Assignment 6: Secure chat using openssl and MITM attacks

Task 1:

Root CA:

Generating 512-bit ECC Private Key of Root CA :
openssl ecparam -name brainpoolP512t1 -genkey -noout -out root.key

Creating Self-Signed Certificate of the Root CA using the above generated private key with validity of 10 years:
Openssl req -x509 -sha256 -new -nodes -key root.key -days 3650 -out root.crt
[we get prompted to give details]

[Two files created : root.key , root.crt]

Alice :

We generate private key and CSR of Alice in it's container :

```
root@alice1:~/securechat/rootCA# cd ..
root@alice1:~/securechat# cd alice/
root@alice1:~/securechat/alice# openssl req -new -newkey rsa:2048 -nodes -keyout alice.key -out alice.csr
Generating a RSA private key
.....................................+++++
.....................................................................................................+++++
writing new private key to 'alice.key'
-----
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]:IN
State or Province Name (full name) [Some-State]:TS
Locality Name (eg, city) []:HYD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:alice.iith.ac.in
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:nsa6
An optional company name []:
root@alice1:~/securechat/alice# ls
alice.csr  alice.key
root@alice1:~/securechat/alice#
```

Fig : alice1 container has alice.csr and alice.key

Sending the CSR and Key (from alice1 container) to Root(VM) to sign it:

```
ns@ns09:~$ sudo su
root@ns09:/home/ns# /home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.csr alice/
bash: /home/ns#: No such file or directory
root@ns09:/home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.csr alice/
root@ns09:/home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.key alice/
root@ns09:/home/ns# exit
exit
```

Verifying Alice's CSR in VM:

```
ns@ns09:~$ openssl req -text -noout -in alice/alice.csr -verify
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = alice.iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ab:ce:94:a2:e6:ab:bb:5b:fd:a7:98:69:70:2a:
                    24:ba:29:cf:85:e0:dc:b2:4a:a3:37:cb:a4:fa:ff:
                    5f:f0:a8:e8:17:e6:4a:26:af:9c:4d:8a:c7:c5:24:
                    12:7a:14:16:b7:73:26:75:1c:bc:86:aa:7b:eb:aa:
                    de:c7:31:80:2e:ad:f2:26:a3:4b:26:59:a5:1e:2b:
                    54:7c:d3:ce:4f:d6:7a:54:23:29:df:06:42:b4:ab:
                    49:a7:2b:36:0d:90:89:1a:67:be:b1:a8:e6:01:a0:
                    9a:80:98:27:d6:6b:b9:5d:f2:07:e3:f2:f2:de:e0:
                    11:13:2e:fa:35:fd:1b:5f:aa:40:ba:b3:ba:91:32:
                    07:ac:d1:ca:d8:84:fc:26:58:23:ab:f5:8c:a5:77:
                    6a:4b:a0:3e:13:3b:e8:ee:15:62:1e:e1:28:90:1d:
                    8e:72:04:8c:39:40:50:ec:49:c8:1d:6c:42:35:ab:
                    dc:24:4a:c9:a0:25:01:ef:e8:a2:58:3e:80:4d:99:
                    45:f6:b3:9f:3a:70:41:db:be:b5:60:6f:86:88:2a:
                    bd:58:4d:a8:b1:0b:b5:e6:60:4b:d9:f1:91:04:4d:
                    54:7e:38:e1:1f:94:e3:31:26:40:f7:37:8f:0f:f8:
                    82:5a:cb:68:3e:8b:e8:4b:d3:30:8e:0c:df:4a:c8:
                    7e:3b
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :nsa6
    Signature Algorithm: sha256WithRSAEncryption
         4d:e4:af:ad:cc:89:7d:28:15:78:53:a6:5b:b8:e5:5f:91:21:
         88:e9:fb:18:a6:93:37:8b:18:62:60:37:88:e7:9e:b5:57:35:
         2e:81:68:b4:2c:a0:c3:38:88:e6:1b:ab:33:ef:f6:94:fd:83:
         c5:3d:1a:65:90:83:ad:3c:05:56:50:66:8f:37:19:13:42:2f:
         24:2a:28:d9:16:41:cd:1b:ee:29:ab:b9:87:89:35:62:5a:88:
         5d:7f:69:57:38:66:57:2f:8c:87:7f:a4:3a:0c:34:42:5c:cc:
         72:e6:42:74:2b:10:0b:05:51:1b:3a:15:a2:ed:cb:6d:45:98:
```

Fig : Verification done successfully

Signing of CSR by Root CA:

```
ns@ns09:~$ openssl x509 -req -days 365 -in alice/alice.csr -CA rootCA/root.crt -CAkey rootCA/root.key -CAcreateserial -out alice/alice.crt
Signature ok
subject=C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = alice.iith.ac.in
Getting CA Private Key
```

Sending Alice certificate from VM to container :

```
ns@ns09:~$ sudo su
root@ns09:/home/ns# cp alice/alice.crt /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/
```

Integrity Check - comparing the modulus :

```
ns@ns09:~$ lxc exec alice1 bash
root@alice1:~# cd securechat/
root@alice1:~/securechat# ls
alice  poison-dns-alice1-bob1.sh  rootCA  secure_chat.py  unpoison-dns-alice1-bob1.sh
root@alice1:~/securechat# openssl x509 -noout -modulus -in alice/alice.crt >a
root@alice1:~/securechat# openssl rsa -noout -modulus -in alice/alice.key >b
root@alice1:~/securechat# diff a b
root@alice1:~/securechat#
```

Bob:

We generate private key and CSR of Bob in it's container :

```
root@bob1:~/securechat# cd bob/
root@bob1:~/securechat/bob# openssl req -new -newkey rsa:2048 -nodes -keyout bob.key -out bob.csr
Generating a RSA private key
.............+++++
.........................+++++
writing new private key to 'bob.key'
-----
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]:IN
State or Province Name (full name) [Some-State]:TS
Locality Name (eg, city) []:HYD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:bob.iith.ac.in
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:nsa6
An optional company name []:
root@bob1:~/securechat/bob# ls
bob.csr  bob.key
root@bob1:~/securechat/bob#
```

Fig : bob1 container has bob.csr and bob.key

Sending the CSR and Key (from bob1 container) to Root(VM) to sign it:

```
ns@ns09:~$ sudo su
root@ns09:/home/ns# /home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.csr alice/
bash: /home/ns#: No such file or directory
root@ns09:/home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.csr alice/
root@ns09:/home/ns# cp /var/snap/lxd/common/lxd/containers/alice1/rootfs/root/securechat/alice/alice.key alice/
root@ns09:/home/ns# exit
exit
```

Verifying Bob's CSR in VM:

```
ns@ns09:~$ openssl req -text -noout -in bob/bob.csr -verify
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = bob.iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b0:89:5a:5f:01:cf:48:48:df:04:0e:da:49:45:
                    b5:2c:20:7d:44:4b:63:91:a6:58:1c:39:ee:19:8c:
                    31:69:23:42:b6:6b:98:88:62:9d:7c:46:45:02:1f:
                    96:f7:5a:1c:dc:7d:97:e5:ab:59:81:8e:83:ee:91:
                    a6:1d:c0:32:ad:a1:28:91:ef:75:db:5d:74:b3:45:
                    9c:bf:bf:ce:2c:05:98:60:e4:ff:ce:42:38:86:a6:
                    9b:bb:e4:ef:b3:67:fd:5a:1b:33:46:c0:82:02:ff:
                    55:90:3b:04:98:ed:d4:b6:d5:b7:29:35:aa:43:76:
                    78:ca:b6:11:9e:c6:99:df:07:39:54:37:25:9e:61:
                    a8:e8:fc:bf:75:9a:b3:ef:76:07:d7:52:48:79:46:
                    e0:66:54:42:27:7a:86:18:d2:8d:34:90:35:9e:ab:
                    a8:00:91:40:26:07:cd:0f:e7:70:e9:45:b1:d1:06:
                    e8:3b:0d:cc:02:20:da:aa:37:d0:d3:58:11:16:e5:
                    e2:4f:8b:17:d7:b8:e0:6f:3a:a7:35:0e:2c:e3:3f:
                    2e:ac:e5:4c:3d:60:e3:e8:b5:fd:d4:49:95:4e:04:
                    e1:f8:54:2e:b4:d6:a6:f3:e6:85:e1:67:18:e1:2f:
                    6f:dd:80:20:3f:2e:16:23:6b:a7:50:e7:08:1a:35:
                    dd:c5
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :nsa6
    Signature Algorithm: sha256WithRSAEncryption
         8f:dc:e1:d0:10:9a:89:0b:35:05:06:dc:31:bb:53:f8:05:21:
         2d:0f:11:37:89:41:8d:d3:fd:58:c5:d5:b7:6b:5d:36:28:48:
         b8:78:c4:a0:96:87:c1:d1:28:bd:2c:cc:1e:91:dc:fc:4c:76:
         dd:57:98:a5:c9:51:bf:82:d9:bb:9d:77:fb:cc:69:ce:57:22:
         f4:17:0d:c5:0a:a2:21:9c:55:62:11:2a:6e:cd:27:87:5d:c0:
         96:71:b4:64:7d:0e:2c:65:b2:7f:94:03:4f:ac:d7:49:d5:c8:
         3f:9d:56:6a:79:25:3e:d9:a2:91:f4:cb:2f:f5:bd:bf:ed:59:
         07:64:58:16:55:65:2f:68:9b:12:31:11:1a:db:9d:59:73:d8:
         e8:da:16:68:d2:9c:40:f4:e1:ef:f8:47:7f:df:ff:ae:be:7c:
         85:11:f3:3b:21:56:13:e8:f2:5f:c1:fb:2a:2f:fa:ff:a5:b2:
         59:36:29:8b:d7:db:72:54:21:ed:af:61:8f:9b:47:c3:1c:dd:
         26:e0:53:e4:14:70:72:87:6e:4e:d4:84:87:d6:db:9e:38:af:
```

Fig : Verification done successfully

Signing of CSR by Root CA:

```
ns@ns09:~$ openssl x509 -req -days 365 -in bob/bob.csr -CA rootCA/root.crt -CAkey rootCA/root.key -CAcreateserial -out bob/bob.crt
Signature ok
subject=C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = bob.iith.ac.in
Getting CA Private Key
```

Sending Bob certificate from VM to container :

```
root@ns09:/home/ns# cp bob/bob.crt /var/snap/lxd/common/lxd/containers/bob1/rootfs/root/securechat/bob/
root@ns09:/home/ns#
```

Integrity Check - comparing the modulus :

```
ns@ns09:~$ lxc exec bob1 bash
root@bob1:~# cd securechat/
root@bob1:~/securechat# openssl x509 -noout -modulus -in bob/bob.crt >a
root@bob1:~/securechat# openssl rsa -noout -modulus -in bob/bob.key >b
root@bob1:~/securechat# diff a b
root@bob1:~/securechat#
```

Task 2:

In this task, we have created a secure chat application which uses TCP, TLS for reliable communication between client and server. For implementing this we have used python's ssl, socket, OpenSSL packages.

Implementation of Client:

- To run the client application, we need to give the server name, port number it wants to connect to.
- Using the above arguments, the client will create the socket and connect to it.
- **Establishing TCP Connection:**
    - Client sends "*chat_hello*" and waits for server's "*chat_reply*". If the server fails to send "*chat_reply*" the TCP Connection will not be established and code will exit.
    - The messages that are exchanged while establishing TCP connection are in plain text format.
- **Establishing TLS Connection:**
    - If a client types "*chat_STARTTLS*", he wants to open a TLS pipe for secure communication.
    - When the server sees the client's "*chat_STARTTLS*" message it sends in "*chat_STARTTLS_ACK*" acknowledging it and saying that it is ready to open a TLS connection.
    - Next the client sends "*client_hello*", to which server sends "*server_hello*".
    - Until here all the messages are exchanged in plain text format.
    - Once "*client_hello*" and "*server_hello*" exchange is done, certificate exchange takes place.
    - Client will receive server's crt, key file and server will receive client's crt, key file respectively.
    - Client will already have rootCA preloaded in its trust store.
    - Client verifies the server's certificate using the crypto function of the OpenSSL package and sends the server "Server Certificate Verification is done".
    - If the server completes the client certificate verification it would send "Client Certificate Verification is done". If it didn't receive that message it means that the certificate is invalid and the connection breaks as mutual auth failed.
    - This is the last step of the handshake. From here on the entire communication is encrypted and secure.
- Once TLS is established, the client and server can chat and send whatever message they want to.

- **Chatting without TLS Connection:**
    - If the client first sends "*chat_STARTNOTLS*" then server and client can chat without a TLS connection.
- **Closing the Connection:**
    - Either the server or client can send "*chat_close*" to close the connection.

Implementation of Server:

- To run the server application, we need to give the port number it wants to connect to.
- Using the above arguments server will create the socket and connect to it.
- **Establishing TCP Connection:**
    - When the client sends "*chat_hello*" the server sends "*chat_reply*" as response to it. If it receives any message other than that the TCP Handshake fails.
    - The messages that are exchanged while establishing TCP connection are in plain text format.
- **Establishing TLS Connection:**
    - When the incoming message is "*chat_STARTTLS*" it means that the client wants to open a TLS connection, so the server will send "*chat_STARTTLS_ACK*" acknowledging it.
    - Next the client will send "*client_hello*", to which server sends "*server_hello*".
    - Until here all the messages are exchanged in plain text format.
    - Once "*client_hello*" and "*server_hello*" exchange is done, certificate exchange takes place.
    - Client will ask server's crt, key file. Similarly the server will ask for the client's crt, key file.
    - Server will already have rootCA preloaded in its trust store.
    - Server verifies the client's certificate using the crypto function of the OpenSSL package and sends the client "*Client Certificate Verification is done*".
    - Server would receive "*Client Certificate Verification is done*" once client does the certificate verification of client. If it didn't receive that message it means that the certificate is invalid and the connection breaks as mutual auth failed.
    - This is the last step of the handshake. From here on the entire communication is encrypted and secure.
    - Once TLS is established, the client and server can chat and send whatever message they want to.
- Once TLS is established, the client and server can chat and send whatever message they want to.
- **Chatting without TLS Connection:**
    - If the client first sends "*chat_STARTNOTLS*" then the server will send "*chat_STARTTLS_NOT_SUPPORTED*"and the server can chat without a TLS connection.
- **Closing the Connection:**
    - Either the server or client can send "*chat_close*" to close the connection.

Chat Snippets of "chat_STARTTLS":

```
root@alice1:~/securechat# python3 secure_chat.py -c bob1 3000
Alice> Connected to bob1
Alice> Sending chat_hello
Alice> chat_STARTTLS
Alice> Sending client_hello
Alice> server_hello recieved, verfying certificates
Alice> Client Certificate Verification is done
Alice> Handshake is completed
====================== Chat Feature Activated ======================
Alice> Hi Bob!
Bob>  Hello Alice!
Alice> chat_close
Alice closed the chat
root@alice1:~/securechat#
```

Fig: Alice sends messages to Bob through a secure channel constructing a TLS pipe.

```
root@bob1:~/securechat# python3 secure_chat.py -s 3000
Bob> Started listening
Bob> Recieved chat_hello
Bob> Sending chat_reply
Bob> Connected to alice1
Bob> Recieved client_hello
Bob> Sending server_hello
Bob> client_hello recieved, verfying certificates
Bob> Server Certificate Verification is done
Bob> Handshake is completed
====================== Chat Feature Activated ======================
Alice>  Hi Bob!
Bob> Hello Alice!
Alice closed the chat
root@bob1:~/securechat#
```

Fig: Bob also communicates with Alice through a secure channel(TLS pipe established).

PCAP Capture Analysis:

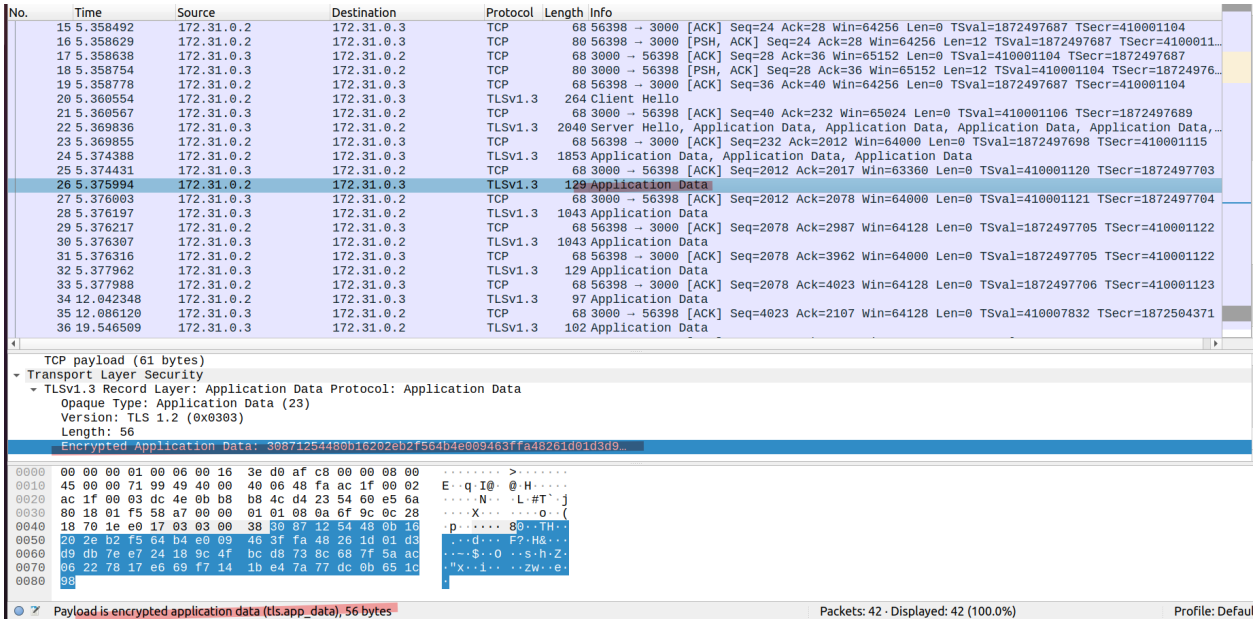


Fig: The TLS handshake has been established correctly. All the messages exchanged between Alice and Bob from chat_STARTTLS is encrypted

Chat Snippets of "chat_STARTNOTLS":

```
root@alice1:~/securechat# python3 secure_chat.py -c bob1 3000
Alice> Connected to bob1
Alice> Sending chat_hello
Alice> chat_STARTTLS
Alice> Sending client_hello
Alice> server_hello recieved, verfying certificates
Alice> Client Certificate Verification is done
Alice> Handshake is completed
======================== Chat Feature Activated ========================
Alice> Hi Bob
Bob>  Hello Alice!
Alice> chat_close
Alice closed the chat
root@alice1:~/securechat# python3 secure_chat.py -c bob1 3000
Alice> Connected to bob1
Alice> Sending chat_hello
Alice> chat_STARTNOTLS
Alice> TLS Connection is not estiblished. Chat is not secure!!
======================== Chat Feature Activated ========================
Alice> Hi Bob!
Bob>  Hello Alice!
Alice> chat_close
Alice closed the chat
root@alice1:~/securechat#
```

Fig: Alice sends messages to Bob through an insecure channel without constructing a TLS pipe.

```
root@bob1:~/securechat# python3 secure_chat.py -s 3000
Bob> Started listening
Bob> Recieved chat_hello
Bob> Sending chat_reply
Bob> Connected to alice1
Bob> Recieved client_hello
Bob> Sending server_hello
Bob> client_hello recieved, verfying certificates
Bob> Server Certificate Verification is done
Bob> Handshake is completed
======================== Chat Feature Activated ========================
Alice>  Hi Bob
Bob> Hello Alice!
Alice closed the chat
root@bob1:~/securechat# python3 secure_chat.py -s 3000
Bob> Started listening
Bob> Recieved chat_hello
Bob> Sending chat_reply
Bob> Connected to alice1
Bob> TLS Connection is not estiblished. Chat is not secure!!
======================== Chat Feature Activated ========================
Alice>  Hi Bob!
Bob> Hello Alice!
Alice closed the chat
root@bob1:~/securechat#
```

Fig: Bob also communicates with Alice through an insecure channel(no TLS pipe established).

PCAP Captures Analysis:

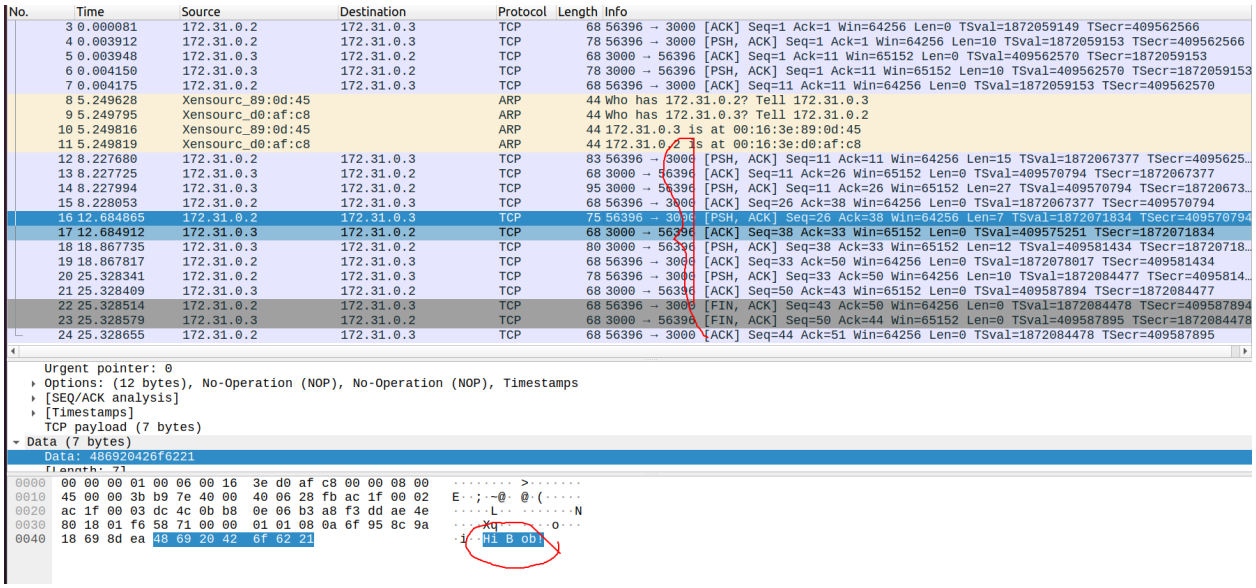


Fig: TLS pipe has not been established , all the conversations between Alice and Bob are unencrypted and can be clearly read.

Task 3: STARTTLS downgrade attack

For this task we should implement a downgrade attack. Here Trudy will act as Man In The Middle(MiTM). First we poison the host files using the bash script.

Implementation of Trudy:

- To run the trudy for downgrade attack, we need to give server name and client name as arguments.
- Using the above arguments Trudy will create two sockets. One for listening to the client and the other for listening to the server.
- Basically Trudy is playing two roles.
- **Listening to Client:**
    - It keeps listening on the server's socket.
    - When a message is received from the server it checks for default messages like "*chat_hello*", "*chat_close*".
    - If "*chat_STARTTLS_NOT_SUPPORTED*" is received then chat without TLS is opened.
    - Other messages are sent unmodified to the client.
- **Listening to Server:**
    - It keeps listening on the client's socket.
    - When a message is received from the server it checks for default messages like "*chat_hello*", "*chat_STARTTLS*", "*chat_close*".
    - If "*chat_STARTTLS*" is detected then it sends "*chat_STARTTLS_NOT_SUPPORTED*" as response to downgrade.
    - Other messages are sent unmodified to the client.

Chat Snippets:

```
ns@ns09:~$ lxc exec alice1 bash
root@alice1:~# cd securechat/
root@alice1:~/securechat# python3 secure_chat.py -c bob1 3000
Alice> Connected to bob1
Alice> Sending chat_hello
Alice> chat_STARTTLS
TLS Connection is not estiblished. Chat is not secure!!
====================== Chat Feature Activated ======================
Alice> Hi
Bob>  Hello
Alice> chat_close
Alice closed the chat
root@alice1:~/securechat#
```

Fig: Alice's side of communication.

```
ns@ns09:~$ lxc exec bob1 bash
root@bob1:~# cd securechat/
root@bob1:~/securechat# python3 secure_chat.py -s 3000
Bob> Started listening
Bob> Recieved chat_hello
Bob> Sending chat_reply
Bob> Connected to alice1
TLS Connection is not estiblished. Chat is not secure!!
======================== Chat Feature Activated ========================
Alice>  Hi
Bob> Hello
Alice closed the chat
root@bob1:~/securechat#
```

Fig: Bob side of communication.

```
ns@ns09:~$ lxc exec trudy1 bash
root@trudy1:~# python3 secure_chat_interceptor.py -d alice1 bob1 3000
Launching downgrade attack!
Trudy> Started listening
======================== Intercepting chat between bob1 and alice1 ========================
Alice>  chat_hello
Bob>  chat_reply
Alice>  chat_STARTTLS
Trudy> Detected chat_STARTTLS and downgraded it to NO TLS
Bob>  chat_STARTTLS_NOT_SUPPORTED
Alice>  Hi
Bob>  Hello
Alice closed the chat
root@trudy1:~#
```

Fig: Trudy Intercepting and launching Downgrade Attack.

PCAP Wireshark Analysis:

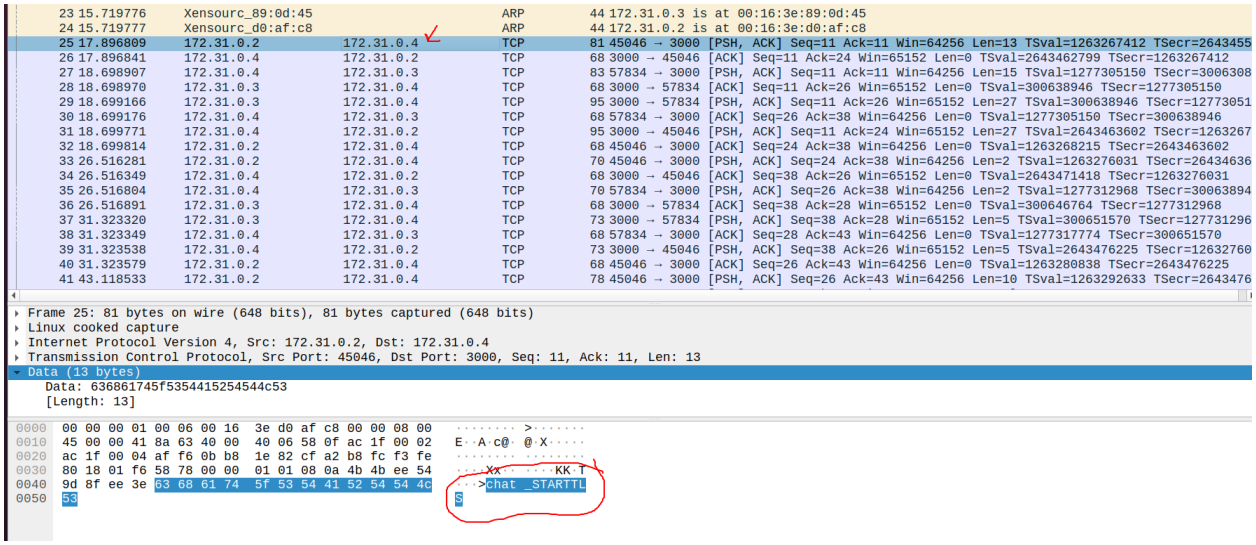


Fig:Alice sends *chat_STARTTLS* to Bob(intercepted by Trudy : IP address 172.31.0.4)

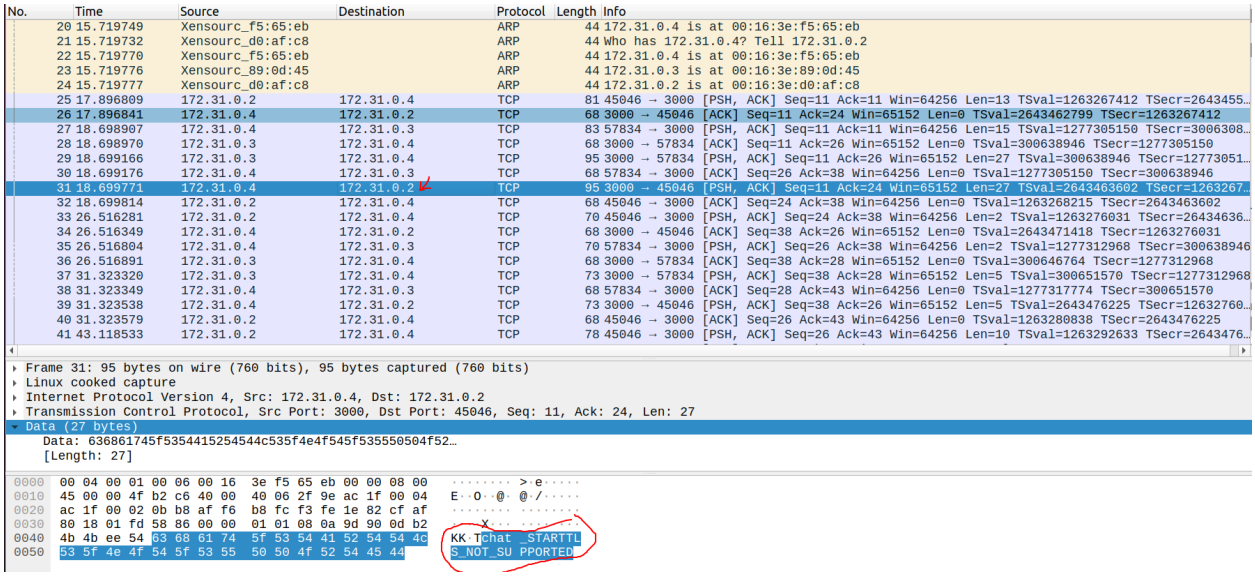


Fig: Trudy sends *chat_STARTTLS_NOT_SUPPORTED* to Alice and thereby forcing Alice and Bob to have unsecure chat communication for successfully intercepting their communication.

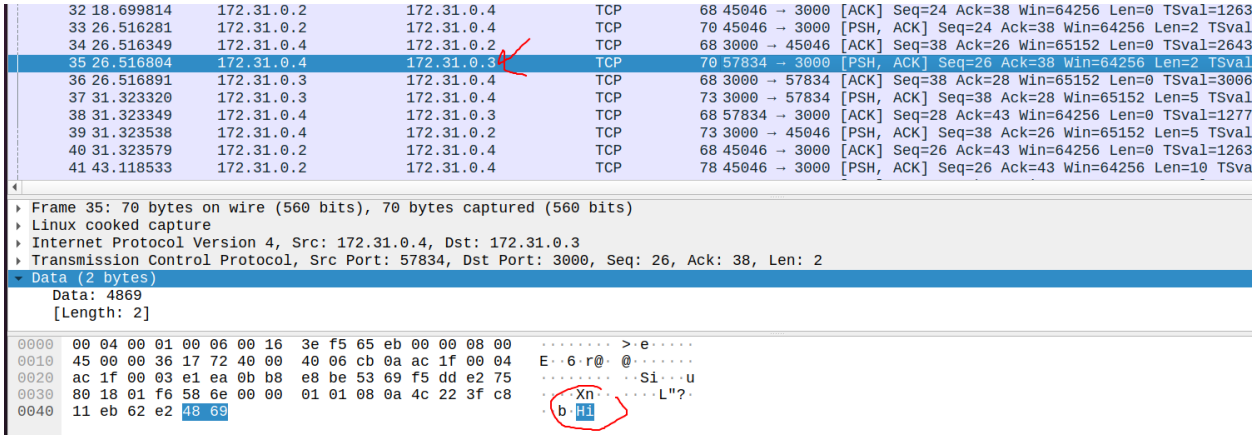


Fig: Insecure unencrypted communication , Trudy forwarding message to Bob from Alice.

Task 4:

Creating fakealice, fake bob crts, key files:

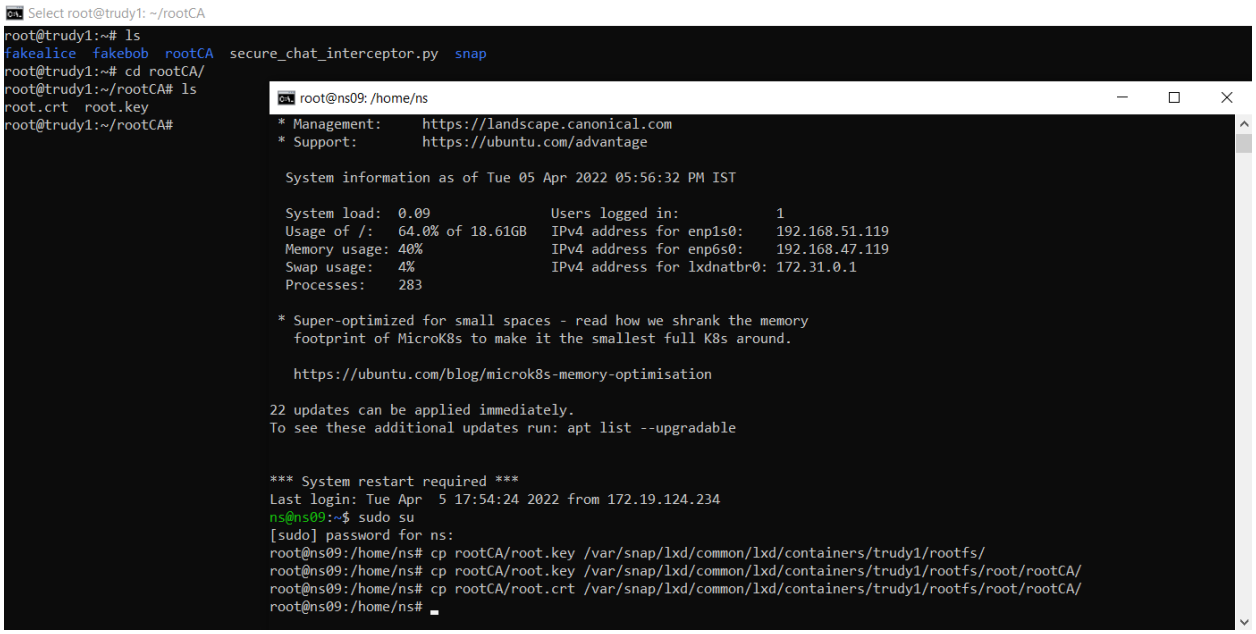
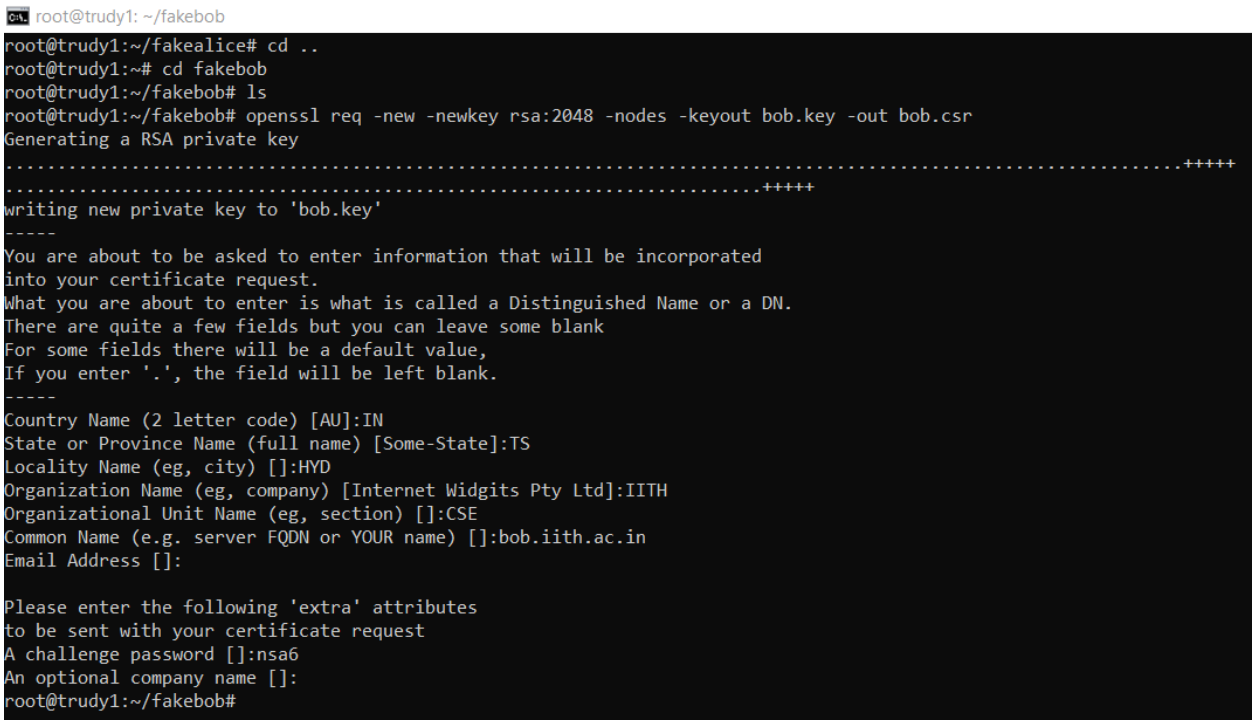


Fig: Trudy hacking RootCA and copying it's certificate and key

a. Bob : New private key and CSR



b. Alice: New Private key and CSR

root@trudy1: ~/fakealice

```
root@trudy1:~# ls
fakealice  fakebob  rootCA  secure_chat_interceptor.py  snap
root@trudy1:~# cd rootCA/
root@trudy1:~/rootCA# ls
root.crt  root.key
root@trudy1:~/rootCA# cd ..
root@trudy1:~# cd fakealice/
root@trudy1:~/fakealice# openssl req -new -newkey rsa:2048 -nodes -keyout alice.key -out alice.csr
Generating a RSA private key
.....................................................................................................+++++
......+++++
writing new private key to 'alice.key'
-----
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]:IN
State or Province Name (full name) [Some-State]:TS
Locality Name (eg, city) []:HYD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:alice.iith.ac.in
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:nsa6
An optional company name []:
root@trudy1:~/fakealice#
```

Trudy issuing the CSR's of Alice and Bob by hacking Root CA:

```
ns@ns09:~$ lxc exec trudy1 bash
root@trudy1:~# ls
fake_alice.crt  fake_bob.crt  fakealice  fakebob  rootCA  secure_chat_interceptor.py  snap
root@trudy1:~# openssl x509 -req -days 365 -in fakebob/bob.csr -CA rootCA/root.crt -CAkey rootCA/root.key -CAcreateserial -out fake_bob.crt
Signature ok
subject=C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = bob.iith.ac.in
Getting CA Private Key
root@trudy1:~# openssl x509 -req -days 365 -in fakealice/alice.csr -CA rootCA/root.crt -CAkey rootCA/root.key -CAcreateserial -out fakealice/
fake_alice.crt
Signature ok
subject=C = IN, ST = TS, L = HYD, O = IITH, OU = CSE, CN = alice.iith.ac.in
Getting CA Private Key
root@trudy1:~#
```

Verifying them :

Check md5 checksums of the certificate and key; the checksums can be compared to verify that the certificate and key match:

```
root@trudy1:~/fakealice# openssl x509 -noout -modulus -in fake_alice.crt| openssl md5 >a
root@trudy1:~/fakealice# openssl rsa -noout -modulus -in alice.key| openssl md5 >b
root@trudy1:~/fakealice# diff a b
root@trudy1:~/fakealice#
```

```
root@trudy1:~/fakebob# openssl x509 -noout -modulus -in fake_bob.crt| openssl md5 >a
root@trudy1:~/fakebob# openssl rsa -noout -modulus -in bob.key| openssl md5 >b
root@trudy1:~/fakebob# diff a b
root@trudy1:~/fakebob#
```

For this task we should implement an Active MiTM attack. Here Trudy will act as Man In The Middle(MiTM). First we poison the host files using the bash script.

Implementation of Trudy:

- Trudy will hack the rootCA and create fake certificates for client and server. It uses the same commands it used for creating crt, key files in task 1.
- To run the trudy for MiTM attack, we need to give server name and client name as arguments.
- Using the above arguments Trudy will create two sockets. One for listening to the client and the other for listening to the server.
- Basically Trudy is playing two roles by establishing two TLS pipes.
- **Listening to Client:**
    - It keeps listening on the server's socket.
    - When a client sends "*chat_STARTTLS*" message it will receive ack "*chat_STARTTLS_ACK*" to Alice and create an SSL socket over the existing connection with "fakealice.crt".
    - This way it creates a TLS connection between client and Trudy.
    - Other messages are sent unmodified to the client.
- **Listening to Server:**
    - It keeps listening on the client's socket.
    - When it detects a "*chat_STARTTLS*" message from the client it will send ack "*chat_STARTTLS_ACK*" to Alice and create an SSL socket over the existing connection with "fakebob.crt".
    - This way it creates a TLS connection between client and Trudy.
    - After the two individual connections are established with client and server by Trudy from now Trudy can modify the messages coming from client and server.

Chat Snippets:

```
root@alice1:~/securechat# python3 secure_chat.py -c bob1 3000
Alice> Connected to bob1
Alice> Sending chat_hello
Alice> chat_STARTTLS
Alice> Sending client_hello
Alice> server_hello recieved, verfying certificates
Alice> Client Certificate Verification is done
Alice> Handshake is completed
====================== Chat Feature Activated ======================
Alice> Hi Bob!
Bob>  Ok
Alice> chat_close
Alice closed the chat
root@alice1:~/securechat#
```

Fig: Alice sends "Hi Bob!"

```
root@trudy1:~# python3 secure_chat_interceptor.py -m alice1 bob1 3000
Trudy> Started listening
====================== Intercepting chat between bob1 and alice1 ======================
Alice>  chat_hello
Bob>  chat_reply
Alice>  chat_STARTTLS
Trudy> Detected chat_STARTTLS and sending fake certificates of Bob
Bob> Recieved client_hello
Bob> Sending server_hello
Alice> Sending client_hello
Alice> server_hello recieved, verfying certificates
Alice>  Server Certificate Verification is done
Bob>  Client Certificate Verification is done
Alice>  Hi Bob!
Do you want to modify the message Trudy (Y/N)? Y
Modified Message: Bye Bob!
Bob>  Send me the file
Do you want to modify the message Trudy (Y/N)? Y
Modified Message: Ok
Alice closed the chat
root@trudy1:~#
```

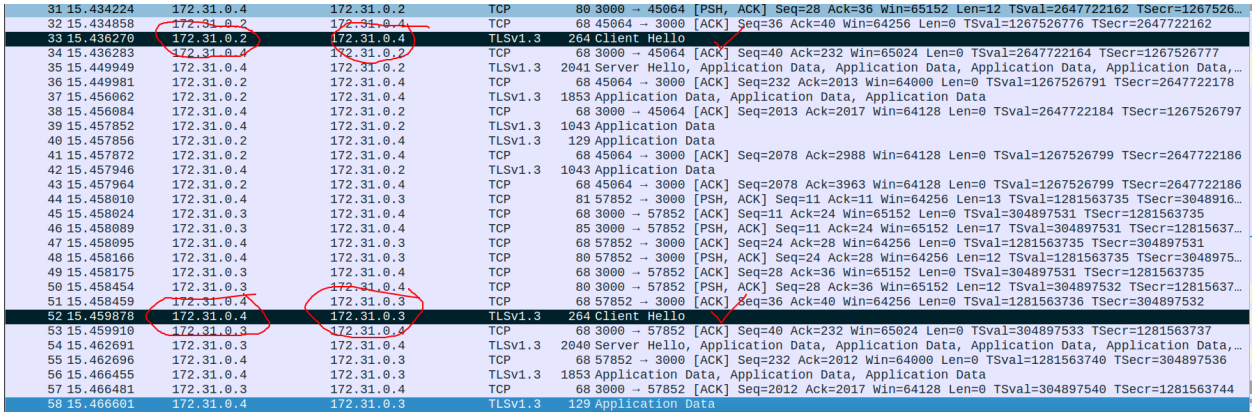
Fig: Trudy Intercepts , modifies “Hi Bob!” to “Bye Bob!”

```
root@bob1:~/securechat# python3 secure_chat.py -s 3000
Bob> Started listening
Bob> Recieved chat_hello
Bob> Sending chat_reply
Bob> Connected to alice1
Bob> Recieved client_hello
Bob> Sending server_hello
Bob> client_hello recieved, verfying certificates
Bob> Server Certificate Verification is done
Bob> Handshake is completed
====================== Chat Feature Activated ======================
Alice>  Bye Bob!
Bob> Send me the file
Alice closed the chat
root@bob1:~/securechat#
```

Fig: Trudy sends Bob “Bye Bob!”

Similarly Bob sends Alice “Send me the file” , which Trudy modifies and sends “Ok” to Alice . Hence MiTM successfully launched.

PCAP Wireshark Analysis:



31	15.434224	172.31.0.4	172.31.0.2	TCP	80	3000 → 45064 [PSH, ACK] Seq=28 Ack=36 Win=65152 Len=12 TSval=2647722162 TSecr=1267526…
32	15.434858	172.31.0.2	172.31.0.4	TCP	68	45064 → 3000 [ACK] Seq=36 Ack=40 Win=64256 Len=0 TSval=1267526776 TSecr=2647722162
33	15.436270	172.31.0.2	172.31.0.4	TLSv1.3	264	Client Hello
34	15.436283	172.31.0.4	172.31.0.2	TCP	68	3000 → 45064 [ACK] Seq=40 Ack=232 Win=65024 Len=0 TSval=2647722164 TSecr=1267526777
35	15.449949	172.31.0.4	172.31.0.2	TLSv1.3	2041	Server Hello, Application Data, Application Data, Application Data, Application Data,…
36	15.449981	172.31.0.2	172.31.0.4	TCP	68	45064 → 3000 [ACK] Seq=232 Ack=2013 Win=64000 Len=0 TSval=1267526791 TSecr=2647722178
37	15.456062	172.31.0.2	172.31.0.4	TLSv1.3	1853	Application Data, Application Data, Application Data
38	15.456084	172.31.0.4	172.31.0.2	TCP	68	3000 → 45064 [ACK] Seq=2013 Ack=2017 Win=64128 Len=0 TSval=2647722184 TSecr=1267526797
39	15.457852	172.31.0.4	172.31.0.2	TLSv1.3	1043	Application Data
40	15.457856	172.31.0.2	172.31.0.4	TLSv1.3	129	Application Data
41	15.457872	172.31.0.2	172.31.0.4	TCP	68	45064 → 3000 [ACK] Seq=2078 Ack=2988 Win=64128 Len=0 TSval=1267526799 TSecr=2647722186
42	15.457946	172.31.0.4	172.31.0.2	TLSv1.3	1043	Application Data
43	15.457964	172.31.0.2	172.31.0.4	TCP	68	45064 → 3000 [ACK] Seq=2078 Ack=3963 Win=64128 Len=0 TSval=1267526799 TSecr=2647722186
44	15.458010	172.31.0.4	172.31.0.3	TCP	81	57852 → 3000 [PSH, ACK] Seq=11 Ack=11 Win=64256 Len=13 TSval=1281563735 TSecr=3048916…
45	15.458024	172.31.0.3	172.31.0.4	TCP	68	3000 → 57852 [ACK] Seq=11 Ack=24 Win=65152 Len=0 TSval=304897531 TSecr=1281563735
46	15.458089	172.31.0.3	172.31.0.4	TCP	85	3000 → 57852 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=17 TSval=304897531 TSecr=12815637…
47	15.458095	172.31.0.4	172.31.0.3	TCP	68	57852 → 3000 [ACK] Seq=24 Ack=28 Win=64256 Len=0 TSval=1281563735 TSecr=304897531
48	15.458166	172.31.0.4	172.31.0.3	TCP	80	57852 → 3000 [PSH, ACK] Seq=24 Ack=28 Win=64256 Len=12 TSval=1281563735 TSecr=3048975…
49	15.458175	172.31.0.3	172.31.0.4	TCP	68	3000 → 57852 [ACK] Seq=28 Ack=36 Win=65152 Len=0 TSval=304897531 TSecr=1281563735
50	15.458454	172.31.0.3	172.31.0.4	TCP	80	3000 → 57852 [PSH, ACK] Seq=28 Ack=36 Win=65152 Len=12 TSval=304897532 TSecr=12815637…
51	15.458459	172.31.0.4	172.31.0.3	TCP	68	57852 → 3000 [ACK] Seq=36 Ack=40 Win=64256 Len=0 TSval=1281563736 TSecr=304897532
52	15.459878	172.31.0.4	172.31.0.3	TLSv1.3	264	Client Hello
53	15.459910	172.31.0.3	172.31.0.4	TCP	68	3000 → 57852 [ACK] Seq=40 Ack=232 Win=65024 Len=0 TSval=304897533 TSecr=1281563737
54	15.462691	172.31.0.3	172.31.0.4	TLSv1.3	2040	Server Hello, Application Data, Application Data, Application Data, Application Data,…
55	15.462696	172.31.0.4	172.31.0.3	TCP	68	57852 → 3000 [ACK] Seq=232 Ack=2012 Win=64000 Len=0 TSval=1281563740 TSecr=304897536
56	15.466455	172.31.0.4	172.31.0.3	TLSv1.3	1853	Application Data, Application Data, Application Data
57	15.466481	172.31.0.3	172.31.0.4	TCP	68	3000 → 57852 [ACK] Seq=2012 Ack=2017 Win=64128 Len=0 TSval=304897540 TSecr=1281563744
58	15.466601	172.31.0.4	172.31.0.3	TLSv1.3	129	Application Data

Fig: Shows two TLS pipes established between Alice and Trudy , Trudy and Bob.

Credit Statement:

Tasks	**Vinta Reethu**	**Anwesha Kar**	**Madhavendra Singh Chouhan**
Task 1		Certificates creation	
Task 2	Establishing TCP connection & TCP handshake, server and client chat	Establishing TLS Pipe & TLS handshake , server and client chat	load the respective private key and certificate, trust store, verifying certificates, Wireshark analysis
Task 3	Performing Interception with Trudy , Bug Fixing	Creating sockets for Trudy (listen mode) ,Wireshark Capture & Analysis	Bug fixing
Task 4	Launched MiTM attack (b,c) , modified secure_chat_interceptor .py	Fake Certificates creation , Bug Fixing	Wireshark Capture & Analysis , Bug Fixing
Report Writing	Yes	Yes	Yes

README.md file is submitted which contains the instructions to run task 2, 3, 4

PLAGIARISM STATEMENT <Include it in your report>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, 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 plagiarized 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 my responsibility to report honor violations by other students if we become aware of it.

Names: Vinta Reethu, Anwesha Kar, Madhvendra Singh Chouhan
Date: 8-4-2022
Signature: Vinta, Kar, Chouhan