



भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Secure Chat Application

Network Security Project Report

Submitted by

Patel Heetkumar Dilipbhai

CS23MTECH11029

KR Anuraj

CS23MTECH13002

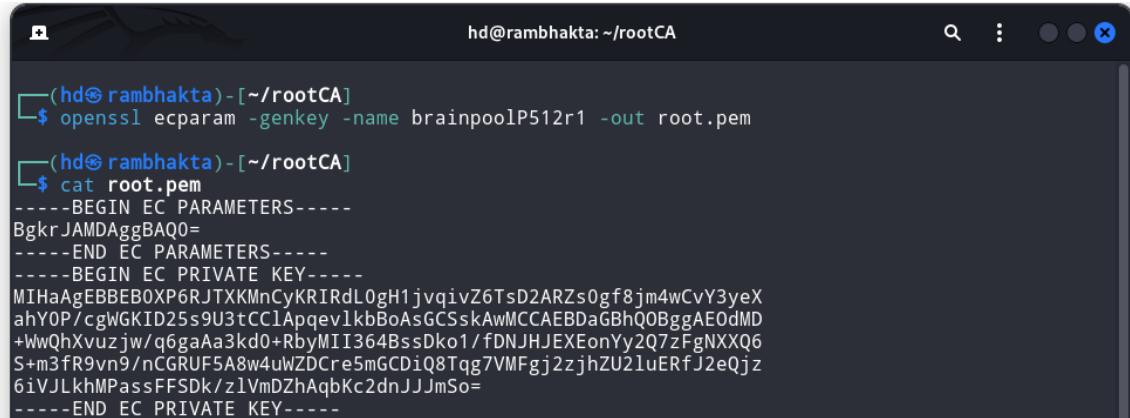
Vishal Patidar

CS23MTECH14017

Task 1 – Keys & Certificate generation

Root CA's Certificate Generation :

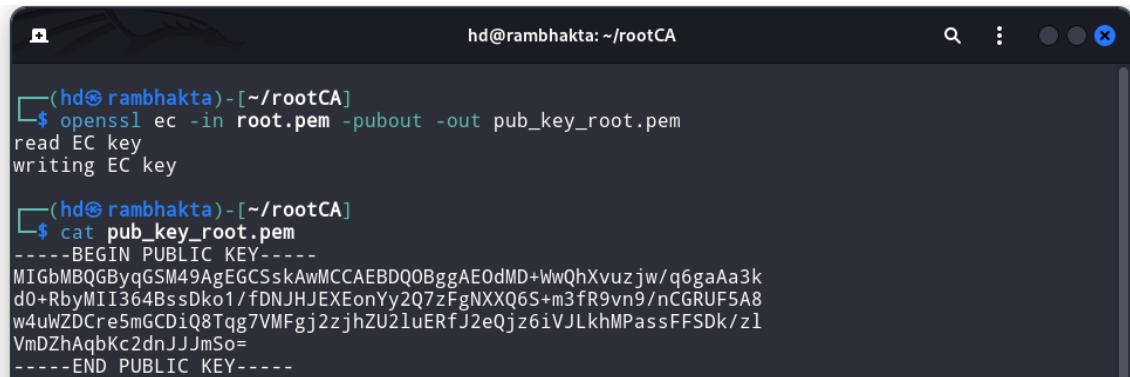
Step 1 : Generating ECC 512-bit Key pair for Root CA



```
hd@rambhakta: ~/rootCA
$ openssl ecparam -genkey -name brainpoolP512r1 -out root.pem

hd@rambhakta: ~/rootCA
$ cat root.pem
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQo=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBBEBOXPGRJTXKMnCyKRIrdL0gH1jvqivZ6TsD2ARZs0gf8jm4wCvY3yeX
ahYOP/cgwGKID25s9U3tC1Apqev1kbBoAsGCSSkAwMCCAEBDaGBhQOBggAE0dMD
+WwQhXvuzjw/q6gaAa3kd0+RbyMII364BssDko1/fDNJHJEXEonYy2Q7zFgNXXQ6
S+m3fR9vn9/nCGRUF5A8w4uWZDCre5mGCDiQ8Tqg7VMFgj2zjhZU2luERfJ2eQjz
6iVJLkhMPassFFSDk/zlVmDZhAqbKc2dnJJmSo=
-----END EC PRIVATE KEY-----
```

Step 2 : Extracting Public key from the above generated key pair

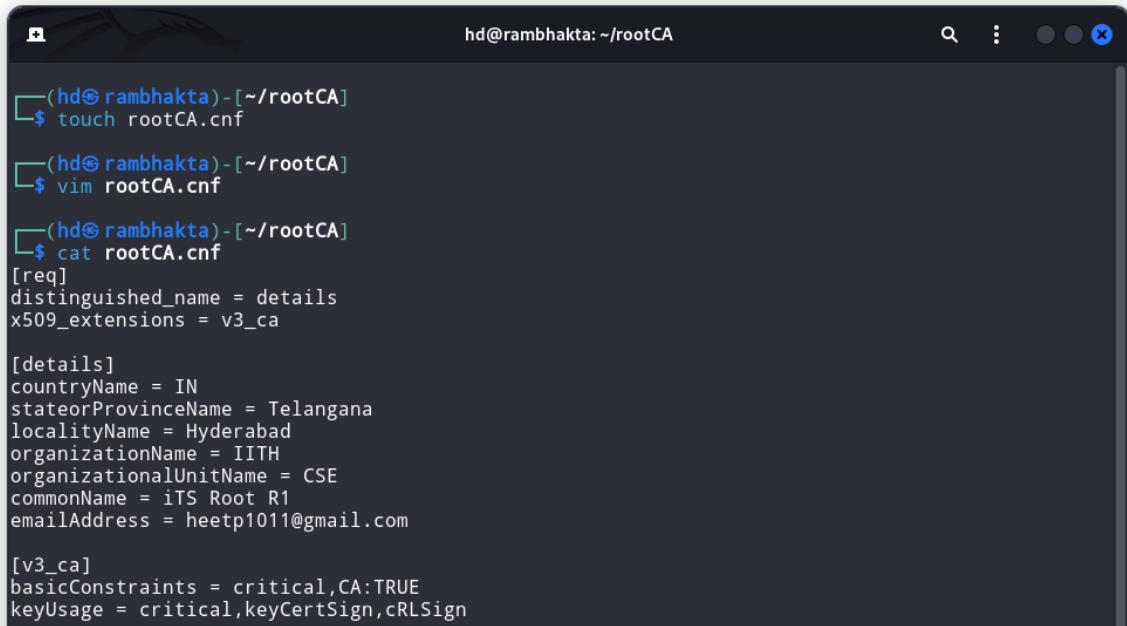


```
hd@rambhakta: ~/rootCA
$ openssl ec -in root.pem -pubout -out pub_key_root.pem
read EC key
writing EC key

hd@rambhakta: ~/rootCA
$ cat pub_key_root.pem
-----BEGIN PUBLIC KEY-----
MIGbMBQGByqGSM49AgEGCSskAwMCCAEBDQOBggAE0dMD+WwQhXvuzjw/q6gaAa3k
d0+RbyMII364BssDko1/fDNJHJEXEonYy2Q7zFgNXXQ6S+m3fR9vn9/nCGRUF5A8
w4uWZDCre5mGCDiQ8Tqg7VMFgj2zjhZU2luERfJ2eQjz6iVJLkhMPassFFSDk/zl
VmDZhAqbKc2dnJJmSo=
-----END PUBLIC KEY-----
```

Now in order to generate the self-sign certificate root CA need one config file which contains all details for generating certificate including the basic constraints, key usage and more.

Step 3 : Creating config file for generating self-signed certificate

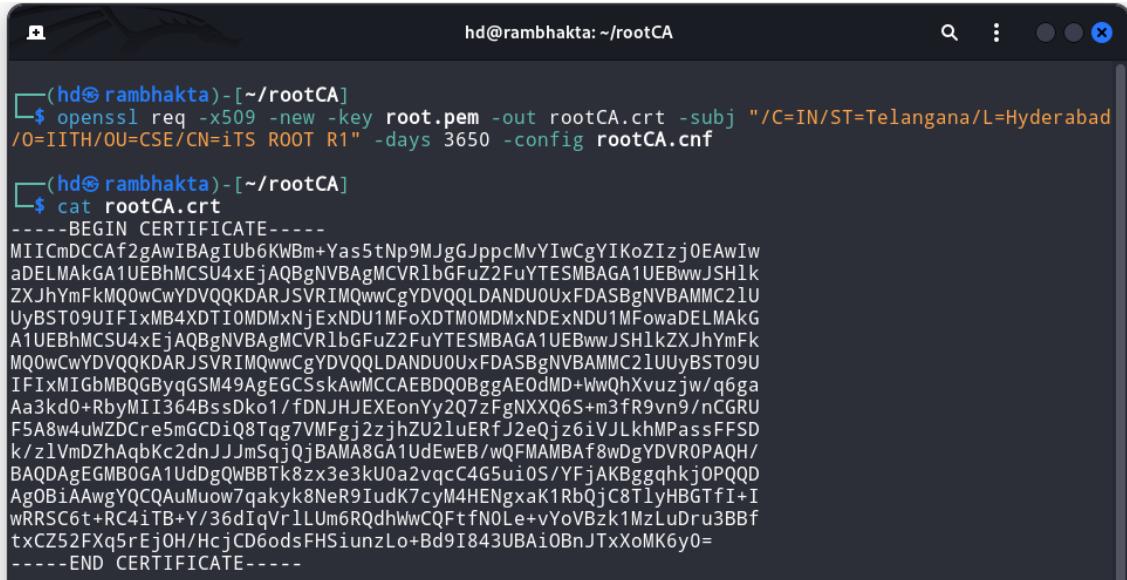


```
hd@rambhakta: ~/rootCA
└─$ touch rootCA.cnf
└─$ vim rootCA.cnf
└─$ cat rootCA.cnf
[req]
distinguished_name = details
x509_extensions = v3_ca

[details]
countryName = IN
stateorProvinceName = Telangana
localityName = Hyderabad
organizationName = IITH
organizationalUnitName = CSE
commonName = iTS Root R1
emailAddress = heetp1011@gmail.com

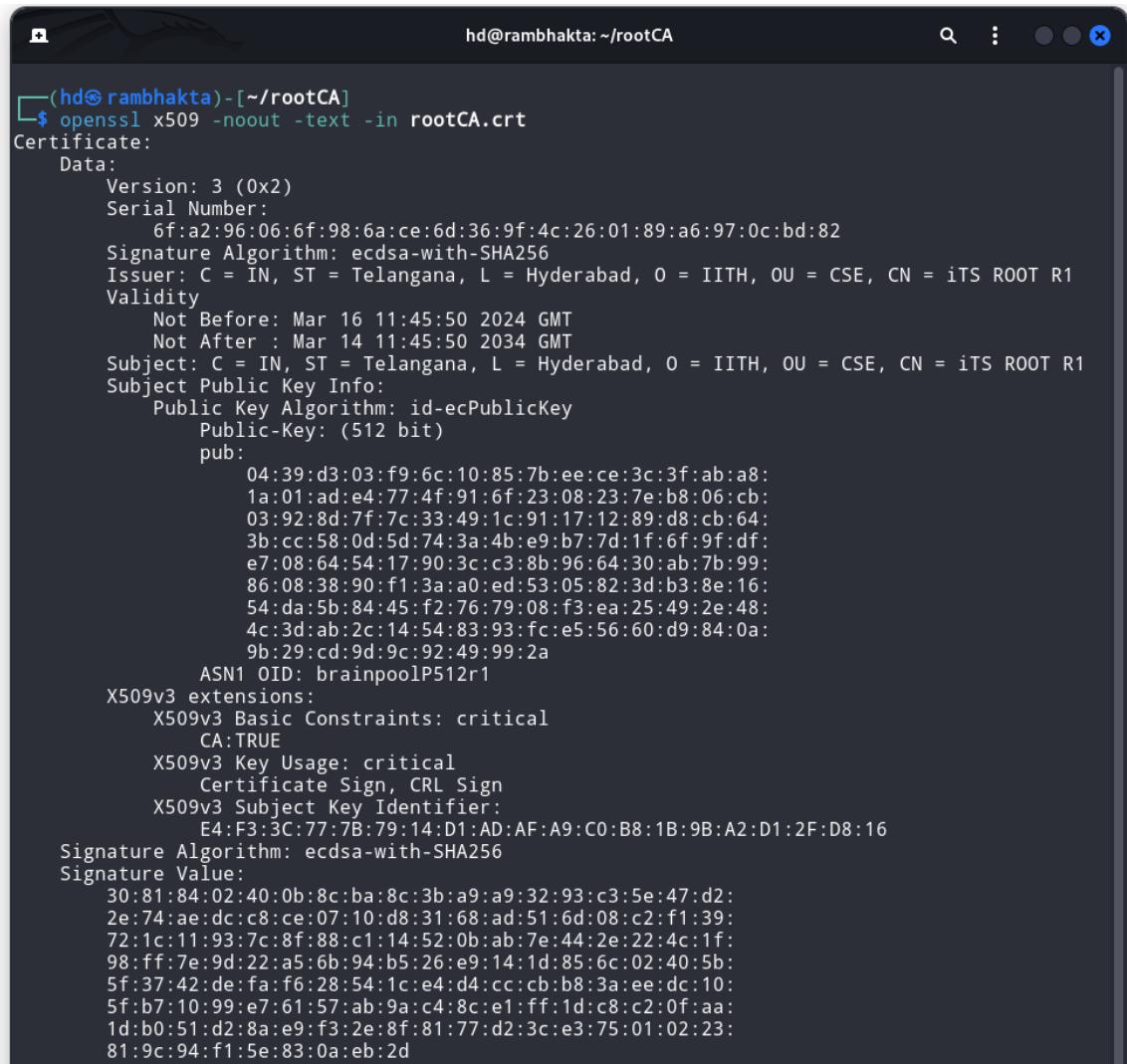
[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

Step 4 : Creating self-signed certificate for root CA



```
hd@rambhakta: ~/rootCA
└─$ openssl req -x509 -new -key root.pem -out rootCA.crt -subj "/C=IN/ST=Telangana/L=Hyderabad
/O=IITH/OU=CSE/CN=iTS ROOT R1" -days 3650 -config rootCA.cnf
└─$ cat rootCA.crt
-----BEGIN CERTIFICATE-----
MIICDCCAf2gAwIBAgIUb6KWBm+Yas5tNp9MjgGJppcMvYIwCgYIKoZIzj0EAwIw
aDELMAkGA1UEBhMCSU4xEjAQBgNVBAgMCVR1bGFuZ2FuYTEsMBAGA1UEBwwJSH1k
ZXJhYmFkMQowCwYDVQQLDARJSVRIMQwwCgYDVQQLDANDU0UxFDASBgNVBAMMC2lU
UyBST09UFIxMB4XDTI0MDMxNjExNDU1MFoXDTM0MDMxNDExNDU1MFowaDELMAkG
A1UEBhMCSU4xEjAQBgNVBAgMCVR1bGFuZ2FuYTEsMBAGA1UEBwwJSH1kZXJhYmFk
MQowCwYDVQQLDARJSVRIMQwwCgYDVQQLDANDU0UxFDASBgNVBAMMC2lUyBST09U
IFIxMIGbMBQGBByqGSM49AgEGCskAwMCACBDQOBggAEodMD+WwQhXvuwjw/q6ga
Aa3kd0+RbyMII364BsxDko1/fDNJHJEXEonYy2Q7zFgNXQ6S+m3fr9vn9/nCGRU
F5A8w4uWZDCre5m6CDiQ8Tqg7VMEgj2zjhZU2luErJ2eQjz6iVJLkhMPassFFSD
k/z1VmDZhAqbKc2dnJJmSqjQjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/
BAQDagEGBM0GA1UdDgQWBBTk8zx3e3kU0a2vqcC4G5uiOS/YFjAKBggqhkjOPQQD
AgOBiAAwgYQCQAUmuow7qakyk8NeR9IudK7cyM4HENgxaK1RbQjC8TlyHBGTfI+i
wRRSC6t+RC4iTb+/36dIqVr1LUm6RQdhlwCQFtfNOLe+vYoVBzk1MzLuDru3BBf
txCZ52FXq5rEjOH/HcjCD6odsFHSiunzLo+Bd9I843UBAiOBnJTxXoMK6y0=
-----END CERTIFICATE-----
```

Output : Root CA's self-signed certificate

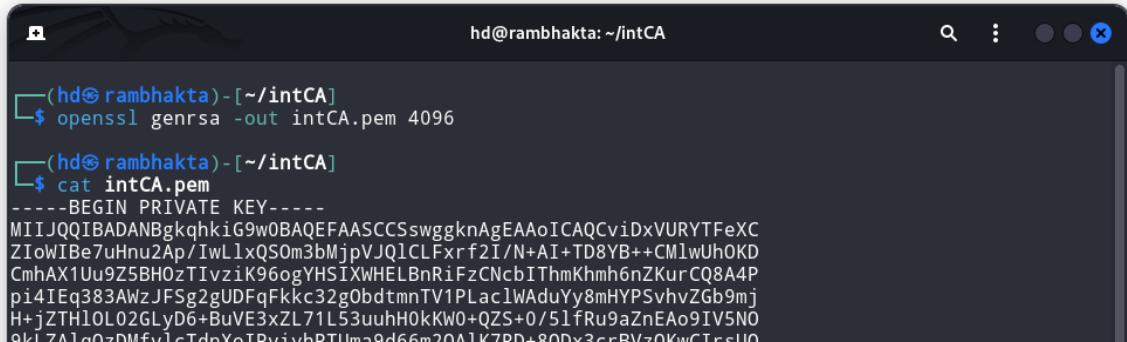


hd@rambhakta: ~/rootCA

```
(hd@rambhakta)-[~/rootCA]
$ openssl x509 -noout -text -in rootCA.crt
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        6f:a2:96:06:6f:98:6a:ce:6d:36:9f:4c:26:01:89:a6:97:0c:bd:82
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = iTS ROOT R1
    Validity
        Not Before: Mar 16 11:45:50 2024 GMT
        Not After : Mar 14 11:45:50 2034 GMT
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = iTS ROOT R1
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
        Public-Key: (512 bit)
        pub:
            04:39:d3:03:f9:6c:10:85:7b:ee:ce:3c:3f:ab:a8:
            1a:01:ad:e4:77:4f:91:6f:23:08:23:7e:b8:06:cb:
            03:92:8d:7f:7c:33:49:1c:91:17:12:89:d8:cb:64:
            3b:cc:58:0d:5d:74:3a:4b:e9:b7:7d:1f:6f:9f:df:
            e7:08:64:54:17:90:3c:c3:8b:96:64:30:ab:7b:99:
            86:08:38:90:f1:3a:a0:ed:53:05:82:3d:b3:8e:16:
            54:da:5b:84:45:f2:76:79:08:f3:ea:25:49:2e:48:
            4c:3d:ab:2c:14:54:83:93:fc:e5:56:60:d9:84:0a:
            9b:29:cd:9d:9c:92:49:99:2a
        ASN1 OID: brainpoolP512r1
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE
        X509v3 Key Usage: critical
            Certificate Sign, CRL Sign
        X509v3 Subject Key Identifier:
            E4:F3:3C:77:7B:79:14:D1:AD:AF:A9:C0:B8:1B:9B:A2:D1:2F:D8:16
    Signature Algorithm: ecdsa-with-SHA256
Signature Value:
30:81:84:02:40:0b:8c:ba:8c:3b:a9:a9:32:93:c3:5e:47:d2:
2e:74:ae:dc:c8:ce:07:10:d8:31:68:ad:51:6d:08:c2:f1:39:
72:1c:11:93:7c:8f:88:c1:14:52:0b:ab:7e:44:2e:22:4c:1f:
98:ff:7e:9d:22:a5:6b:94:b5:26:e9:14:1d:85:6c:02:40:5b:
5f:37:42:de:fa:f6:28:54:1c:e4:d4:cc:cb:b8:3a:ee:dc:10:
5f:b7:10:99:e7:61:57:ab:9a:c4:8c:e1:ff:1d:c8:c2:0f:aa:
1d:b0:51:d2:8a:e9:f3:2e:8f:81:77:d2:3c:e3:75:01:02:23:
81:9c:94:f1:5e:83:0a:eb:2d
```

Intermediate CA's Certificate Generation :

Step 1 : Generating 4096-bit RSA Key pair for Int CA



```
hd@rambhakta: ~/intCA
$ openssl genrsa -out intCA.pem 4096

(hd@rambhakta) [~/intCA]
$ cat intCA.pem
-----BEGIN PRIVATE KEY-----
MIJQQIBADANBgkqhkiG9w0BAQEFAASCCSswggknAgEAAoICAQCviDxVURYTFeXC
ZIowIBe7uhnu2Ap/IwlLxQSm3bMjpVJQ1CLFxrf2I/N+A1+TD8YB++CMlwUhOKD
CmhAX1Uu9Z5BH0zTIVziK96ogYHSIXWHELBNriFzCNcbITHmKhmh6nZKurCQ8A4P
pi4IEq383AWzJSg2gUDFqFkkc32g0bdtnnTV1PLaclWAduYy8mHYPsvhvZGb9mj
H+jZTH1OL02GLyD6+BuVE3xZL1L53uuh0kKw0+QZS+0/5lfRu9aZnEAo9IV5NO
-----END PRIVATE KEY-----
```

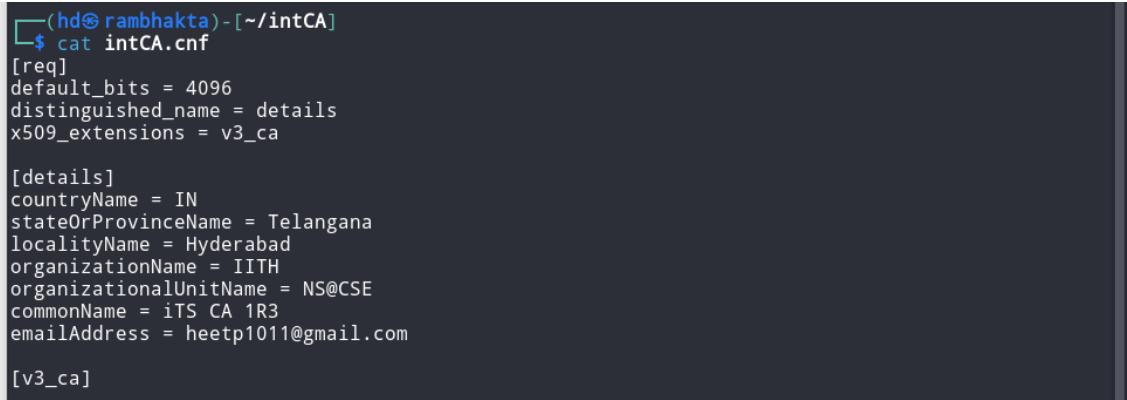
Step 2 : Extracting Public key from the above generated key pair



```
hd@rambhakta: ~/intCA
$ openssl rsa -in intCA.pem -pubout -out pub_key_intCA.pem
writing RSA key

(hd@rambhakta) [~/intCA]
$ cat pub_key_intCA.pem
-----BEGIN PUBLIC KEY-----
MIICIJANBgkqhkiG9w0BAQEFAOCAg8AMIIICgKCAGEA4g8VVEWExXlwmSKFiAX
u7h57tgKfyMC5cUEjpt2z16VSUQixca39iPzfgCPkw/GAfvgjJcFITigwpoQF9V
LvwleQRzs0yL84iveqIGBoiF1hxCwZ0YhcwjXGyE4ZioZoep2SrqwPAOD6YuCBKt
/NwfSyRUoNoFAxahZJHN9oDm3bZp01dTy2nJVGHbmMvJh2DOr4b2Rm/Zox/o2Ux5
Ti9Nhi8g+vgb1RN8WS+9S+d7roR9JCltPkGUvtP+ZX0bvWmZxAKPSFeTTvCZ2QJa
kMwzH75XE3Z16CD74soT01JmvXeuptkAJSu0Q/vDg8d3KwVc0CsAiK7FEGl3sSqj
uD7vDGzPcv5d14ZWteH+2FvfPm26t05ZPJe1WPNTA10HphjbYHDzDWmt2bIZx10
JONydxJgeFrxtQoXrqP2/zs8YLTEhC87hq0Byh1pskY4GMgu29frJM9Q2A7QWeQ
poWUDhbqa1vKvGX6oTPypkaTG0ZFgVc9x/8ttYnfBZfdq68zQw5LP9vn+PseaHGH
HcxZqOA80/B/uXhG3r/D0/exus3ZFM9/0r58tGzfzb0i/2tqsEJd0CINY0HzoYqD
lW1UUqNZIK5tfvUmjIvN+5ATvbtdaDXF4FGgvbMZcu1C1m+bS9C035Fc1HFPArv6
th2dG9MU5IXhLLGVdkE0yicCAwEAAQ==
-----END PUBLIC KEY-----
```

Step 3 : Creating config file for Int CA to generate certificate signing request

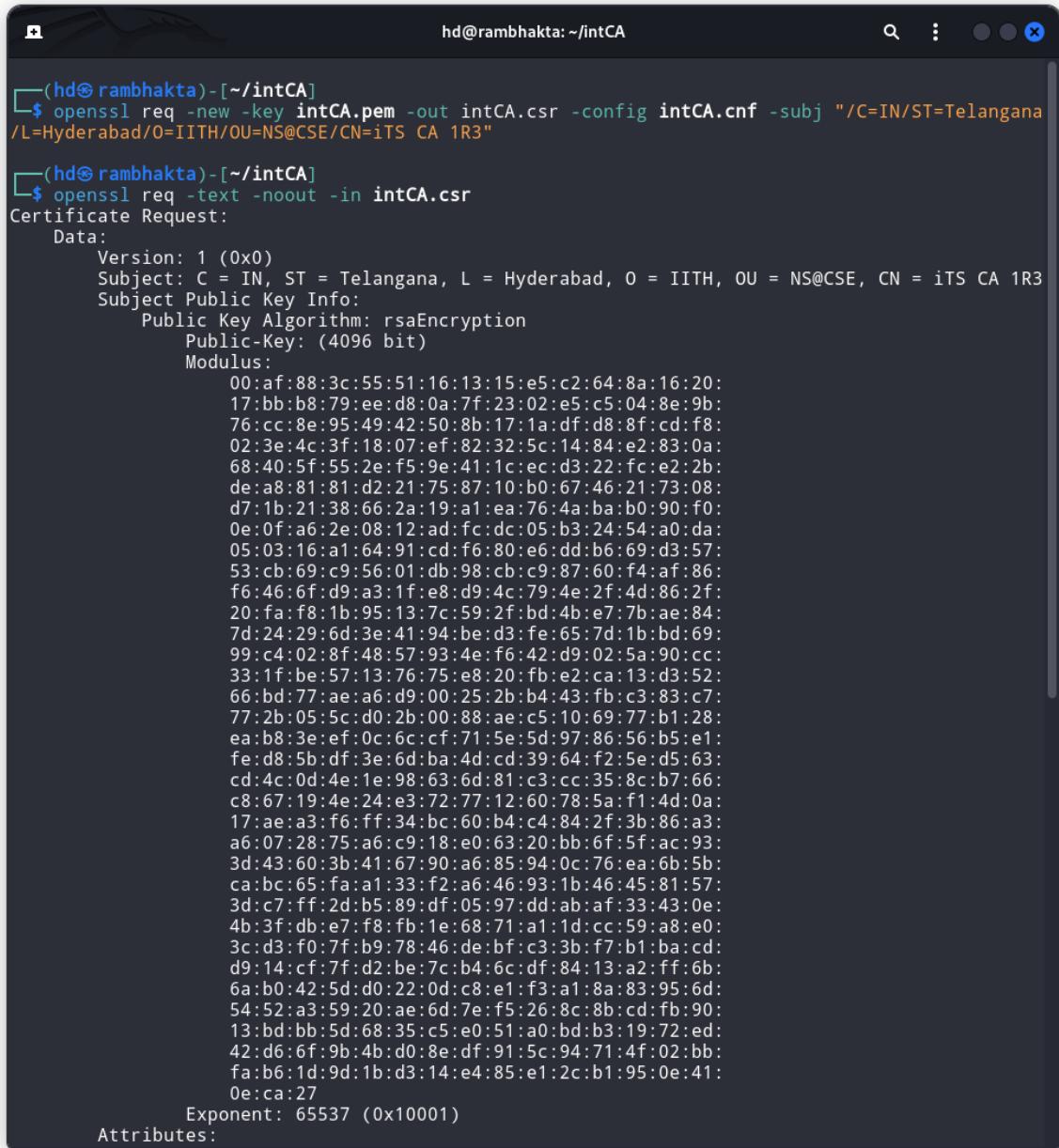


```
[hd@rambhakta) [~/intCA]
$ cat intCA.cnf
[req]
default_bits = 4096
distinguished_name = details
x509_extensions = v3_ca

[details]
countryName = IN
stateOrProvinceName = Telangana
localityName = Hyderabad
organizationName = IIITH
organizationalUnitName = NS@CSE
commonName = iTS CA 1R3
emailAddress = heetp1011@gmail.com

[v3_ca]
```

Step 4 : Creating certificate signing request with above created config file and sending it to Root CA

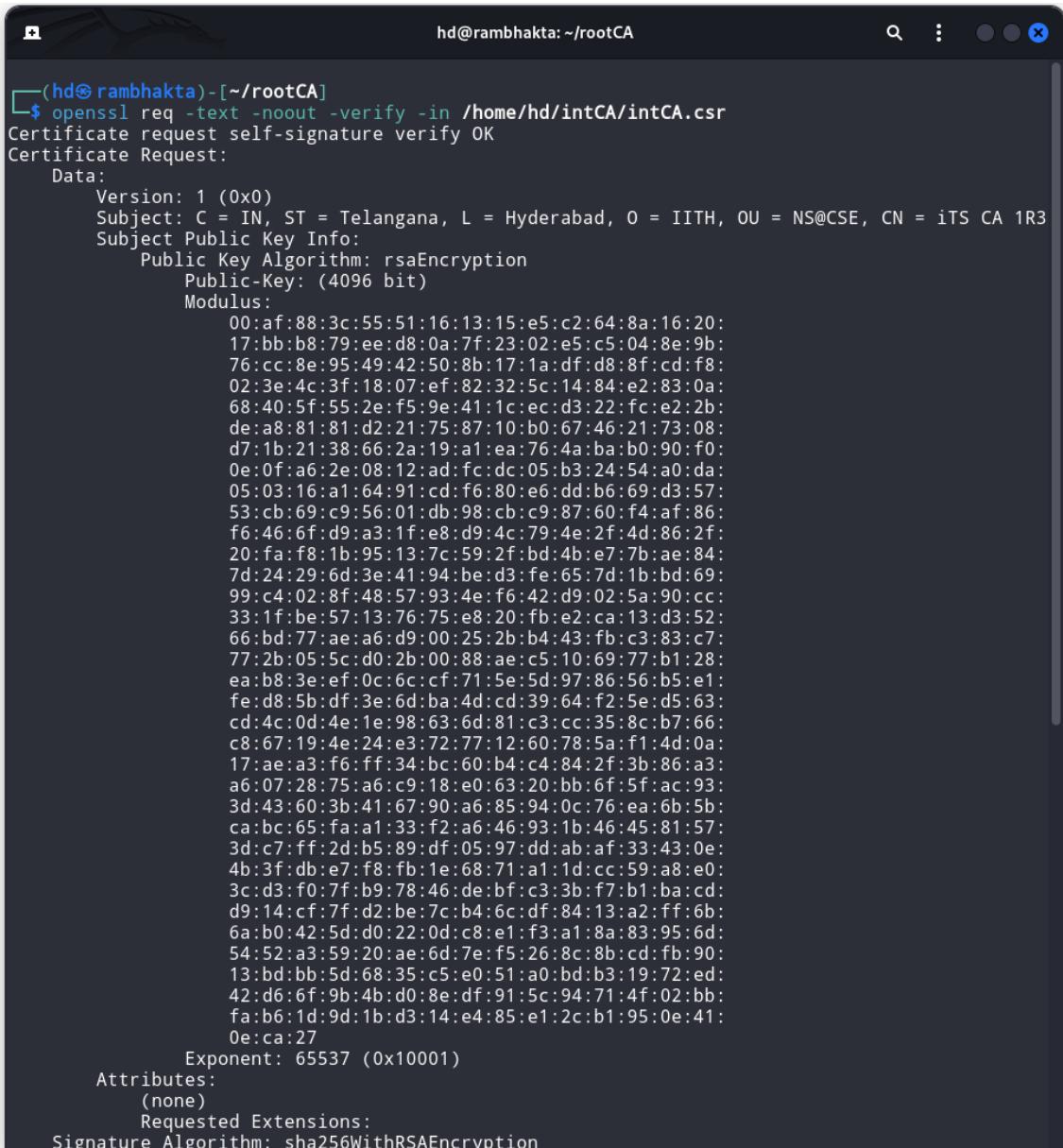


hd@rambhakta: ~/intCA

```
(hd@rambhakta)-[~/intCA]$ openssl req -new -key intCA.pem -out intCA.csr -config intCA.cnf -subj "/C=IN/ST=Telangana/L=Hyderabad/O=IITH/OU=NS@CSE/CN=its CA 1R3"
(hd@rambhakta)-[~/intCA]$ openssl req -text -noout -in intCA.csr
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = its CA 1R3
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
            Modulus:
                00:af:88:3c:55:51:16:13:15:e5:c2:64:8a:16:20:
                17:bb:b8:79:ee:d8:0a:7f:23:02:e5:c5:04:8e:9b:
                76:cc:8e:95:49:42:50:8b:17:1a:df:d8:8f:cd:f8:
                02:3e:4c:3f:18:07:ef:82:32:5c:14:84:e2:83:0a:
                68:40:5f:55:2e:f5:9e:41:1c:ec:d3:22:fc:e2:2b:
                de:a8:81:81:d2:21:75:87:10:b0:67:46:21:73:08:
                d7:1b:21:38:66:2a:19:a1:ea:76:4a:ba:b0:90:f0:
                0e:0f:a6:2e:08:12:ad:fc:dc:05:b3:24:54:a0:da:
                05:03:16:a1:64:91:cd:f6:80:e6:dd:b6:69:d3:57:
                53:cb:69:c9:56:01:db:98:cb:c9:87:60:f4:af:86:
                f6:46:6f:d9:a3:1f:e8:d9:4c:79:4e:2f:4d:86:2f:
                20:fa:f8:1b:95:13:7c:59:2f:bd:4b:e7:7b:ae:84:
                7d:24:29:6d:3e:41:94:be:d3:fe:65:7d:1b:bd:69:
                99:c4:02:8f:48:57:93:4e:f6:42:d9:02:5a:90:cc:
                33:1f:be:57:13:76:75:e8:20:fb:e2:ca:13:d3:52:
                66:bd:77:ae:a6:d9:00:25:2b:b4:43:fb:c3:83:c7:
                77:2b:05:5c:d0:2b:00:88:ae:c5:10:69:77:b1:28:
                ea:b8:3e:ef:0c:6c:cf:71:5e:5d:97:86:56:b5:e1:
                fe:d8:5b:df:3e:6d:ba:4d:cd:39:64:f2:5e:d5:63:
                cd:4c:0d:4e:1e:98:63:6d:81:c3:cc:35:8c:b7:66:
                c8:67:19:4e:24:e3:72:77:12:60:78:5a:f1:4d:0a:
                17:ae:a3:f6:ff:34:bc:60:b4:c4:84:2f:3b:86:a3:
                a6:07:28:75:a6:c9:18:e0:63:20:bb:6f:5f:ac:93:
                3d:43:60:3b:41:67:90:a6:85:94:0c:76:ea:6b:5b:
                ca:bc:65:fa:a1:33:f2:a6:46:93:1b:46:45:81:57:
                3d:c7:ff:2d:b5:89:df:05:97:dd:ab:af:33:43:0e:
                4b:3f:db:e7:f8:fb:1e:68:71:a1:1d:cc:59:a8:e0:
                3c:d3:f0:7f:b9:78:46:de:bf:c3:3b:f7:b1:ba:cd:
                d9:14:cf:7f:d2:be:7c:b4:6c:df:84:13:a2:ff:6b:
                6a:b0:42:5d:d0:22:0d:c8:e1:f3:a1:8a:83:95:6d:
                54:52:a3:59:20:ae:6d:7e:f5:26:8c:8b:cd:fb:90:
                13:bd:bb:5d:68:35:c5:e0:51:a0:bd:b3:19:72:ed:
                42:d6:6f:9b:4b:d0:8e:df:91:5c:94:71:4f:02:bb:
                fa:b6:1d:9d:1b:d3:14:e4:85:e1:2c:b1:95:0e:41:
                0e:ca:27
Exponent: 65537 (0x10001)
Attributes:
```

```
(none)
Requested Extensions:
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
65:7a:4e:20:20:5a:27:03:3a:ab:b2:19:3c:5d:c6:49:1e:ac:
f9:b6:af:82:be:e8:d5:6f:43:6f:53:d7:d0:c4:93:69:86:49:
f0:cb:7a:bd:56:2a:81:ec:8b:9e:30:ba:10:2f:3f:cb:cf:77:
a5:58:fb:d3:c4:b2:35:df:b9:85:b0:01:80:c7:3d:a7:59:5b:
90:ff:c4:85:db:4a:d3:e4:9e:21:55:e8:00:af:e4:60:bd:44:
4f:d6:3e:5d:eb:f9:bc:ec:92:4b:bc:a3:a7:17:ab:b3:fa:69:
13:7a:bf:b3:45:4e:85:86:02:e6:d3:de:9a:b2:b7:2f:b4:3e:
a3:d1:55:37:7b:2e:7f:d8:50:e5:9f:90:7f:ac:37:ce:be:e7:
82:7f:ce:68:ea:1f:2a:93:25:ef:ca:8b:8f:b2:47:18:fc:89:
f6:ed:a9:64:1b:b4:36:9d:cf:23:ff:94:a1:5a:5c:0e:23:08:
85:1a:00:ca:d4:19:32:e7:94:a6:90:c9:b8:d0:39:93:40:e7:
71:15:e1:dd:ef:40:b0:98:af:96:63:df:23:50:ff:bd:6c:59:
d3:2c:c7:51:81:d8:25:ef:5f:37:e6:44:ae:2a:72:31:ea:82:
f6:47:ac:51:c4:da:18:30:f1:b2:30:5a:a9:2e:0b:a1:db:b2:
7d:c9:92:25:44:0c:55:e7:dd:6a:e7:a0:3a:b7:3c:b7:00:0b:
07:60:e4:44:be:c5:b3:5f:f5:b7:60:e9:42:00:fc:b7:8c:5a:
bf:28:07:b7:00:25:8a:25:23:1a:1b:94:71:90:20:d9:2f:a7:
60:71:ee:d9:04:f6:77:0a:50:15:14:53:7f:9a:99:79:4d:bc:
8d:61:76:35:ae:40:f3:2c:a4:5f:57:5c:b9:3b:75:fa:94:cd:
bb:b3:76:b5:de:3f:dc:cd:57:b2:7f:de:7c:90:a8:45:e7:79:
b4:39:e2:8e:a8:66:ff:c3:91:a6:ba:c5:dc:f6:b4:3a:f8:c5:
b0:be:ea:69:7d:9f:76:01:ba:d6:99:47:38:29:65:33:db:19:
ed:bb:e5:1d:eb:71:9b:6d:8a:f6:98:0b:20:01:64:a6:ce:e5:
3a:3a:84:7f:d0:55:73:08:80:d3:20:30:63:4b:d9:04:fc:24:
ac:2b:34:b9:3c:06:11:f4:fe:d7:cb:64:58:b7:a4:51:67:d7:
c1:78:eb:a7:85:3a:94:69:9c:e7:57:61:5a:9a:73:3b:6f:21:
cc:35:9e:9f:bb:a9:aa:aa:b3:b3:53:1b:ab:05:f1:be:a0:3a:
72:33:50:28:35:5c:45:ff:3c:9b:33:9d:8d:2a:2c:b2:1b:d0:
49:2a:05:0e:94:46:53:e1
```

Step 5 : On receiving csr, Root CA will verify the cse that it is authentic.



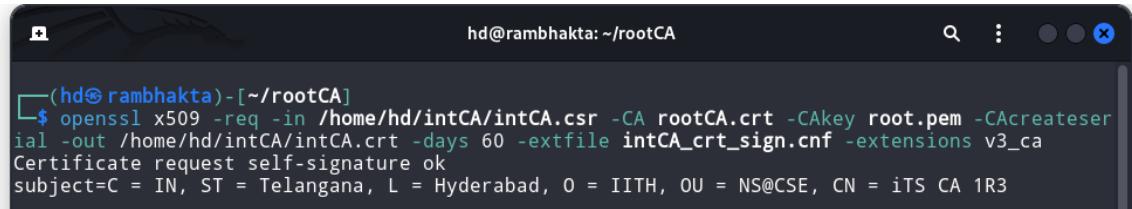
```
hd@rambhakta:~/rootCA
$ openssl req -text -noout -verify -in /home/hd/intCA/intCA.csr
Certificate request self-signature verify OK
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = iTS CA 1R3
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
            Modulus:
                00:af:88:3c:55:51:16:13:15:e5:c2:64:8a:16:20:
                17:bb:b8:79:ee:d8:0a:7f:23:02:e5:c5:04:8e:9b:
                76:cc:8e:95:49:42:50:8b:17:1a:df:d8:8f:cd:f8:
                02:3e:4c:3f:18:07:ef:82:32:5c:14:84:e2:83:0a:
                68:40:5f:55:2e:f5:9e:41:1c:ec:d3:22:fc:e2:2b:
                de:a8:81:81:d2:21:75:87:10:b0:67:46:21:73:08:
                d7:1b:21:38:66:2a:19:a1:ea:76:4a:ba:b0:90:f0:
                0e:0f:a6:2e:08:12:ad:fc:dc:05:b3:24:54:a0:da:
                05:03:16:a1:64:91:cd:f6:80:e6:dd:b6:69:d3:57:
                53:cb:69:c9:56:01:db:98:cb:c9:87:60:f4:af:86:
                f6:46:6f:d9:a3:1f:e8:d9:4c:79:4e:2f:4d:86:2f:
                20:fa:f8:1b:95:13:7c:59:2f:bd:4b:e7:7b:ae:84:
                7d:24:29:6d:3e:41:94:be:d3:fe:65:7d:1b:bd:69:
                99:c4:02:8f:48:57:93:4e:f6:42:d9:02:5a:90:cc:
                33:1f:be:57:13:76:75:e8:20:fb:e2:ca:13:d3:52:
                66:bd:77:ae:a6:d9:00:25:2b:b4:43:fb:c3:83:c7:
                77:2b:05:5c:d0:2b:00:88:ae:c5:10:69:77:b1:28:
                ea:b8:3e:ef:0c:6c:cf:71:5e:5d:97:86:56:b5:e1:
                fe:d8:5b:df:3e:6d:ba:4d:cd:39:64:f2:5e:d5:63:
                cd:4c:0d:4e:1e:98:63:6d:81:c3:cc:35:8c:b7:66:
                c8:67:19:4e:24:e3:72:77:12:60:78:5a:f1:4d:0a:
                17:ae:a3:f6:ff:34:bc:60:b4:c4:84:2f:3b:86:a3:
                a6:07:28:75:a6:c9:18:e0:63:20:bb:6f:5f:ac:93:
                3d:43:60:3b:41:67:90:a6:85:94:0c:76:ea:6b:5b:
                ca:bc:65:fa:a1:33:f2:a6:46:93:1b:46:45:81:57:
                3d:c7:ff:2d:b5:89:df:05:97:dd:ab:af:33:43:0e:
                4b:3f:db:e7:f8:fb:1e:68:71:a1:1d:cc:59:a8:e0:
                3c:d3:f0:7f:b9:78:46:de:bf:c3:3b:f7:b1:ba:cd:
                d9:14:cf:7f:d2:be:7c:b4:6c:df:84:13:a2:ff:6b:
                6a:b0:42:5d:d0:22:0d:c8:e1:f3:a1:8a:83:95:6d:
                54:52:a3:59:20:ae:6d:7e:f5:26:8c:8b:cd:fb:90:
                13:bd:bb:5d:68:35:c5:e0:51:a0:bd:b3:19:72:ed:
                42:d6:6f:9b:4b:d0:8e:df:91:5c:94:71:4f:02:bb:
                fa:b6:1d:9d:1b:d3:14:e4:85:e1:2c:b1:95:0e:41:
                0e:ca:27
            Exponent: 65537 (0x10001)
Attributes:
    (none)
Requested Extensions:
Signature Algorithm: sha256WithRSAEncryption
```

Step 6 : Root CA then creates the config file which contains the basic constraints and key usage details.



```
hd@rambhakta:~/rootCA
$ touch intCA_crt_sign.cnf
hd@rambhakta:~/rootCA
$ vim intCA_crt_sign.cnf
hd@rambhakta:~/rootCA
$ cat intCA_crt_sign.cnf
[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

Step 7 : Creating certificate with above generate config file and received csr file.

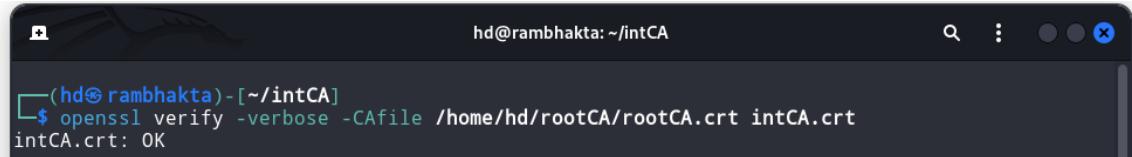


```
(hd@rambhakta)-[~/rootCA]
$ openssl x509 -req -in /home/hd/intCA/intCA.csr -CA rootCA.crt -CAkey root.pem -CAcreateserial -out /home/hd/intCA/intCA.crt -days 60 -extfile intCA_crt_sign.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = iTS CA 1R3
```

Step 8 : Creating certificate chain with rootCA.crt and IntCA.crt and sending it to IntCA.

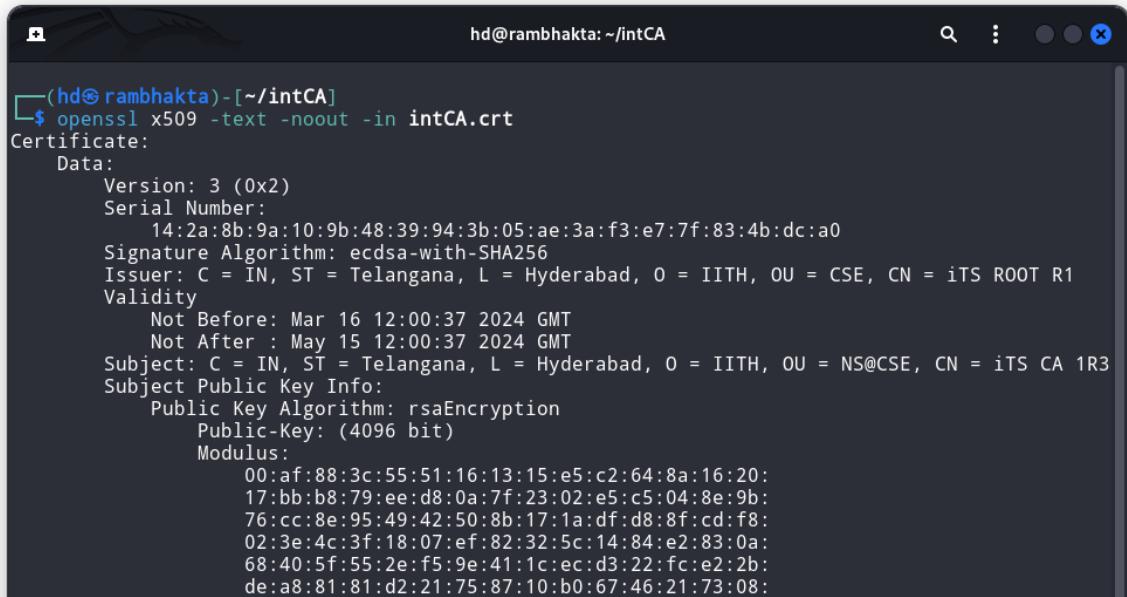
“`cat intCA.crt rootCA.crt > /home/hd/intCA/intCA_crt_chain.crt`”

Step 9 : Verifying authenticity of the certificate means verifying that it is signed by trusted authority.

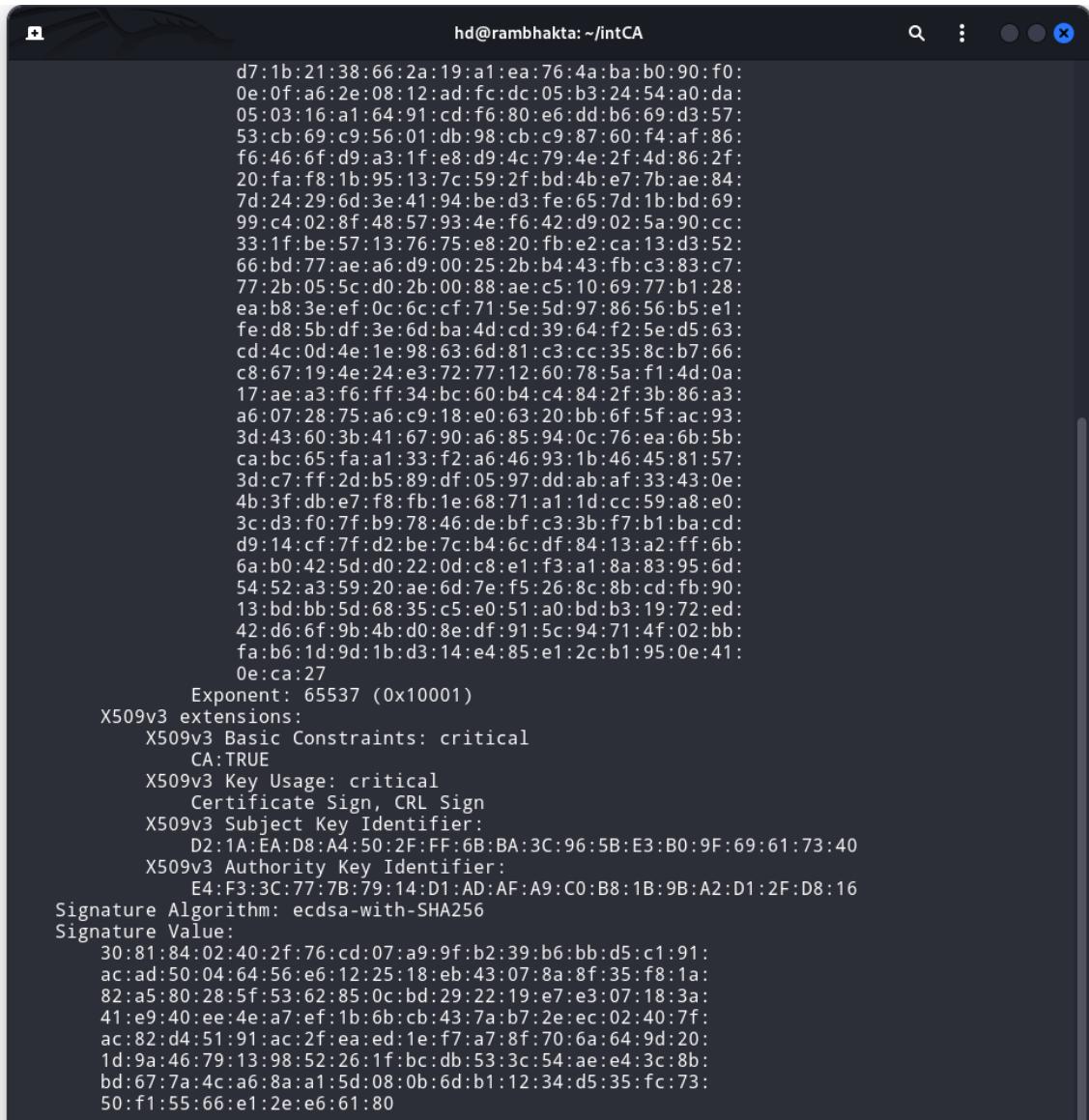


```
(hd@rambhakta)-[~/intCA]
$ openssl verify -verbose -CAfile /home/hd/rootCA/rootCA.crt intCA.crt
intCA.crt: OK
```

Output : Int CA's certificate signed and verified by Root CA.



```
(hd@rambhakta)-[~/intCA]
$ openssl x509 -text -in intCA.crt
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        14:2a:8b:9a:10:9b:48:39:94:3b:05:ae:3a:f3:e7:7f:83:4b:dc:a0
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = iTS ROOT R1
    Validity
        Not Before: Mar 16 12:00:37 2024 GMT
        Not After : May 15 12:00:37 2024 GMT
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = iTS CA 1R3
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (4096 bit)
            Modulus:
                00:af:88:3c:55:51:16:13:15:e5:c2:64:8a:16:20:
                17:bb:b8:79:ee:d8:0a:7f:23:02:e5:c5:04:8e:9b:
                76:cc:8e:95:49:42:50:8b:17:1a:df:d8:8f:cd:f8:
                02:3e:4c:3f:18:07:ef:82:32:5c:14:84:e2:83:0a:
                68:40:5f:55:2e:f5:9e:41:1c:ec:d3:22:fc:e2:2b:
                de:a8:81:81:d2:21:75:87:10:b0:67:46:21:73:08:
            Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    X509v3 Subject Key Identifier:
        D2:1A:EA:D8:A4:50:2F:FF:6B:BA:3C:96:5B:E3:B0:9F:69:61:73:40
    X509v3 Authority Key Identifier:
        E4:F3:3C:77:7B:79:14:D1:AD:AF:A9:C0:B8:1B:9B:A2:D1:2F:D8:16
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:81:84:02:40:2f:76:cd:07:a9:9f:b2:39:b6:bb:d5:c1:91:
    ac:ad:50:04:64:56:e6:12:25:18:eb:43:07:8a:8f:35:f8:1a:
    82:a5:80:28:5f:53:62:85:0c:bd:29:22:19:e7:e3:07:18:3a:
    41:e9:40:ee:4e:a7:ef:1b:6b:cb:43:7a:b7:2e:ec:02:40:7f:
    ac:82:d4:51:91:ac:2f:ea:ed:1e:f7:a7:8f:70:6a:64:9d:20:
    1d:9a:46:79:13:98:52:26:1f:bc:db:53:3c:54:ae:e4:3c:8b:
    bd:67:7a:4c:a6:8a:a1:5d:08:0b:6d:b1:12:34:d5:35:fc:73:
    50:f1:55:66:e1:2e:e6:61:80
```



Alice's Certificate Generation :

Step 1 : Generating 1024-bit RSA Key pair for Alice



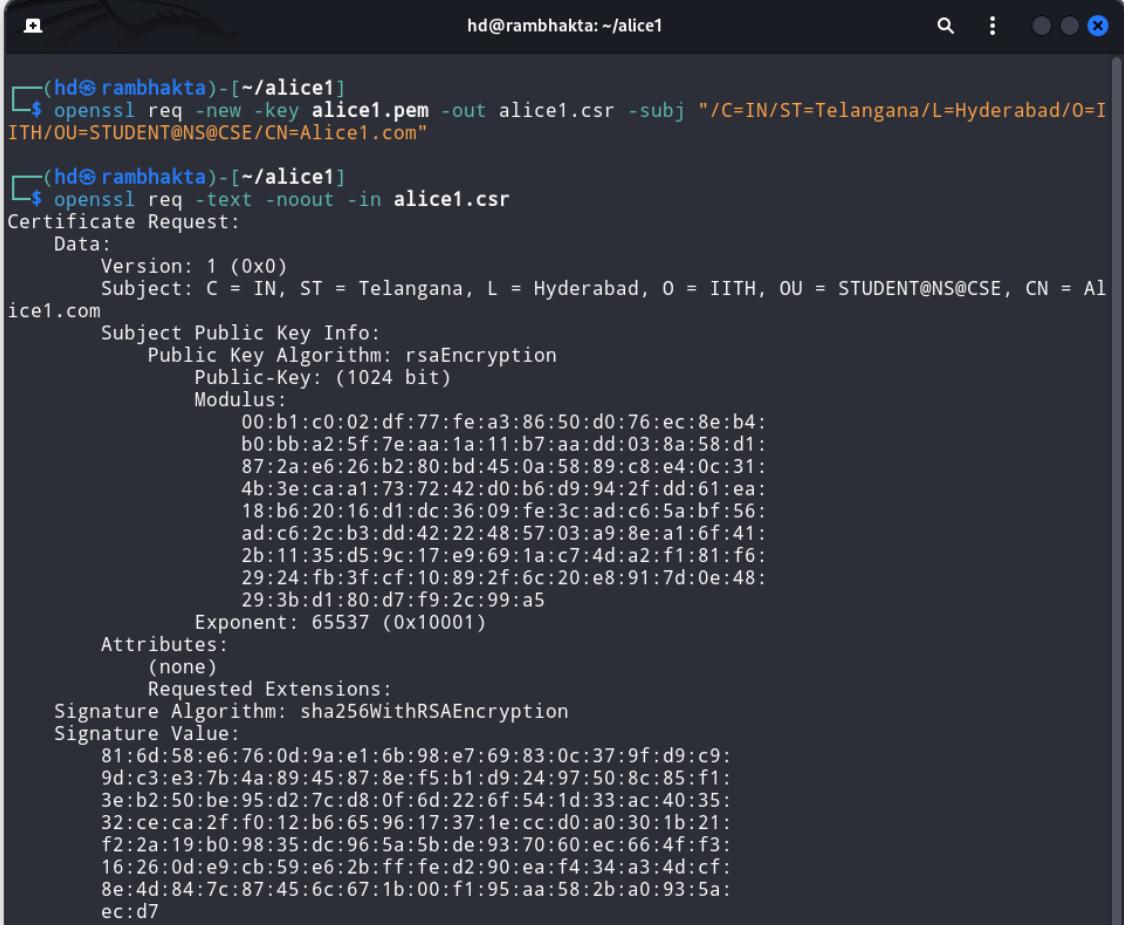
```
hd@rambhakta: ~/alice1
└─$ openssl genrsa -out alice1.pem 1024
└─$ cat alice1.pem
-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASAmEwggJdAgEAAoGBALHAAt93/q0GUNB2
7I60sLuiX36qGhG3qt0DiljRhymJrKAuUUKWInI5AwxSz7KoXNyQtC22ZQv3WHq
GLYgFtHcNgn+PK3GWr9WrcYss91CIkhXA6mOoW9BkxE11ZwX6Wkax02i8YH2KST7
P88QiS9sIOiRfQ5IKTvrGnf5LJmlAgMBAECgYEAgEz7K2Ztp53UxR31bFwwBW1/
6+nZ1y7FzJC+nMx/pGHZACKfjNB39rZNHzJJYchhwzuPAWejX6RTNEBnYnhGSKDye
hQbAZtpBp6j110glW0mIUSD2FTd3RhPgGR+1dIyclaDXj3g5Xvh3kFq3MsXwMrR
n6mzWYRmhQI9kmVN5j0CQQDdns+DVxgjDWpwt1bZtoC9apVNR4KY1/ZSaS8RGUNr
omXxaM4bj8oK1JCHWk3azW12SRzRoc5vP/LW93rOAPLAkEAzkE71h00hud07YIQ
29uncV3dYSqVmJCKpqY5G/KM0saHjrevMb9+heKeDTVDI0JDEkt700xpHaNKxjf
541KTwJAevWQF1V+QFsDzaCKeUC8M9WpbK7EXKO1R30UHGWLAEQ7CFLbhUPuCHh
A28ТИWJInnRNA0IbvpypJzVt+fTQJBAD0g/AdDZXZUuKderChWsHhINhjSXw/
nti5w0a3197zu6CbncjTpUTiggzL5EMQ9n4VuIGSmYpRZuIbxWWxG0kCQHeS/M4W
lz4CrCwnKH+0MgaciNsbwL/vX4LtzxYZCebnhhCepid8Q7v/Kiva++oW9KRJ3sbQ
BDuTNQ4mb+PV/0k=
-----END PRIVATE KEY-----
```

Step 2 : Extracting Public key from the above generated key pair



```
hd@rambhakta: ~/alice1
└─$ openssl rsa -in alice1.pem -pubout -out pub_key_alice1.pem
writing RSA key
└─$ cat pub_key_alice1.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxwALfd/6jh1DQduy0tLC7o19+
qhoRt6rdA4pY0Ycq5iaygl1FCliJy0QMMUs+yqFzckLQttmUL91h6hi2IBbR3DYJ
/jytxlq/Vg3GLLPdqIJIWopjqFvQSsRNdwCf+lpGsdNovGB9ikk+z/PEIkvbCD
kX00Sck70YDX+SyzpQIDAQAB
-----END PUBLIC KEY-----
```

Step 3 : Alice creates it certificate signing request and send it to Int CA

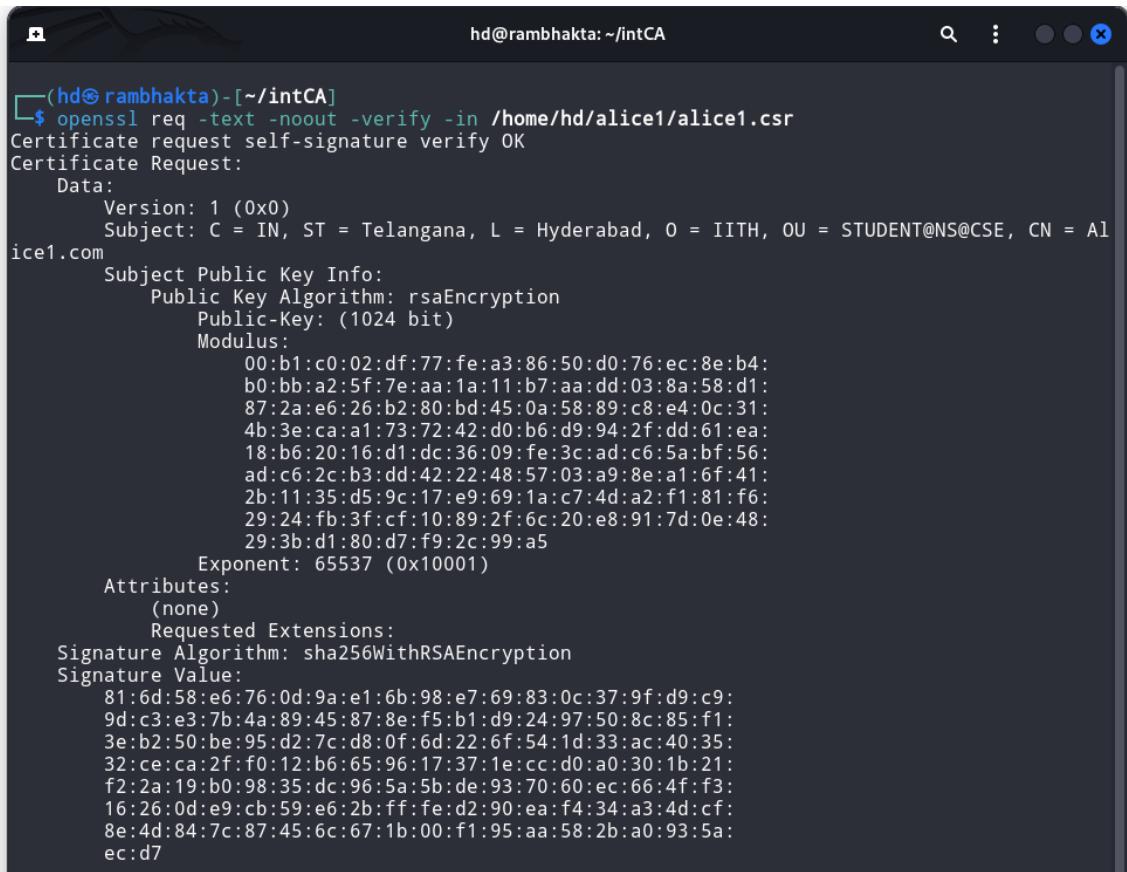


The screenshot shows a terminal window titled "hd@rambhakta: ~/alice1". The terminal displays the following command and its output:

```
(hd@rambhakta)-[~/alice1]
$ openssl req -new -key alice1.pem -out alice1.csr -subj "/C=IN/ST=Telangana/L=Hyderabad/O=IITH/OU=STUDENT@NS@CSE/CN=Alice1.com"

(hd@rambhakta)-[~/alice1]
$ openssl req -text -noout -in alice1.csr
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Alice1.com
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (1024 bit)
        Modulus:
            00:b1:c0:02:df:77:fe:a3:86:50:d0:76:ec:8e:b4:
            b0:bb:a2:5f:7e:aa:1a:11:b7:aa:dd:03:8a:58:d1:
            87:2a:e6:26:b2:80:bd:45:0a:58:89:c8:e4:0c:31:
            4b:3e:ca:a1:73:72:42:d0:b6:d9:94:2f:dd:61:ea:
            18:b6:20:16:d1:dc:36:09:fe:3c:ad:c6:5a:bf:56:
            ad:c6:2c:b3:dd:42:22:48:57:03:a9:8e:a1:6f:41:
            2b:11:35:d5:9c:17:e9:69:1a:c7:4d:a2:f1:81:f6:
            29:24:fb:3f:cf:10:89:2f:6c:20:e8:91:7d:0e:48:
            29:3b:d1:80:d7:f9:2c:99:a5
        Exponent: 65537 (0x10001)
    Attributes:
        (none)
    Requested Extensions:
        Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        81:6d:58:e6:76:0d:9a:e1:6b:98:e7:69:83:0c:37:9f:d9:c9:
        9d:c3:e3:7b:4a:89:45:87:8e:f5:b1:d9:24:97:50:8c:85:f1:
        3e:b2:50:be:95:d2:7c:d8:0f:6d:22:6f:54:1d:33:ac:40:35:
        32:ce:ca:2f:f0:12:b6:65:96:17:37:1e:cc:d0:a0:30:1b:21:
        f2:2a:19:b0:98:35:dc:96:5a:5b:de:93:70:60:ec:66:4f:f3:
        16:26:0d:e9:cb:59:e6:2b:ff:fe:d2:90:ea:f4:34:a3:4d:cf:
        8e:4d:84:7c:87:45:6c:67:1b:00:f1:95:aa:58:2b:a0:93:5a:
        ec:d7
```

Step 4 : Int CA verifies the received csr from Alice



```
hd@rambhakta: ~/intCA
$ openssl req -text -noout -verify -in /home/hd/alice1/alice1.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Alice1.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (1024 bit)
            Modulus:
                00:b1:c0:02:df:77:fe:a3:86:50:d0:76:ec:8e:b4:
                b0:bb:a2:5f:7e:aa:1a:11:b7:aa:dd:03:8a:58:d1:
                87:2a:e6:26:b2:80:bd:45:0a:58:89:c8:e4:0c:31:
                4b:3e:ca:a1:73:72:42:d0:b6:d9:94:2f:dd:61:ea:
                18:b6:20:16:d1:dc:36:09:fe:3c:ad:c6:5a:bf:56:
                ad:c6:2c:b3:dd:42:22:48:57:03:a9:8e:a1:f6:41:
                2b:11:35:d5:9c:17:e9:69:1a:c7:4d:a2:f1:81:f6:
                29:24:fb:3f:cf:10:89:2f:6c:20:e8:91:7d:0e:48:
                29:3b:d1:80:d7:f9:2c:99:a5
            Exponent: 65537 (0x10001)
        Attributes:
            (none)
        Requested Extensions:
            Signature Algorithm: sha256WithRSAEncryption
        Signature Value:
            81:6d:58:e6:76:0d:9a:e1:6b:98:e7:69:83:0c:37:9f:d9:c9:
            9d:c3:e3:7b:4a:89:45:87:8e:f5:b1:d9:24:97:50:8c:85:f1:
            3e:b2:50:be:95:d2:7c:d8:0f:6d:22:6f:54:1d:33:ac:40:35:
            32:ce:ca:2f:f0:12:b6:65:96:17:37:1e:cc:d0:a0:30:1b:21:
            f2:2a:19:b0:98:35:dc:96:5a:5b:de:93:70:60:ec:66:4f:f3:
            16:26:0d:e9:cb:59:e6:2b:ff:fe:d2:90:ea:f4:34:a3:4d:cf:
            8e:4d:84:7c:87:45:6c:67:1b:00:f1:95:aa:58:2b:a0:93:5a:
            ec:d7
```

Step 5 : Int CA then creates the config file which contains the basic constraints and key usage details.



```
hd@rambhakta: ~/intCA
$ touch end_user_crt_sign.cnf
$ vim end_user_crt_sign.cnf
[hd@rambhakta: ~/intCA]
$ cat end_user_crt_sign.cnf
[v3_ca]
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature,keyEncipherment,keyAgreement
```

Step 6 : Generating certificate for Alice using received csr and above generated config file.



```
hd@rambhakta: ~/intCA
$ openssl x509 -req -in /home/hd/alice1/alice1.csr -CA intCA_crt_chain.crt -CAkey intCA.pem
-CAcreateserial -out alice1.crt -days 30 -extfile end_user_crt_sign.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Alice1.com
```

Step 7 : Generating certificate chain by attaching rootCA.crt and intCA.crt and sending to Alice.

```
[~(hd@rambhakta)-[~/intCA]$ cat alice1.crt intCA_crt_chain.crt /home/hd/rootCA/rootCA.crt > /home/hd/alice1/alice1_crt_chain.crt
```

Step 8 : Alice verifies authenticity of the certificate means verifying that it is signed by trusted authority i.e. Int CA.

```
[~(hd@rambhakta)-[~/alice1]$ openssl verify -verbose -CAfile /home/hd/intCA/intCA_crt_chain.crt alice1_crt_chain.crt
alice1_crt_chain.crt: OK
```

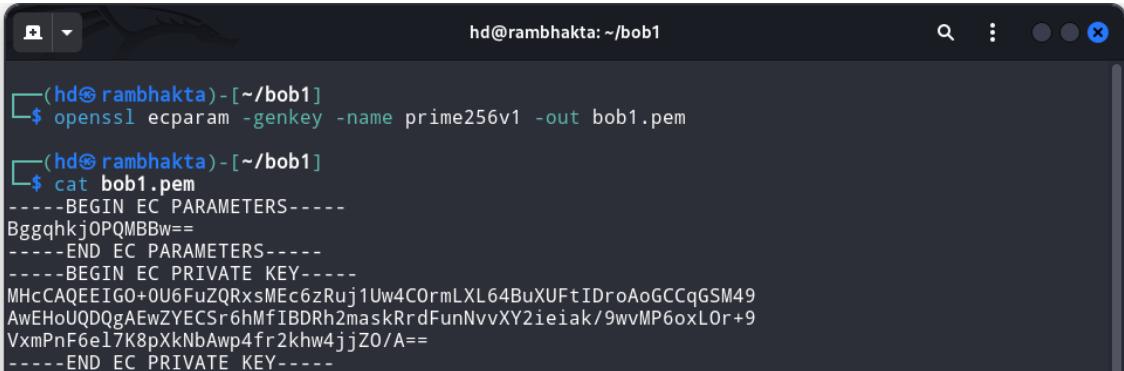
Output : Alice's certificate chain signed by Int CA.

```
hd@rambhakta:~/alice1$ openssl x509 -text -outout -verify -in alice1_crt_chain.crt
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      40:1b:47:eb:bc:8f:20:d5:ae:7e:98:3e:7e:0c:c8:b5:1c:32:70:ec
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = iTS CA 1R3
    Validity
      Not Before: Mar 16 13:02:41 2024 GMT
      Not After : Apr 15 13:02:41 2024 GMT
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Alice1.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        Public-Key: (1024 bit)
          Modulus:
            00:b1:c0:02:df:77:fe:a3:86:50:d0:76:ec:8e:b4:
            b0:bb:a2:5f:7e:aa:1a:11:b7:aa:dd:03:8a:58:d1:
            87:2a:e0:26:b2:80:bd:45:0a:58:89:c8:e4:0c:31:
            4b:3e:ca:a1:73:72:42:d0:b6:d9:94:2f:dd:61:ea:
            18:b6:20:16:d1:dc:36:09:fe:3c:ad:c6:5a:bf:56:
            ad:c6:2c:b3:dd:42:22:48:57:03:a9:8e:a1:6f:41:
            2b:11:35:d5:9c:17:e9:69:1a:c7:4d:a2:f1:81:f6:
            29:24:fb:3f:cf:10:89:2f:6c:20:e8:91:7d:0e:48:
            29:3b:d1:80:d7:f9:2c:99:a5
          Exponent: 65537 (0x10001)
      X509v3 extensions:
        X509v3 Basic Constraints: critical
          CA:FALSE
        X509v3 Key Usage: critical
          Digital Signature, Key Encipherment, Key Agreement
        X509v3 Subject Key Identifier:
          90:4F:8C:36:7D:8E:B1:52:FA:EC:96:BE:8D:CC:33:BC:8E:E9:8E:7A
        X509v3 Authority Key Identifier:
          D2:1A:EA:D8:A4:50:2F:FF:6B:BA:3C:96:5B:E3:B0:9F:69:61:73:40
```

```
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
97:e4:8c:cf:2b:3c:80:7b:94:31:c0:52:34:83:fe:6c:5d:1e:
cb:dc:a2:c9:1e:44:5c:56:0b:11:3c:0f:e1:a8:bc:8d:49:d1:
20:7c:15:e5:4a:c8:3b:eb:3a:7c:c5:02:33:60:21:e3:b1:0d:
4e:84:b9:e7:6f:9b:f7:a4:80:33:ac:ea:79:d3:2e:a4:17:7c:
4b:16:11:4c:a0:65:63:f0:9b:b9:3d:e7:91:45:c0:c3:a1:34:
42:f1:b6:26:58:f5:a0:b2:f0:74:b8:06:21:d4:e8:65:2a:bb:
d0:20:ab:a5:80:8a:49:08:4e:94:48:50:2a:3a:44:21:1d:15:
7d:40:d8:25:71:c2:f0:1c:9d:67:2a:76:57:bc:43:71:58:dc:
3a:c8:38:07:7e:cb:47:1c:8c:5a:15:b2:9e:37:55:fd:43:54:
7a:a0:6c:f9:51:ae:b0:f0:ac:9e:67:82:9b:88:23:9b:45:be:
2a:8e:d5:91:4c:ad:d9:8d:5d:bc:70:cf:60:60:b6:15:0d:8f:
d5:2d:0f:41:21:6a:52:dc:49:d0:59:b4:f5:89:1d:cb:fe:53:
d6:aa:64:c3:0c:98:99:ad:88:56:a6:1a:61:0f:c0:94:4c:aa:
9a:85:84:57:d4:32:ad:67:91:5d:9d:1d:cf:42:17:c8:28:18:
71:a5:62:fb:07:7e:23:bb:cb:cf:92:07:13:4e:c0:1f:07:65:
04:e1:7f:f4:e0:ca:84:75:4d:29:d4:f8:5c:63:5a:e1:ea:fc:
5c:b5:4d:80:a7:5b:a9:c7:fc:72:21:7f:fe:b0:c3:96:0d:c1:
68:8d:81:35:aa:36:44:f3:e8:9f:1d:6c:1b:02:a8:bd:67:3c:
c6:9e:29:bf:85:0d:f2:70:b0:32:c6:e9:da:56:64:e3:d1:4f:
1a:2d:4a:2e:99:1e:4d:1a:32:82:da:8f:96:d5:73:cf:e3:91:
9b:04:01:a5:df:dc:5c:c3:84:14:1e:bd:94:cd:c5:81:88:0a:
a9:dd:c7:4a:fa:89:40:e3:b5:25:89:8c:1d:89:89:b5:6b:08:
cb:67:0e:cc:dc:62:a7:03:92:68:3e:bf:96:22:f0:5a:7b:10:
af:60:d9:8a:52:af:4f:6a:db:cb:59:cf:a7:2a:b2:1b:ec:07:
50:4f:bc:10:6e:24:af:84:1e:3d:e8:7f:38:80:0e:ff:66:b7:
7b:9b:10:5c:53:1f:95:fe:e6:c3:8c:78:ea:bd:3e:02:28:4a:
e6:05:74:e2:0e:d3:c0:a6:8c:d9:02:43:79:5f:c2:22:3b:29:
d7:e9:ab:a1:d1:73:a0:23:fd:3b:c4:26:f2:83:82:e8:bc:77:
b1:1c:7e:ae:c9:44:47:8f
```

Bob's Certificate Generation :

Step 1 : Generating 256-bit ECC Key pair for Bob.



```
hd@rambhakta: ~/bob1
[hd@rambhakta ~]$ openssl ecparam -genkey -name prime256v1 -out bob1.pem
[hd@rambhakta ~]$ cat bob1.pem
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIGO+OU6FuZQRxsMEc6zRuj1Uw4COrmLXL64BuXUftIDroAoGCCqGSM49
AwEhoUQDQgAEwZYECSr6hMFIBDRh2maskRrdFunNvvXY2ieiak/9wvMP6oxL0r+9
VxmPnF6e17K8pXkNbAwp4fr2khw4jjZ0/A==
-----END EC PRIVATE KEY-----
```

Step 2 : Extracting Public key from the above generated key pair.



```
hd@rambhakta: ~/bob1
[hd@rambhakta ~]$ openssl ec -in bob1.pem -pubout -out pub_key_bob1.pem
read EC key
writing EC key
[hd@rambhakta ~]$ cat pub_key_bob1.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEwZYECSr6hMFIBDRh2maskRrdFunN
vvXY2ieiak/9wvMP6oxL0r+9VxmPnF6e17K8pXkNbAwp4fr2khw4jjZ0/A==
-----END PUBLIC KEY-----
```

Step 3 : Bob creates its certificate signing request and sends it to Int CA.

```
hd@rambhakta:~/bob1
└─$ openssl req -new -key bob1.pem -out bob1.csr -subj "/C=IN/ST=Telangana/L=Hyderabad/O=IITH/OU=STUDENT@NS@CSE/CN=Bob1.com"

└─$ openssl req -text -noout -in bob1.csr
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Bob1.com
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
            04:c1:96:04:09:2a:fa:84:c7:c8:04:34:61:da:66:
            ac:91:1a:dd:16:e9:cd:be:f5:d8:da:27:a2:6a:4f:
            fd:c2:f3:0f:ea:8c:4b:3a:bf:bd:57:19:8f:9c:5e:
            9e:97:b2:bc:a5:79:0d:6c:0c:29:e1:fa:f6:92:1c:
            38:8e:36:4e:fc
        ASN1 OID: prime256v1
        NIST CURVE: P-256
    Attributes:
        (none)
    Requested Extensions:
        Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:6a:c4:b3:2a:4f:fd:0b:94:01:9d:a9:d3:86:a5:
        6d:bb:2c:76:07:0d:89:6f:28:5f:77:d5:14:42:a5:63:aa:2c:
        02:20:47:db:cd:27:c2:c4:6b:7d:42:dd:9e:67:38:8a:0b:4f:
        c5:c6:fb:50:2a:d0:ce:74:98:09:d7:57:f2:d8:6d:e0
```

Step 4 : Int CA verifies the received CSR from Bob.

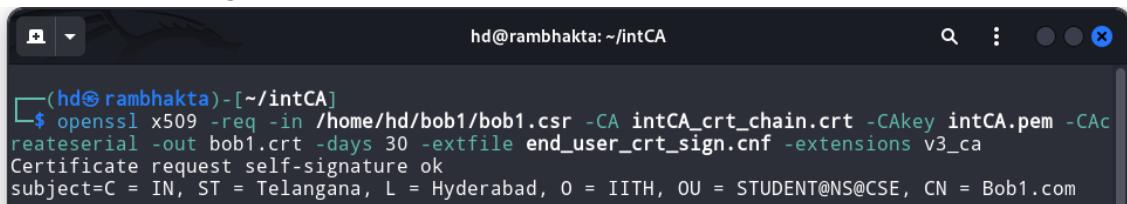
```
hd@rambhakta:~/intCA
└─$ openssl req -text -noout -verify -in /home/hd/bob1/bob1.csr
Certificate request self-signature verify OK
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Bob1.com
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
            04:c1:96:04:09:2a:fa:84:c7:c8:04:34:61:da:66:
            ac:91:1a:dd:16:e9:cd:be:f5:d8:da:27:a2:6a:4f:
            fd:c2:f3:0f:ea:8c:4b:3a:bf:bd:57:19:8f:9c:5e:
            9e:97:b2:bc:a5:79:0d:6c:0c:29:e1:fa:f6:92:1c:
            38:8e:36:4e:fc
        ASN1 OID: prime256v1
        NIST CURVE: P-256
    Attributes:
        (none)
    Requested Extensions:
        Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:6a:c4:b3:2a:4f:fd:0b:94:01:9d:a9:d3:86:a5:
        6d:bb:2c:76:07:0d:89:6f:28:5f:77:d5:14:42:a5:63:aa:2c:
        02:20:47:db:cd:27:c2:c4:6b:7d:42:dd:9e:67:38:8a:0b:4f:
        c5:c6:fb:50:2a:d0:ce:74:98:09:d7:57:f2:d8:6d:e0
```

Step 5 : Int CA then creates the config file which contains the basic constraints and key usage details.



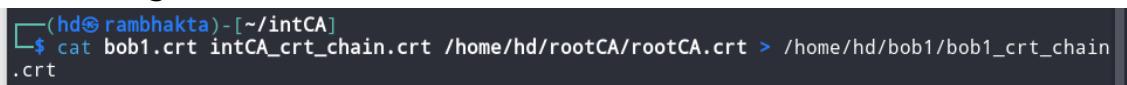
```
hd@rambhakta: ~/intCA
└─(hd@rambhakta)-[~/intCA]
$ touch end_user_crt_sign.cnf
└─(hd@rambhakta)-[~/intCA]
$ vim end_user_crt_sign.cnf
└─(hd@rambhakta)-[~/intCA]
$ cat end_user_crt_sign.cnf
[v3_ca]
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature,keyEncipherment,keyAgreement
```

Step 6 : Generating certificate for Bob using received csr and above generated config file.



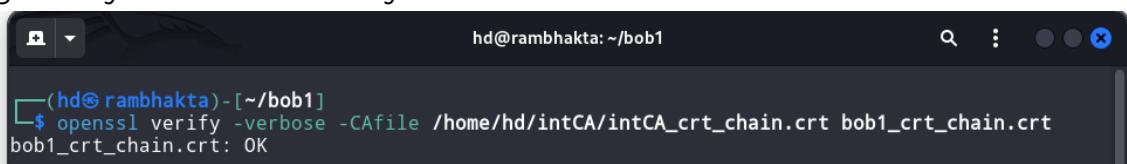
```
hd@rambhakta: ~/intCA
└─(hd@rambhakta)-[~/intCA]
$ openssl x509 -req -in /home/hd/bob1/bob1.csr -CA intCA_crt_chain.crt -CAkey intCA.pem -CAcreateserial -out bob1.crt -days 30 -extfile end_user_crt_sign.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Bob1.com
```

Step 7 : Generating certificate chain by attaching rootCA.crt and intCA.crt and sending to Bob.



```
hd@rambhakta: ~/intCA
└─(hd@rambhakta)-[~/intCA]
$ cat bob1.crt intCA_crt_chain.crt /home/hd/rootCA/rootCA.crt > /home/hd/bob1/bob1_crt_chain.crt
```

Step 8 : Bob verifies authenticity of the certificate means verifying that it is signed by trusted authority i.e. Int CA.



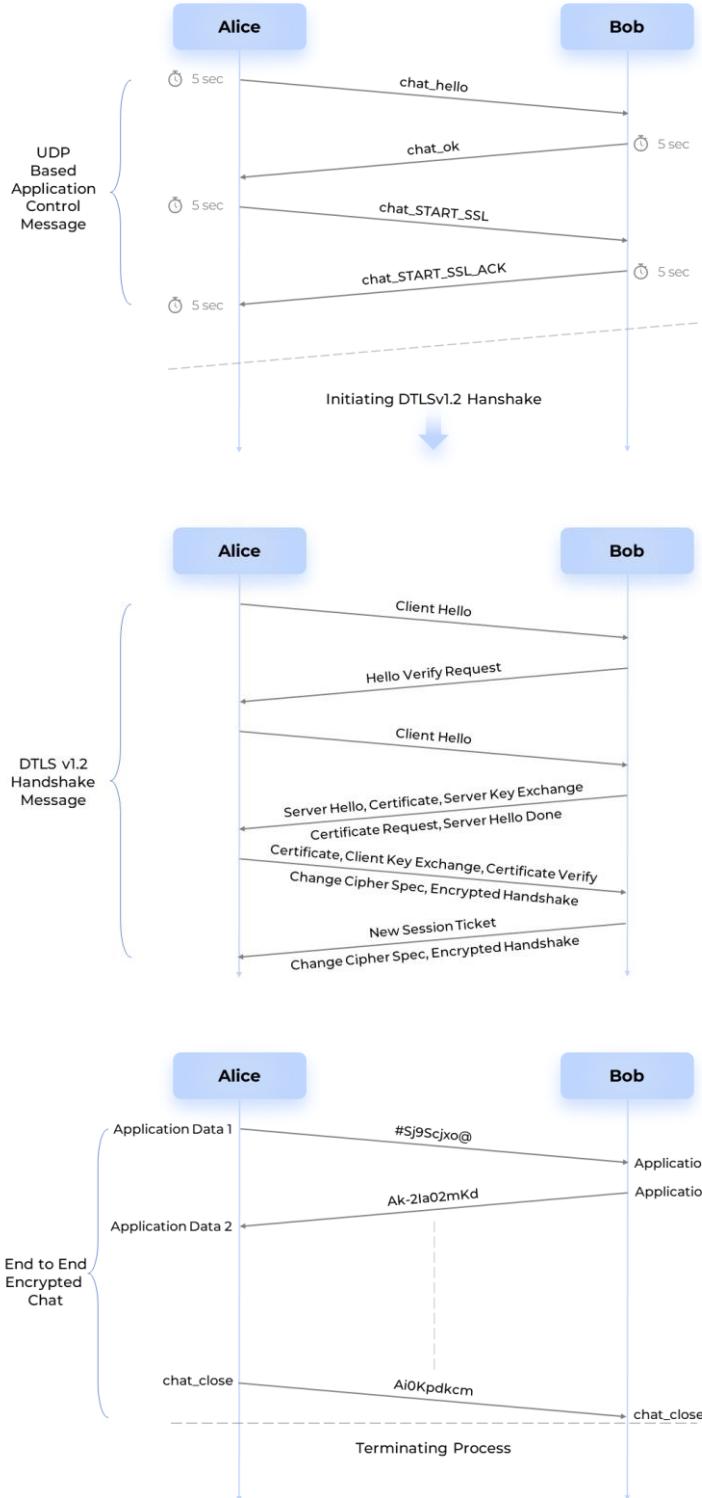
```
hd@rambhakta: ~/bob1
└─(hd@rambhakta)-[~/bob1]
$ openssl verify -verbose -CAfile /home/hd/intCA/intCA_crt_chain.crt bob1_crt_chain.crt
bob1_crt_chain.crt: OK
```

Output : Bob's certificate chain signed by Int CA.

```
hd@rambhakta:~/bob1
$ openssl x509 -text -noout -verify -in bob1_crt_chain.crt
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        40:1b:47:eb:bc:8f:20:d5:ae:7e:98:3e:7e:0c:c8:b5:1c:32:70:f0
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = NS@CSE, CN = iTS CA 1R3
    Validity
        Not Before: Apr 5 15:40:23 2024 GMT
        Not After : May 5 15:40:23 2024 GMT
    Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = STUDENT@NS@CSE, CN = Bob
b1.com
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
            pub:
                04:c1:96:04:09:2a:fa:84:c7:c8:04:34:61:da:66:
                ac:91:1a:dd:16:e9:cd:be:f5:d8:da:27:a2:6a:4f:
                fd:c2:f3:0f:ea:8c:4b:3a:bf:bd:57:19:8f:9c:5e:
                9e:97:b2:bc:a5:79:0d:6c:0c:29:e1:fa:f6:92:1c:
                38:8e:36:4e:fc
            ASN1 OID: prime256v1
            NIST CURVE: P-256
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment, Key Agreement
    X509v3 Subject Key Identifier:
        7A:45:24:34:C5:49:D1:44:39:F1:28:E9:96:E3:7F:98:A5:1F:BE:27
    X509v3 Authority Key Identifier:
        D2:1A:EA:D8:A4:50:2F:FF:6B:BA:3C:96:5B:E3:B0:9F:69:61:73:40
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
51:bd:33:aa:90:29:73:5e:3f:fd:26:ba:a7:30:ff:cd:05:2a:
aa:6d:55:ee:11:23:56:5d:5d:d4:e3:45:95:fd:78:8f:45:b8:
2e:c2:77:fc:96:c1:85:d6:26:1a:47:23:9d:9a:62:b0:b0:75:
06:d8:56:c9:a1:1a:2d:82:fc:f2:4d:23:4f:15:5a:b4:26:3e:
86:ea:76:67:ce:01:50:5a:01:e2:9d:43:c2:9b:ed:32:ee:35:
c7:a1:6c:60:78:e6:09:65:1f:0a:f8:35:76:b8:e9:1f:fe:8c:
5e:05:a7:1d:c2:45:33:c4:6a:98:89:93:50:ea:7b:1b:b9:07:
2a:51:12:51:80:1a:8c:cc:80:e2:2d:45:e9:a6:34:07:eb:a7:
f1:12:fb:e6:53:7e:2a:5e:5a:6b:1c:82:17:ef:79:d4:bf:37:
f2:19:4e:8e:48:0c:48:f9:53:9d:42:91:a4:91:0e:76:09:b2:
b5:1e:60:1b:80:38:bb:13:3c:14:02:dc:50:26:b6:8a:62:b8:
a6:5f:88:77:b0:60:96:df:48:8e:2c:20:45:76:e5:cb:3f:e5:
07:b6:96:d3:42:60:6b:c1:b8:ad:ed:de:56:0e:6b:15:d5:a3:
a6:0f:37:b5:29:c9:73:ea:c1:cc:7a:d1:97:2e:b8:91:0b:c2:
fe:62:76:22:8b:ac:97:eb:ec:59:4e:e0:74:d4:37:22:b6:42:
55:82:05:06:85:05:08:db:b9:35:55:7b:75:9b:89:b9:38:ca:5c:
82:f2:0a:ca:d1:8b:ab:86:8a:96:ed:94:b9:e4:51:2d:ff:ae:
e8:61:5e:52:92:90:a3:b1:ba:ce:0d:7f:58:46:eb:cd:b3:85:
c5:c2:e4:39:32:4d:fc:c9:e1:e9:11:8e:2b:a8:b2:15:b1:46:
7a:65:e4:9f:12:9b:7e:1f:8c:36:4e:78:6c:6d:ff:0a:48:f2:
ee:4c:2c:b4:ec:bb:31:bf:db:7a:0d:d3:c2:a9:b9:da:be:37:
59:2f:f5:03:99:7e:77:2f:21:cb:12:0c:b7:4f:9a:f1:4c:1f:
f2:6b:96:28:53:78:8d:ff:13:18:39:a3:fa:d4:d7:a3:58:78:
ff:8f:17:0a:e6:c3:fd:fb:67:ae:55:f5:ef:2a:68:38:e9:35:
18:cd:c3:20:b4:12:7a:86:19:87:c4:02:0e:d8:14:da:e8:f3:
40:93:e8:bb:40:5d:39:e9:a2:25:c0:68:e0:da:1b:eb:5e:db:
1e:15:79:ca:4c:d1:e4:f3:25:d6:97:09:3a:89:a8:4d:dd:e1:
53:40:d6:c6:27:29:4b:75:6a:ae:4d:4d:e7:10:d7:2f:15:ec:
38:1b:92:17:62:7c:4a:7d
```

Task 2 – Secure Chat Application

Application Flow :



Above exceptional flow diagram created by one of our team member shows the basic working of our chat application.

Assuming Client & Server both have certificates of Root CA and Int CA preloaded in their respective trust stores.

Some code snippets :

Below code snippet is responsible for loading necessary modules under openssl library, and for creating the SSL context of client & server.

```
28 void initialize_openssl() {
29     SSL_library_init();
30     SSL_load_error_strings();
31     OpenSSL_add_ssl_algorithms();
32     ERR_load_crypto_strings();
33 }
34
35 SSL_CTX* create_server_context() {
36     const SSL_METHOD *method = DTLSv1_2_server_method();
37     SSL_CTX *ctx = SSL_CTX_new(method);
38     if (!ctx) {
39         perror("Unable to create SSL context");
40         ERR_print_errors_fp(stderr);
41         exit(EXIT_FAILURE);
42     }
43     return ctx;
44 }
45
46 SSL_CTX* create_client_context() {
47     const SSL_METHOD* method = DTLSv1_2_client_method();
48     SSL_CTX* ctx = SSL_CTX_new(method);
49     if (!ctx) {
50         std::cerr << "Unable to create SSL context" << std::endl;
51         ERR_print_errors_fp(stderr);
52         exit(EXIT_FAILURE);
53     }
54     return ctx;
55 }
56
```

Below code snippet helps to set socket in non-blocking mode which here helps as a part of retries and retransmission functionality.

```
105 void setup_nonblocking(){
106     int fcntl_check = fcntl(socket_descriptor, F_GETFL, 0);
107     if (fcntl_check == -1) {
108         perror("F_GETFL error\n");
109         exit(1);
110     }
111
112     if (fcntl(socket_descriptor, F_SETFL, fcntl_check | O_NONBLOCK) == -1) {
113         perror("F_SETFL error\n");
114         exit(1);
115     }
116 }
117
```

Now as part of authenticated and encrypted chatting we are verifying keys and certificate of client and server as shown in below code snippet.

```

66
67 void configure_context(SSL_CTX *ctx, const char * crt_path, const char * key_path, const char * CA_crt_path) {
68     SSL_CTX_set_ecdh_auto(ctx, 1);
69     if (SSL_CTX_use_certificate_file(ctx, crt_path, SSL_FILETYPE_PEM) <= 0) {
70         ERR_print_errors_fp(stderr);
71         exit(EXIT_FAILURE);
72     }
73
74     if (SSL_CTX_use_PrivateKey_file(ctx, key_path, SSL_FILETYPE_PEM) <= 0 ) {
75         ERR_print_errors_fp(stderr);
76         exit(EXIT_FAILURE);
77     }
78
79     if(!SSL_CTX_check_private_key(ctx)){
80         cout << " Private Key Verification failed!\n";
81         exit(0);
82     }
83
84     if (!SSL_CTX_load_verify_locations(ctx, CA_crt_path, NULL)) {
85         ERR_print_errors_fp(stderr);
86         exit(EXIT_FAILURE);
87     }
88
89     SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
90
91     SSL_CTX_set_security_level(ctx, 1);
92     SSL_CTX_set_cipher_list(ctx,"ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384");
93     SSL_CTX_set_mode(ctx,SSL_MODE_AUTO_RETRY);
94     SSL_CTX_set_session_cache_mode(ctx,SSL_SESS_CACHE_OFF);
95     SSL_CTX_set_verify(ctx,SSL_VERIFY_PEER,NULL);
96     SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_SERVER);
97     SSL_CTX_set_session_id_context(ctx, (const unsigned char *)"DTLS", strlen("DTLS"));
98
99     if(mode=='s'){
100         SSL_CTX_set_cookie_generate_cb(ctx,generateCookie);
101         SSL_CTX_set_cookie_verify_cb(ctx,&verifyCookie);
102     }
103 }
```

Now as a part of reliability after setting socket in not blocking mode we are using select function to listen if any activity is their till 5 sec or timeout it will wait otherwise it print Timeout and start timer again and wait for incoming activity. If any message arrives then it exit from condition and go to printing and rest part.

```

    fd_set readfds;
FD_ZERO(&readfds);
FD_SET(socket_descriptor, &readfds);
timeout.tv_sec = 5;
timeout.tv_usec = 0;
int activity = select(socket_descriptor + 1, &readfds, NULL, NULL, &timeout);
if (activity < 0) {
    perror("Error in select\n");
    exit(EXIT_FAILURE);
} else if (activity == 0) {
    printf("Timeout occurred. No data received.\n");
    continue;
}
int received_message_len = SSL_read(ssl, buffer, sizeof(buffer) - 1);
```

Client-Side actual output of secure chat application.

```
root@alice1:~# ./secure_chat_app -c bob1
Client started...

Client connecting to server bob1 with 172.31.0.3

You : chat_hello sent
Server : chat_ok received
You : chat_START_SSL sent
Server : chat_START_SSL_ACK received
Client OpenSSL initialized
Client Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
You : hii
Server: ok
You : chat_close
Client closed
root@alice1:~#
```

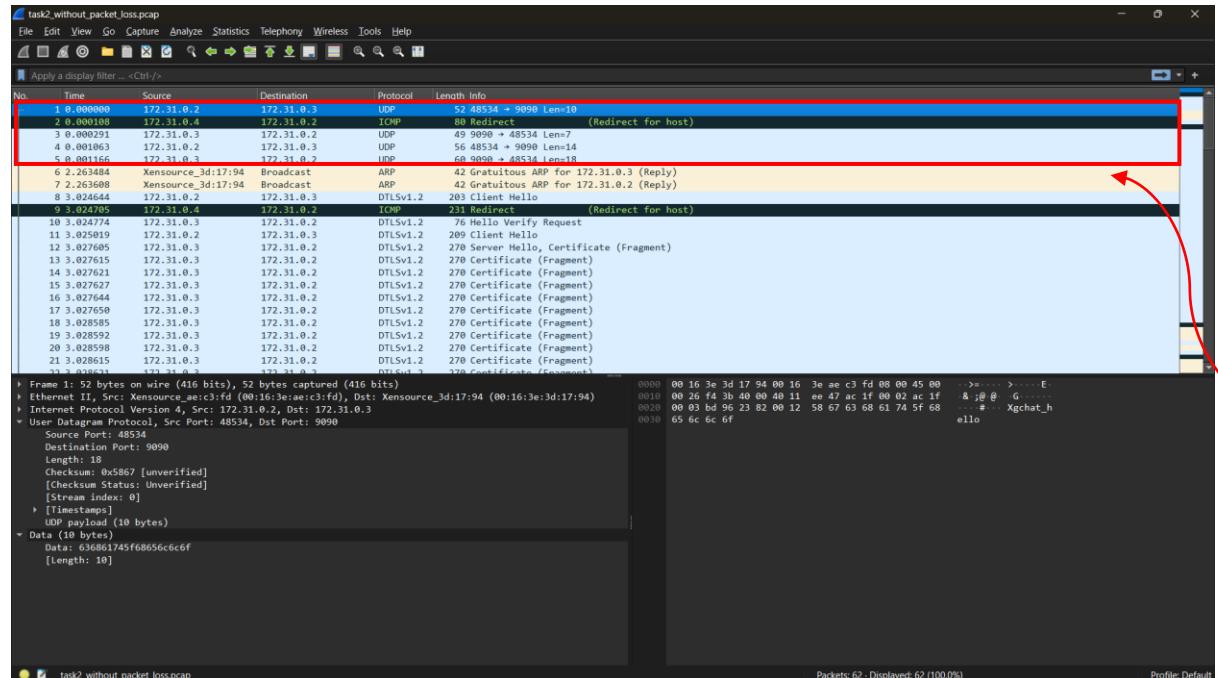
Server-side actual output of secure chat application.

```
root@bob1:~# ./secure_chat_app -s
Server started...

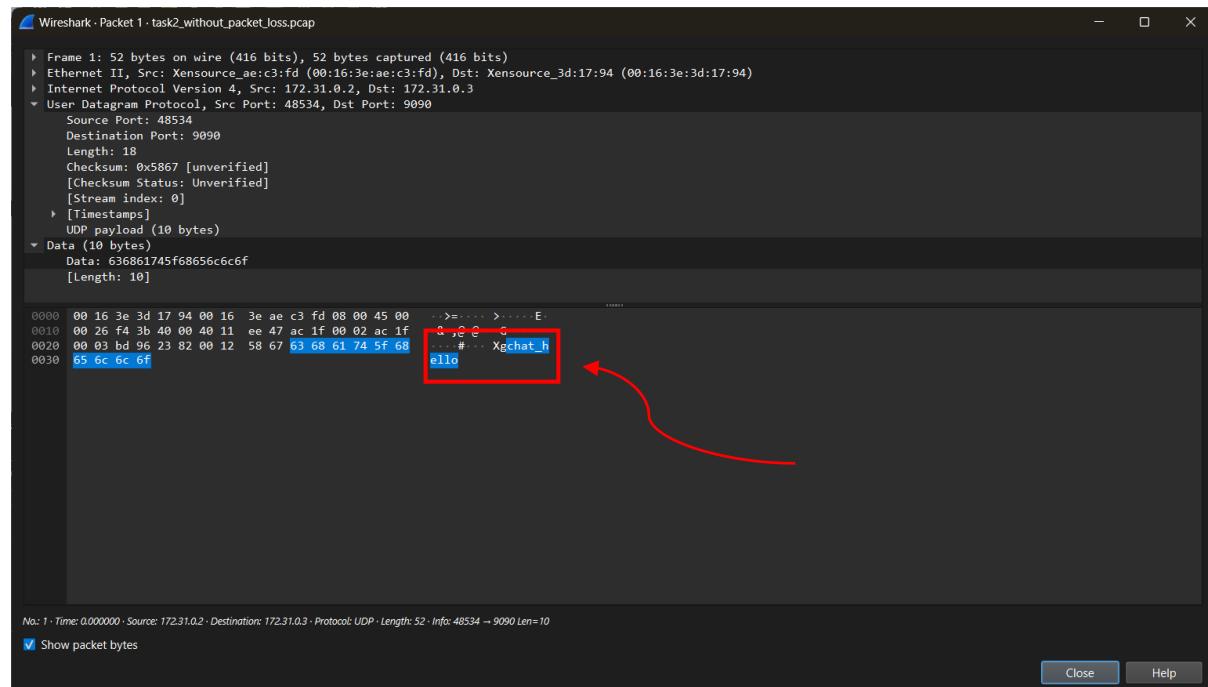
Client : chat_hello received
Server (You) : chat_ok sent
Client : chat_START_SSL received
Server (You) : chat_START_SSL_ACK sent
Server OpenSSL initialized
Server Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
Client: hii
Server (You) : ok
Client: chat_close
Server closed
root@bob1:~#
```

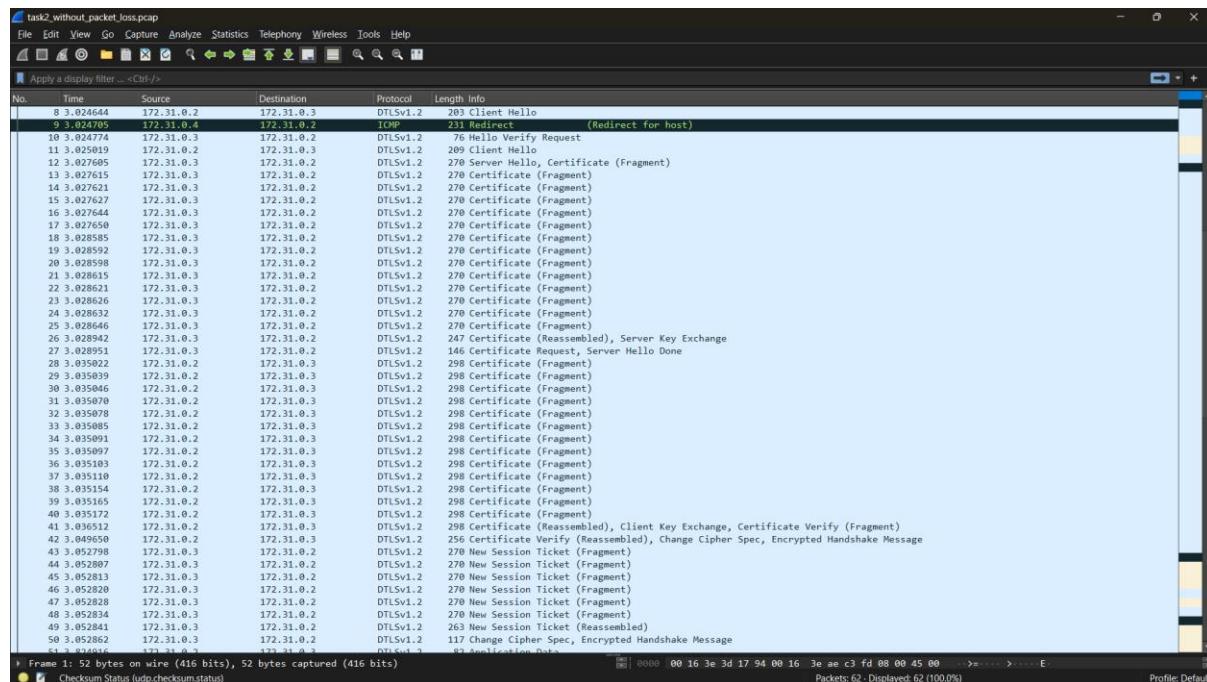
As our control message are goes under UDP socket, we can see the message by capturing it in packet capturing tool. Here below image shows the captured packets.



Below image shows actual control string message.

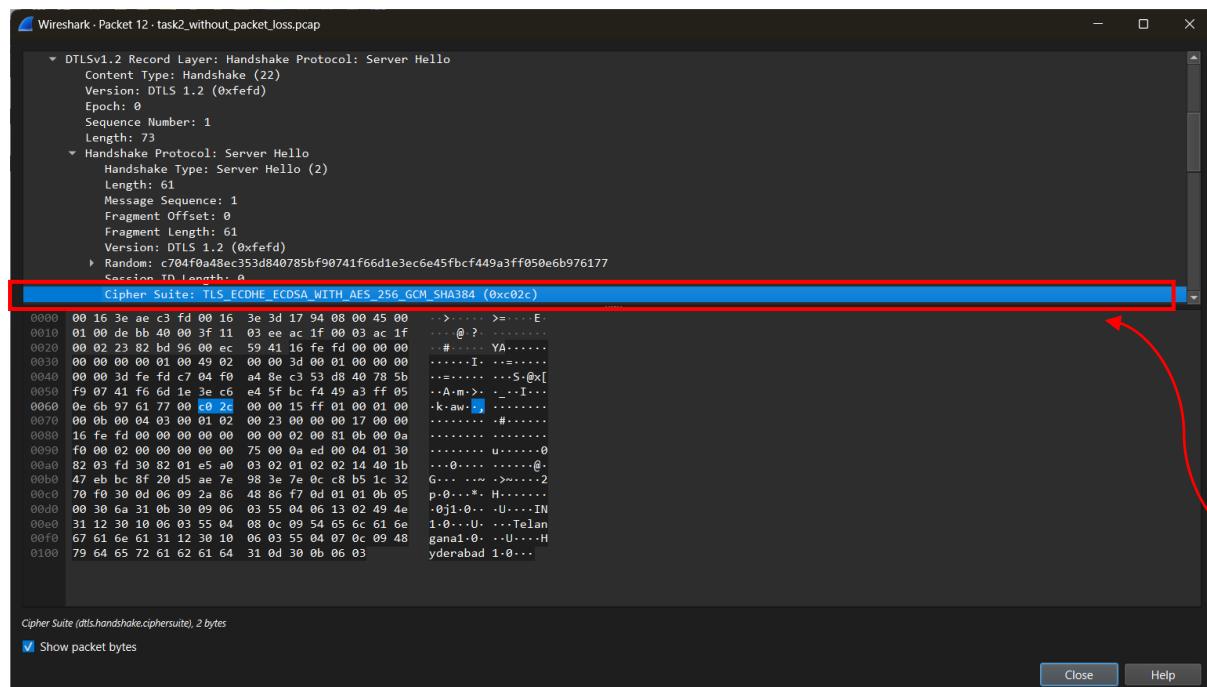


Below image shows the packet capture of DTLS v1.2 handshake.

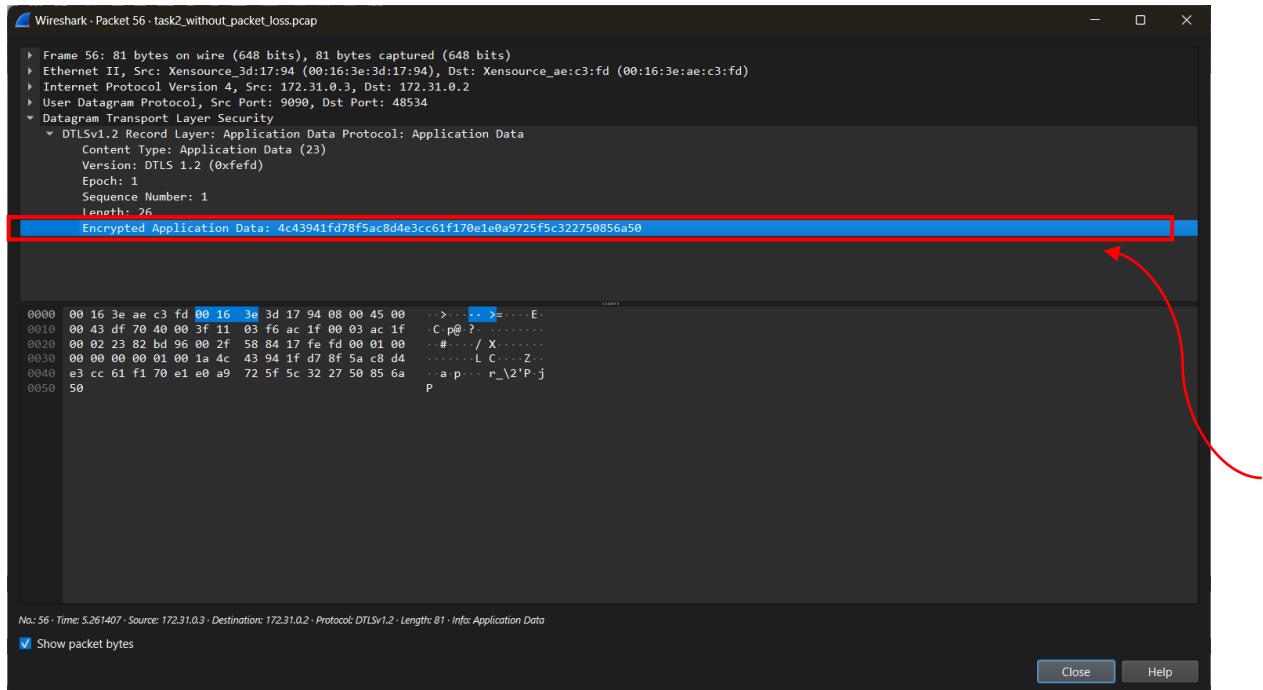


As a part of **Session Resumption** our server is sending New Session Ticket to client.

Here in below image we can see client and server agrees on **Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384** which is a one of the best cipher suite which provides **Perfect Forward Secrecy**.



Below given image shows the Packet capture of Application data which is shared after establishment of DTLS v1.2 pipe. We can see the data is in encrypted form.



Comparing DTLS 1.2 & TLS 1.2

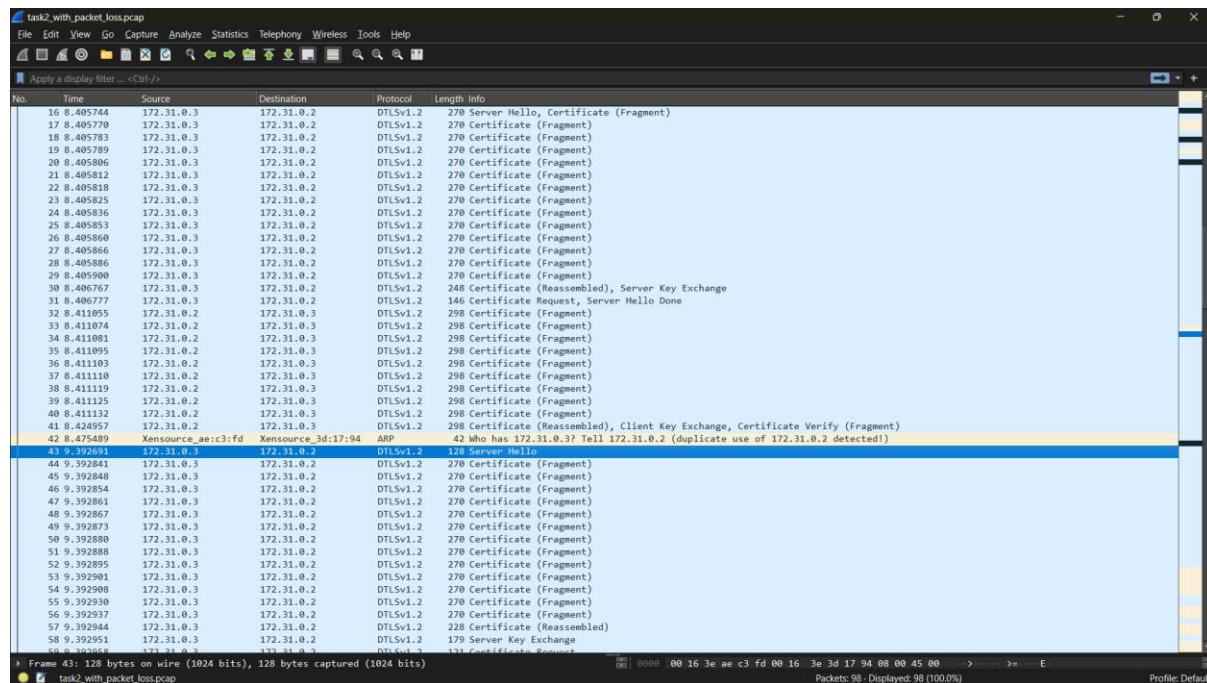
	DTLS v1.2	TLS v1.2
Base Protocol	It works over UDP.	It works over TCP.
Delay	It is slow compared to TLS as it works over UDP it has to manage loss and retransmission.	It is fast compared to DTLS as underlying layer TCP handles loss and congestion control.
Handshake message dependency	Client and server can share handshake message independently which helps in out-of-order delivery of packets.	Client and Server handshake message are dependent and must be shared in a serial form.
Usage	Used for real-time applications like streaming.	Used where the reliability is a necessity.

For **testing reliability** of our application, add packet loss using netem tool with Traffic control (tc).

Command : sudo tc qdisc add dev [interface] root netem loss [loss in percentage]

```
root@alice1:~#
ubuntu@cs23mtech11029:~$ lxc exec alice1 bash
root@alice1:~# sudo tc qdisc change dev eth0 root netem loss 20%
root@alice1:~#
```

We can see in the below image that the Packets from Server – Client was dropped and our chat app is able to retransmit the packets after timeout because of which Server Hello, and certificate sharing is done twice here.



How reliability works here in our application :

In our case in simple UDP communication we have added the timer while sending message from both server and client side. Now when any one sends the message timer starts with timeout = 5 sec, and if no ack or msg received within that time period, that particular message is retransmitted.

In DTLS communication we setup Nonblocking mode in our socket so that if message is coming then it receives else it will re-enter the loop and try to receive message for timeout period and if message not received then it prints Timeout and re-enter loop and received again for timeout period.

Client-Side actual output of secure chat application with 20% packet loss.

```
root@alice1:~# ./secure_chat_app -c bob1
Client started...

Client connecting to server bob1 with 172.31.0.3

You : chat_hello sent
Server : chat_ok received
You : chat_START_SSL sent
Timeout occurred. No data received.
You : chat_START_SSL sent
Server : chat_START_SSL_ACK received
Client OpenSSL initialized
Client Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
You : hi
Server: fff
You : ddd
Server: chat_close
Client closed
root@alice1:~#
```

Server-Side actual output of secure chat application with 20% packet loss.

```
root@bob1:~# ./secure_chat_app -s
Server started...

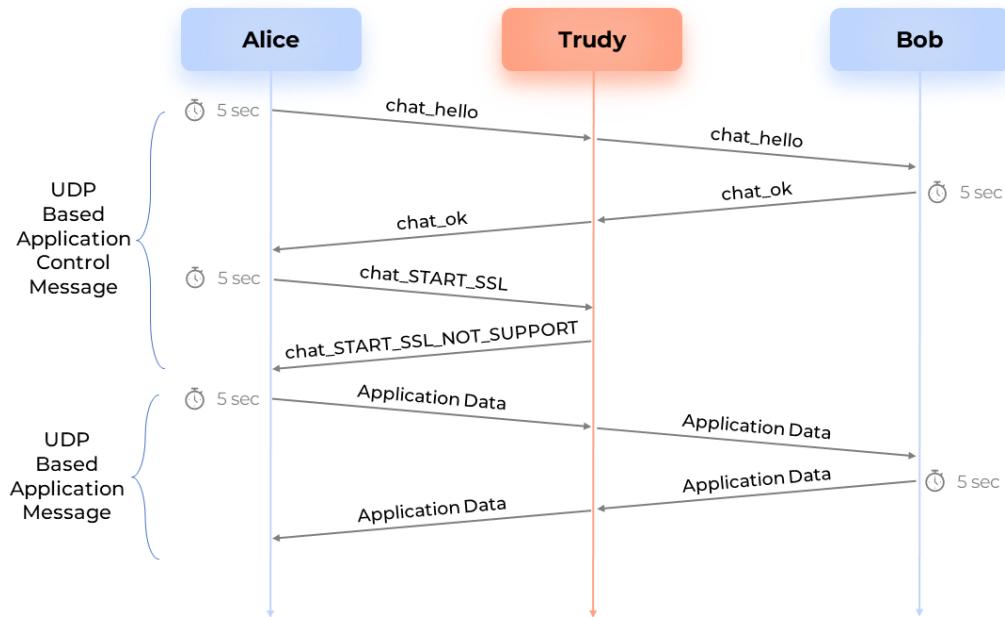
Client : chat_hello received
Server (You) : chat_ok sent
Timeout occurred. No data received.
Client : chat_START_SSL received
Server (You) : chat_START_SSL_ACK sent
Server OpenSSL initialized
Server Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
Client: hi
Server (You) : fff
Client: ddd
Server (You) : chat_close
Server closed
root@bob1:~#
```

Task 3 – SSL Downgrade Attack for eavesdropping

Assuming Trudy has successfully poisoned /etc/hosts file.

Application Flow :



Above exceptional flow diagram created by one of our team member shows the basic working of our chat application with SSL downgrade attack.

Here Trudy intercept all messages from Alice to Bob, Now when Alice sends the `chat_START_SSL`, Trudy block that message and reply it as `chat_START_SSL_NOT_SUPPORTED` which Alice understand that it is coming from Bob and Bob doesn't able to support SSL communication so Alice continues with the UDP open communication.

This code snippet given below consist of main logic behind the downgrading at Trudy's side where it checks every received message and if chat_START_SSL is received it sends chat_START_SSL_NOT_SUPPORTED.

```

received_message_len = recvfrom(fake_server_socket_descriptor, (char *)buffer, MAX_BUFFER_SIZE-1, 0, (struct sockaddr *) &client_address, (socklen_t *)&client_address_len);
cout << "Received message : " << buffer << " received from client.\n";
if(strcmp(buffer,"chat_START_SSL")==0){
    ack = "chat_START_SSL_NOT_SUPPORTED";
    goto jump;
}
msg = buffer;
sendto(fake_client_socket_descriptor, msg.c_str(), msg.length(), 0, (const struct sockaddr *) &server_address, sizeof(server_address));
cout << "Forwarding message : " << msg << " to actual server.\n";
if(msg=="chat_close"){
    break;
}

memset(buffer, '\0', sizeof(buffer));
received_message_len = recvfrom(fake_client_socket_descriptor, (char *)buffer, MAX_BUFFER_SIZE-1, 0, (struct sockaddr *) &server_address, (socklen_t *)&server_address_len);
buffer[received_message_len] = '\0';
cout << "Server : " << buffer << " received from server.\n";

ack = buffer;
jump:
sendto(fake_server_socket_descriptor, ack.c_str(), ack.length(), 0, (const struct sockaddr *) &client_address, sizeof(client_address));
cout << "Forwarding message : " << ack << " to actual client.\n";
if(ack=="chat_close"){
    break;
}

```

Server-side actual output of secure chat application.

```

root@bob1:~# ./secure_chat_app -s
Server started...

Client : chat_hello received
Server (You) : chat_ok sent
Client : hii received
Server (You) : hello
Client : how are u received
Server (You) : chat_close
Server closed
root@bob1:~#

```

Client-side actual output of secure chat application.

```

root@alice1:~# ./secure_chat_app -c bob1
Client started...

Client connecting to server bob1 with 172.31.0.4

You : chat_hello sent
Server : chat_ok received
You : chat_START_SSL sent
Server : chat_START_SSL_NOT_SUPPORTED received
You :hii
Server : hello received
You :how are u
Server : chat_close received
Client closed
root@alice1:~#

```

Interceptor-side actual output of SSL Downgrade attack

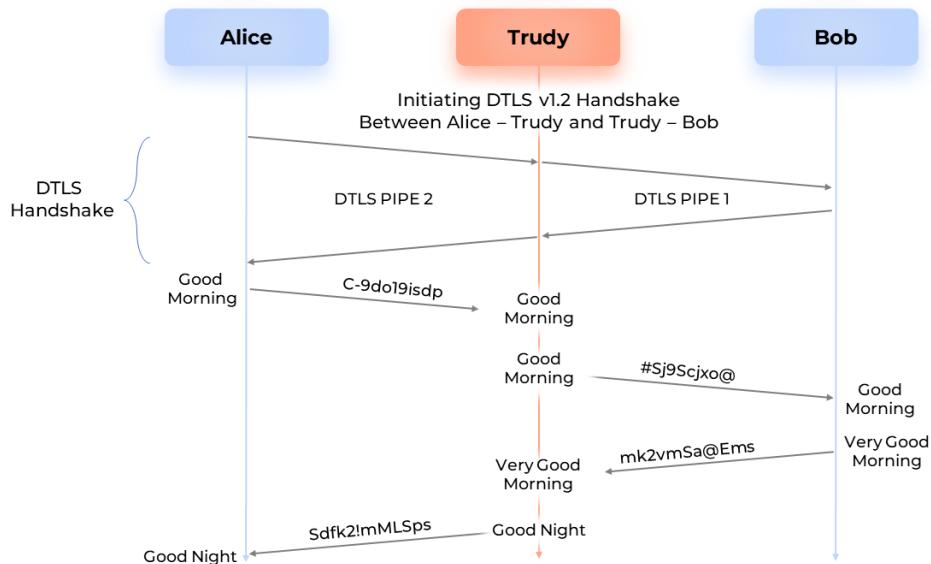
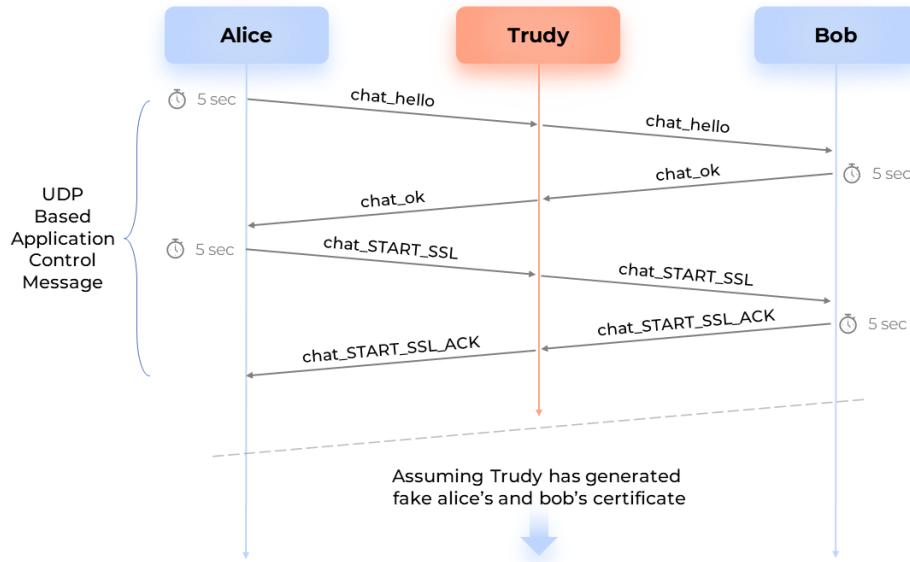
```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
Client : chat_hello received from client.
Forwarding message : chat_hello to actual server.
Server : chat_ok received from server.
Forwarding message : chat_ok to actual client.
Client : chat_START_SSL received from client.
Forwarding message : chat_START_SSL_NOT_SUPPORTED to actual client.
Client : hii received from client.
Forwarding message : hii to actual server.
Server : hello received from server.
Forwarding message : hello to actual client.
Client : how are u received from client.
Forwarding message : how are u to actual server.
Server : chat_close received from server.
Forwarding message : chat_close to actual client.

Interceptor Closed
```

Task 4 – Active MITM Attack

Assuming Trudy has successfully poisoned /etc/hosts file.

Application Flow :



Above exceptional flow diagram created by one of our team member shows the basic working of how Active MITM attack is performed here.

Alice-side actual output of Active MITM attack

```
root@alice1:~# ./secure_chat_app -c bob1
Client started...

Client connecting to server bob1 with 172.31.0.4

You : chat_hello sent
Server : chat_ok received
You : chat_START_SSL sent
Server : chat_START_SSL_ACK received
Client OpenSSL initialized
Client Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
You : hii
Server: hello
You : chat_close
Client closed
root@alice1:~#
```

Bob-side actual output of Active MITM attack

```
root@bob1:~# ./secure_chat_app -s
Server started...

Client : chat_hello received
Server (You) : chat_ok sent
Client : chat_START_SSL received
Server (You) : chat_START_SSL_ACK sent
Server OpenSSL initialized
Server Context Configured Success
Certificate Verification successfull.
Client connected with DTLSv1.2

-----
Client: hii
Server (You) : hello
Client: chat_close
Server closed
root@bob1:~#
```

Trudy-side actual output of Active MITM attack

```
root@trudy1:~  
Server : chat_START_SSL_ACK received  
Forwarding message : chat_START_SSL_ACK to actual client  
OpenSSL initialized  
Fake_client Context Configured Success  
Fake_server Context Configured Success  
Certificate Verification successfull.  
Fake_server connected with DTLSv1.2  
-----  
Certificate Verification successfull.  
Fake_client connected with DTLSv1.2  
-----  
Client: hii received  
Message hii passed to actual server  
Server: hello received  
Message hello passed to actual client  
Client: chat_close received  
Message chat_close passed to actual server  
Active Interceptor Closed  
root@trudy1:~#
```

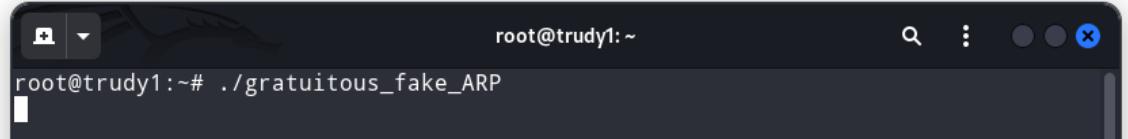
We are implementing MITM in two mode if Interceptor has to modify message then he/she enter the modified string or if he/she don't want to modify then simply enter will send the original content as shown in below snippet.

```
}  
received_message_len = SSL_read(ssl_server, buffer, sizeof(buffer) - 1);  
if (received_message_len <= 0) {  
    int ssl_error = SSL_get_error(ssl_server, received_message_len);  
    if (ssl_error == SSL_ERROR_ZERO_RETURN || ssl_error == SSL_ERROR_SYSCALL) {  
        break;  
    }  
} else {  
    buffer[received_message_len] = '\0';  
    cout << "Client: " << buffer << " received\n";  
}  
string modified;  
cout << "Type to modify else Enter : ";  
getline(cin,modified);  
if (modified == ""){  
    msg = modified;  
}  
else{  
    msg = buffer;  
}  
SSL_write(ssl_client, msg.c_str(), msg.length());  
cout << "Message " << msg << " passed to actual server\n";  
*-----*
```

Task 5 – ARP Cache Poisoning

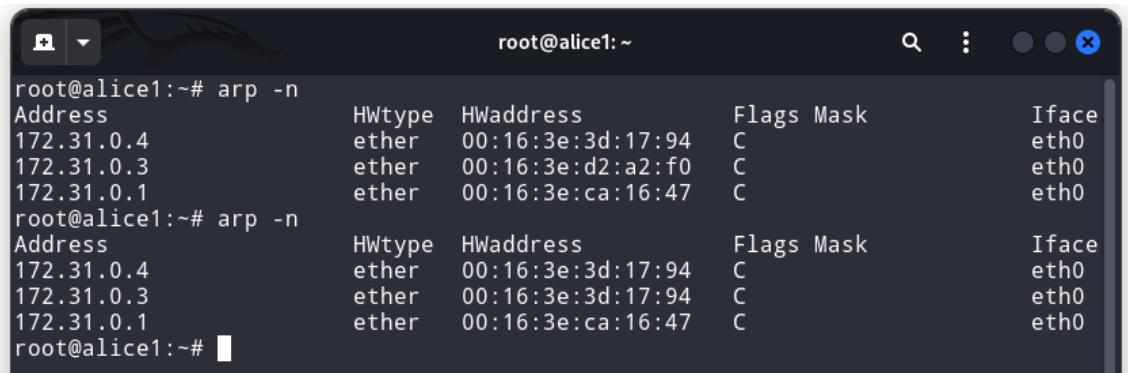
For ARP cache poisoning we are creating a fake ARP response packet and broadcast it using `gratuitous_fake_ARP.cpp`

Sending Fake Gratuitous ARP reponse packet from Trudy



```
root@trudy1:~# ./gratuitous_fake_ARP
```

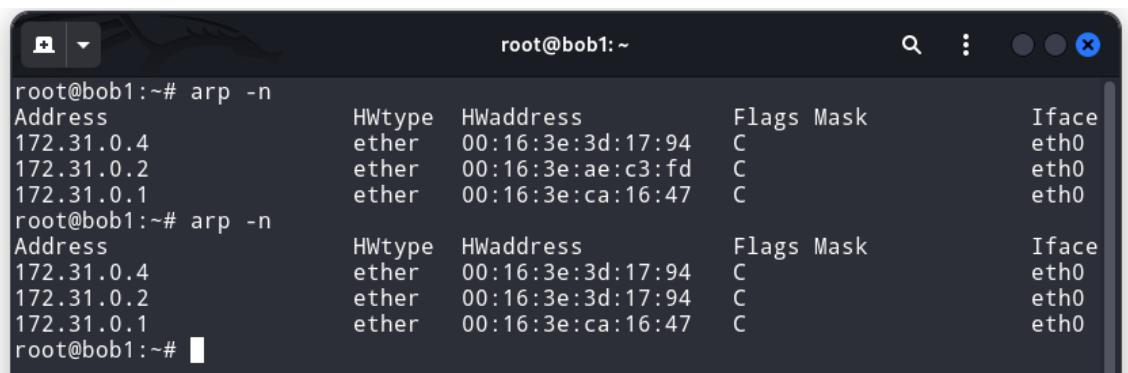
We can see before and after sending the fake ARP that at Alice's side MAC address corresponding to 172.31.0.3 got changed with Trudy's MAC.



Address	HWtype	HWaddress	Flags	Mask	Iface
172.31.0.4	ether	00:16:3e:3d:17:94	C		eth0
172.31.0.3	ether	00:16:3e:d2:a2:f0	C		eth0
172.31.0.1	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
172.31.0.3	ether	00:16:3e:3d:17:94	C		eth0
172.31.0.1	ether	00:16:3e:ca:16:47	C		eth0

Also at Bob's side, we can see before and after sending the fake ARP that MAC address corresponding to 172.31.0.2 got changed with Trudy's MAC.



Address	HWtype	HWaddress	Flags	Mask	Iface
172.31.0.4	ether	00:16:3e:3d:17:94	C		eth0
172.31.0.2	ether	00:16:3e:ae:c3:fd	C		eth0
172.31.0.1	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
172.31.0.2	ether	00:16:3e:3d:17:94	C		eth0
172.31.0.1	ether	00:16:3e:ca:16:47	C		eth0

Major Libraries Used

- Openssl : For DTLS
- pcap.h : For generating ARP response packet (Task 5)

Tools Used

- Tcpdump : For packet capturing
- Microsoft Powerpoint : For creating flow diagram
- Microsoft Word : For report
- After Effect : For 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: Patel Heetkumar D, KR Anuraj, Vishal Patidar

Date: 10/04/2024

Signature: HD, Anuraj, VP