LAB

1. Create certs
2. Run HTTPS server
3. Capture encrypted traffic
4. Decrypt in Wireshark

Steps

1. **Create a self-signed *CA*, issue a *server certificate* and a *client certificate*.**

2. **Run a Python HTTPS server using the server certificate (optionally configure RSA key-exchange for key-file decryption).**

3. **Browse the server while capturing in Wireshark; observe encrypted TLS application data.**

4. **Decrypt in Wireshark by (A) loading the browser SSLKEYLOGFILE or (B) loading the server private key**
5. **Verify decrypted HTTP content in Wireshark and export objects; answer short questions.**

1- Create a private key for CA and self-signed CA certificate

```
openssl genpkey -algorithm RSA -out ca.key -pkeyopt rsa_keygen_bits:2048

openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 \
  -subj "/C=PK/ST=Punjab/L=Lahore/O=BSIT/OU=PUCIT/CN=NetSecLab-CA" \
  -out ca.crt
```

2- Create server key and CSR, then sign with CA

```
openssl genpkey -algorithm RSA -out server.key -pkeyopt rsa_keygen_bits:2048

openssl req -new -key server.key  -subj "/C=PK/ST=Punjab/L=Lahore/O=BSIT/OU=Servers/CN=localhost" \
  -out server.csr

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial \
  -out server.crt -days 365 -sha256 \
  -extfile <(printf "subjectAltName=DNS:localhost,IP:127.0.0.1")
```

Server Code:

```
import http.server, ssl, pathlib, os, sys
```

```
PORT = 4443
WWW = pathlib.Path.cwd()/ "www"
WWW.mkdir(exist_ok=True)
(WWW/"index.html").write_text("<html><body><h1>TLS Lab</h1><p>Hello!!!!!!!!! BSIT from a secured
server, Can you read this?</p></body></html>")

class Handler(http.server.SimpleHTTPRequestHandler):
    def __init__(self, *a, **kw):
        super().__init__(*a, directory=str(WWW), **kw)

if __name__ == "__main__":
    server_address = ('0.0.0.0', PORT)
    httpd = http.server.HTTPServer(server_address, Handler)


    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)

    context.load_cert_chain(certfile="server.crt", keyfile="server.key")


    try:
        context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1  # disable old TLS versions

        context.minimum_version = ssl.TLSVersion.TLSv1_2
        context.maximum_version = ssl.TLSVersion.TLSv1_2

        context.set_ciphers("AES128-SHA")
    except Exception as e:
        print("Cipher selection may not be supported on this platform:", e)

    httpd.socket = context.wrap_socket(httpd.socket, server_side=True)
    print(f"Serving on https://localhost:{PORT} (pid {os.getpid()})")
    httpd.serve_forever()
```

Start:

```
python3 https_server.py
```

Start capture. In browser, go to: https://localhost:4443

If a browser complains about an untrusted CA,(this is expected since CA is self-signed) — or add ca.crt to their browser OS trust store for a smooth experience.

1. Stop capture once page loads. Save capture as tls_lab_capture.pcapng.

What students will observe in Wireshark (before decryption):

- Filter tls or tcp.port==4443.
- Handshake packets: Client Hello, Server Hello, Certificate, Server Key Exchange (if ECDHE), Client Key Exchange, Change Cipher Spec, Application Data.
- Application Data packets shown as **“Application Data”** (encrypted blob). No readable HTTP payload.

Explanation: The TLS handshake negotiated keys; the subsequent Application Data packets are encrypted

Decryption:

```
export SSLKEYLOGFILE=<path>/tls_lab/sslkeylog.log
```

1. Close all browser instances.
2. In a terminal set environment variable and start the browser from that terminal:

```
export SSLKEYLOGFILE=$HOME/tls_lab/sslkeylog.log   # Linux/macOS
# On Windows (PowerShell): $env:SSLKEYLOGFILE = "C:\path\to\tls_lab\sslkeylog.log"
# or CMD: set SSLKEYLOGFILE=C:\path\to\tls_lab\sslkeylog.log
```

3. Launch the browser from that shell, e.g. google-chrome or firefox. Visit https://localhost:4443. The browser will append session keys to sslkeylog.log.
4. In Wireshark: Edit → Preferences → Protocols → TLS

Close all browser instances.

In a terminal set environment variable and start the browser from that terminal:

```
export SSLKEYLOGFILE=$HOME/tls_lab/sslkeylog.log   # Linux/macOS
# On Windows (PowerShell): $env:SSLKEYLOGFILE = "C:\path\to\tls_lab\sslkeylog.log"
# or CMD: set SSLKEYLOGFILE=C:\path\to\tls_lab\sslkeylog.log
```

Launch the browser from that shell, e.g. google-chrome or firefox. Visit https://localhost:4443. The browser will append session keys to sslkeylog.log.

In Wireshark: Edit → Preferences → Protocols → TLS

Set (Pre)-Master-Secret log filename to the path sslkeylog.log.

Click OK, then reload/open the capture.
What to expect: Wireshark will now decrypt TLS sessions (including ECDHE ones). Use filter tls and http or http to see HTTP requests (GET /) and HTML response.
Why it works: The browser wrote the secrets during the handshake; anyone possessing that file can decrypt corresponding captured TLS sessions. This demonstrates how compromise of session secrets can reveal plaintext even when PFS is used.

Method B

In Wireshark go to Edit → Preferences → Protocols → TLS.
Under **(RSA) Keys list** (or "RSA keys list") click **Edit** → **New** and add:
IP address: 127.0.0.1 (or leave blank)

- Port: 4443
- Protocol: http
- Key File: path to server.key (PEM).

Click OK. Re-open the capture.