

# Lab 10: TLS

50.020 Security

Hand-out: December 1

Hand-in: December 8, 11am

## 1 Objective

By the end of this lab, you should be able to:

- Create simple key-management system
- Create certificate for TLS/SSL connection
- Wrap socket using TLS/SSL
- Use Wireshark to capture packets transmission

## 2 Part 0: Do this before you come to class

- Read Python SSL module: <https://docs.python.org/2.6/library/ssl.html>

## 3 Part I: Key Management System Client-Server

- We are going to implement a simple protocol for key management system similar to Kerberos. If A wants to send a secure message to B, A would request the shared key from a trusted server. The server would respond by generating the shared key between A and B, and encrypt them for both A and B. The protocol can be written as follows.

$$A \rightarrow S : A, B \quad (1)$$

$$S \rightarrow A : \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \quad (2)$$

$$A \rightarrow B : \{T_S, L, K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}} \quad (3)$$

$$B \rightarrow A : \{T_A + 1\}_{K_{AB}} \quad (4)$$

The above protocol means that Alice asks the server for a shared key with Bob. The server then creates the ticket  $\{\{T_S, L, K_{AB}, A\}_{K_{BS}}\}$  containing the shared key  $K_{AB}$  and given to Alice to use. Alice also gets a copy of the key in a form readable by her encrypted under  $K_{AS}$ . She now sends the ticket and verifies the ticket by sending a timestamp  $T_A$  to Bob. Bob responds by sending back the timestamp incremented by 1. This means that Bob can decrypt the ticket correctly and extract the key  $K_{AB}$ .

- Create studentserver.py:

- Create a socket connection using `socket.socket()`.
- Bind socket to localhost with port 10023 or any other unused port number.
- Listen to the port using `socket.listen()`.
- Create a loop to accept socket using `socket.accept()`.
- Complete `deal_with_client`:
  - \* Prompt user with a menu: 0) Exit 1) Create user account, 2) Request Keys. Use `socket.socket.send(s)` to send the prompt.
  - \* Read reply from client using `socket.socket.recv(buffsize)`.
    - If client send '0', then server will reply with a string '%' to quit.
    - If client send '1', then create a new user account. Request for a username and password. Ask client to enter the password one more time. Check whether username is already in a file called 'passwd'. If not, create username and hash password using md5. Everytime a new user is created, generate an 80-bit shared key between the client and the server. Store the new username, hashed password, and shared key in a file called 'passwd'. Send the shared key to the client in PEM format. Send '%' for client to quit.
    - If client send '2', then create shared keys for client and destination user. Ask client for its username and password. The client must be found in the file 'passwd'. Prompt for the destination username. Find the usernames in the file 'passwd'. If found, generate a shared key between the client and the destination user. Send the following data back to the client:

$$\{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}. \quad (5)$$

Where,  $T_S$  is the current timestamp,  $L$  is the lifetime.  $K_{AB}$  is the shared key between the client (A) and the destination user (B),  $K_{BS}$  is the shared key between the destination username and the server,  $K_{AS}$  is the shared key between the client and the server. Use '\$\$' to separate between different fields in the message. Note that the shared key is encrypted two times, one is for the destination user using the key  $K_{BS}$ , and the other one is for the client using the key  $K_{AS}$ . Send the message in PEM format. Use PRESENT in ECB mode to encrypt.

- If client send others, prompt the menu again.

- Create `studentclient.py`:

- Use localhost with port 10023 (or any other port number) as server to connect to.
- Create a socket connection using `socket.socket()`.
- Connect to your `server.py` using `socket.socket.connect()`.
  - \* Display the common name, email, and date for the valid certificate of the server.
  - \* Read prompt from server using `socket.socket.recv(buffsize)`.
  - \* Display prompt, and if the last character is '%', then exit and terminate the program. If not exit, then read from keyboard.
  - \* If the server is sending the requested shared key, store it into a file.
  - \* Send the characters read from keyboard to server.

- Create openkey.py that enables you to do the following:
  - Read shared key between A and server from a file.
  - Read received encrypted message from the server containing the shared key.
  - Decrypt message from the server using PRESENT in ECB mode with the shared key between A and server.
  - Parse the message to obtain: 1) timestamp, 2) lifetime, 3) shared key between A and B, 4) B username, 5) encrypted message to be send to B.
  - Read encrypted message sent by A. The message is encrypted by the shared key between B and server.
  - Parse the message to obtain: 1) timestamp, 2) lifetime, 3) shared key between A and B, 5) A username.

## 4 Part II: Sniffing Packet using Wireshark

- Run wireshark from the terminal:

```
$ sudo wireshark
```

- Run studentserver.py in one terminal:

```
$ python studentserver.py
```

- Run studentclient.py in another terminal:

```
$ python studentclient.py
```

- Use studentclient.py to request key. Go to wireshark capture to find the username entered and the password.
- Refer to appendix for some screenshot for wireshark capture.

## 5 Part III: Creating Certificate X509

- Use your pyrsa.py from Lab 9 to generate a new RSA private key in PEM format.
- Create a certificate request from your private key using OpenSSL. Use your student ID as your *Common Name*, and key in your student email address.

```
$ openssl req -new -key yourprivkey.pem -out req.pem
```

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:SG  
State or Province Name (full name) [Some-State]:Singapore  
Locality Name (eg, city) []:Singapore  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XXX  
Organizational Unit Name (eg, section) []:YYY  
Common Name (e.g. server FQDN or YOUR name) []:1000001  
Email Address []:1000001@sutd.edu.sg

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

- Submit to eDimension for CA to sign your certificate by Wednesday 12pm. TA will notify you once the certificate is ready for you to download from eDimension.

## 6 Part IV: TLS/SSL

- We are going to wrap the socket connection using SSL.

```
import ssl
```

- Inside studentserver.py and studentclient.py:
  - Wrap the accepted socket using ssl with `ssl.wrap_socket()`. For server side, set `server_side` to `True`, and specify the private key you have generated as well as the signed certificate from CA. For client side, specify the location of CA certificate, and set `cert_reqs` to `ssl.CERT_REQUIRED`.
  - Change `socket.socket.recv(buffsize)` to `ssl.SSLSocket.read()`.
  - Change `socket.socket.send(s)` to `ssl.SSLSocket.write(s)`.
- Try to capture the data transferred using Wireshark.

## 7 Checkoff

- Show the tls/ssl data sniffed by wireshark.
- How would you verify the certificate of the root CA?
- Explain what's the use of certificate in this protocol.
- How can the client check on the identity of the server?
- How does server authenticate the users requesting for keys?
- What's the purpose of timestamp and lifetime?

## 8 Test Scenario

- Beginning Prompt of client.py:

```
$ python studentclient.py
Connecting to:
commonName = 1000001
email = 1000001@sutd.edu.sg
certificate valid until = Nov 12 06:56:44 2015 GMT
=====
```

```
0) Exit
1) Create account
2) Request Key
```

```
>
```

- If user enter '0':

```
0) Exit
1) Create account
2) Request Key
```

```
> 0
```

```
Quitting
```

- Create first user, enter '1' to create new user:

```
0) Exit
1) Create account
2) Request Key
```

```
> 1
```

```
Enter username:
```

```
> 1000003
```

```
Enter password:
```

```
> 1234
```

```
Enter password one more time:
```

```
> 1234
```

```
MHZ2ntBUCYNxeQ==
```

```
Quitting
```

Copy the shared key in PEM format to a text file.

- Create second user:

```
0) Exit
1) Create account
2) Request Key
```

```
> 1
Enter username:
> 1000004
Enter password:
> 5678
Enter password one more time:
> 5678
tQJAjThb96LfDg==
Quitting
```

- Request key between 1000003 and 1000004 from server:

```
0) Exit
1) Create account
2) Request Key

> 2
Enter your username:
> 1000003
Enter password:
> 1234
Enter destination username:
> 1000004
Receiving key at: receivedkey.txt

Quitting
```

- Running openkey.py to parse receivedkey.txt:

```
=====
Timestamp in second: 1416972723.92
Lifetime in second: 86400
Shared key with B in PEM: t2j0Sf8LlanDGQ==
B username: 1000004
=====
Timestamp in second: 1416972723.92
Lifetime in second: 86400
Shared key with A in PEM: t2j0Sf8LlanDGQ==
A username: 1000003
```

## 9 Appendix: Screenshot for Wireshark capture

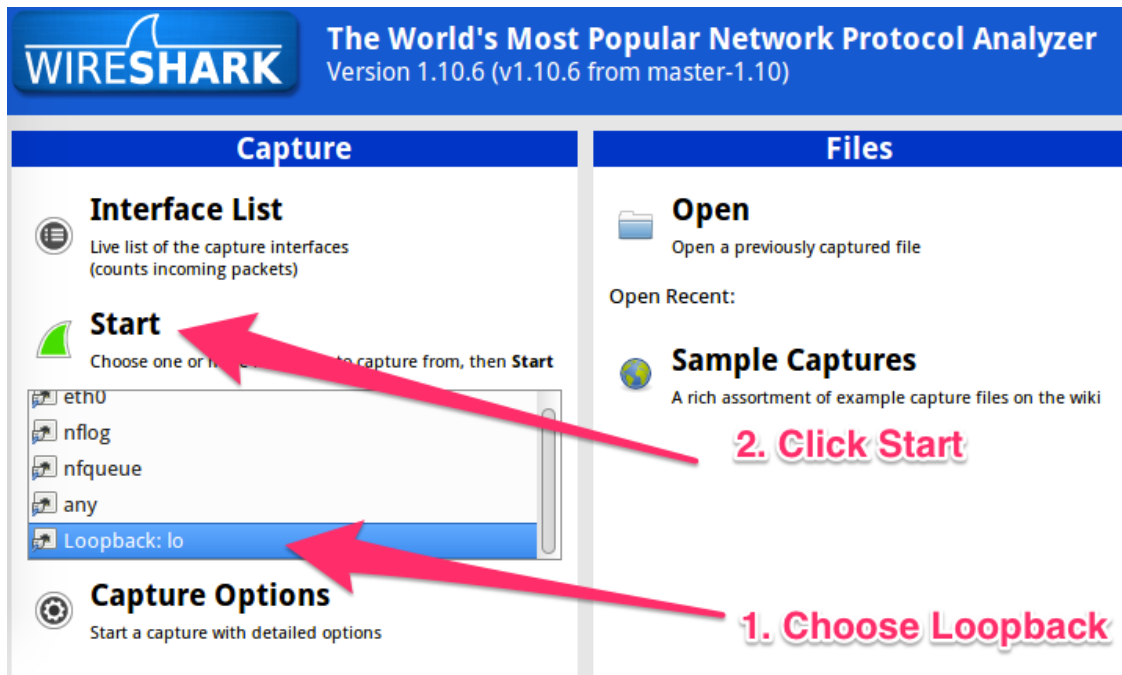


Figure 1: Start capturing localhost in wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
41	116.8262350	127.0.0.1	127.0.0.1	TCP	87	10023 > 56434 [PSH, ACK] Seq=80 Ack=14 Win=
42	116.8263440	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=14 Ack=101 Win=4377
43	122.6578860	127.0.0.1	127.0.0.1	TCP	73	56434 > 10023 [PSH, ACK] Seq=14 Ack=101 Win=
44	122.6579560	127.0.0.1	127.0.0.1	TCP	82	10023 > 56434 [PSH, ACK] Seq=101 Ack=21 Win=
45	122.6579810	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=21 Ack=117 Win=4377
46	124.2416920	127.0.0.1	127.0.0.1	TCP	70	56434 > 10023 [PSH, ACK] Seq=21 Ack=117 Win=
47	124.2419420	127.0.0.1	127.0.0.1	TCP	94	10023 > 56434 [PSH, ACK] Seq=117 Ack=25 Win=

▶ Frame 41: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0  
 ▶ Ethernet II, Src: 00:00:00 00:00:00 (00:00:00:00:00:00), Dst: 00:00:00 00:00:00 (00:00:00:00:00:00)  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)  
 ▶ Transmission Control Protocol, Src Port: 10023 (10023), Dst Port: 56434 (56434), Seq: 80, Ack: 14, Len: 21  
 ▼ Data (21 bytes)  
 Data: 456e74657220796f757220757365726e616d653a20  
 [Length: 21]

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  ....E.
0010  00 49 86 44 40 00 00 06 b6 68 7f 00 00 01 7f 00  .I.D@. .h....
0020  00 01 27 27 dc 72 42 8e 4c 0c fc c6 ad af 80 18  .'.rB. L.....
0030  01 56 fe 3d 00 00 01 01 08 0a 00 0b 39 09 00 0b  .V.=.....9...
0040  39 09 45 6e 74 65 72 20 79 6f 75 72 20 75 73 65  9.Enter your use
0050  72 6e 61 6d 65 3a 20                               rname:
  
```

Figure 2: Captured enter username prompt.

No.	Time	Source	Destination	Protocol	Length	Info
41	116.8262350	127.0.0.1	127.0.0.1	TCP	87	10023 > 56434 [PSH, ACK] Seq=
42	116.8263440	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=14 Ac
43	122.6578860	127.0.0.1	127.0.0.1	TCP	73	56434 > 10023 [PSH, ACK] Seq=
44	122.6579560	127.0.0.1	127.0.0.1	TCP	82	10023 > 56434 [PSH, ACK] Seq=
45	122.6579810	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=21 Ac
46	124.2416920	127.0.0.1	127.0.0.1	TCP	70	56434 > 10023 [PSH, ACK] Seq=
47	124.2419420	127.0.0.1	127.0.0.1	TCP	94	10023 > 56434 [PSH, ACK] Seq=
▶ Frame 43: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00) ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1) ▶ Transmission Control Protocol, Src Port: 56434 (56434), Dst Port: 10023 (10023), Seq: 14, Ack: 101, Len: 7 ▼ Data (7 bytes) Data: 31303030303031 [Length: 7]						
0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00	.....E.			
0010	00 3b 60 65 40 00 40 06	dc 55 7f 00 00 01 7f 00	.;e@.@..U.....			
0020	00 01 dc 72 27 27 fc c6	ad af 42 8e 4c 21 80 18	...r'...'..B.L!..			
0030	01 56 fe 2f 00 00 01 01	08 0a 00 0b 3e bb 00 0b	.V./.....>...			
0040	39 09 31 30 30 30 30 30	31	9.100000 1			

Figure 3: Captured username entered.

No.	Time	Source	Destination	Protocol	Length	Info
41	116.8262350	127.0.0.1	127.0.0.1	TCP	87	10023 > 56434 [PSH, ACK] Seq=
42	116.8263440	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=14 A
43	122.6578860	127.0.0.1	127.0.0.1	TCP	73	56434 > 10023 [PSH, ACK] Seq=
44	122.6579560	127.0.0.1	127.0.0.1	TCP	82	10023 > 56434 [PSH, ACK] Seq=
45	122.6579810	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq=21 A
46	124.2416920	127.0.0.1	127.0.0.1	TCP	70	56434 > 10023 [PSH, ACK] Seq=
47	124.2419420	127.0.0.1	127.0.0.1	TCP	94	10023 > 56434 [PSH, ACK] Seq=
▶ Frame 44: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00) ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1) ▶ Transmission Control Protocol, Src Port: 10023 (10023), Dst Port: 56434 (56434), Seq: 101, Ack: 21, Len: 16 ▼ Data (16 bytes) Data: 456e7465722070617373776f72643a20 [Length: 16]						
0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00	.....E.			
0010	00 44 86 45 40 00 40 06	b6 6c 7f 00 00 01 7f 00	.D.E@.@..l.....			
0020	00 01 27 27 dc 72 42 8e	4c 21 fc c6 ad b6 80 18	...'rB. L!.....			
0030	01 56 fe 38 00 00 01 01	08 0a 00 0b 3e bb 00 0b	.V.8....>...			
0040	3e bb 45 6e 74 65 72 20	70 61 73 73 77 6f 72 64	>.Enter password			
0050	3a 20		:			

Figure 4: Captured enter password prompt.



No.	Time	Source	Destination	Protocol	Length	Info
41	116.82623506	127.0.0.1	127.0.0.1	TCP	87	10023 > 56434 [PSH, ACK
42	116.82634406	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq
43	122.65788606	127.0.0.1	127.0.0.1	TCP	73	56434 > 10023 [PSH, ACK
44	122.65795606	127.0.0.1	127.0.0.1	TCP	82	10023 > 56434 [PSH, ACK
45	122.65798106	127.0.0.1	127.0.0.1	TCP	66	56434 > 10023 [ACK] Seq
46	124.24169206	127.0.0.1	127.0.0.1	TCP	70	56434 > 10023 [PSH, ACK
47	124.24194206	127.0.0.1	127.0.0.1	TCP	94	10023 > 56434 [PSH, ACK
▶ Frame 46: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00) ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1) ▶ Transmission Control Protocol, Src Port: 56434 (56434), Dst Port: 10023 (10023), Seq: 21, Ack: 117, Len ▼ Data (4 bytes) Data: 31323334 [Length: 4]						
0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00	.....E.			
0010	00 38 60 67 40 00 40 06	dc 56 7f 00 00 01 7f 00	.8`g@.@. .V.....			
0020	00 01 dc 72 27 27 fc c6	ad b6 42 8e 4c 31 80 18	...r'...'..B.L1..			
0030	01 56 fe 2c 00 00 01 01	08 0a 00 0b 40 47 00 0b	.V.,.....@G..			
0040	3e bb 31 32 33 34		>.1234			

Figure 5: Captured password entered.