Assignment-3

Secure chat using OpenSSL and MITM attacks

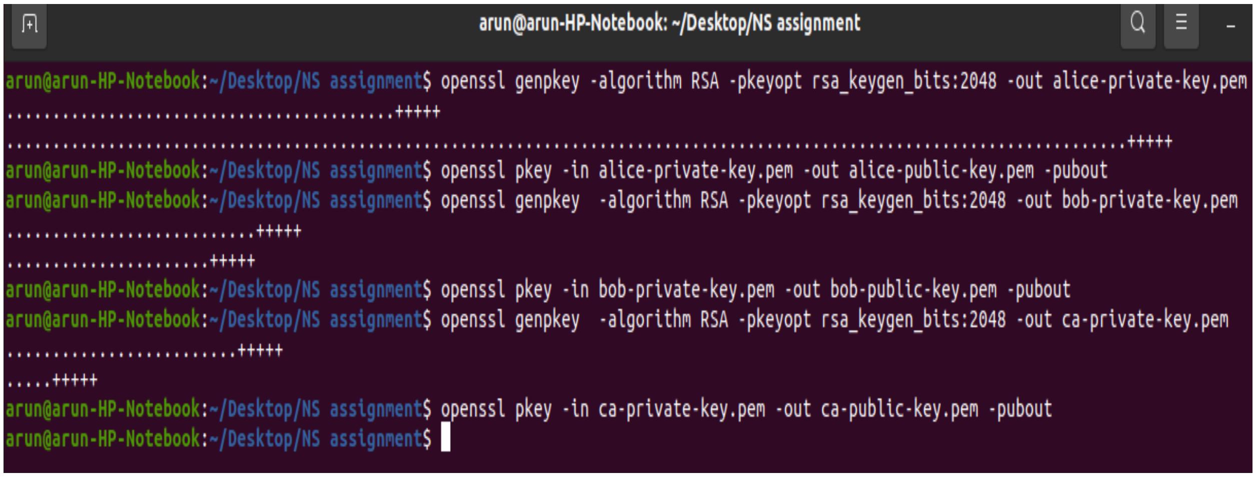
Roles

Alice : Arun Kant Dubey (CS20MTECH12008)
Trudy : Jagmeet Singh Ichhprani (CS20MTECH11006)
Bob : Saim Khan (CS20MTECH14008)

Task-1

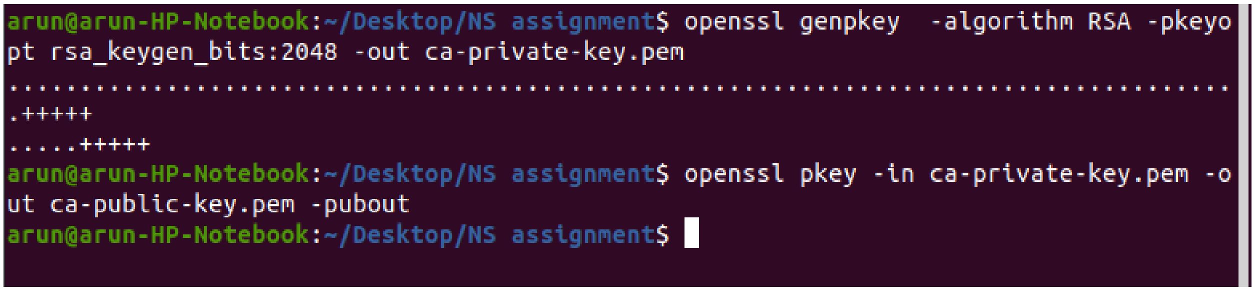
We generate public and private key pair for Alice and Bob using the commands as mentioned below:

1. **For Alice's private key**: openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out alice-private-key.pem
2. **For Alice's public key**: openssl pkey -in alice-private-key.pem -out alice-public-key.pem -pubout
3. **For Bob's private key**: openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out bob-private-key.pem
4. **For Bob's public key**:openssl pkey -in bob-private-key.pem -out bob-public-key.pem -pubout



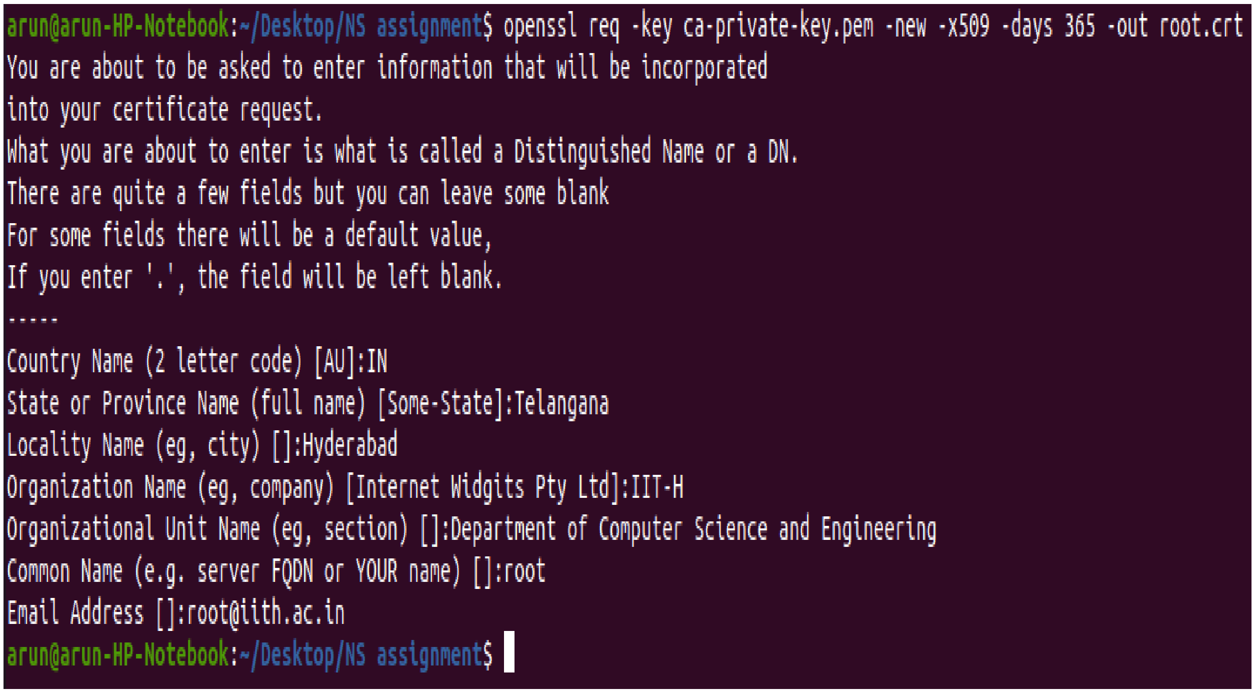
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out alice-private-key.pem
...........................................+++++
...........................................................................................................................+++++
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl pkey -in alice-private-key.pem -out alice-public-key.pem -pubout
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl genpkey  -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out bob-private-key.pem
...........................+++++
......................+++++
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl pkey -in bob-private-key.pem -out bob-public-key.pem -pubout
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl genpkey  -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out ca-private-key.pem
.........................+++++
.....+++++
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl pkey -in ca-private-key.pem -out ca-public-key.pem -pubout
arun@arun-HP-Notebook:~/Desktop/NS assignment$ 
```

After generating the key pair for Alice and Bob, key pair for CA is generated:



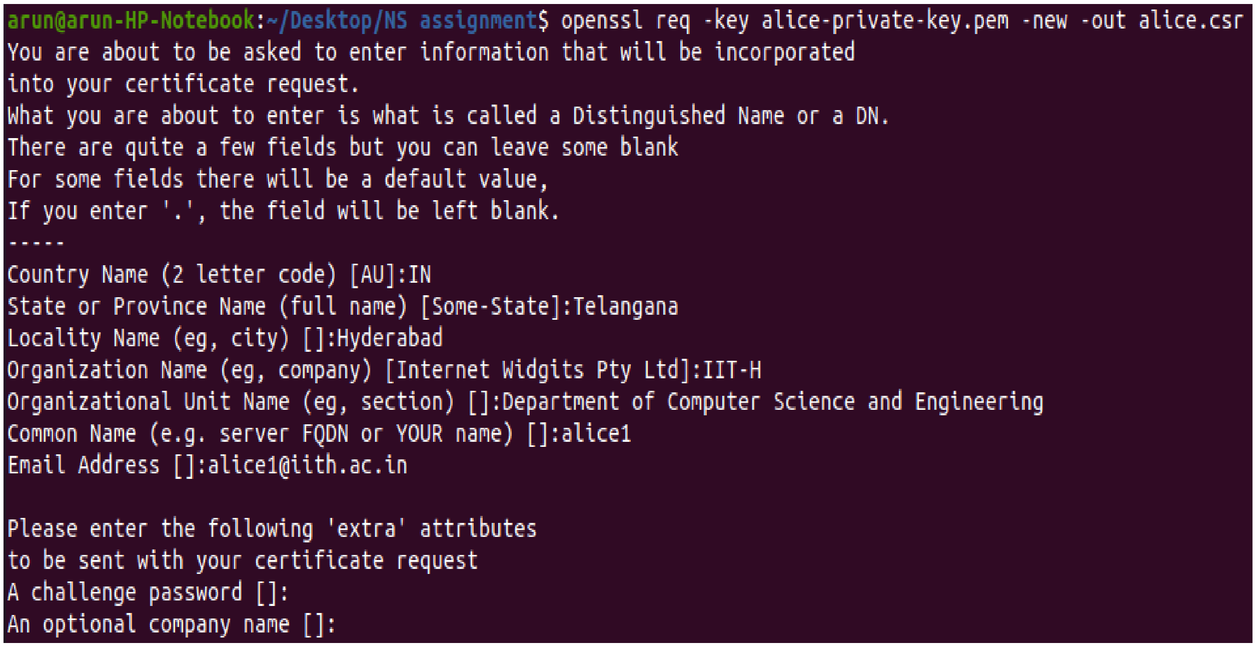
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl genpkey  -algorithm RSA -pkeyo
pt rsa_keygen_bits:2048 -out ca-private-key.pem
.....................................................................................
.+++++
.....+++++
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl pkey -in ca-private-key.pem -o
ut ca-public-key.pem -pubout
arun@arun-HP-Notebook:~/Desktop/NS assignment$ 
```

A self-signed certificate is generated for the root CA using the following command:



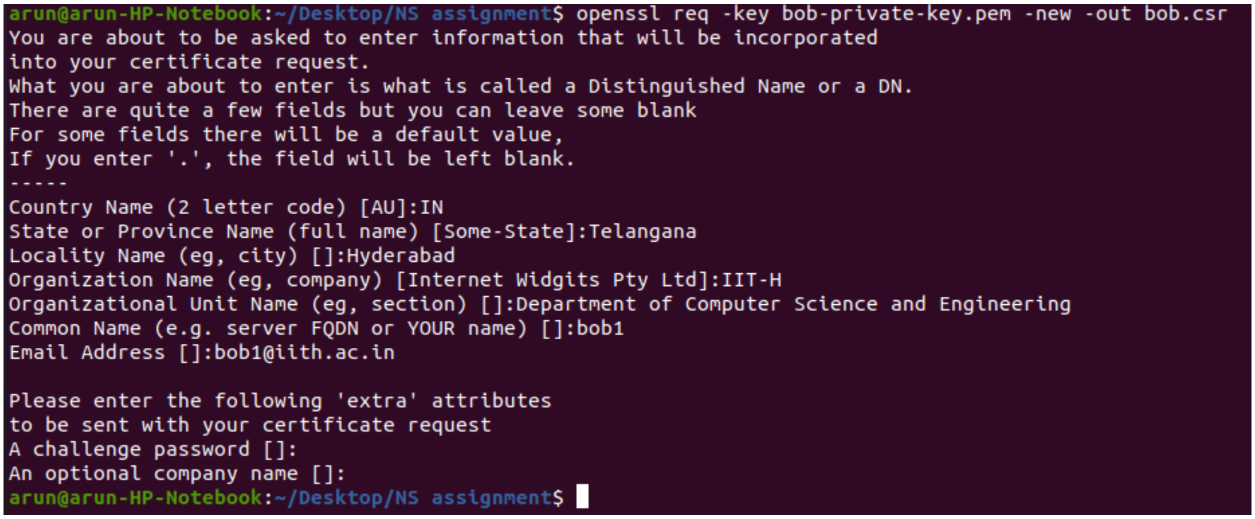
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl req -key ca-private-key.pem -new -x509 -days 365 -out root.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Telangana
Locality Name (eg, city) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT-H
Organizational Unit Name (eg, section) []:Department of Computer Science and Engineering
Common Name (e.g. server FQDN or YOUR name) []:root
Email Address []:root@iith.ac.in
arun@arun-HP-Notebook:~/Desktop/NS assignment$ 
```

Alice generates a CSR by providing several details as mentioned below:



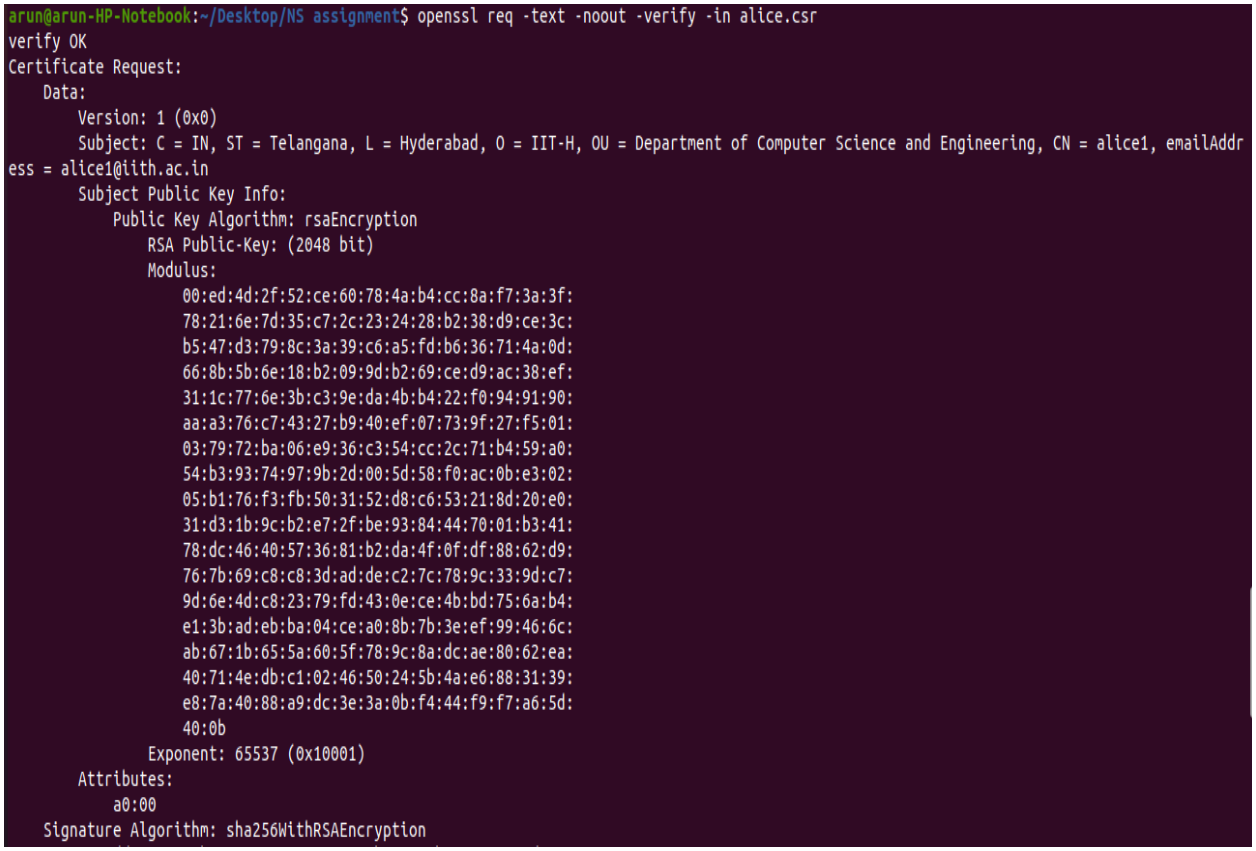
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl req -key alice-private-key.pem -new -out alice.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Telangana
Locality Name (eg, city) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT-H
Organizational Unit Name (eg, section) []:Department of Computer Science and Engineering
Common Name (e.g. server FQDN or YOUR name) []:alice1
Email Address []:alice1@iith.ac.in

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

Bob generates a CSR by providing several details as mentioned below:

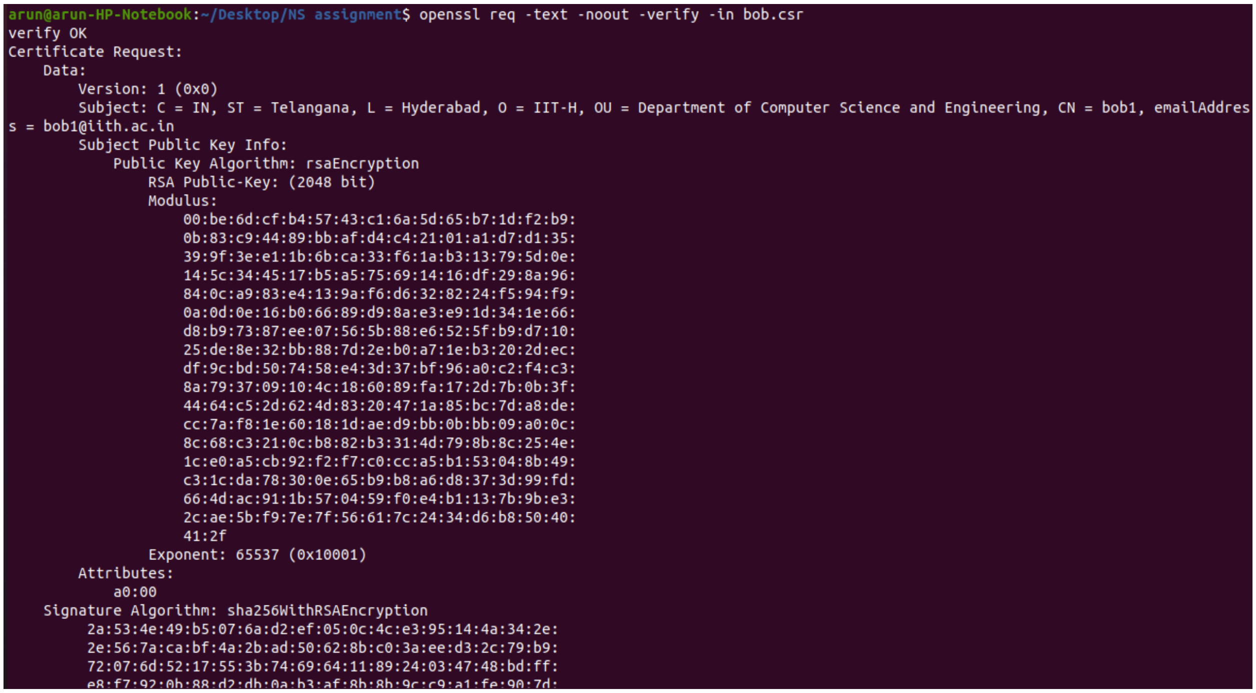


```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl req -key bob-private-key.pem -new -out bob.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Telangana
Locality Name (eg, city) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIT-H
Organizational Unit Name (eg, section) []:Department of Computer Science and Engineering
Common Name (e.g. server FQDN or YOUR name) []:bob1
Email Address []:bob1@iith.ac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
arun@arun-HP-Notebook:~/Desktop/NS assignment$
```

Now, CA can verify if the CSR using the openssl commands:



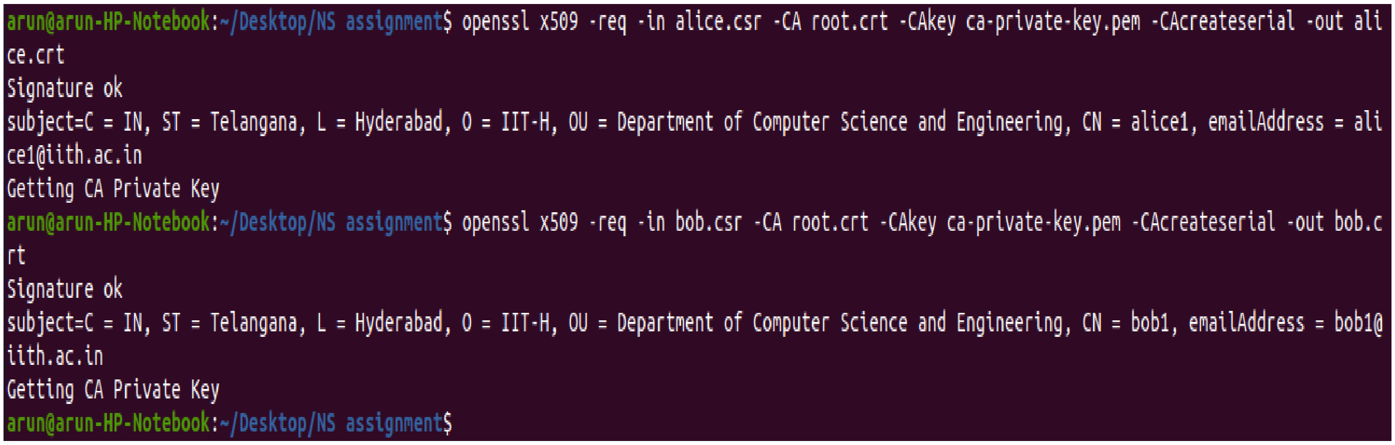
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl req -text -noout -verify -in alice.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT-H, OU = Department of Computer Science and Engineering, CN = alice1, emailAddr
ess = alice1@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ed:4d:2f:52:ce:60:78:4a:b4:cc:8a:f7:3a:3f:
                    78:21:6e:7d:35:c7:2c:23:24:28:b2:38:d9:ce:3c:
                    b5:47:d3:79:8c:3a:39:c6:a5:fd:b6:36:71:4a:0d:
                    66:8b:5b:6e:18:b2:09:9d:b2:69:ce:d9:ac:38:ef:
                    31:1c:77:6e:3b:c3:9e:da:4b:b4:22:f0:94:91:90:
                    aa:a3:76:c7:43:27:b9:40:ef:07:73:9f:27:f5:01:
                    03:79:72:ba:06:e9:36:c3:54:cc:2c:71:b4:59:a0:
                    54:b3:93:74:97:9b:2d:00:5d:58:f0:ac:0b:e3:02:
                    05:b1:76:f3:fb:50:31:52:d8:c6:53:21:8d:20:e0:
                    31:d3:1b:9c:b2:e7:2f:be:93:84:44:70:01:b3:41:
                    78:dc:46:40:57:36:81:b2:da:4f:0f:df:88:62:d9:
                    76:7b:69:c8:c8:3d:ad:de:c2:7c:78:9c:33:9d:c7:
                    9d:6e:4d:c8:23:79:fd:43:0e:ce:4b:bd:75:6a:b4:
                    e1:3b:ad:eb:ba:04:ce:a0:8b:7b:3e:ef:99:46:6c:
                    ab:67:1b:65:5a:60:5f:78:9c:8a:dc:ae:80:62:ea:
                    40:71:4e:db:c1:02:46:50:24:5b:4a:e6:88:31:39:
                    e8:7a:40:88:a9:dc:3e:3a:0b:f4:44:f9:f7:a6:5d:
                    40:0b
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
```



```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl req -text -noout -verify -in bob.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT-H, OU = Department of Computer Science and Engineering, CN = bob1, emailAddres
s = bob1@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:be:6d:cf:b4:57:43:c1:6a:5d:65:b7:1d:f2:b9:
                    0b:83:c9:44:89:bb:af:d4:c4:21:01:a1:d7:d1:35:
                    39:9f:3e:e1:1b:6b:ca:33:f6:1a:b3:13:79:5d:0e:
                    14:5c:34:45:17:b5:a5:75:69:14:16:df:29:8a:96:
                    84:0c:a9:83:e4:13:9a:f6:d6:32:82:24:f5:94:f9:
                    0a:0d:0e:16:b0:66:89:d9:8a:e3:e9:1d:34:1e:66:
                    d8:b9:73:87:ee:07:56:5b:88:e6:52:5f:b9:d7:10:
                    25:de:8e:32:bb:88:7d:2e:b0:a7:1e:b3:20:2d:ec:
                    df:9c:bd:50:74:58:e4:3d:37:bf:96:a0:c2:f4:c3:
                    8a:79:37:09:10:4c:18:60:89:fa:17:2d:7b:0b:3f:
                    44:64:c5:2d:62:4d:83:20:47:1a:85:bc:7d:a8:de:
                    cc:7a:f8:1e:60:18:1d:ae:d9:bb:0b:bb:09:a0:0c:
                    8c:68:c3:21:0c:b8:82:b3:31:4d:79:8b:8c:25:4e:
                    1c:e0:a5:cb:92:f2:f7:c0:cc:a5:b1:53:04:8b:49:
                    c3:1c:da:78:30:0e:65:b9:b8:a6:d8:37:3d:99:fd:
                    66:4d:ac:91:1b:57:04:59:f0:e4:b1:13:7b:9b:e3:
                    2c:ae:5b:f9:7e:7f:56:61:7c:24:34:d6:b8:50:40:
                    41:2f
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         2a:53:4e:49:b5:07:6a:d2:ef:05:0c:4c:e3:95:14:4a:34:2e:
         2e:56:7a:ca:bf:4a:2b:ad:50:62:8b:c0:3a:ee:d3:2c:79:b9:
         72:07:6d:52:17:55:3b:74:69:64:11:89:24:03:47:48:bd:ff:
         e8:f7:92:0b:88:d2:db:0a:b3:af:8b:8b:9c:c9:a1:fe:90:7d:
```

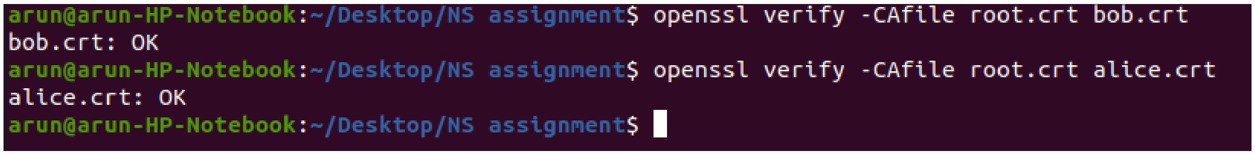
Inorder to provide more security we can keep the public key of Alice and Bob in the list of trust public key at the root.

Now, root CA issues certificate by to both Alice and Bob:



```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl x509 -req -in alice.csr -CA root.crt -CAkey ca-private-key.pem -CAcreateserial -out ali
ce.crt
Signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT-H, OU = Department of Computer Science and Engineering, CN = alice1, emailAddress = ali
ce1@iith.ac.in
Getting CA Private Key
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl x509 -req -in bob.csr -CA root.crt -CAkey ca-private-key.pem -CAcreateserial -out bob.c
rt
Signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT-H, OU = Department of Computer Science and Engineering, CN = bob1, emailAddress = bob1@
iith.ac.in
Getting CA Private Key
arun@arun-HP-Notebook:~/Desktop/NS assignment$
```

The issued certificates can be verified as shown in the following screenshot :



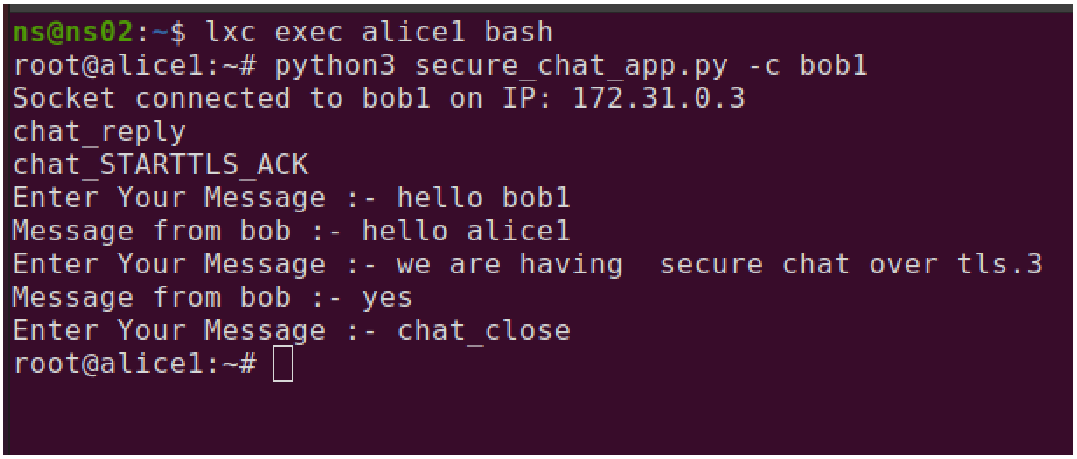
```
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl verify -CAfile root.crt bob.crt
bob.crt: OK
arun@arun-HP-Notebook:~/Desktop/NS assignment$ openssl verify -CAfile root.crt alice.crt
alice.crt: OK
arun@arun-HP-Notebook:~/Desktop/NS assignment$ 
```

TASK-2

Secure Chat Between Alice and bob.

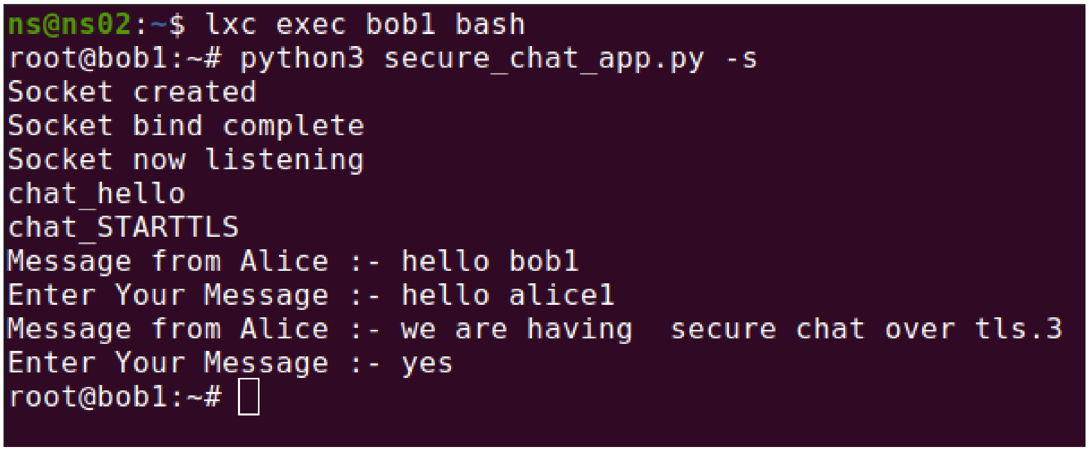
Algorithm

1. Alice initiates a TCP connection to Bob and sends a **chat_hello** message to bob.
2. Bob sends a **chat_reply** message in response and they complete a handshake at the application level.
3. Alice then sends a chat_STARTTLS message to know if Bob wants secure communication.
4. If Bob replies **chat_STARTTLS_ACK** then they initiate a TLS handshake so now their chat is encrypted as attached in the Wireshark screenshot below.



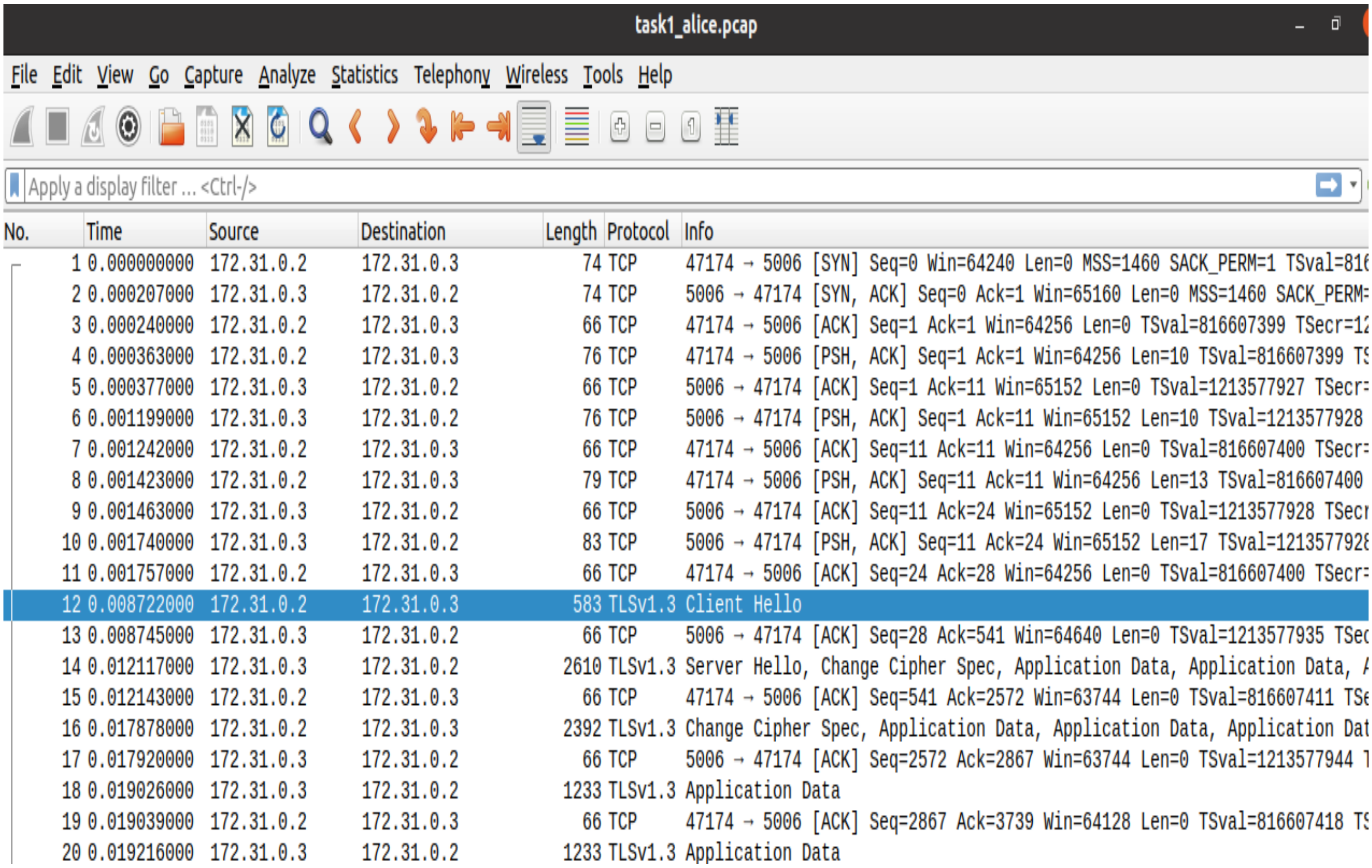
```
ns@ns02:~$ lxc exec alice1 bash
root@alice1:~# python3 secure_chat_app.py -c bob1
Socket connected to bob1 on IP: 172.31.0.3
chat_reply
chat_STARTTLS_ACK
Enter Your Message :- hello bob1
Message from bob :- hello alice1
Enter Your Message :- we are having  secure chat over tls.3
Message from bob :- yes
Enter Your Message :- chat_close
root@alice1:~# 
```

Alice



```
ns@ns02:~$ lxc exec bob1 bash
root@bob1:~# python3 secure_chat_app.py -s
Socket created
Socket bind complete
Socket now listening
chat_hello
chat_STARTTLS
Message from Alice :- hello bob1
Enter Your Message :- hello alice1
Message from Alice :- we are having  secure chat over tls.3
Enter Your Message :- yes
root@bob1:~# 
```

Bob


task1_alice.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time        Source      Destination  Length Protocol Info
   1 0.000000000 172.31.0.2  172.31.0.3       74 TCP     47174 → 5006 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=81
   2 0.000207000 172.31.0.3  172.31.0.2       74 TCP     5006 → 47174 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=
   3 0.000240000 172.31.0.2  172.31.0.3       66 TCP     47174 → 5006 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=816607399 TSecr=1
   4 0.000363000 172.31.0.2  172.31.0.3       76 TCP     47174 → 5006 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=816607399 T
   5 0.000377000 172.31.0.3  172.31.0.2       66 TCP     5006 → 47174 [ACK] Seq=1 Ack=11 Win=65152 Len=0 TSval=1213577927 TSecr=
   6 0.001199000 172.31.0.3  172.31.0.2       76 TCP     5006 → 47174 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=10 TSval=1213577928
   7 0.001242000 172.31.0.2  172.31.0.3       66 TCP     47174 → 5006 [ACK] Seq=11 Ack=11 Win=64256 Len=0 TSval=816607400 TSecr=
   8 0.001423000 172.31.0.2  172.31.0.3       79 TCP     47174 → 5006 [PSH, ACK] Seq=11 Ack=11 Win=64256 Len=13 TSval=816607400
   9 0.001463000 172.31.0.3  172.31.0.2       66 TCP     5006 → 47174 [ACK] Seq=11 Ack=24 Win=65152 Len=0 TSval=1213577928 TSec
  10 0.001740000 172.31.0.3  172.31.0.2       83 TCP     5006 → 47174 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=17 TSval=121357792
  11 0.001757000 172.31.0.2  172.31.0.3       66 TCP     47174 → 5006 [ACK] Seq=24 Ack=28 Win=64256 Len=0 TSval=816607400 TSecr=
  12 0.008722000 172.31.0.2  172.31.0.3      583 TLSv1.3 Client Hello
  13 0.008745000 172.31.0.3  172.31.0.2       66 TCP     5006 → 47174 [ACK] Seq=28 Ack=541 Win=64640 Len=0 TSval=1213577935 TSe
  14 0.012117000 172.31.0.3  172.31.0.2     2610 TLSv1.3 Server Hello, Change Cipher Spec, Application Data, Application Data,
  15 0.012143000 172.31.0.2  172.31.0.3       66 TCP     47174 → 5006 [ACK] Seq=541 Ack=2572 Win=63744 Len=0 TSval=816607411 TS
  16 0.017878000 172.31.0.2  172.31.0.3     2392 TLSv1.3 Change Cipher Spec, Application Data, Application Data, Application Da
  17 0.017920000 172.31.0.3  172.31.0.2       66 TCP     5006 → 47174 [ACK] Seq=2572 Ack=2867 Win=63744 Len=0 TSval=1213577944
  18 0.019026000 172.31.0.3  172.31.0.2     1233 TLSv1.3 Application Data
  19 0.019039000 172.31.0.2  172.31.0.3       66 TCP     47174 → 5006 [ACK] Seq=2867 Ack=3739 Win=64128 Len=0 TSval=816607418 T
  20 0.019216000 172.31.0.3  172.31.0.2     1233 TLSv1.3 Application Data
```

Wireshark trace for secure Alice-Bob communication

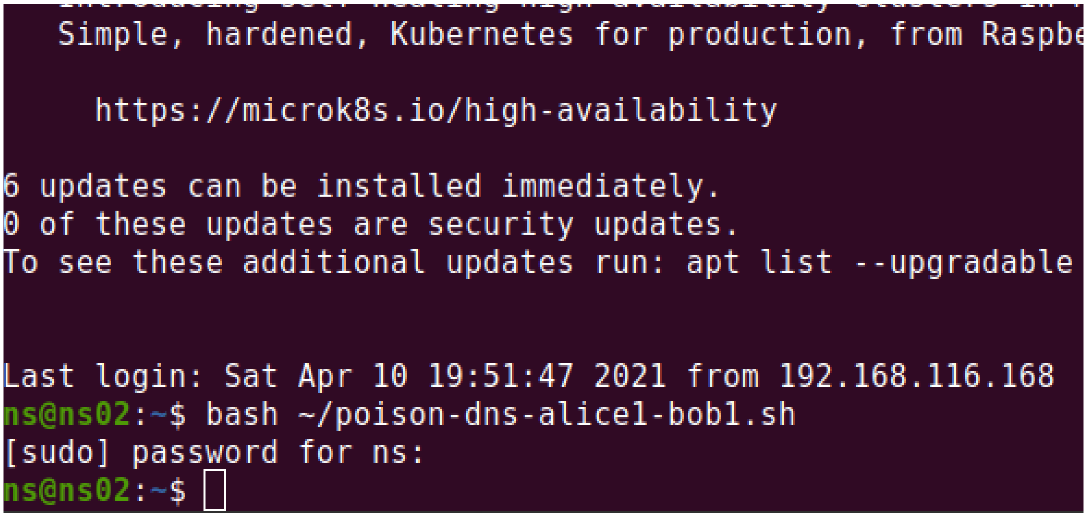
TASK-3

Downgrade attack by Trudy

Algorithm

1. Trudy did DNS poisoning and now the traffic between Alice and Bob passes through Trudy.
2. Alice initiates **chat_hello** same as in task 1. Trudy in this downgrade attack blocks the chat_STARTTLS message from Alice and replies chat_STARTTLS_NOT_SUPPORTED to Alice. So now Alice thinks that bob doesn't want secure communication so he now initiates chat in plain text form as can be seen in the Wireshark screenshot attached below no TLS pipe is formed.

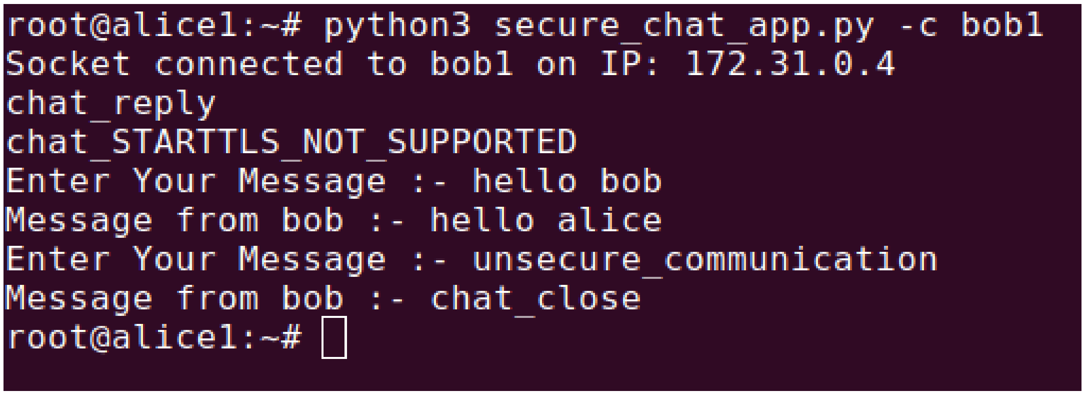
DNS-POISONING



```
  Introducing self-healing high availability clusters in
  Simple, hardened, Kubernetes for production, from Raspbe

    https://microk8s.io/high-availability

6 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable


Last login: Sat Apr 10 19:51:47 2021 from 192.168.116.168
ns@ns02:~$ bash ~/poison-dns-alice1-bob1.sh
[sudo] password for ns:
ns@ns02:~$ 
```

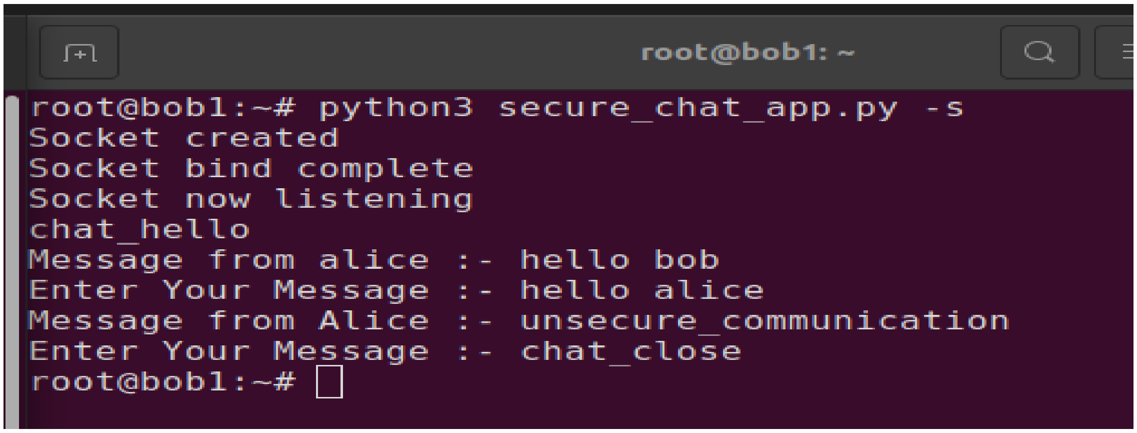
ALICE SIDE

Alice receives chat_STARTTLS_NOT_SUPPORTED and he thinks bob doesn't want secure communication.



```
root@alice1:~# python3 secure_chat_app.py -c bob1
Socket connected to bob1 on IP: 172.31.0.4
chat_reply
chat_STARTTLS_NOT_SUPPORTED
Enter Your Message :- hello bob
Message from bob :- hello alice
Enter Your Message :- unsecure_communication
Message from bob :- chat_close
root@alice1:~# 
```

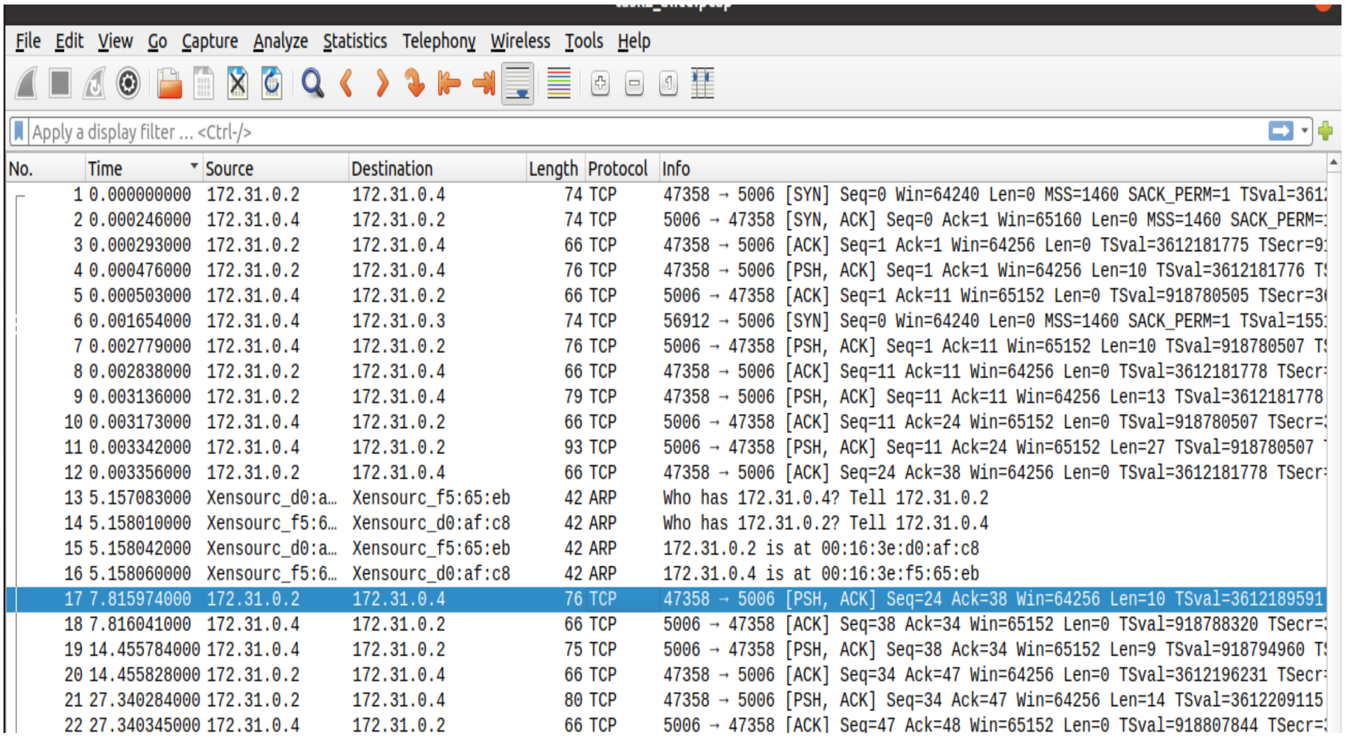
BOB SIDE



```
root@bob1:~# python3 secure_chat_app.py -s
Socket created
Socket bind complete
Socket now listening
chat_hello
Message from alice :- hello bob
Enter Your Message :- hello alice
Message from Alice :- unsecure_communication
Enter Your Message :- chat_close
root@bob1:~# 
```

Wireshark at Alice

It can be seen that no TLS connection is set up between Alice and Bob.



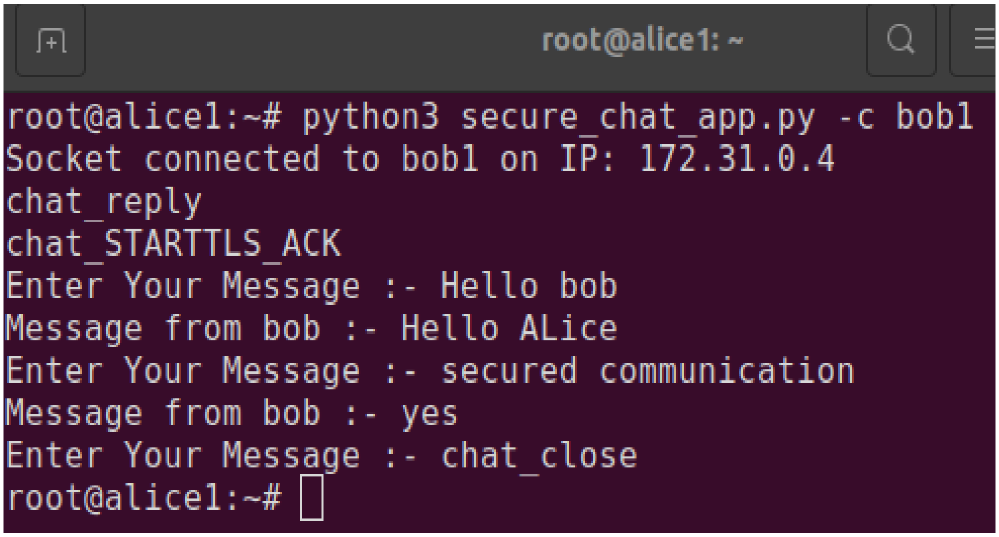
No.	Time	Source	Destination	Length	Protocol	Info
1	0.000000000	172.31.0.2	172.31.0.4	74	TCP	47358 → 5006 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=361
2	0.000246000	172.31.0.4	172.31.0.2	74	TCP	5006 → 47358 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=
3	0.000293000	172.31.0.2	172.31.0.4	66	TCP	47358 → 5006 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3612181775 TSecr=9
4	0.000476000	172.31.0.2	172.31.0.4	76	TCP	47358 → 5006 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=3612181776 T
5	0.000503000	172.31.0.4	172.31.0.2	66	TCP	5006 → 47358 [ACK] Seq=1 Ack=11 Win=65152 Len=0 TSval=918780505 TSecr=3
6	0.001654000	172.31.0.4	172.31.0.3	74	TCP	56912 → 5006 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=155
7	0.002779000	172.31.0.4	172.31.0.2	76	TCP	5006 → 47358 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=10 TSval=918780507 T
8	0.002838000	172.31.0.2	172.31.0.4	66	TCP	47358 → 5006 [ACK] Seq=11 Ack=11 Win=64256 Len=0 TSval=3612181778 TSecr
9	0.003136000	172.31.0.2	172.31.0.4	79	TCP	47358 → 5006 [PSH, ACK] Seq=11 Ack=11 Win=64256 Len=13 TSval=3612181778
10	0.003173000	172.31.0.4	172.31.0.2	66	TCP	5006 → 47358 [ACK] Seq=11 Ack=24 Win=65152 Len=0 TSval=918780507 TSecr=
11	0.003342000	172.31.0.4	172.31.0.2	93	TCP	5006 → 47358 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=27 TSval=918780507
12	0.003356000	172.31.0.2	172.31.0.4	66	TCP	47358 → 5006 [ACK] Seq=24 Ack=38 Win=64256 Len=0 TSval=3612181778 TSecr
13	5.157083000	Xensourc_d0:a…	Xensourc_f5:65:eb	42	ARP	Who has 172.31.0.4? Tell 172.31.0.2
14	5.158010000	Xensourc_f5:6…	Xensourc_d0:af:c8	42	ARP	Who has 172.31.0.2? Tell 172.31.0.4
15	5.158042000	Xensourc_d0:a…	Xensourc_f5:65:eb	42	ARP	172.31.0.2 is at 00:16:3e:d0:af:c8
16	5.158060000	Xensourc_f5:6…	Xensourc_d0:af:c8	42	ARP	172.31.0.4 is at 00:16:3e:f5:65:eb
17	7.815974000	172.31.0.2	172.31.0.4	76	TCP	47358 → 5006 [PSH, ACK] Seq=24 Ack=38 Win=64256 Len=10 TSval=3612189591
18	7.816041000	172.31.0.4	172.31.0.2	66	TCP	5006 → 47358 [ACK] Seq=38 Ack=34 Win=65152 Len=0 TSval=918788320 TSecr=
19	14.455784000	172.31.0.4	172.31.0.2	75	TCP	5006 → 47358 [PSH, ACK] Seq=38 Ack=34 Win=65152 Len=9 TSval=918794960 T
20	14.455828000	172.31.0.2	172.31.0.4	66	TCP	47358 → 5006 [ACK] Seq=34 Ack=47 Win=64256 Len=0 TSval=3612196231 TSecr
21	27.340284000	172.31.0.2	172.31.0.4	80	TCP	47358 → 5006 [PSH, ACK] Seq=34 Ack=47 Win=64256 Len=14 TSval=3612209115
22	27.340345000	172.31.0.4	172.31.0.2	66	TCP	5006 → 47358 [ACK] Seq=47 Ack=48 Win=65152 Len=0 TSval=918807844 TSecr=

TASK-4: MITM attack by Trudy

Algorithm

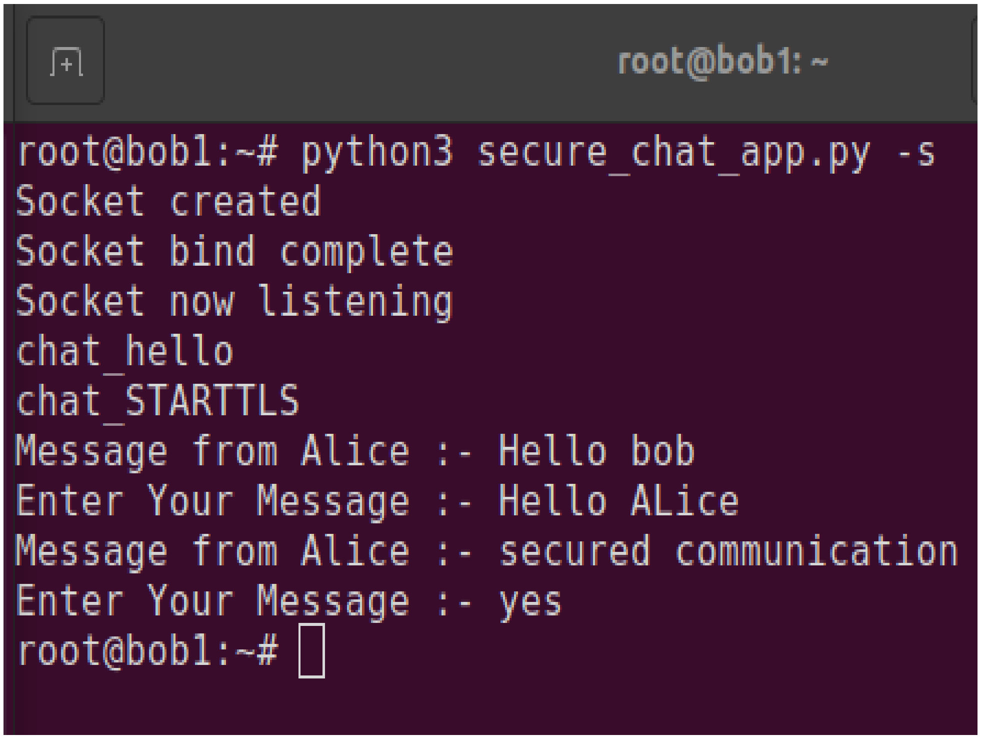
1. In this attack, Trudy hacks into the server of root CA and issues fake/shadow certificates for Alice and Bob and now Trudy can form two TLS pipe one with Alice and one with Bob.
   Trudy issues two certificates **fakealice.crt** and **fakebob.crt** in the name of alice1 and bob1 by using **ca-private-key.pem**.
   1. First, generate two key pair for fake-alice and fake-bob
   2. Create two CSR and then issue a fake certificate using the root private key.
2. Alice and Bob are now fooled they think that they are talking to each other but both of them are talking to Trudy. Thus Trudy can change the chat messages transferred between them.

Alice Side



```
root@alice1:~# python3 secure_chat_app.py -c bob1
Socket connected to bob1 on IP: 172.31.0.4
chat_reply
chat_STARTTLS_ACK
Enter Your Message :- Hello bob
Message from bob :- Hello ALice
Enter Your Message :- secured communication
Message from bob :- yes
Enter Your Message :- chat_close
root@alice1:~# 
```

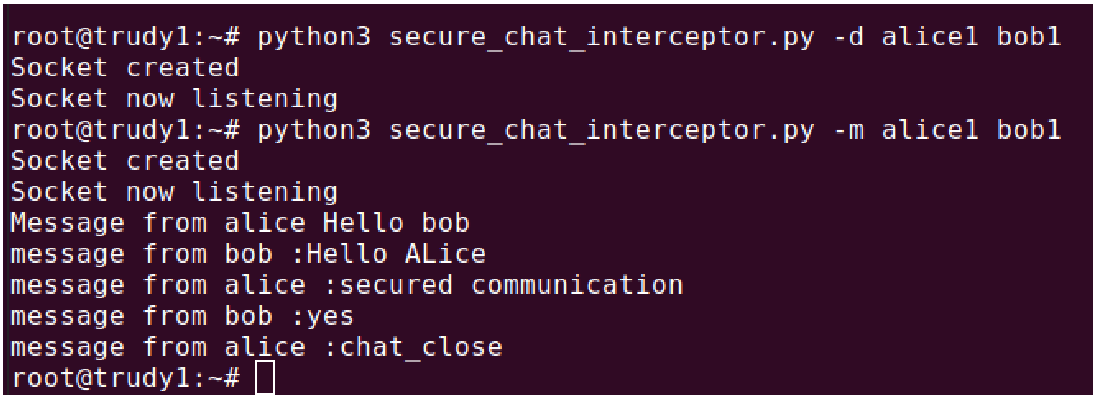
Bob Side



```
root@bob1:~# python3 secure_chat_app.py -s
Socket created
Socket bind complete
Socket now listening
chat_hello
chat_STARTTLS
Message from Alice :- Hello bob
Enter Your Message :- Hello ALice
Message from Alice :- secured communication
Enter Your Message :- yes
root@bob1:~# 
```

Trudy Side

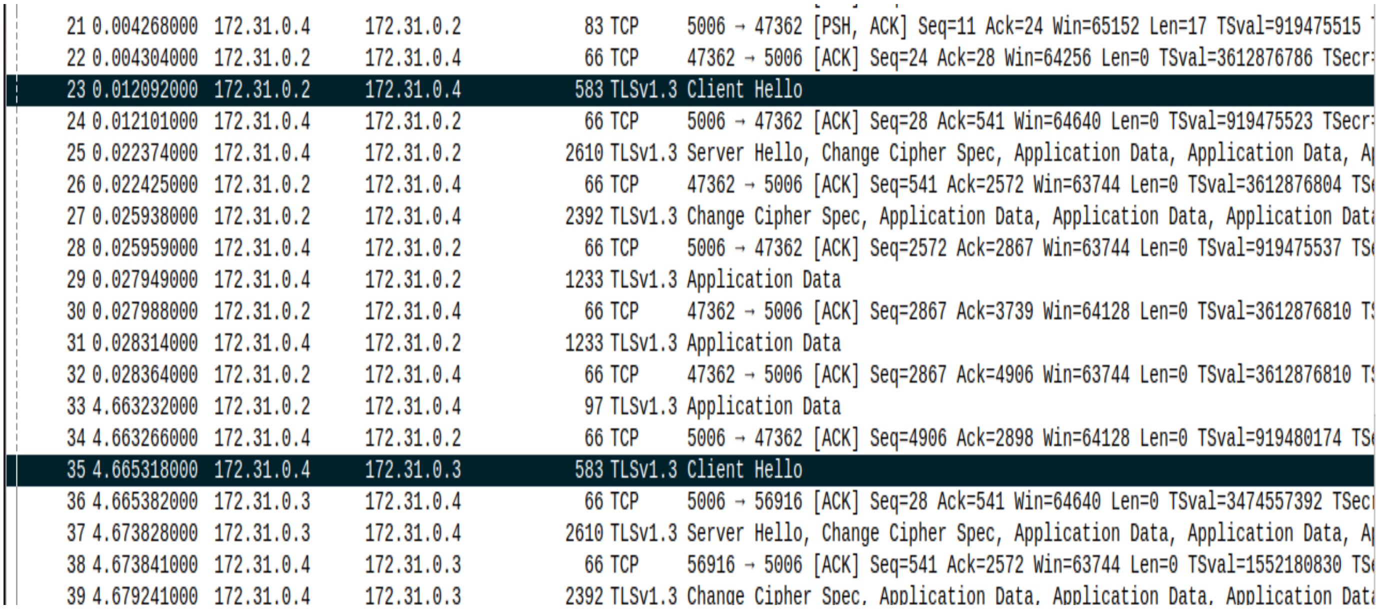
Trudy can read and even change the whole communication between Alice and Bob.



```
root@trudy1:~# python3 secure_chat_interceptor.py -d alice1 bob1
Socket created
Socket now listening
root@trudy1:~# python3 secure_chat_interceptor.py -m alice1 bob1
Socket created
Socket now listening
Message from alice Hello bob
message from bob :Hello ALice
message from alice :secured communication
message from bob :yes
message from alice :chat_close
root@trudy1:~# 
```

Wireshark Trudy Side

It can be seen in Wireshark's trace that two TLS pipes are set up between (Alice - Trudy) and (Trudy - Bob). a



```
21 0.004268000  172.31.0.4    172.31.0.2      83 TCP     5006 → 47362 [PSH, ACK] Seq=11 Ack=24 Win=65152 Len=17 TSval=919475515
22 0.004304000  172.31.0.2    172.31.0.4      66 TCP     47362 → 5006 [ACK] Seq=24 Ack=28 Win=64256 Len=0 TSval=3612876786 TSecr
23 0.012092000  172.31.0.2    172.31.0.4     583 TLSv1.3 Client Hello
24 0.012101000  172.31.0.4    172.31.0.2      66 TCP     5006 → 47362 [ACK] Seq=28 Ack=541 Win=64640 Len=0 TSval=919475523 TSecr
25 0.022374000  172.31.0.4    172.31.0.2    2610 TLSv1.3 Server Hello, Change Cipher Spec, Application Data, Application Data, A
26 0.022425000  172.31.0.2    172.31.0.4      66 TCP     47362 → 5006 [ACK] Seq=541 Ack=2572 Win=63744 Len=0 TSval=3612876804 TS
27 0.025938000  172.31.0.2    172.31.0.4    2392 TLSv1.3 Change Cipher Spec, Application Data, Application Data, Application Dat
28 0.025959000  172.31.0.4    172.31.0.2      66 TCP     5006 → 47362 [ACK] Seq=2572 Ack=2867 Win=63744 Len=0 TSval=919475537 TS
29 0.027949000  172.31.0.4    172.31.0.2    1233 TLSv1.3 Application Data
30 0.027988000  172.31.0.2    172.31.0.4      66 TCP     47362 → 5006 [ACK] Seq=2867 Ack=3739 Win=64128 Len=0 TSval=3612876810 T
31 0.028314000  172.31.0.4    172.31.0.2    1233 TLSv1.3 Application Data
32 0.028364000  172.31.0.2    172.31.0.4      66 TCP     47362 → 5006 [ACK] Seq=2867 Ack=4906 Win=63744 Len=0 TSval=3612876810 T
33 4.663232000  172.31.0.2    172.31.0.4      97 TLSv1.3 Application Data
34 4.663266000  172.31.0.4    172.31.0.2      66 TCP     5006 → 47362 [ACK] Seq=4906 Ack=2898 Win=64128 Len=0 TSval=919480174 TS
35 4.665318000  172.31.0.4    172.31.0.3     583 TLSv1.3 Client Hello
36 4.665382000  172.31.0.3    172.31.0.4      66 TCP     5006 → 56916 [ACK] Seq=28 Ack=541 Win=64640 Len=0 TSval=3474557392 TSec
37 4.673828000  172.31.0.3    172.31.0.4    2610 TLSv1.3 Server Hello, Change Cipher Spec, Application Data, Application Data, A
38 4.673841000  172.31.0.4    172.31.0.3      66 TCP     56916 → 5006 [ACK] Seq=541 Ack=2572 Win=63744 Len=0 TSval=1552180830 TS
39 4.679241000  172.31.0.4    172.31.0.3    2392 TLSv1.3 Change Cipher Spec, Application Data, Application Data, Application Dat
```

PLAGIARISM STATEMENT

We certify that this assignment/report is our own work, based on our combined personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, tutorials, sample programs, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment in any other course, 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 plagiarised 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 our responsibility to report honour violations by other students if any of us become aware of it.

Names : Arun Kant Dubey , Jagmeet Singh Ichhprani , Saim Khan
Date : 10th April, 2021

Signature : Arunkant

J Singh

Saim Khan