Computer Network Security

PKI

Satish Reddy.Y (cs14btech11030)

1) Creating a Public key infrastructure of IITH

->I have create a root key and certificate which is self signed

->create intermediate keys and certification request like for CSE,EE,admin etc and got it signed by the root ca to generate certificates for all these intermediate C.A's

->Now I created the servers(one or two) for each of the intermediate CA and generated keys and csr's for these and got them signed by intermediate CA's to get the server certificates.

Each of the keys are locked and all requires the passwords.

I have created the chain certificates(for all intermediate CA's) which is the concatenation of all certificates from root to that Intermediate CA .

I have made certificate revocation lists.

This is PK infrastructure I made for iith.

Commands Used:

I have created a config file included in submission which has all configuration i used for this assignment for openssl like version of x509 the encryptions used and various other configurations.

Creating a Key:

openssl genrsa -aes256 -out <key name> 2048

For creating a certificate request(csr):

```
openssl req -config <conf file of openssh> \
      -key <location of root key> \
      -new -sha256 -out <location for root certificate>
```

For creating certificate:

```
openssl ca -config <conf file of openssh> \
      -extensions server_cert -days 375 -notext -md sha256 \
      -in <location of csr> \
      -out <location for certificate>
```

For root certificate is created directly without csr:

```
openssl req -config <openssl config file> \
      -key <root key> \
      -new -x509 -days 7300 -sha256 -extensions v3_ca \
      -out <the location of certificate>
```

When creating certificates it will ask for details and detials should be entered correctly :

For the root and intermediate the common name could be anything but for the server certificate the common name should be the dns server name for that ip of server(web service)

I had my dns name server_cse for cse server and had this included in the /etc/hosts to get the correct mapping

The browser should be given the root certificate of the ca so that it can verify any service of the server under this ca by looking into the hierarchy in the order root(browser has this certificate after we give it),the server will give its own certificate and the chain of certificate till its intermediate.The browser/client evaluates this validity by looking into these files.

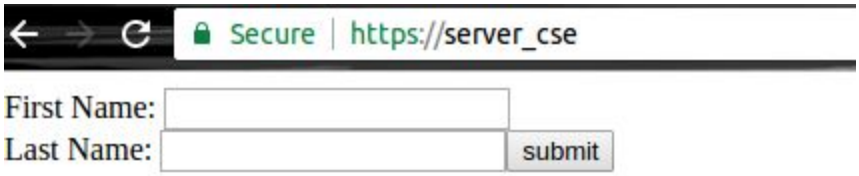
2) Create a web server with https support and send a form

->I have added the root certificate of the ca to my browser.

->I add the ca-chain till that server(catenation from root till that intermediate CA) ,server certificate and server key to the server and run as server.I have written a python program to work as the web server.

->the common name of my server certificate is server_cse and that is the dns name for mylocal host.This gives me a valid https page.

->The certificates are added by making a ssl layer wrapped on the http socket and the certificates are used in making this ssl layer.This layer converts http -> https



->This has the form which takes input and i used get method on this post.

The wireshark results of this are:

102	21.358793622	192.168.0.102	216.58.204.35	TLSv1.2	112 Application Data
103	21.360743603	74.125.200.189	192.168.0.102	TLSv1.2	126 Application Data
107	21.369444105	216.58.197.46	192.168.0.102	TLSv1.2	112 Application Data
112	21.810405053	216.58.204.35	192.168.0.102	TLSv1.2	112 Application Data
113	21.810418996	216.58.204.35	192.168.0.102	TLSv1.2	138 Application Data
114	21.810429210	216.58.204.35	192.168.0.102	TLSv1.2	112 Application Data
116	21.811735663	192.168.0.102	216.58.204.35	TLSv1.2	112 Application Data
117	21.812929512	192.168.0.102	216.58.197.35	TLSv1.2	142 Application Data
118	21.814193690	192.168.0.102	216.58.197.35	TLSv1.2	437 Application Data, Application Data
121	21.837707188	216.58.197.35	192.168.0.102	TLSv1.2	112 Application Data
123	21.929006521	216.58.197.35	192.168.0.102	TLSv1.2	135 Application Data
125	21.929756484	216.58.197.35	192.168.0.102	TLSv1.2	112 Application Data
127	21.929915464	192.168.0.102	216.58.197.35	TLSv1.2	112 Application Data
132	22.708154522	216.58.197.46	192.168.0.102	TLSv1.2	139 Application Data
134	22.708231286	216.58.197.46	192.168.0.102	TLSv1.2	112 Application Data
137	22.709705529	192.168.0.102	216.58.197.46	TLSv1.2	112 Application Data
141	25.329750491	192.168.0.102	216.58.197.46	TLSv1.2	289 Application Data
144	25.332161723	192.168.0.102	216.58.197.46	TLSv1.2	1202 Application Data
149	25.701635851	216.58.197.46	192.168.0.102	TLSv1.2	139 Application Data
150	25.701672178	216.58.197.46	192.168.0.102	TLSv1.2	112 Application Data
153	25.703077832	192.168.0.102	216.58.197.46	TLSv1.2	112 Application Data
159	26.623410320	74.125.200.189	192.168.0.102	TLSv1.2	126 Application Data
161	27.032538929	74.125.200.189	192.168.0.102	TLSv1.2	126 Application Data

3)**Secure Peer to peer chat application using pki:**

I have made a p2p chat application which runs server and client at same time.The client will have the root certificates and server will gives the certificate of server and the chain certificate.This happens from both sides once the certificates are verified from both sides the communication can happen.

I have done this part using peers in the same intermediate ca as well peers belonging to different intermediate ca's but same root ca.In both cases verification happens by looking at chain and the root certificate of the peers.The client sockets wraps the root certificate on its socket and server socket wraps ssl layer of server certificate ,chain of certificates and server key ,the ssl layer can be unwrapped at client only if it has the correct root certificate ,so messages are passed securely.

Running of the program:

User1:

```
→ certs python q_3_chat.py
2450
2449
0
socket created
socket created
Bind worked

accepted client connection to address ('127.0.0.1', 60678)

Enter PEM pass phrase:
ssl wrap succeeded for server
wrapped client socket for SSL
server: 1234
client socket connected
Enter message to send
server: 1234
hi
Enter message to send
server: i am user 2 sending message to user 1
i am user 1 sending message to user 2
Enter message to send
```

User 2:

```
→ certs python q_3_chat.py
2449
2450
0
socket created
socket created
Bind worked

wrapped client socket for SSL
client socket connected
Enter message to send
accepted client connection to address ('127.0.0.1', 59770)

Enter PEM pass phrase:Enter message to send

ssl wrap succeeded for server
1234
Enter message to send
server: hi
i am user 2 sending message to user 1
Enter message to send
server: i am user 1 sending message to user 2
```

Wireshark capture for The chat application(3rd part):

```
113 24.869736903  192.168.0.102   216.58.197.46   TLSv1.2   119 Application Data
114 24.869948826  192.168.0.102   216.58.197.46   TLSv1.2   164 Application Data, Application Data
115 24.870177144  192.168.0.102   216.58.197.46   TLSv1.2  1146 Application Data
116 24.870309828  192.168.0.102   216.58.197.46   TLSv1.2  1434 Application Data, Application Data, Application Data, Application Data, Application Data
117 24.870849943  192.168.0.102   216.58.197.46   TLSv1.2  1200 Application Data
118 24.882803446  216.58.197.46   192.168.0.102   TLSv1.2   122 Application Data
119 24.882822131  216.58.197.46   192.168.0.102   TLSv1.2   108 Application Data
120 24.882827091  216.58.197.46   192.168.0.102   TLSv1.2   122 Application Data
121 24.882832789  216.58.197.46   192.168.0.102   TLSv1.2   108 Application Data
122 24.882833441  192.168.0.102   216.58.197.46   TCP        66 36010 → 443 [ACK] Seq=4467 Ack=259 Win=30336 Len=0 TSval=7166283 TSecr=4245900697
123 24.882911258  192.168.0.102   216.58.197.46   TLSv1.2   104 Application Data
124 24.889357300  216.58.197.46   192.168.0.102   TLSv1.2   122 Application Data
125 24.889369734  216.58.197.46   192.168.0.102   TLSv1.2   108 Application Data
126 24.891341872  216.58.197.46   192.168.0.102   TLSv1.2   122 Application Data
127 24.891353590  216.58.197.46   192.168.0.102   TLSv1.2   108 Application Data
128 24.891358121  216.58.197.46   192.168.0.102   TCP        66 443 → 36010 [ACK] Seq=259 Ack=885 Win=44672 Len=0 TSval=4245900707 TSecr=7166280
129 24.891363626  216.58.197.46   192.168.0.102   TLSv1.2   104 Application Data
130 24.892830701  216.58.197.46   192.168.0.102   TCP        66 443 → 36010 [ACK] Seq=297 Ack=3333 Win=49536 Len=0 TSval=4245900709 TSecr=7166280
131 24.904418883  216.58.197.46   192.168.0.102   TCP        66 443 → 36010 [ACK] Seq=297 Ack=4505 Win=52224 Len=0 TSval=4245900721 TSecr=7166280
```

Q4) ***Running the p2p application between peers of different CA's***

I have created another pki for IITB similar to that of IITH with different roots.

Here Instead of giving the root of peers to client which were same in previous case ,I have made the chain of the root certificates of the peers which would be the same for all the peers and can be used for verification and thus achieved the communication between peers belonging to different roots.

Wireshark Capture for this part:

```
 77 22.425628501  74.125.200.189   192.168.0.102   TLSv1.2   112 Application Data
 78 22.425633272  192.168.0.102   74.125.200.189   TCP        66 56870 → 443 [ACK] Seq=429 Ack=549 Win=1444 Len=0 TSval=7202175 TSecr=4177849696
 79 22.426012786  192.168.0.102   74.125.200.189   TLSv1.2   112 Application Data
 80 22.528793780  74.125.200.189   192.168.0.102   TCP        66 443 → 56870 [ACK] Seq=549 Ack=475 Win=1622 Len=0 TSval=4177849895 TSecr=7202175
 81 22.732404334  74.125.200.189   192.168.0.102   TLSv1.2   127 Application Data
 82 22.732999659  192.168.0.102   74.125.200.189   TLSv1.2   395 Application Data
 83 22.787119732  74.125.200.189   192.168.0.102   TCP        66 443 → 56870 [ACK] Seq=610 Ack=804 Win=1620 Len=0 TSval=4177850153 TSecr=7202252
 84 23.245040433  74.125.200.189   192.168.0.102   TLSv1.2   146 Application Data
 85 23.245067677  74.125.200.189   192.168.0.102   TLSv1.2   125 Application Data
 86 23.245154905  192.168.0.102   74.125.200.189   TCP        66 56870 → 443 [ACK] Seq=804 Ack=749 Win=1444 Len=0 TSval=7202380 TSecr=4177850528
 87 23.962157197  74.125.200.189   192.168.0.102   TLSv1.2   125 Application Data
 88 24.000005708  192.168.0.102   74.125.200.189   TCP        66 56870 → 443 [ACK] Seq=804 Ack=808 Win=1444 Len=0 TSval=7202569 TSecr=4177851254
 89 24.009537885  192.168.0.102   216.58.197.46   TLSv1.2   345 Application Data
 90 24.009752884  192.168.0.102   216.58.197.46   TLSv1.2  1229 Application Data, Application Data
 91 24.031360166  216.58.197.46   192.168.0.102   TCP        66 443 → 35910 [ACK] Seq=1 Ack=280 Win=922 Len=0 TSval=3322274365 TSecr=7202571
 92 24.031654968  216.58.197.46   192.168.0.102   TCP        66 443 → 35910 [ACK] Seq=1 Ack=1443 Win=944 Len=0 TSval=3322274365 TSecr=7202571
 93 24.031661411  216.58.197.46   192.168.0.102   TLSv1.2   112 Application Data
 94 24.067990468  192.168.0.102   216.58.197.46   TCP        66 35910 → 443 [ACK] Seq=1443 Ack=47 Win=572 Len=0 TSval=7202586 TSecr=3322274365
 95 24.117926447  216.58.197.46   192.168.0.102   TLSv1.2   131 Application Data
 96 24.117942577  192.168.0.102   216.58.197.46   TCP        66 35910 → 443 [ACK] Seq=1443 Ack=112 Win=572 Len=0 TSval=7202598 TSecr=3322274450
 97 24.117954802  216.58.197.46   192.168.0.102   TLSv1.2   112 Application Data
 98 24.117960237  192.168.0.102   216.58.197.46   TCP        66 35910 → 443 [ACK] Seq=1443 Ack=158 Win=572 Len=0 TSval=7202598 TSecr=3322274450
 99 24.118451941  192.168.0.102   216.58.197.46   TLSv1.2   112 Application Data
100 24.179320081  216.58.197.46   192.168.0.102   TCP        66 443 → 35910 [ACK] Seq=158 Ack=1489 Win=944 Len=0 TSval=3322274513 TSecr=7202598
101 26.009451181  216.58.197.46   192.168.0.102   TLSv1.2   129 [TCP ACKed unseen segment] Application Data
102 26.009478143  216.58.197.46   192.168.0.102   TCP        66 [TCP ACKed unseen segment] 443 → 35986 [FIN, ACK] Seq=64 Ack=2 Win=415 Len=0 TSval=586849345 TSecr=7143
103 26.009590358  192.168.0.102   216.58.197.46   TCP        66 [TCP Previous segment not captured] 35986 → 443 [FIN, ACK] Seq=2 Ack=65 Win=581 Len=0 TSval=7203071 TSe
104 26.030318756  216.58.197.46   192.168.0.102   TCP        66 [TCP ACKed unseen segment] 443 → 35986 [ACK] Seq=65 Ack=3 Win=415 Len=0 TSval=586849421 TSecr=7203071
105 27.750500295  74.125.200.189   192.168.0.102   TLSv1.2   127 Application Data
106 27.750531515  192.168.0.102   74.125.200.189   TCP        66 56870 → 443 [ACK] Seq=804 Ack=869 Win=1444 Len=0 TSval=7203506 TSecr=4177855005
107 31.900003888  192.168.0.102   34.251.25.218   TCP        66 36688 → 443 [ACK] Seq=1 Ack=1 Win=229 Len=0 TSval=7204544 TSecr=23659539
108 31.901604445  74.125.200.189   192.168.0.102   TLSv1.2   168 Application Data
109 31.901626029  192.168.0.102   74.125.200.189   TCP        66 56870 → 443 [ACK] Seq=804 Ack=971 Win=1444 Len=0 TSval=7204544 TSecr=4177859250
110 32.028000439  192.168.0.102   216.58.197.46   TCP        66 35972 → 443 [ACK] Seq=1 Ack=1 Win=262 Len=0 TSval=7204576 TSecr=738451014
```

All the codes,wireshark captures and the readme.txt to run those are included in the submission.