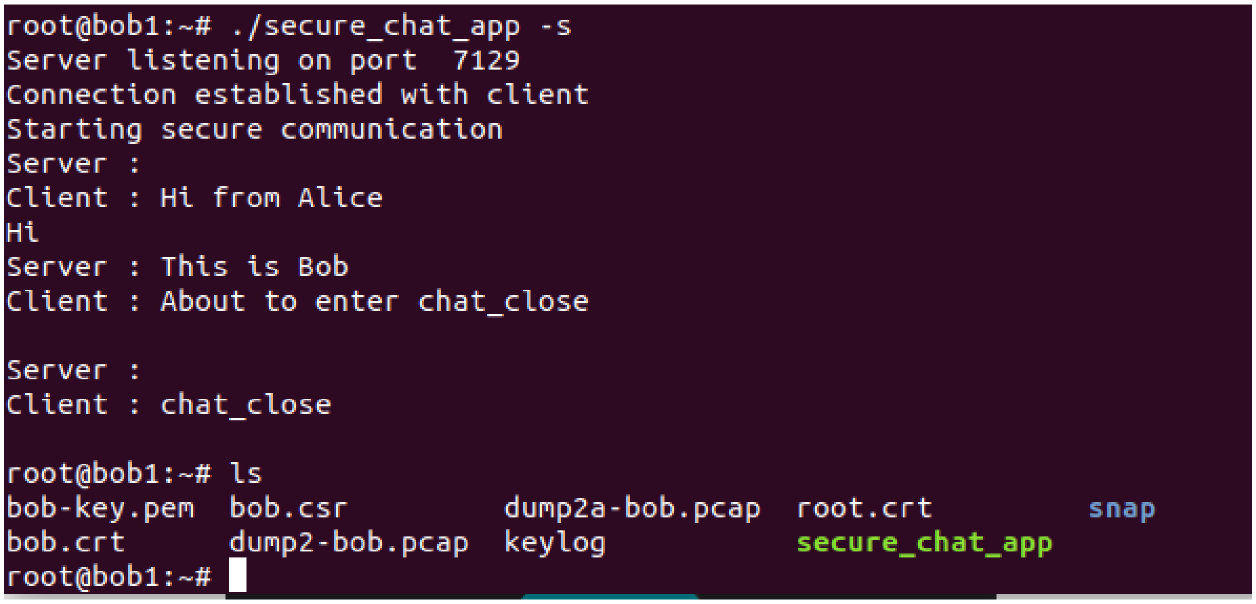
Alice's :



```
root@alice1:~# ./secure_chat_app -c bob1
This is client connecting to bob1
Connection established with  bob1
Starting secure communication
Client : Hi from Alice
Client :
Server : Hi
About to enter chat_close
Client : chat_close
Server : This is Bob

root@alice1:~# ls
```

Bob's :



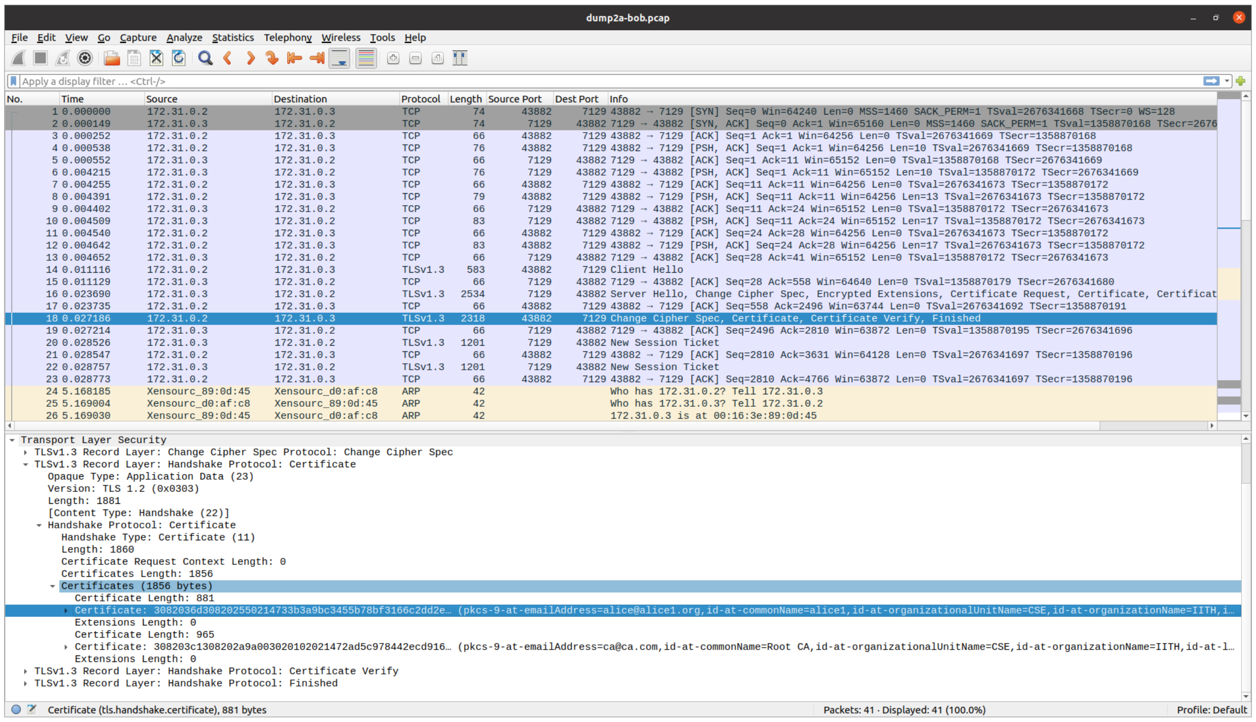
```
root@bob1:~# ./secure_chat_app -s
Server listening on port  7129
Connection established with client
Starting secure communication
Server :
Client : Hi from Alice
Hi
Server : This is Bob
Client : About to enter chat_close

Server :
Client : chat_close

root@bob1:~# ls
bob-key.pem  bob.csr        dump2a-bob.pcap  root.crt         snap
bob.crt      dump2-bob.pcap keylog           secure_chat_app
root@bob1:~#
```

Certificates transfer from pcap: (After decryption using SSLKEYLOGFILE)

Alice’s:

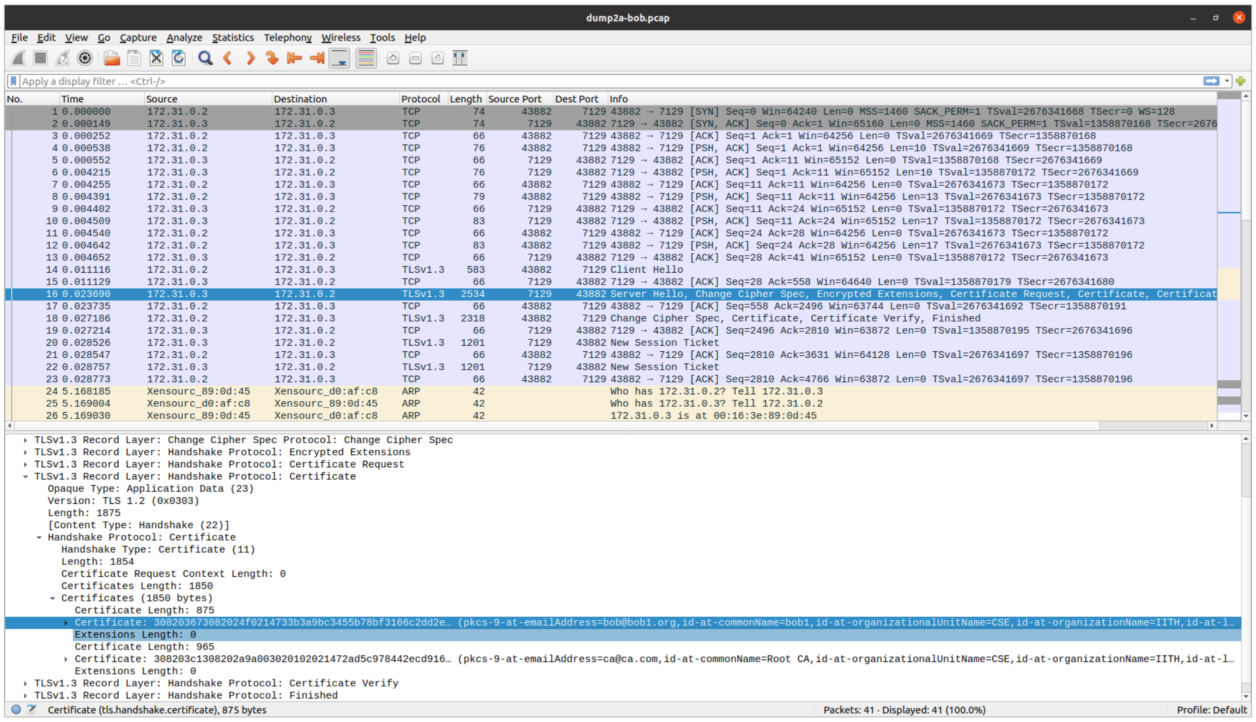

dump2a-bob.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time       Source             Destination        Protocol Length Source Port Dest Port Info
   1 0.000000   172.31.0.2         172.31.0.3         TCP        74      43882      7129 43882 → 7129 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2676341668 TSecr=0 WS=128
   2 0.000149   172.31.0.3         172.31.0.2         TCP        74       7129     43882 7129 → 43882 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1358870168 TSecr=2676
   3 0.000252   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2676341669 TSecr=1358870168
   4 0.000538   172.31.0.2         172.31.0.3         TCP        76      43882      7129 43882 → 7129 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=2676341669 TSecr=1358870168
   5 0.000552   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=1 Ack=11 Win=65152 Len=0 TSval=1358870168 TSecr=2676341669
   6 0.004215   172.31.0.3         172.31.0.2         TCP        76       7129     43882 7129 → 43882 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=10 TSval=1358870172 TSecr=2676341669
   7 0.004255   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=11 Ack=11 Win=64256 Len=0 TSval=2676341673 TSecr=1358870172
   8 0.004391   172.31.0.2         172.31.0.3         TCP        79      43882      7129 43882 → 7129 [PSH, ACK] Seq=11 Ack=11 Win=64256 Len=13 TSval=2676341673 TSecr=1358870172
   9 0.004402   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=11 Ack=24 Win=65152 Len=0 TSval=1358870172 TSecr=2676341673
  10 0.004509   172.31.0.3         172.31.0.2         TCP        83       7129     43882 7129 → 43882 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=17 TSval=1358870172 TSecr=2676341673
  11 0.004540   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=24 Ack=28 Win=64256 Len=0 TSval=2676341673 TSecr=1358870172
  12 0.004642   172.31.0.2         172.31.0.3         TCP        83      43882      7129 43882 → 7129 [PSH, ACK] Seq=24 Ack=28 Win=64256 Len=17 TSval=2676341673 TSecr=1358870172
  13 0.004652   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=28 Ack=41 Win=65152 Len=0 TSval=1358870172 TSecr=2676341673
  14 0.011116   172.31.0.2         172.31.0.3         TLSv1.3   583      43882      7129 Client Hello
  15 0.011129   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=28 Ack=558 Win=64640 Len=0 TSval=1358870179 TSecr=2676341680
  16 0.023690   172.31.0.3         172.31.0.2         TLSv1.3  2534       7129     43882 Server Hello, Change Cipher Spec, Encrypted Extensions, Certificate Request, Certificate, Certificat
  17 0.023735   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=558 Ack=2496 Win=63744 Len=0 TSval=2676341692 TSecr=1358870191
  18 0.027186   172.31.0.2         172.31.0.3         TLSv1.3  2318      43882      7129 Change Cipher Spec, Certificate, Certificate Verify, Finished
  19 0.027214   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=2496 Ack=2810 Win=63872 Len=0 TSval=1358870195 TSecr=2676341696
  20 0.028526   172.31.0.3         172.31.0.2         TLSv1.3  1201       7129     43882 New Session Ticket
  21 0.028547   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=2810 Ack=3631 Win=64128 Len=0 TSval=2676341697 TSecr=1358870196
  22 0.028757   172.31.0.3         172.31.0.2         TLSv1.3  1201       7129     43882 New Session Ticket
  23 0.028773   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=2810 Ack=4766 Win=63872 Len=0 TSval=2676341697 TSecr=1358870196
  24 5.168185   Xensourc_89:0d:45  Xensourc_d0:af:c8  ARP        42                       Who has 172.31.0.2? Tell 172.31.0.3
  25 5.169004   Xensourc_d0:af:c8  Xensourc_89:0d:45  ARP        42                       Who has 172.31.0.3? Tell 172.31.0.2
  26 5.169030   Xensourc_89:0d:45  Xensourc_d0:af:c8  ARP        42                       172.31.0.3 is at 00:16:3e:89:0d:45
```

```
Transport Layer Security
  TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  TLSv1.3 Record Layer: Handshake Protocol: Certificate
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 1881
    [Content Type: Handshake (22)]
    Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 1860
      Certificate Request Context Length: 0
      Certificates Length: 1856
      Certificates (1856 bytes)
        Certificate Length: 881
        Certificate: 3082036d308202550214733b3a9bc3455b78bf3166c2dd2e… (pkcs-9-at-emailAddress=alice@alice1.org,id-at-commonName=alice1,id-at-organizationalUnitName=CSE,id-at-organizationName=IITH,i…
        Extensions Length: 0
        Certificate Length: 965
        Certificate: 308203c1308202a9a003020102021472ad5c978442ecd916… (pkcs-9-at-emailAddress=ca@ca.com,id-at-commonName=Root CA,id-at-organizationalUnitName=CSE,id-at-organizationName=IITH,id-at-l…
        Extensions Length: 0
  TLSv1.3 Record Layer: Handshake Protocol: Certificate Verify
  TLSv1.3 Record Layer: Handshake Protocol: Finished
```

```
Certificate (tls.handshake.certificate), 881 bytes          Packets: 41 · Displayed: 41 (100.0%)          Profile: Default
```

Bob’s:


dump2a-bob.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time       Source             Destination        Protocol Length Source Port Dest Port Info
   1 0.000000   172.31.0.2         172.31.0.3         TCP        74      43882      7129 43882 → 7129 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2676341668 TSecr=0 WS=128
   2 0.000149   172.31.0.3         172.31.0.2         TCP        74       7129     43882 7129 → 43882 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1358870168 TSecr=2676
   3 0.000252   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2676341669 TSecr=1358870168
   4 0.000538   172.31.0.2         172.31.0.3         TCP        76      43882      7129 43882 → 7129 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=2676341669 TSecr=1358870168
   5 0.000552   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=1 Ack=11 Win=65152 Len=0 TSval=1358870168 TSecr=2676341669
   6 0.004215   172.31.0.3         172.31.0.2         TCP        76       7129     43882 7129 → 43882 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=10 TSval=1358870172 TSecr=2676341669
   7 0.004255   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=11 Ack=11 Win=64256 Len=0 TSval=2676341673 TSecr=1358870172
   8 0.004391   172.31.0.2         172.31.0.3         TCP        79      43882      7129 43882 → 7129 [PSH, ACK] Seq=11 Ack=11 Win=64256 Len=13 TSval=2676341673 TSecr=1358870172
   9 0.004402   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=11 Ack=24 Win=65152 Len=0 TSval=1358870172 TSecr=2676341673
  10 0.004509   172.31.0.3         172.31.0.2         TCP        83       7129     43882 7129 → 43882 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=17 TSval=1358870172 TSecr=2676341673
  11 0.004540   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=24 Ack=28 Win=64256 Len=0 TSval=2676341673 TSecr=1358870172
  12 0.004642   172.31.0.2         172.31.0.3         TCP        83      43882      7129 43882 → 7129 [PSH, ACK] Seq=24 Ack=28 Win=64256 Len=17 TSval=2676341673 TSecr=1358870172
  13 0.004652   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=28 Ack=41 Win=65152 Len=0 TSval=1358870172 TSecr=2676341673
  14 0.011116   172.31.0.2         172.31.0.3         TLSv1.3   583      43882      7129 Client Hello
  15 0.011129   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=28 Ack=558 Win=64640 Len=0 TSval=1358870179 TSecr=2676341680
  16 0.023690   172.31.0.3         172.31.0.2         TLSv1.3  2534       7129     43882 Server Hello, Change Cipher Spec, Encrypted Extensions, Certificate Request, Certificate, Certificat
  17 0.023735   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=558 Ack=2496 Win=63744 Len=0 TSval=2676341692 TSecr=1358870191
  18 0.027186   172.31.0.2         172.31.0.3         TLSv1.3  2318      43882      7129 Change Cipher Spec, Certificate, Certificate Verify, Finished
  19 0.027214   172.31.0.3         172.31.0.2         TCP        66       7129     43882 7129 → 43882 [ACK] Seq=2496 Ack=2810 Win=63872 Len=0 TSval=1358870195 TSecr=2676341696
  20 0.028526   172.31.0.3         172.31.0.2         TLSv1.3  1201       7129     43882 New Session Ticket
  21 0.028547   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=2810 Ack=3631 Win=64128 Len=0 TSval=2676341697 TSecr=1358870196
  22 0.028757   172.31.0.3         172.31.0.2         TLSv1.3  1201       7129     43882 New Session Ticket
  23 0.028773   172.31.0.2         172.31.0.3         TCP        66      43882      7129 43882 → 7129 [ACK] Seq=2810 Ack=4766 Win=63872 Len=0 TSval=2676341697 TSecr=1358870196
  24 5.168185   Xensourc_89:0d:45  Xensourc_d0:af:c8  ARP        42                       Who has 172.31.0.2? Tell 172.31.0.3
  25 5.169004   Xensourc_d0:af:c8  Xensourc_89:0d:45  ARP        42                       Who has 172.31.0.3? Tell 172.31.0.2
  26 5.169030   Xensourc_89:0d:45  Xensourc_d0:af:c8  ARP        42                       172.31.0.3 is at 00:16:3e:89:0d:45
```

```
  TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  TLSv1.3 Record Layer: Handshake Protocol: Encrypted Extensions
  TLSv1.3 Record Layer: Handshake Protocol: Certificate Request
  TLSv1.3 Record Layer: Handshake Protocol: Certificate
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 1875
    [Content Type: Handshake (22)]
    Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 1854
      Certificate Request Context Length: 0
      Certificates Length: 1850
      Certificates (1850 bytes)
        Certificate Length: 875
        Certificate: 308203673082024f0214733b3a9bc3455b78bf3166c2dd2e… (pkcs-9-at-emailAddress=bob@bob1.org,id-at-commonName=bob1,id-at-organizationalUnitName=CSE,id-at-organizationName=IITH,id-at-l…
        Extensions Length: 0
        Certificate Length: 965
        Certificate: 308203c1308202a9a003020102021472ad5c978442ecd916… (pkcs-9-at-emailAddress=ca@ca.com,id-at-commonName=Root CA,id-at-organizationalUnitName=CSE,id-at-organizationName=IITH,id-at-l…
        Extensions Length: 0
  TLSv1.3 Record Layer: Handshake Protocol: Certificate Verify
  TLSv1.3 Record Layer: Handshake Protocol: Finished
```

```
Certificate (tls.handshake.certificate), 875 bytes          Packets: 41 · Displayed: 41 (100.0%)          Profile: Default
```

TASK3:

We designed a secure_chat_interceptor in order for Trudy to intercept the chat between Alice and bob. First we poison the DNS using the given script to change the IP addresses in hosts files of alice and bob.
This makes Alice and BobBob believe they are chatting with each other while they are originally connected to Trudy's IP. Trudy does a passive MITM attack which implies that Trudy doesn’t alter the messages between them but directly transfers them across. So the chat is no longer confidential.
When Alice requests for a secure TLS connection with chat_STARTTLS message, Trudy replies with chat_STARTTLS_NOT_SUPPORTED but doesn’t transfer the request for TLS over to Bob.
The secure_chat_interceptor program first connects to Alice acting as Bob and then to Bob acting as Alice.
after connecting to both, two new threads are spawn by the main thread. One thread waits on receiving from Alice and sends to Bob and the other thread waits on receiving from Bob and sends to Alice.
Other than interfering in the chat_STARTTLS messages Trudy sends every message from Alice to Bob(and vice versa) as-it-as. It also prints every message onto standard input.

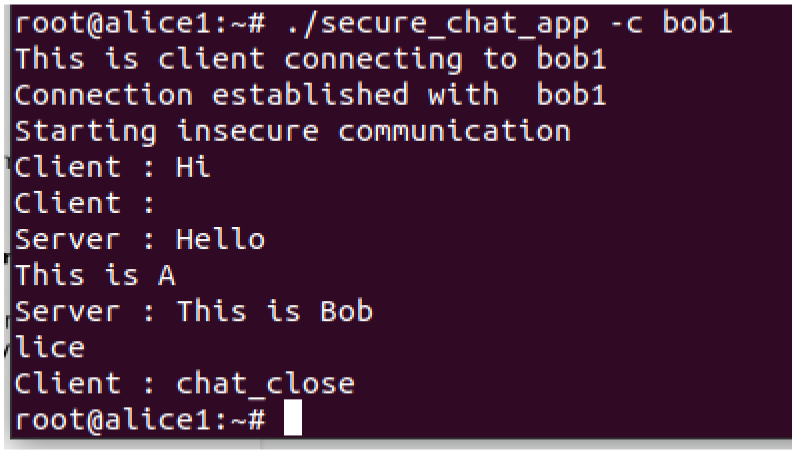
When a "chat_close" message is received from one side, it sends the message to the other side and closes both the connections.

Alice <--------> Trudy <--------> Bob

Here 2 TCP connections are formed, one between Alice and Trudy, other between Trudy and Bob. So the chat remains insecure as TLS connection is not implemented.

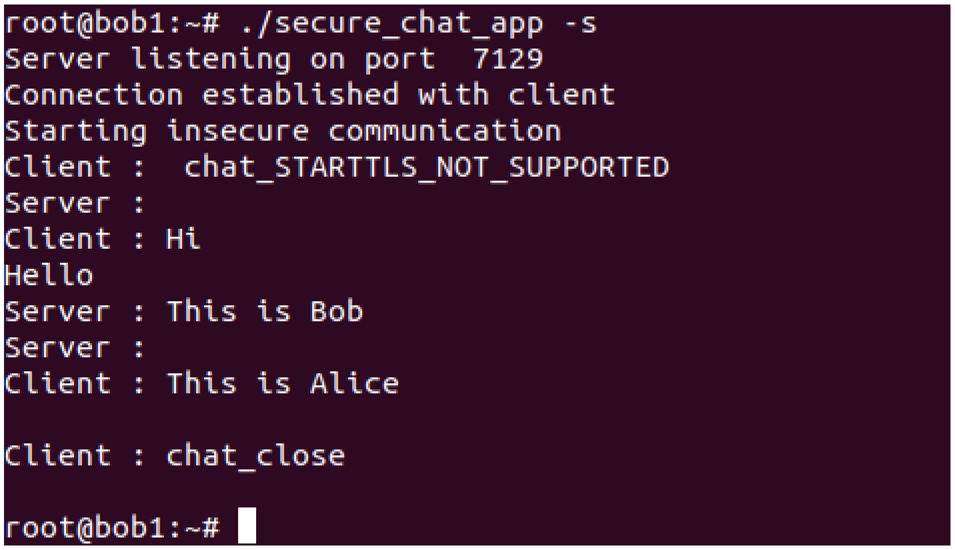
Working :

Alice’s :



```
root@alice1:~# ./secure_chat_app -c bob1
This is client connecting to bob1
Connection established with  bob1
Starting insecure communication
Client : Hi
Client :
Server : Hello
This is A
Server : This is Bob
lice
Client : chat_close
root@alice1:~#
```

Bob’s :

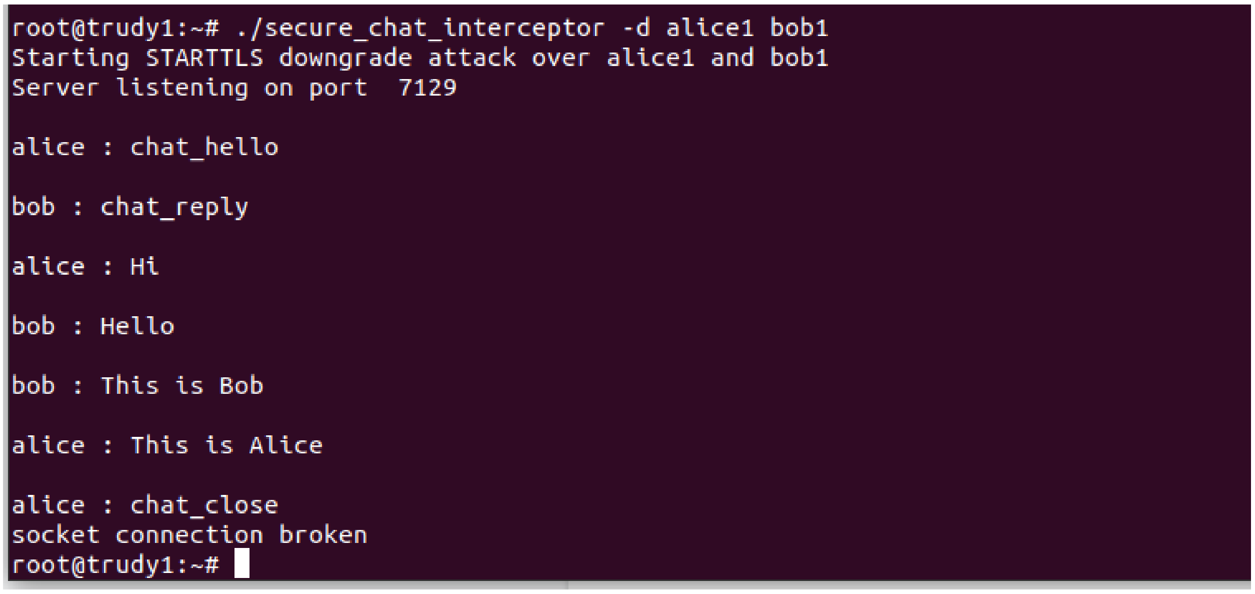


```
root@bob1:~# ./secure_chat_app -s
Server listening on port  7129
Connection established with client
Starting insecure communication
Client :  chat_STARTTLS_NOT_SUPPORTED
Server :
Client : Hi
Hello
Server : This is Bob
Server :
Client : This is Alice

Client : chat_close

root@bob1:~#
```

Trudy's:



```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
Starting STARTTLS downgrade attack over alice1 and bob1
Server listening on port  7129

alice : chat_hello

bob : chat_reply

alice : Hi

bob : Hello

bob : This is Bob

alice : This is Alice

alice : chat_close
socket connection broken
root@trudy1:~#
```

TASK 4:

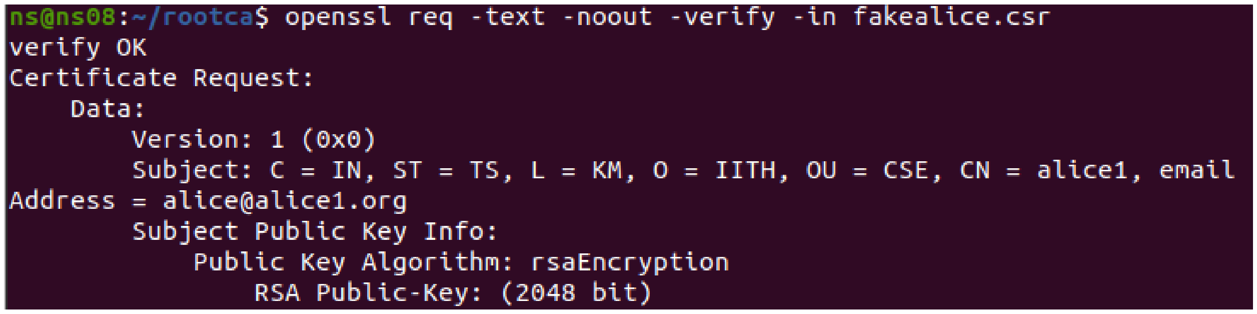
Generating key file and CSR for fake alice by trudy

```
    $ openssl req -newkey rsa:2048 -nodes -keyout
fakealice-key.pem -out fakealice.csr
```

Generating key file and CSR for fake bob by trudy

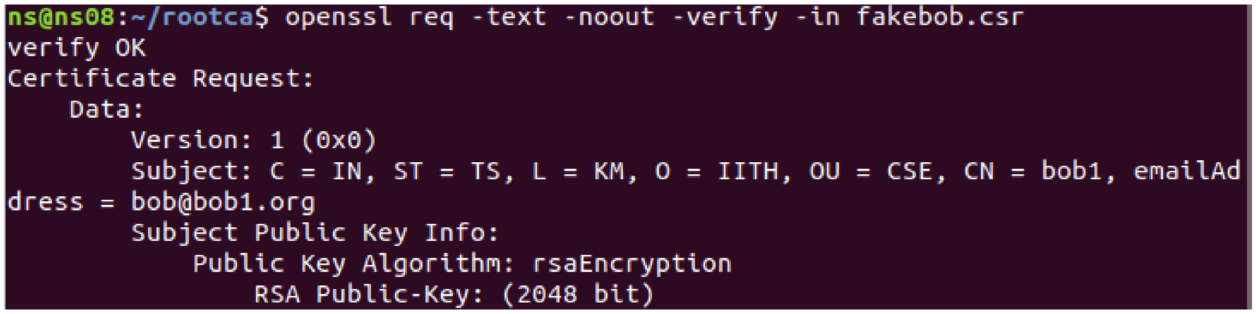
```
    $ openssl req -newkey rsa:2048 -nodes -keyout
fakebob-key.pem -out fakebob.csr
```

Verifying CSR of fake alice



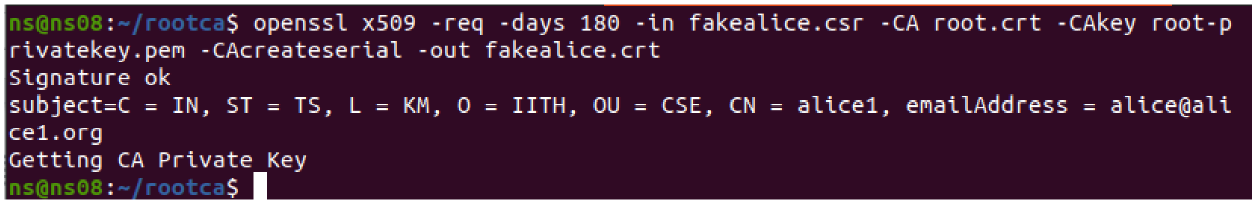
```
ns@ns08:~/rootca$ openssl req -text -noout -verify -in fakealice.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = alice1, email
Address = alice@alice1.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
```

Verifying CSR of fake bob

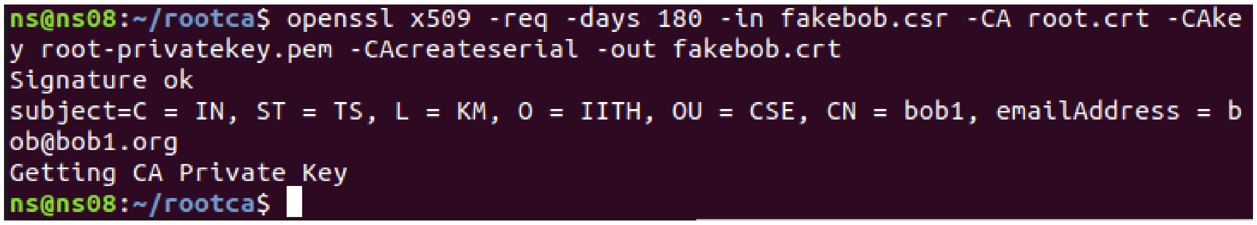


```
ns@ns08:~/rootca$ openssl req -text -noout -verify -in fakebob.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = bob1, emailAd
dress = bob@bob1.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
```

Generating certificates for fake alice and fake bob

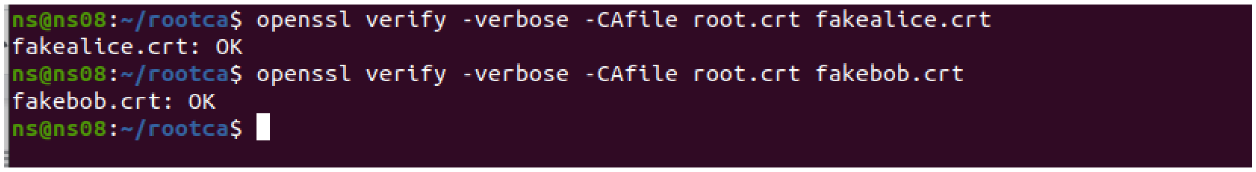


```
ns@ns08:~/rootca$ openssl x509 -req -days 180 -in fakealice.csr -CA root.crt -CAkey root-p
rivatekey.pem -CAcreateserial -out fakealice.crt
Signature ok
subject=C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = alice1, emailAddress = alice@ali
ce1.org
Getting CA Private Key
ns@ns08:~/rootca$ 
```



```
ns@ns08:~/rootca$ openssl x509 -req -days 180 -in fakebob.csr -CA root.crt -CAke
y root-privatekey.pem -CAcreateserial -out fakebob.crt
Signature ok
subject=C = IN, ST = TS, L = KM, O = IITH, OU = CSE, CN = bob1, emailAddress = b
ob@bob1.org
Getting CA Private Key
ns@ns08:~/rootca$
```

Verifying Certificate of Fake Alice and Fake Bob:



```
ns@ns08:~/rootca$ openssl verify -verbose -CAfile root.crt fakealice.crt
fakealice.crt: OK
ns@ns08:~/rootca$ openssl verify -verbose -CAfile root.crt fakebob.crt
fakebob.crt: OK
ns@ns08:~/rootca$
```

For this task, we've used the same functions as in the previous task except that now the sending and receiving happens through an SSL socket.

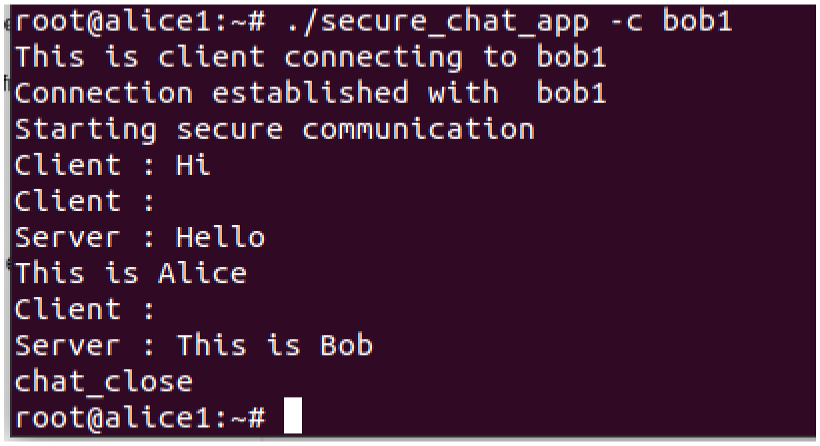
Trudy first connects to Alice impersonating Bob and initiates application layer handshake and STARTTLS handshake. Trudy uses the fakebob certificate which is generated as above to prove her authenticity. Since this is validated by root CA, Alice trusts this certificate. TLS handshake completes and a TLS pipe is established between Alice and Trudy.
After Trudy connects to Alice, she now connects to Bob, does the application layer and STARTTLS handshakes. Now, she initiates TLS handshake with Bob impersonating Alice. Since she has fakealice certificate which is signed by the root CA, Bob trusts this connection and establishes a TLS connection. Trudy now has a TLS pipe with Bob.

After this, Trudy will transfer the messages as-it-is from Alice to Bob and vice versa. She can read all the messages.

In the program, we have used the same functions for this purpose also except that in the mode Trudy does not check for chat_STARTTLS message from Alice.
Every message is TLS-encrypted between Alice-Trudy and Trudy-Bob. We are printing all the messages onto the standard input.

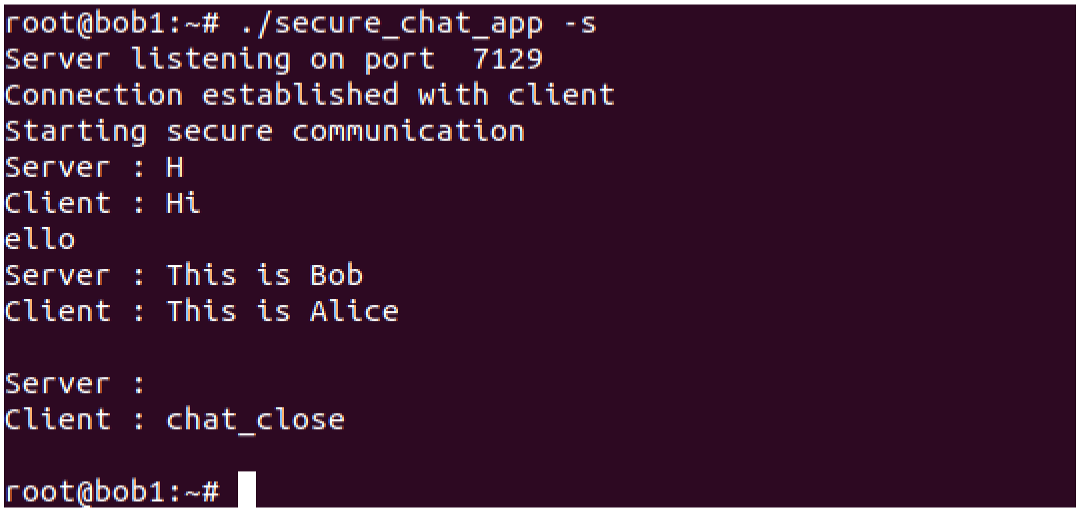
Working :

Alice’s :



```
root@alice1:~# ./secure_chat_app -c bob1
This is client connecting to bob1
Connection established with  bob1
Starting secure communication
Client : Hi
Client :
Server : Hello
This is Alice
Client :
Server : This is Bob
chat_close
root@alice1:~#
```

Bob’s :

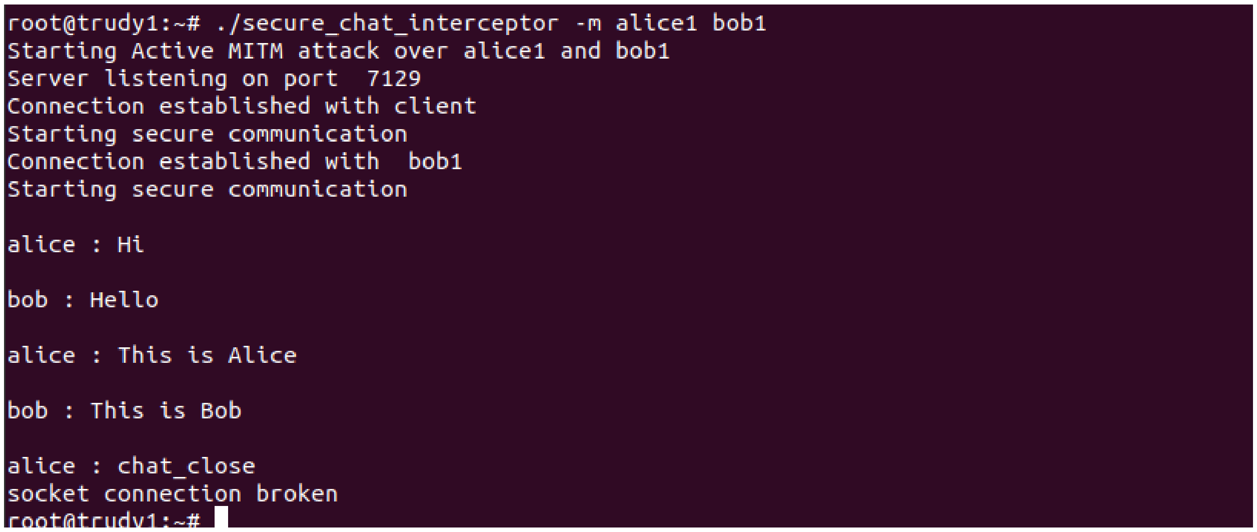


```
root@bob1:~# ./secure_chat_app -s
Server listening on port  7129
Connection established with client
Starting secure communication
Server : H
Client : Hi
ello
Server : This is Bob
Client : This is Alice

Server :
Client : chat_close

root@bob1:~#
```

Trudy’s :



```
root@trudy1:~# ./secure_chat_interceptor -m alice1 bob1
Starting Active MITM attack over alice1 and bob1
Server listening on port  7129
Connection established with client
Starting secure communication
Connection established with  bob1
Starting secure communication

alice : Hi

bob : Hello

alice : This is Alice

bob : This is Bob

alice : chat_close
socket connection broken
root@trudy1:~#
```

References:

ssl — TLS/SSL wrapper for socket objects — Python 3.9.2 documentation
OpenSSL Cookbook: Chapter 1. OpenSSL Command Line