

Assignment 7

Secure chat using openssl and MITM attacks

Kritik Agarwal (CS23MTECH11009)

Naveen Nayak (CS23MTECH11011)

Girish Thatte (CS22MTECH11005)



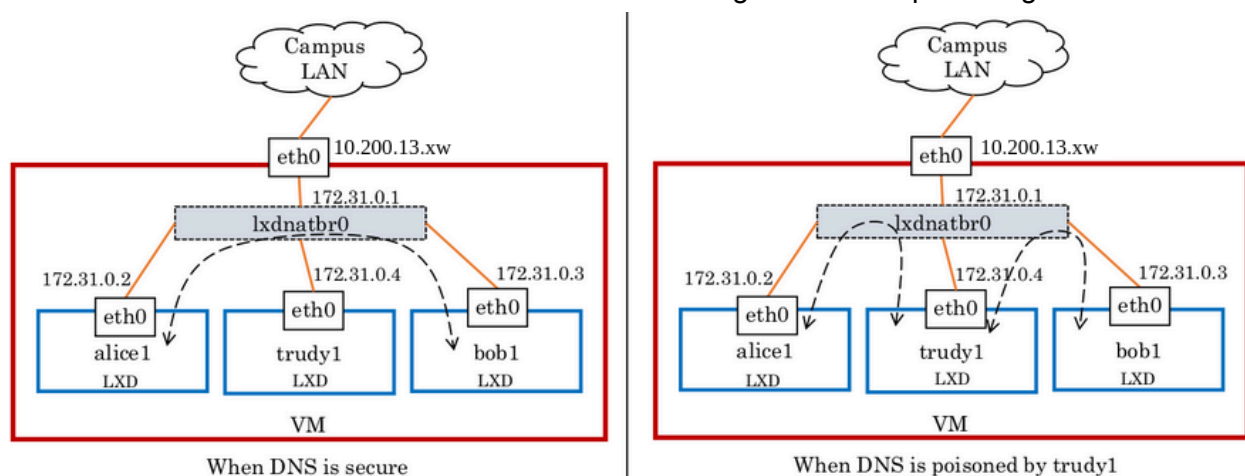
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Introduction :The secure chat application is designed to facilitate secure communication between a client and a server using Datagram Transport Layer Security (DTLS) protocol over User Datagram Protocol (UDP). The application leverages the OpenSSL library to implement secure communication channels and ensure data confidentiality, integrity, and authenticity.

In this programming assignment, your group will implement a secure peer-to-peer chat application using openssl in C/C++ and demonstrate how Alice and Bob could chat with each other using it. Plus you will also implement evil Trudy who is trying to intercept the chat messages between Alice and Bob by launching various MITM attacks.

Setup:

Each group will be given a dedicated QEMU-KVM VM with the IP address (10.200.13.xw) in the CSE dept's over-cloud for completing this assignment. Refer Appendix B for some help on how to work with containers. The VM runs three LXD containers (one LXD container each for Alice, Trudy and Bob in the VM provided by the TAs) which are configured in a star topology i.e., with the switch at the center, Alice, Bob, Trudy are connected to the switch ports. **Make a note of hostnames assigned to Alice, Trudy and Bob from your VM.** DNS is already set up properly so ping using hostnames from Alice to Bob and vice-versa works. Under normal circumstances, Trudy does not come in the traffic forwarding path between Alice and Bob. But when DNS is poisoned by Trudy, all traffic between Alice and Bob is intercepted by Trudy as the MITM attacker. You can also launch a similar MITM attack using ARP cache poisoning.



Please upload ssh public key to your github profile and share your github user-id with TAs to get access to your group's VM through IITH's Wireguard VPN (if you are not on the campus). The three LXDs are reachable from inside the VM, and cannot be reached from the campus LAN.

Task 1: Generate keys and certificates (10M)

Use OpenSSL to create a root CA certificate (Subject name: iTS Root R1, V3 X.509 certificate, self-signed using 512-bit ECC Private Key of the root), an intermediate CA certificate (Subject Name: iTS CA 1R3, V3 X.509 certificate with 4096-bit RSA public key, signed by iTS Root R1), a certificate of Alice (Subject Name: Alice1.com with 1024-bit RSA public key, issued i.e., signed by the intermediate CA, iTS CA 1R3) and a certificate of Bob (Subject Name: Bob1.com with 256-bit ECC public key, issued i.e., signed by the intermediate CA, iTS CA 1R3). Ensure that you provide realistic meta-data while creating these X.509 V3 certificates like values for OU, L, Country, etc of your choice with appropriate key usage/constraints. Save these certificates as root.crt, int.crt, alice.crt and bob.crt, save their CSRs and key-pairs in .pem files and verify that they are valid using openssl. You can complete this task either on the VM provided (recommended) or on your personal machine.

Below are the commands we used to do Task 1 and there use is mention:-

Commands for Creating root CA certificate in VM (remote system)

Generate 512-bit ECC Private Key

\$ openssl ecparam -name brainpoolP512r1 -genkey -noout -out root-ca-private-kepemey.

```

[ubuntu@cs6903-1:~/intermediate$ openssl rsa -in int-ca-private-key.pem -check
RSA key ok
writing RSA key
-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQDjk52DhdL4px8Y
+1vldwjaD/8pmew6/PeIB0r8lyqDJ5X0/7kzN/+f2KmL/ila1L4Et7g8TVkhWVNU
yM2N3xa4I5B2phLE5hsSHAV4HD1KSpMrRS1jP/g28opIUqgz+UW2v3I1J5VFCqh8
Epf4hEEdx9ZVGa4oBkju3+L1bsACU6ELkbbtVX0YbHr3i2cvdjxHWKRtNT3ysVc3
c5lJlmac4760fVZo7zpJ31T2HgGf1JgwupAWTDyU22s55szQqUprffVYT62SKuy1
gaH1sgst003Jz0DDaMDhpkpukU45uzUUpFKB07r2tr1tP6j0MtveuzBUe1BgiSSd
BB4RgNbWfEKBYhcbHvXsfkSWMCMoo2qRMYvTDh3ydKngB1FkEtoZleCk1x6RAJVF
2r1PfL5wk1kZw4X0nmn5rKQA7myQ3c87Tatq01BBZXkpW/B2vK/seumI6rX0msay
+LZGUbqajd1/sAKABpcUd+GyrhYC6UDBo1FWL5pDG+2KLmD+0QXYPZvCqsXfVZwq
A1Mkgk0dhIC1TSPQXf3wQZf+qWbBpkj4buwaA/LJkmp/qAwhu52g9omCxd+/f3
wXt/R2qAVo4NfFFHkp81Qpxadu4LA0EexjxNBEw3K02EqIIgCrXByiRaKXklcIJh
T1yxZRN/uz6pvaYhJvrxDrGQTkqz+QIDAQABaoICACFggy+zfw4t60SC37xR+XqL
RxKcZcZUV0oxkPYRMPsnC/obTq0W5PGrhAOpwQUHjPjsfbkGZRtG2mRrG6QCQ0xC
wbYibvVG2aiNyIrlpWLnI2+Asq1FnWC9sLn2ZCn/QLy9QQGkweSMBsrvoqSYgmOy
Qpqc7MB9BNGtWhWUXSEq3r1E0piQuwtLzxB4TGwXF18laTQvz1lFWN55S1LncT7z
CK4Ns3CCVEEFcR/Fwy/tn2aM2Ni/BKbrtBhmuRpPV0ADAL/wXE+dkQidxcRfAP80
ue6pFr1czeZ/uOyUzhgZ1K1QM1422u8X5S8v8KvOu8A5Iaz6Sv8Q0b6JqjlJ9VEz
1o283iDPj6qEhOqQ7CrCzaYHJqqVqz7+IbdWQao6awSNq2wm6Fhm2Stha/xChraf
4NAjzDYP9I9QyhbGWNqsjq0r9LODVnzmrNS27WUd07TG5y0SjHX0gqQzo05aXBGRg
U/+qDUpzxUSYBk1eI/F2sIZJODU/j3PB5RxA7/5wPaeEkU1WtGa/CPgxd5chY4m
iNNHimHbn3jz9PHnzqxHwP3mEbvtDc/mXW1YG55We93E0QXhJE88x5MG4f8McP4
F/MbxvYcuy6xSNsVhMks9VRMcD/ELsKR4euTuoLfvVD0TTdeFaF1FrJMGSRX3ATZ
/hVaNgviRjKCRxzRP1hLAoIBAQAQspNg3KPG0Fw0Qg57W8Gcfa0vsp2tcwj8Yh+Nk
pVCQY+qoFrzE0PZ5fNf2qm+SVxCUxhI7+Utsx2Tzvdj7YsELNx0Bj/rUR5CokI/L
r0xb4TETFnxax0dRTtr/ffz8tinI6HFa+T8XNghY7uSoCDVelim6/IuM+E00NSnn
eZc4TEEF6KBcAsu+1yZ1i0rA1EAibcA1+xspeuNkMELis1S6JAxhj68SP0LbXGLZ
xUEQsukP1bh9gkvQ8LXgggR4/a6Ns5W/vUQE5kMZsntAuqp3xugMmHEb6BU2GMrQ
B96qE4TZUyJCY9M5rXXA1UylGvKbxQo7dIAmg5X8ywdtLh6bAoIBAQAQD2M0hofKXn
Rf8z0EmKUXNnyvRyCk6+5/D0MpE2q7HisSkHZvcxmV089Cys67aSG+G9HU2NJDb
MIWcWkI6bGx0IN2jLn7WWhtrXy6ZY63wDty+GFaE0Yk3pDk+mi8+J6c9EyJ5kffD
Zux427GJnkPLMP90In7ksyq0BBcIWQ5DdiISbAJSp2tu4XGI6v3k4KxBc/8g7pr
gk00BYS/kSS+VBdpAAuvhMVXPd0igv834JDzF2H4F6Kv4gMgvrSK4SjLJfXcCx+0
5VZzThXUW5KyF3xf18NN6ysZK7hpjLpKV04c0vWF7FAT4AnS3Ne0gAA6SjHNkw0o
YRrBHCIRujb7AoIBAQAQZ+OUnudEY/vqLsZEHjpyONjjTHUspP8r6f4CW3icdOMuW
VItABvJg6PvM0yVV+FHpkYtQ0f4xYADoewUoyE76Y4/mGRHDYQ6sdeTracerwfuA
B7jyhMxG0md8ujm16xZYL65CEhytRZhoKbvCPEr0R09dZc3Pkp7sBQdJEzzB1A
pgu/dNXWkjrcnGcPgvIQUXdwwYE9bfIyALo7Ms/iPP4ih0TNW/PrNmffc3BpP0Kp
11E/Wvqr/eyphVklUQYNg9N1r5INqUM5UktXTf9jIJXxtCDLgbS4BZIAMiUix6Pg
wLCWmNkHcZ067MqScgXyMubBiQfZce8fzywzRv3tAoIBAQAQDs9Cnc14w+ypUtnDz
wAek0TopIS6lpExQXZNS0XKHRAi/bSUjAf+XuTp+v0iePGLR7bq0CF1JgCTenQg
E1cy44Mvm0LnB3MgBnwfGsfbNCzWckpjPZWkjM/ihzhNVJESwYv/Aq8TPQsBUELS
uBd8LI7S0Q1GWObo1uN4KVNzbpYopvPeHYkiPEKHajIM01gh1fxXiNoWu+wVd85L
K4H4Bqv5j3gVmw7vBzz5P6gxud0JfEn5JZaaPqjQDX1r5kDBn6nBvpsDSE0q8c8k
83aAHwbAoMUD7AMxWN1IDgEHpjL/z0efnNF+L7aE07ZNG3ocoGjK+mxBnyTHFTLD
aS0dAoIBAQAQD0F87LSKFLlcY1qz8zvGQ2UluDdlfyEHOafd0gPLYLUot64d3R1DT
H++0NvYaHB3CfXsJj4yk0ZT+eH24PScVtaf0yit+0XBFztMDh5MX/ySi/twPEmAz
d0C0ztX/7dV+k9iTSPrjMvg/Y6v1ft1d6PA2Rde4/QH6/37BpsEBIKCqaNDWK4sA
i9dVZr6+dCmt02FL+859yK+NLMMa3dzuNAIAWHMUBDPfpBim0sT970KXBpD1qBqR
uBSAUp18DhWhR3UAXrH5ZQwXTv1cJ+fFo2k+TXdbfDJApvpvBVAjSYe6NjIyenJq
u1UqQoMaRUXUvBIM2mBn0TE21Y2r3tsD
-----END PRIVATE KEY-----
[ubuntu@cs6903-1:~/intermediate$ ]

```

Generate Public Key from 512-bit ECC Private Key

\$ openssl ec -in root-ca-private-key.pem -pubout -out root-ca-public-key.pem

Create a root.cnf file to specify the Certificate details like Extensions, Key Usages

\$ nano root.cnf

```
[ubuntu@cs6903-1:~/root$ cat root.cnf
[ req ]
default_bits = 4096
prompt = no
default_md = sha256
distinguished_name = dn
x509_extensions = v3_ca

[ dn ]
C=IN
ST=Telangana
L=Hyderabad
O=IIT Hyderabad
OU=CSE@IITH
CN=iTS Root R1

[ v3_ca ]
basicConstraints = critical, CA:TRUE, pathlen:1
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
ubuntu@cs6903-1:~/root$
```

Root CA self signed certificate generation from private key

**\$ openssl req -x509 -new -nodes -key root-ca-private-key.pem -days 1825 -out root.pem
-config root.cnf -extensions v3_ca**

Viewing the Root CA self-signed certificate

\$ openssl x509 -in root.pem -text -noout


```

[ubuntu@cs6983-1:~/root$ openssl x509 -in root.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            4b:1c:d5:06:0b:ff:0f:88:2e:ec:05:03:6f:fd:32:d0:f8:61:0c:18
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE@IITH, CN = iTS Root R1
        Validity
            Not Before: Mar  7 08:52:07 2024 GMT
            Not After : Mar  6 08:52:07 2029 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE@IITH, CN = iTS Root R1
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:70:bb:10:89:75:0f:6e:47:82:53:c5:ec:5b:2c:
                    20:77:15:dd:0b:c2:00:80:a0:b8:6f:c6:e1:43:89:
                    68:b6:ec:0e:57:e9:32:f1:0c:16:6e:8c:f8:e9:b9:
                    81:8b:fa:68:c8:87:99:d5:4b:80:27:91:10:15:16:
                    dc:07:85:8c:0e:24:95:5b:21:5d:e5:eb:ca:7a:29:
                    55:f8:4d:89:fc:f8:82:4a:24:03:b6:69:ae:3e:ca:
                    ee:ec:b7:02:7c:d9:62:72:5f:83:e2:1a:98:69:7a:
                    77:7d:8b:fc:a6:d7:3c:66:9c:50:c0:a4:60:25:83:
                    4a:da:0e:a7:f0:aa:d6:8d:94
                ASN1 OID: brainpoolP512r1
            X509v3 extensions:
                X509v3 Basic Constraints: critical
                    CA:TRUE, pathlen:1
                X509v3 Key Usage: critical
                    Digital Signature, Certificate Sign, CRL Sign
                X509v3 Subject Key Identifier:
                    D2:1E:37:AF:4B:C7:46:C6:B0:86:04:01:4C:42:46:DF:D0:08:7D:73
        Signature Algorithm: ecdsa-with-SHA256
        Signature Value:
            30:81:84:02:40:60:e1:82:30:25:04:35:8b:7a:a6:5f:99:bf:
            db:6f:00:a3:bd:cc:24:cc:44:cd:b2:82:03:0d:99:a9:2c:61:
            f8:7d:5a:d7:53:77:01:5d:a9:47:88:1b:4c:d3:29:55:f5:d0:
            34:be:7d:db:72:76:8c:8a:4d:6f:e1:39:9b:b8:8d:02:40:4f:
            10:91:36:9b:34:c7:67:31:d5:db:e2:92:05:76:80:ba:5a:fe:
            5c:2a:c1:27:cd:8b:8a:c5:c8:da:9b:01:ff:3f:49:a3:84:15:
            4a:af:74:93:3b:ca:d9:21:ef:0f:ea:29:57:e7:fc:76:da:c3:
            b7:e9:29:b5:e0:1b:f7:77:e3
ubuntu@cs6983-1:~/root$

```

Commands For Creating Intermediate CA certificate

Intermediate CA private key generation

\$ openssl genpkey -algorithm RSA -out int-ca-private-key.pem -pkeyopt rsa_keygen_bits:4096

```

[ubuntu@cs6983-1:~$ openssl genpkey -algorithm RSA -out int-ca-private-key.pem -pkeyopt rsa_keygen_bits:4096
.....
[ubuntu@cs6983-1:~$

```

Intermediate CA public key generation from private key

\$ openssl rsa -in int-ca-private-key.pem -pubout -out int-ca-public-key.pem

```
ubuntu@cs6903-1:~$ openssl rsa -in int-ca-private-key.pem -pubout -out int-ca-public-key.pem
writing RSA key
ubuntu@cs6903-1:~$
```

Create a int.cnf file to specify the Certificate details like Extensions, Key Usages

\$ nano int.cnf

CSR by Intermediate CA certificate that must be signed by the root CA

\$ openssl req -config int.cnf -new -sha256 -key int-ca-private-key.pem -out int.csr -extensions v3_ca

CSR verified and signed by Root CA

\$ openssl x509 -req -in int.csr -CA root.pem -CAkey root-ca-private-key.pem -CAcreateserial -out int.pem -days 365 -extensions v3_ca -extfile int.cnf

```
ubuntu@cs6903-1:~/root$ openssl x509 -req -in int.csr -CA root.crt -CAkey root-ca-private-key.pem -CAcreateserial -out int.crt -days 365 -extensions v3_ca -extfile int.cnf
Certificate request self-signature ok
subject=C = IN, ST = Maharashtra, L = Mumbai, O = IIT Bombay, OU = CSE@IITB, CN = ITS CA 1R3
ubuntu@cs6903-1:~/root$
```

View the Intermediate CA certificate

\$ openssl x509 -in int.pem -text -noout Alice

```

[ubuntu@cs6903-1:~/root$ openssl x509 -in int.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            15:02:24:11:af:ae:f4:0f:cf:af:25:47:1e:1c:ee:69:e8:aa:84:85
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE@IITH, CN = iTS Root R1
        Validity
            Not Before: Mar  7 09:24:11 2024 GMT
            Not After : Mar  7 09:24:11 2025 GMT
        Subject: C = IN, ST = Maharashtra, L = Mumbai, O = IIT Bombay, OU = CSE@IITB, CN = iTS CA 1R3
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (4096 bit)
            Modulus:
                00:e3:93:9d:83:85:d2:f8:a7:1f:18:fb:5b:e5:77:
                08:da:0f:ff:29:99:ec:3a:fc:f7:88:07:4a:fc:97:
                2a:83:27:95:f4:ff:b9:33:37:ff:9f:d8:a9:8b:fe:
                29:5a:d4:be:04:b7:b8:3c:4d:59:21:59:53:54:c8:
                cd:8d:df:16:b8:23:90:76:a6:12:c4:e6:1b:12:1c:
                05:78:1c:3d:4a:4a:93:2b:45:2d:63:3f:f8:36:f2:
                8a:48:52:a8:33:f9:45:b6:bf:72:35:25:5e:45:0a:
                a8:7c:12:97:f8:84:41:1d:c7:d6:55:19:ae:28:06:
                48:ee:df:e2:f5:6e:c0:02:53:a1:0b:91:b6:ed:55:
                7d:18:6c:7a:f7:8b:67:2f:76:3c:47:58:a4:6d:35:
                3d:f2:b1:57:37:73:99:49:96:66:9c:e3:be:b4:7d:
                56:68:ef:3a:49:df:54:f6:1e:01:9f:94:98:30:ba:
                90:16:4c:3c:94:db:6b:39:e6:cc:d0:a9:4a:6b:7d:
                f5:58:4c:6d:92:2a:ec:a5:81:a1:f5:b2:0b:2d:d0:
                ed:c9:cf:40:c3:68:c0:e1:a6:4a:6e:91:4e:39:bb:
                35:14:a4:52:81:3b:ba:f6:b6:bd:6d:3f:a8:f4:32:
                db:de:bb:30:54:7b:50:60:89:2b:03:04:1e:11:80:
                d6:f0:15:e2:81:62:17:1b:1e:f5:ec:7e:44:96:30:
                23:28:a3:6a:91:31:8b:d3:0e:1d:f2:74:a9:e0:06:
                51:64:12:da:19:95:e0:a4:97:1e:91:00:95:45:da:
                b9:4f:7c:be:70:93:59:19:c3:85:ce:9e:69:f9:ac:
                a4:00:ee:6c:90:dd:cf:3b:4d:ab:6a:3b:50:41:65:
                79:29:c3:f0:76:bc:af:ec:7a:e9:88:ea:b5:f4:9a:
                c6:b2:f8:b6:46:51:ba:9a:8d:dd:7f:b0:02:80:06:
                97:14:77:e1:b2:ae:16:02:e9:40:c1:a3:51:56:2f:
                9a:43:1b:ed:8a:2e:60:fe:d1:05:d8:3d:9b:c2:aa:
                c5:df:55:9c:2a:02:53:24:80:ad:1d:84:62:02:d4:
                84:8f:42:75:df:df:04:19:7f:ea:96:6c:1a:64:8f:
                86:ee:c1:a0:3f:94:99:26:a7:fa:80:c2:1b:b9:da:
                0f:68:98:2c:5d:fb:f7:f7:c1:7b:7f:47:6a:80:56:
                8e:0d:7c:51:61:92:9f:35:42:9c:5a:76:ee:0b:00:
                e1:1e:c6:3c:4d:04:4c:37:2b:4d:84:a8:82:20:0a:
                b5:c1:ca:24:5a:29:79:25:70:82:61:4f:5c:b1:65:
                13:7f:ba:be:a9:bc:06:21:26:fa:f1:0e:b1:90:4e:
                4a:b3:f9
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                E5:66:80:CC:B6:3A:48:61:56:85:51:35:62:DC:97:2A:DB:61:5C:35
            X509v3 Authority Key Identifier:
                41:8F:7E:D6:B4:37:AC:1E:1D:F5:BD:0E:5C:C9:6C:FB:BD:4E:40:C2
        Signature Algorithm: ecdsa-with-SHA256
        Signature Value:
            30:81:85:02:40:0e:6c:e0:bd:35:ea:a2:2a:15:22:57:de:5f:
            f7:68:ba:86:c3:69:09:93:c2:5d:df:dc:56:5a:d3:0b:e9:d5:
            be:cc:bb:73:b8:f9:54:9f:4c:e1:0e:02:7d:fe:76:22:dc:a0:
            32:94:d8:13:e8:b3:17:ee:1f:7c:5f:46:12:5d:e2:02:41:00:
            88:5d:2a:8f:64:41:07:56:69:7a:f4:37:f2:20:fc:f0:b7:15:
            9a:5e:41:23:12:35:a9:ad:d7:93:9c:fb:7c:2a:9a:fc:6a:2b:
            47:11:4f:4c:78:57:56:b2:42:19:a2:20:14:23:1c:54:c2:74:
            cf:25:89:a7:b4:bc:a5:a1:35:55
    
```

Generate 1024-bit RSA private key

```
$ openssl genpkey -algorithm RSA -out alice-private-key.pem -pkeyopt
rsa_keygen_bits:1024
```

Commands For Creating Alice certificate

Generate public key

```
$ openssl rsa -in alice-private-key.pem -pubout -out alice-public-key.pem
```



```
ubuntu@cs6903-1:~/alice1$ openssl genpkey -algorithm RSA -out alice-private-key.pem -pkeyopt rsa_keygen_bits:1024
...+++++
.....+++++
ubuntu@cs6903-1:~/alice1$
```

Create a alice.cnf file to specify the Certificate details like Extensions, Key Usages

\$ nano alice.cnf

CSR by Alice that must be signed by the Intermediate CA

\$ openssl req -config alice.cnf -new -sha256 -key alice-private-key.pem -out alice.csr -extensions v3_req

CSR verified and signed by Intermediate CA

\$ openssl x509 -req -in alice.csr -CA int.pem -CAkey int-ca-private-key.pem -CAcreateserial -out alice.pem -days 90 -extensions v3_req -extfile alice.cnf

Viewing the Certificate

\$ openssl x509 -in alice.pem -text -noout

// add screenshot of alice certificate

Commands For Creating Bob certificate

Generate 256-bit ECC private key

\$ openssl ecparam -name brainpoolP256r1 -genkey -noout -out bob-private-key.pem

Generate public key

\$ openssl ec -in bob-private-key.pem -pubout -out bob-public-key.pem

Create a bob.cnf file to specify the Certificate details like Extensions, Key Usages

\$ nano bob.cnf

CSR by Alice that must be signed by the Intermediate CA

\$ openssl req -config bob.cnf -new -sha256 -key bob-private-key.pem -out bob.csr -extensions v3_req

CSR verified and signed by Intermediate CA

\$ openssl x509 -req -in bob.csr -CA int.pem -CAkey int-ca-private-key.pem -CAcreateserial -out bob.pem -days 90 -extensions v3_req -extfile bob.cnf

Viewing the Certificate

\$ openssl x509 -in bob.pem -text -noout

```
ubuntu@cs6903-1:~/intermediate$ openssl x509 -in bob.crt -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      09:74:65:0e:97:bc:dc:c1:a6:a6:17:58:e4:9c:4c:23:5e:70:c5:f4
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = IN, ST = Maharashtra, L = Mumbai, O = IIT Bombay, OU = CSE@IITB, CN = iTS CA 1R3
    Validity
      Not Before: Mar  7 09:47:28 2024 GMT
      Not After : Jun  5 09:47:28 2024 GMT
    Subject: C = IN, ST = New Delhi, L = Delhi, O = IIT Delhi, OU = CSE@IITD, CN = Bob1.com
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
      Public-Key: (256 bit)
      pub:
        04:5b:46:b2:f4:3b:0b:54:37:a3:b0:c6:29:d4:a2:
        2a:18:d2:70:be:df:b5:71:7c:7a:4b:42:4d:3d:98:
        d1:f0:8f:63:2a:37:e1:88:c6:e9:ab:e1:f3:10:df:
        c6:55:ef:9d:98:74:5b:d5:13:f8:c2:b9:c2:c4:e0:
        bf:4c:87:c7:ef
      ASN1 OID: brainpoolP256r1
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Key Usage:
        Digital Signature, Key Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
      X509v3 Subject Key Identifier:
        B1:BC:38:ED:72:B1:0B:11:DA:B1:0C:DE:48:86:67:27:2B:B4:35:FF
      X509v3 Authority Key Identifier:
        E5:66:80:CC:B6:3A:48:61:56:85:51:35:62:DC:97:2A:DB:61:5C:35
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
      29:ec:e3:b5:ff:65:49:d2:cd:36:ca:1f:bd:5c:ee:11:80:fa:
      eb:d5:b0:56:07:d7:02:cd:0a:dc:09:ff:3b:bb:08:c4:80:5d:
      24:15:1c:0c:81:44:de:5a:45:1b:77:d0:75:b8:bc:33:7d:6a:
      b0:32:31:04:10:e0:cd:45:57:54:dc:90:c8:f2:0f:08:6f:66:
      b5:ba:2c:56:3c:14:61:4f:2f:4c:83:2d:8b:cd:f6:ed:ee:2f:
      37:f1:36:b7:00:f7:53:b9:7e:ee:fc:72:ba:5d:b0:d1:75:84:
      92:8f:38:95:ac:a7:18:05:f7:6a:29:7b:e2:62:dc:92:d3:df:
      2c:bf:74:a1:90:be:c3:40:f4:26:a2:e1:7f:d0:40:46:dc:c1:
      5c:42:3d:fb:74:c0:af:4d:5a:63:be:3e:6e:b2:05:8d:59:db:
      4f:60:f3:12:b8:a2:3a:58:11:f0:a5:10:d7:8e:ae:c4:0d:20:
      99:f3:1d:5c:7d:b5:4e:5a:12:fa:ee:0c:6d:65:dd:77:ce:dc:
      04:f3:f4:c1:57:be:f5:12:87:fd:00:42:d9:e5:98:fc:f7:e7:
      1a:c3:da:f5:16:53:d3:53:54:f1:8c:33:7e:a1:db:b4:4b:82:
      6c:90:0c:2c:1d:f4:1d:1c:c9:d5:72:63:43:f6:b7:7d:4c:92:
      ea:92:77:18:ba:5c:2c:28:e7:21:61:42:c7:0c:21:12:fb:88:
      2b:e8:1b:ff:4d:3b:13:ba:ae:0d:db:8c:83:00:7e:de:ad:2d:
      8f:af:0b:58:d5:4a:06:6b:82:72:f8:89:2d:51:5f:1a:d5:c7:
      ce:3a:90:08:64:d0:4c:a9:fa:e5:44:08:1a:42:63:de:6a:46:
      1a:35:63:68:42:d4:70:d7:9e:be:c1:3e:51:b5:ad:08:95:53:
      c6:73:7e:0c:f0:64:62:eb:1d:62:db:93:8c:27:d6:60:2a:8c:
      5e:c6:d1:1c:18:ed:2c:23:3b:76:b3:8d:4f:83:0f:2f:3d:f8:
      a5:b9:f7:55:fd:bf:6b:55:77:68:be:3b:e1:d3:b4:88:d5:be:
      91:6d:63:4f:59:81:8a:9c:95:69:a2:02:27:ae:7e:70:5f:64:
      f2:42:02:61:9c:bf:1e:96:e6:25:96:41:4b:11:8e:e8:bf:37:
      7f:21:55:b1:e2:d6:20:cc:0b:2f:37:6d:49:b3:25:22:d3:66:
      ca:f6:4d:e6:95:fc:c7:53:67:bc:4a:a6:25:a2:97:e8:d8:55:
      9b:a3:9e:ea:f4:48:f7:db:ec:38:b5:e4:8e:40:b2:83:23:8b:
      e8:08:48:b4:a8:ed:30:eb:75:37:98:3e:a8:ce:eb:98:31:33:
      c1:fc:1c:cb:43:d1:59:a6
  ubuntu@cs6903-1:~/intermediate$
```

Task 2: Secure Chat App (30M)

Write a peer-to-peer application (`secure_chat_app`) for chatting which uses **DTLS 1.2** and **UDP** as the underlying protocols for secure communication. *Note that the `secure_chat_app` works somewhat like HTTPS except that here it's a peer-to-peer paradigm with no reliability where Alice plays the role of the client and Bob plays the role of the server and vice versa.* That means the same program should have different functions for server and client code which can be chosen using command line options “-s” and “-c <serverhostname>” respectively. Feel free to define your own chat headers (if necessary) and add them to the chat payload before giving it to DTLS/UDP. Make sure that your application uses only the hostnames for communication between Alice and Bob but not hard-coded IP addresses (refer `gethostbyname(3)`). The application should perform the following operations:

- a) Bob starts the app using “`secure_chat_app -s`” and Alice starts the app using “`secure_chat_app -c bob1`”
- b) Alice sends a `chat_hello` application layer control message to Bob and Bob replies with a `chat_ok_reply` message. It works like a handshake between peers at the application layer. Note that these control messages are sent in plain-text. Show that it is indeed the case by capturing Pcap traces at Alice-LXD and Bob-LXD.
- c) Alice initiates a secure chat session by sending out a `chat_START_SSL` application layer control message and getting `chat_START_SSL_ACK` from Bob. Your program should then load the respective private keys and certificates for both Alice and Bob. Furthermore, each of them should have pre-loaded the certificates of the root CA and the intermediate CA in their respective trust stores.
 - i) For example, if Alice sends a `chat_START_SSL` control message to Bob, upon parsing the message, Bob initiates replies with `chat_START_SSL_ACK`. Upon parsing this ACK from Bob, Alice initiates **DTLS 1.2** handshake by first sending a `client_hello` message as we discussed in the TLS lesson.
 - ii) Alice gets the certificate of Bob and verifies that. She also provides her certificate to Bob for verification. So, Alice and Bob use their certificates to perform mutual aka two-way authentication.
 - iii) DTLS 1.2 handshake between Alice and Bob should contain Alice specifying a list of one or more ciphersuites that offer perfect forward secrecy (PFS) as part of `client_hello` and Bob picking one of them if his application is pre-configured to support any of them. That means, client and server programs should be pre-configured to support PFS cipher suites using openssl API and then they can agree on some common ciphersuite. Make sure your program generates appropriate error messages if a secure connection could not be established due to mismatch in the supported ciphersuites at client and server.
- d) Upon establishing a secure DTLS 1.2 pipe, it will be used by Alice and Bob to exchange their encrypted chat messages. Show that it is indeed the case by capturing Pcap traces at Alice-LXD/Bob-LXD.

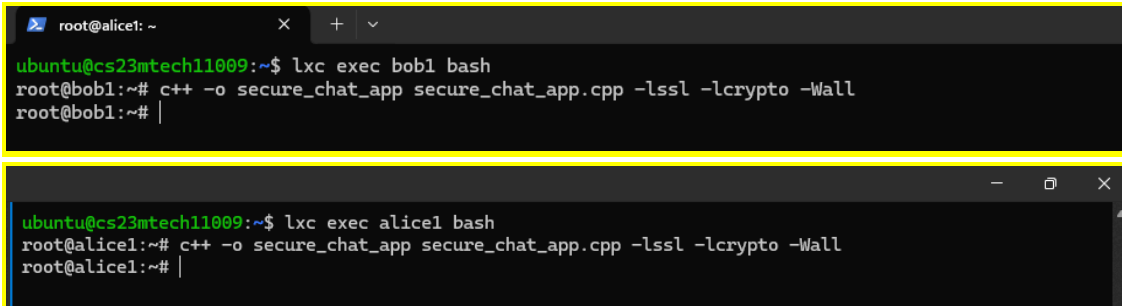
- e) Compare and contrast DTLS 1.2 handshake with that of TLS 1.2 handshake.
- f) Your `secure_chat_app` should support session resumption using session tickets. Show that it is indeed the case by capturing Pcap traces at Alice-LXD/Bob-LXD.
- g) Either of them sends a `chat_close` message which in turn triggers closure of TLS connection and finally TCP connection.

Since the underlying transport protocol is UDP, messages between Alice and Bob may not get delivered reliably. Use `tc` command to force some packet loss on the links. Make sure your secure chat application handles loss of the control messages specific to the secure chat application appropriately with timers and retries. Your application does not have to ensure reliable delivery of application DATA (i.e., chat) messages as only fast delivery of chat messages is important.

We have written chat application for chatting which uses DTLS 1.2 and UDP as the underlying protocols for secure communication

Compiling the Code

\$ `c++ -o secure_chat_app secure_chat_app.cpp -lssl -lcrypto -Wall`



```
root@alice1: ~  
ubuntu@cs23mtech11009:~$ lxc exec bob1 bash  
root@bob1:~# c++ -o secure_chat_app secure_chat_app.cpp -lssl -lcrypto -Wall  
root@bob1:~#  
  
ubuntu@cs23mtech11009:~$ lxc exec alicel bash  
root@alicel:~# c++ -o secure_chat_app secure_chat_app.cpp -lssl -lcrypto -Wall  
root@alicel:~#
```

To start the server we will use the command :

\$ `./secure_chat_app -s`

```

ubuntu@cs23mtech11009:~$ lxc exec bob1 bash
root@bob1:~# c++ -o secure_chat_app secure_chat_app.cpp -lssl -lcrypto -Wall
root@bob1:~# ./secure_chat_app -s
<~> SERVER STARTED!
<~> Waiting for client to connect...
^C
root@bob1:~# ./secure_chat_app -s
<~> SERVER STARTED!
<~> Waiting for client to connect...
<~> Client connected: alice1 (172.31.0.2)
<~> Received from Client: chat_hello
<~> Sent: chat_ok_reply
<~> Received from Client: chat_START_SSL
<~> Sent: chat_START_SSL_ACK
<~> Initializing SSL Handshake...
<~> Server accepted DTLS connection
<~> DTLS Handshake successful
<~> Received from Client: Hello Bob, Kritik here
</> Server: Hi Kritik(Alice), Bob OK
<~> Sent to Client: Hi Kritik(Alice), Bob OK
<~> Received chat_close from Client.
<~> Closing the connection...
root@bob1:~# |

```

For Starting the client we will use :

\$./secure_chat_app -c bob1

```

ubuntu@cs23mtech11009:~$ lxc exec alice1 bash
root@alice1:~# c++ -o secure_chat_app secure_chat_app.cpp -lssl -lcrypto -Wall
root@alice1:~# ./secure_chat_app -c bob1
<~> CLIENT STARTED!
<~> Trying to connect to bob1
<~> Connected to bob1
<~> Sent: chat_hello
<~> Received from Server: chat_ok_reply
<~> Sent: chat_START_SSL
<~> Received from Server: chat_START_SSL_ACK
<~> Initiating DTLS Handshake...
<~> Server accepted DTLS connection
<~> DTLS Handshake successful
</> Client: Hello Bob, Kritik here
<~> Sent to Server: Hello Bob, Kritik here
<~> Received from Server: Hi Kritik(Alice), Bob OK
</> Client: chat_close
<~> Sent chat_close to Client.
<~> Closing the connection...
root@alice1:~# |

```

The client and server exchange “chat_hello” and “chat_ok_reply” and then they also exchange “chat_START_SSL” and “chat_START_SSL_ACK”.

General Flow of the task that is based on our code:-

- 1) Depending on the command-line arguments, an instance of either the Alice or Bob class is created, and the application layer handshake process is initiated.

- 2) Both client and server classes perform an application layer handshake to ensure mutual readiness for secure communication. The client sends a "hello" message to the server, and upon receiving an acknowledgment, initiates the DTLS handshake.
- 3) The server waits for the "hello" message from the client and responds with an acknowledgment before initiating the DTLS handshake.
- 4) After successful application layer handshakes, both client and server initiate the DTLS handshake process. This involves setting up SSL/TLS contexts, loading certificates and private keys, configuring security parameters, and performing the handshake.
- 5) Certificate verification is conducted to ensure the authenticity of peer certificates. Once the DTLS handshake is completed, secure communication channels are established between the client and server.
- 6) With the secure channels established, both client and server can securely exchange messages.
- 7) Messages are encrypted and decrypted using SSL/TLS encryption algorithms. The client and server continue to exchange messages until the communication is terminated.
- 8) The communication session can be terminated by either party sending a predefined termination message (e.g., CHAT_CLOSE).
- 9) Upon receiving the termination message, the recipient closes the secure channel and terminates the program.
- 10) Resources such as SSL/TLS contexts, objects, sockets, and memory allocations are freed and closed appropriately to ensure proper cleanup and release of system resources.

NOTE:- PCAP traces are included in the README file.

Task 3: START_SSL downgrade attack for eavesdropping (20M)

Downgrade attack by Trudy by intercepting the chat_START_SSL control message from Alice (Bob) to Bob (Alice).

- a) If Alice receives a chat_START_SSL_NOT_SUPPORTED message after sending chat_START_SSL to Bob, it assumes that Bob does not have capability to set up secure chat communication.
- b) In this attack, Trudy blocks chat_START_SSL from reaching Bob and sends a forged reply message chat_START_SSL_NOT_SUPPORTED to Alice and thereby forcing Alice and Bob to have unsecure chat communication for successfully intercepting their communication. Show that it is indeed the case by capturing Pcap traces at Trudy/Alice/Bob LXDs.
- c) You need to write a program (secure_chat_interceptor) to launch this downgrade attack (-d command line option) from Trudy-LXD. For this task, you can assume that Trudy poisoned the /etc/hosts file of Alice (Bob) and replaced the IP address of Bob (Alice) with that of her. It's a kind of DNS spoofing for launching MITM attacks. In this attack, Trudy only plays with chat_START_SSL message(s) and forwards the rest of the traffic as it is i.e., eavesdropper.

To poison /etc/hosts file of Alice and Bob containers, use the following command from inside the VM

```
bash ~/poison-dns-alice1-bob1.sh
```

To revert back the /etc/hosts file of Alice, Bob containers, use the following command from inside the VM.

```
bash ~/unpoison-dns-alice1-bob1.sh
```

To start a downgrade attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.

```
./secure_chat_interceptor -d alice1 bob1
```

Trudy executes a downgrade attack by maliciously intercepting the initial chat_START_SSL message sent between Alice and Bob, disrupting their attempt to establish a secure connection.

A chat_START_SSL_NOT_SUPPORTED response from Bob following Alice's chat_START_SSL initiation signals a potential lack of secure chat capability on Bob's device.

In this attack scenario, Trudy demonstrates the ability to force insecure communication between Alice and Bob. This is done by intercepting the chat_START_SSL message, forging a chat_START_SSL_NOT_SUPPORTED response to Alice, and effectively tricking both parties into believing secure communication is impossible.

The task involves intercepting the chat_START_SSL message, where Trudy intercepts the communication from Alice to Bob.

- 1) Trudy then crafts a reply message, chat_START_SSL_NOT_SUPPORTED, and sends it to Alice, indicating that Bob does not support secure chat communication.
- 2) Following this, Trudy proceeds to forward the remaining traffic between Alice and Bob without modification, effectively acting as an eavesdropper.
- 3) As part of the demonstration, network traffic at Trudy, Alice, and Bob's LXDs is captured to illustrate the interception and modification of messages.
- 4) To simulate DNS spoofing, Trudy poisons the /etc/hosts file of Alice and Bob containers to redirect their traffic to Trudy.
- 5) Additionally, a script is provided to revert the changes made to the /etc/hosts file of Alice and Bob containers, undoing the DNS poisoning.
- 6) Finally, a command-line option (-d) is implemented to indicate the downgrade attack mode, offering flexibility in executing the attack scenario.

After poisoning the server we executed the commands:

And got the below result :

```
root@alice1:~# ./secure_chat_app -s
<=> SERVER STARTED!
<=> Waiting for client to connect...
<=> Client connected: alice1 (172.31.0.4)

root@bob1:~# ./secure_chat_app -c bob1
<=> CLIENT STARTED!
<=> Trying to connect to bob1
<=> Connected to bob1
<=> Sent: chat_hello

ubuntu@cs23mtech11009:~$ bash ~/poison-dns-alice1-bob1.sh
ubuntu@cs23mtech11009:~$

root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
<=> Starting Downgrading Attack...
<=> Attacker: Trudy
<=> Server socket binded...
</> Received a message: chat_hello
</> Sent a message: chat_hello
```

Task 4: Active MITM attack for tampering chat messages and dropping DTLS handshake messages (40M)

Active MITM attack by Trudy to tamper the chat communication between Alice and Bob. For this task also, you can assume that Trudy poisoned the `/etc/hosts` file of Alice (Bob) and replaced the IP address of Bob (Alice) with that of her.

- Also assume that Trudy hacks into the server of the intermediate CA and issues fake/shadow certificates for Alice and Bob. Save these fake certificates as `fakealice.crt` and `fakebob.crt`, save their CSRs and key-pairs in `.pem` files and verify that they are indeed valid using `openssl`!
- Rather than launching the `START_SSL` downgrade attack, in this attack Trudy sends the fake certificate of Bob when Alice sends a `client_hello` message and vice versa. This certificate is indeed signed by the trusted intermediate CA, so its verification succeeds at Alice. So, two DTLS 1.2 pipes are set up: one between Alice and Trudy; the other between Trudy and Bob. Trudy is now like a malicious intercepting proxy who can decrypt messages from Alice to Bob (and from Bob to Alice) and re-encrypt them as-it-is or by altering message contents as she desires! Show that it is indeed the case by capturing Pcaps at Trudy/Alice/Bob LXDs.
- Modify the `secure_chat_interceptor` program to launch this active MITM attack from Trudy-LXD, name it as `secure_chat_active_interceptor`
To start the MITM attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.
`./secure_chat_active_interceptor -m alice1 bob1`

```
root@alice1:~# ./secure_chat_app -s
<=> SERVER STARTED!
<=> Waiting for client to connect...
<=> Client connected: alice1 (172.31.0.4)

root@bob1:~# ./secure_chat_app -c bob1
<=> CLIENT STARTED!
<=> Trying to connect to bob1
<=> Connected to bob1
<=> Sent: chat_hello

ubuntu@cs23mtech11009:~$ bash ~/poison-dns-alice1-bob1.sh
ubuntu@cs23mtech11009:~$

root@trudy1:~# ./secure_chat_interceptor -m alice1 bob1
<=> Starting MITM Attack...
<=> Attacker: Trudy
<=> Server socket binded...
</> Received a message: chat_hello
</> Sent a message: chat_hello
```

Task V: Optional

Bonus Marks (25 M): In this assignment, you emulated DNS poisoning by running **poison-dns-alice1-bob1.sh** script written by TAs to manipulate the entries in the `/etc/hosts` file. **Instead you need to implement one of the following to get the bonus marks:**

1. You need to first ensure that Alice/Bob sends DNS queries (over UDP) to a local resolver which in turn contacts an emulated DNS infra (root servers and Authoritative Name servers) and gets DNS response. Trudy tampers these responses so that the DNS cache of the resolver is poisoned.
2. ARP cache poisoning where Trudy sends gratuitous fake ARP messages to Alice/Bob. Refer this assignment https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/ for launching this real MITM attack in your set up.

```
root@trudy1:~# cat arppoison.sh
#!/bin/bash

arpspoof -i "eth0" -t "172.31.0.2" "172.31.0.3"
arpspoof -i "eth0" -t "172.31.0.3" "172.31.0.2"

root@trudy1:~# |
```

[illegible]