TLS and SSL

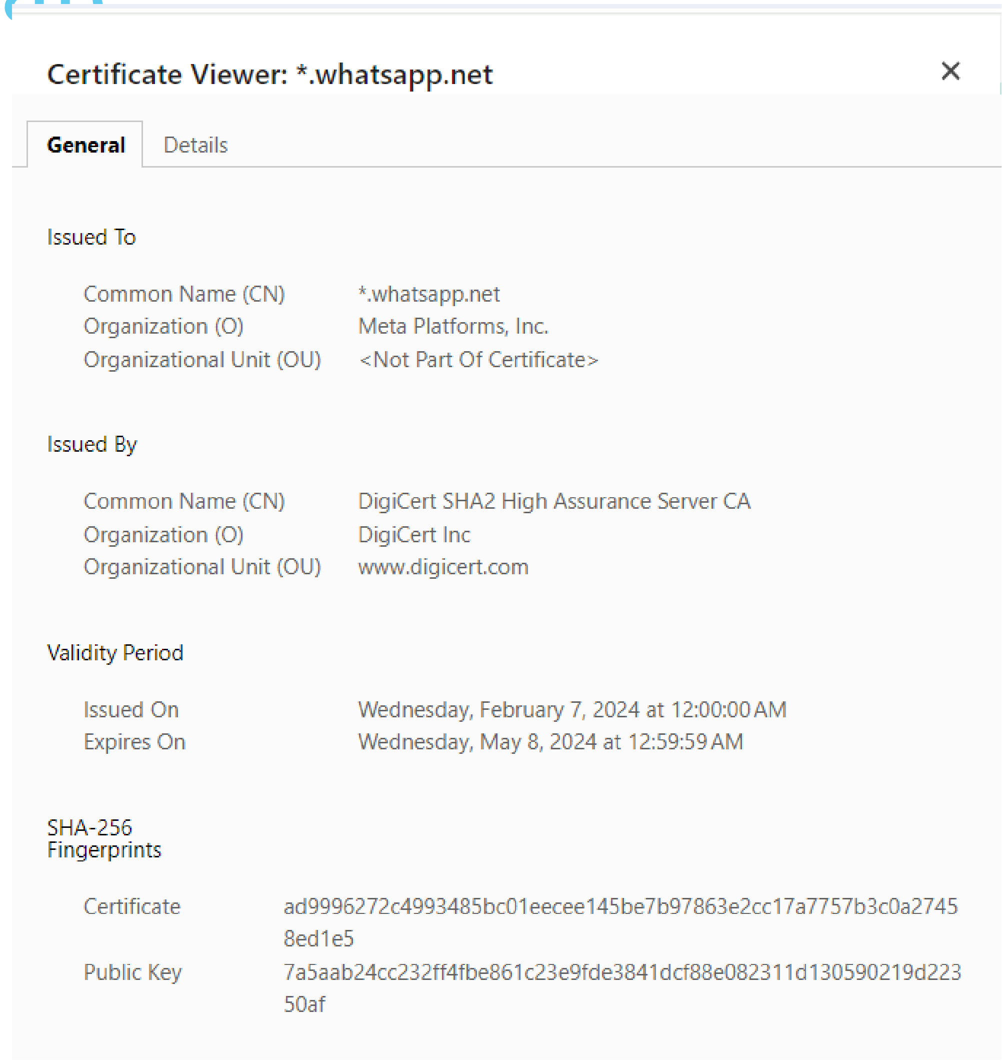
Transport Layer Security

- The TLS protocol, protects data using encryption.
- When users send their information to a website, TLS encrypts it before sending it.
- Then, only the server with the same public key as the client can open the message.
- This rule also applies when the server sends information back to the client.
- Only the client with the corresponding key can read the data.

Transport Layer Security Cont.

- For a website to use TLS protocol, you must install a valid TLS/SSL certificate (often called an SSL certificate).
- the certificate is a data file that contains the website’s identity and the public key for opening payload messages.
- An SSL certificate must be valid to work. This means that not only must a credible certificate authority (CA) sign it, but the certificate also must be active.
- Every certificate has an issuance date and an expiration date. A certificate is no longer valid after its expiration date.

SSL Certificate



Certificate Viewer: *.whatsapp.net

Issued To

Common Name (CN)	*.whatsapp.net
Organization (O)	Meta Platforms, Inc.
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	DigiCert SHA2 High Assurance Server CA
Organization (O)	DigiCert Inc
Organizational Unit (OU)	www.digicert.com

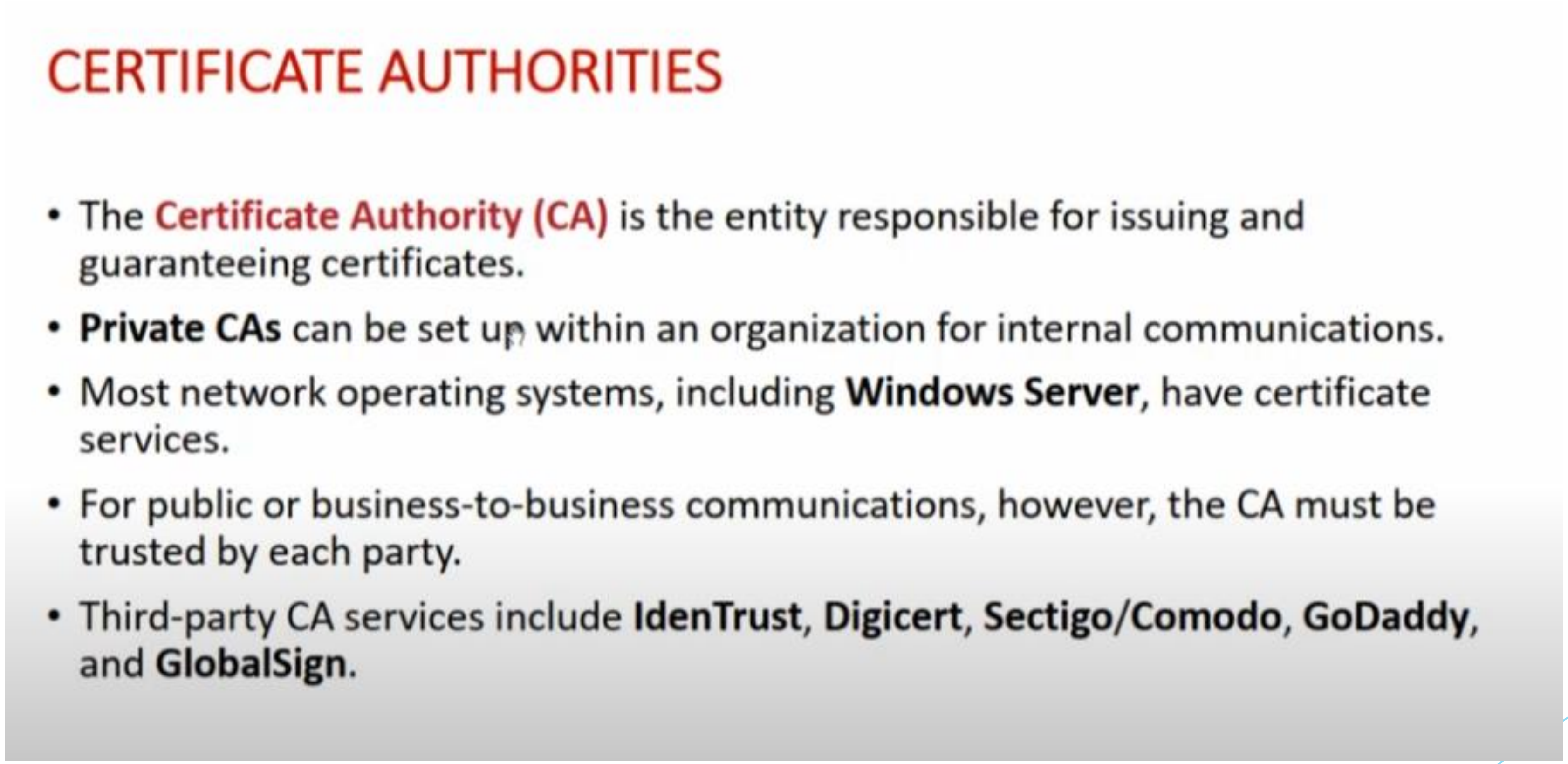
Validity Period

Issued On	Wednesday, February 7, 2024 at 12:00:00 AM
Expires On	Wednesday, May 8, 2024 at 12:59:59 AM

SHA-256 Fingerprints

Certificate	ad9996272c4993485bc01eecee145be7b97863e2cc17a7757b3c0a27458ed1e5
Public Key	7a5aab24cc232ff4fbe861c23e9fde3841dcf88e082311d130590219d22350af

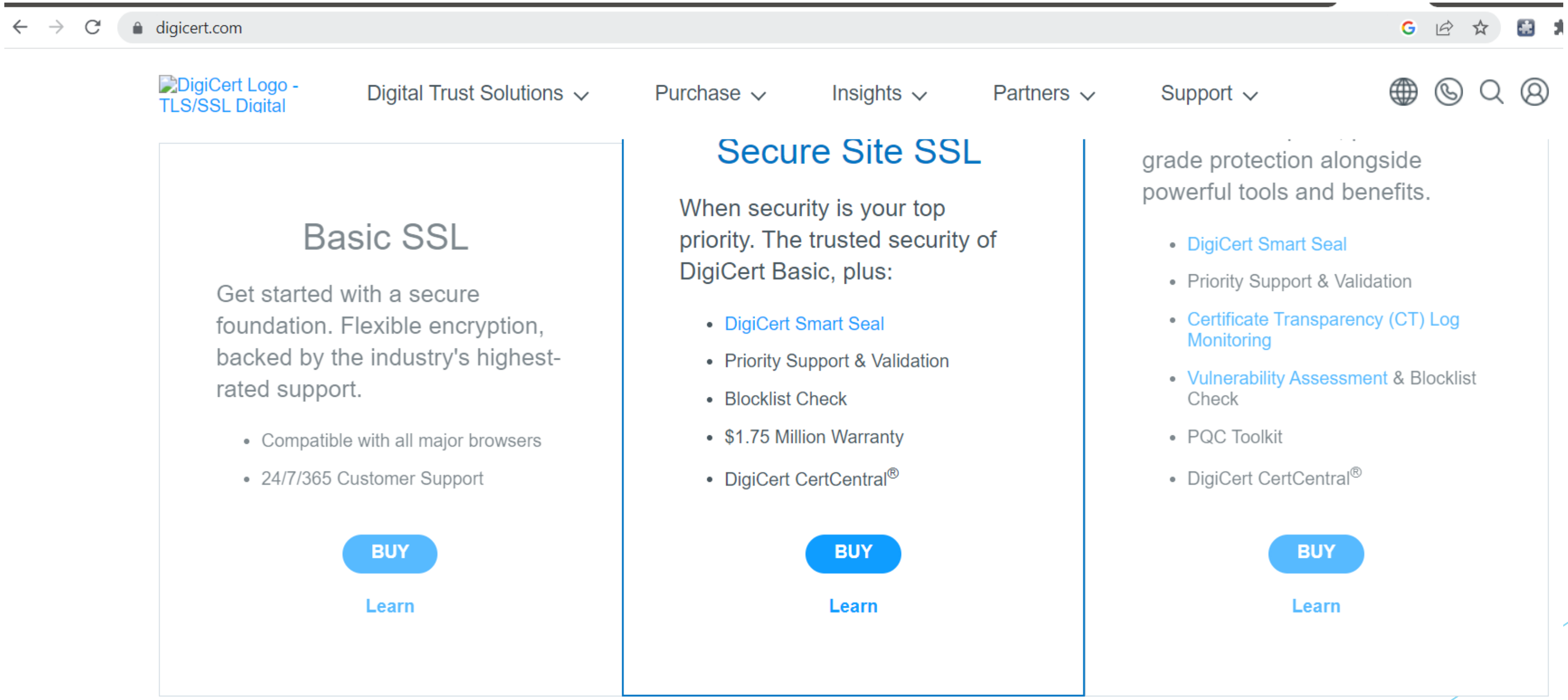
Certificate Authorities



CERTIFICATE AUTHORITIES

- The **Certificate Authority (CA)** is the entity responsible for issuing and guaranteeing certificates.
- **Private CAs** can be set up within an organization for internal communications.
- Most network operating systems, including **Windows Server**, have certificate services.
- For public or business-to-business communications, however, the CA must be trusted by each party.
- Third-party CA services include **IdenTrust**, **Digicert**, **Sectigo/Comodo**, **GoDaddy**, and **GlobalSign**.

digicert



Basic SSL

Get started with a secure foundation. Flexible encryption, backed by the industry's highest-rated support.

- Compatible with all major browsers
- 24/7/365 Customer Support

Secure Site SSL

When security is your top priority. The trusted security of DigiCert Basic, plus:

- DigiCert Smart Seal
- Priority Support & Validation
- Blocklist Check
- $1.75 Million Warranty
- DigiCert CertCentral®

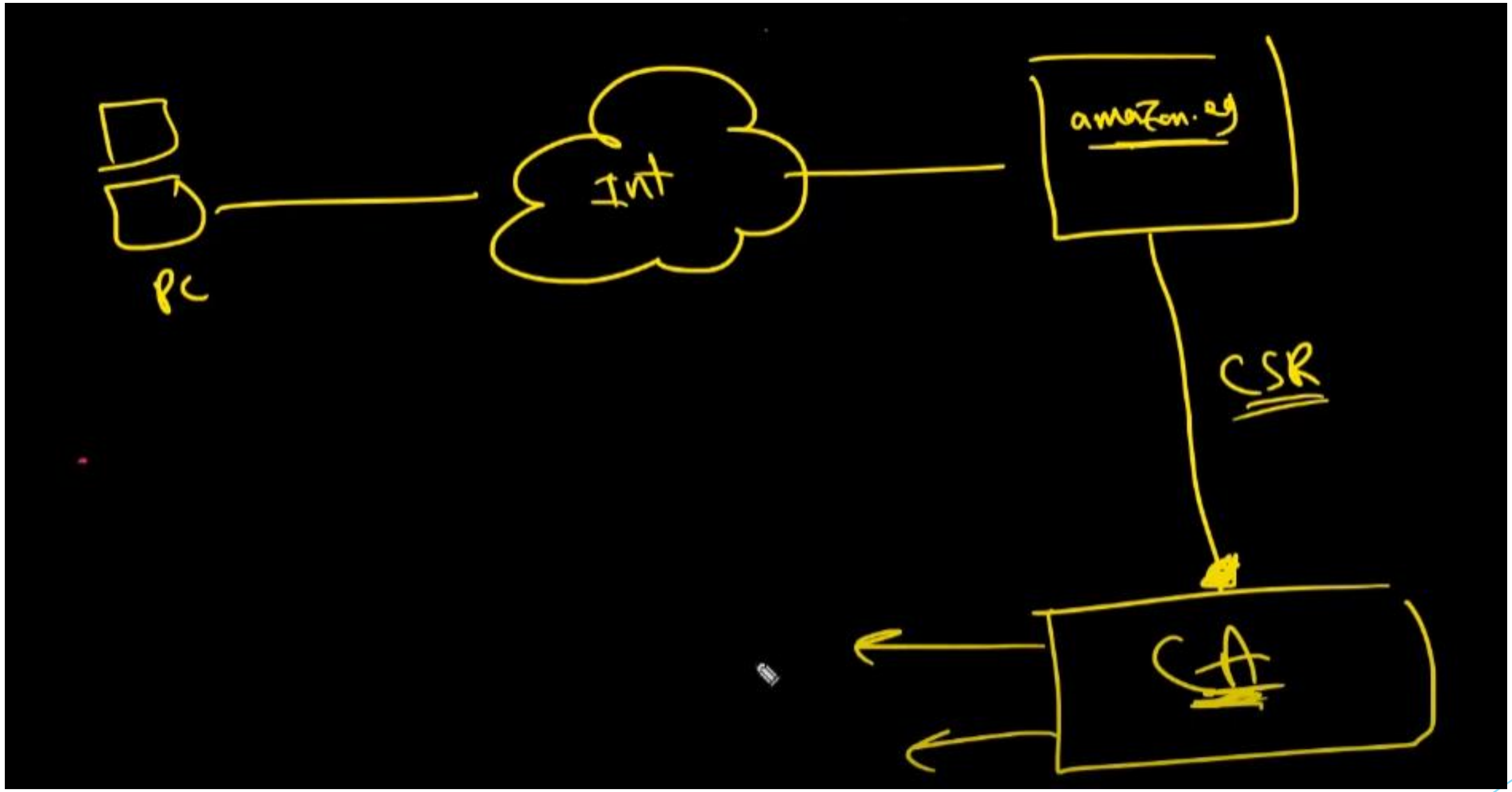
grade protection alongside powerful tools and benefits.

- DigiCert Smart Seal
- Priority Support & Validation
- Certificate Transparency (CT) Log Monitoring
- Vulnerability Assessment & Blocklist Check
- PQC Toolkit
- DigiCert CertCentral®

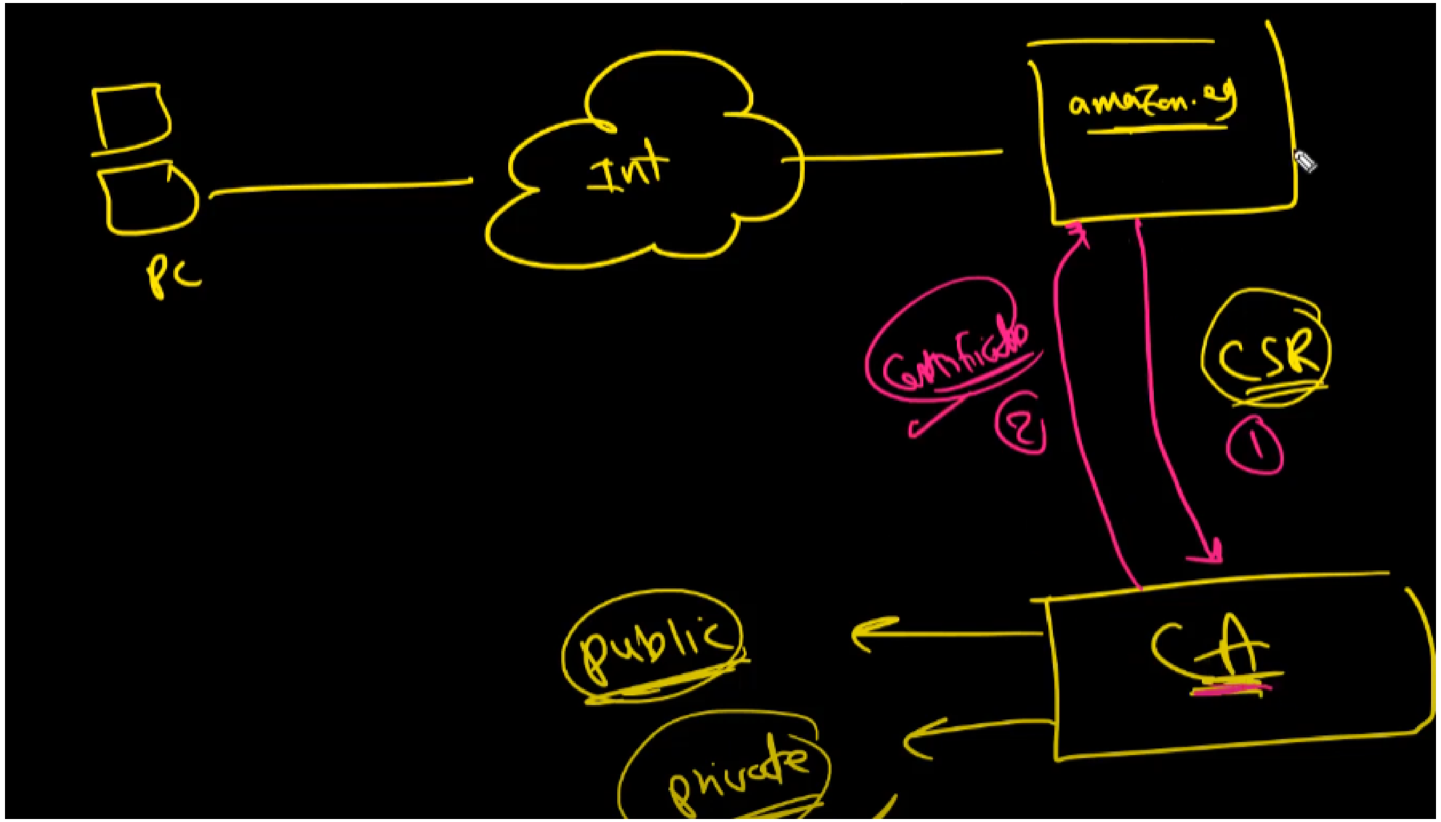
Certificate steps

1. Website sent CSR (certificate Signed Request)
2. CA replay with signed certificate
3. When Pc open website that has valid certificate, the website replay with response + certificate

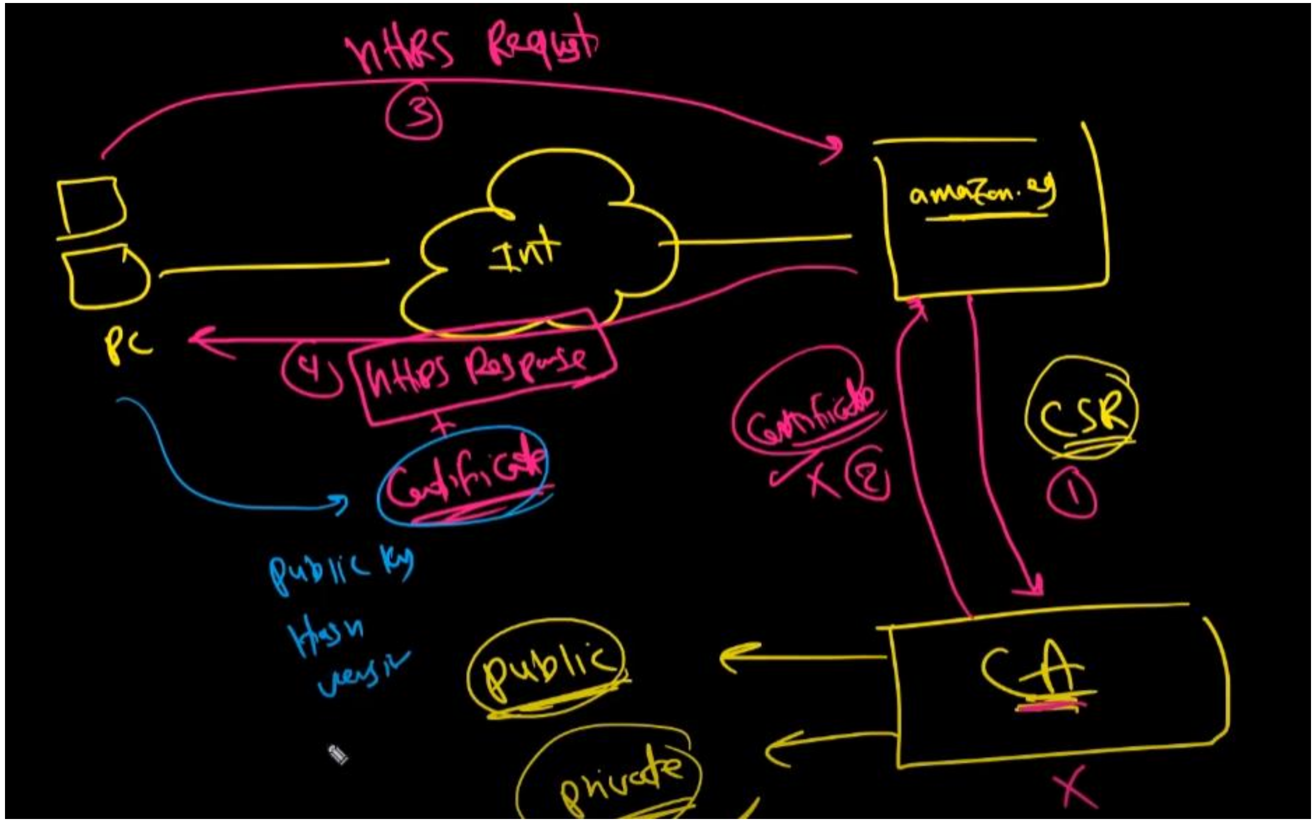
Website sent CSR (certificate signed request



CA replay with signed certificate

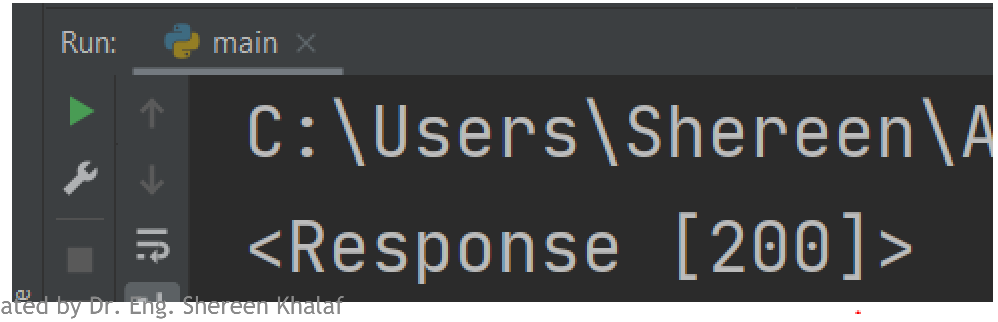


Pc send request and website replay with response + certificate



Checking if a website has a valid SSL certificate

```
import requests

response=requests.get('https://twitter.com/')

print(response)
```

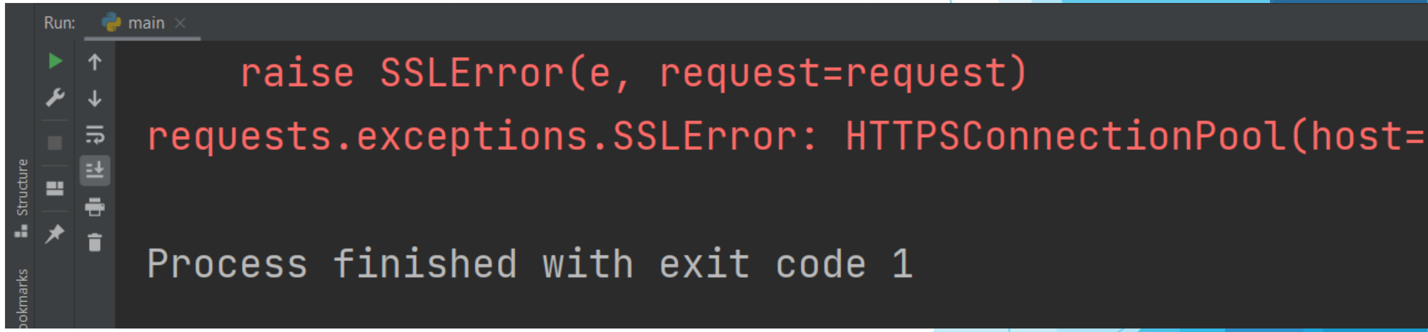


```
C:\Users\Shereen\A
<Response [200]>
```

- When we execute the code, we get a <Response [200]> (OK) message, meaning that the Twitter site is using a valid SSL certificate (as expected).

Checking if a website has a valid SSL certificate

```
import requests


response=requests.get('https://www.expired.badssl.com/')

print(response)
```



```
    raise SSLError(e, request=request)
requests.exceptions.SSLError: HTTPSConnectionPool(host=

Process finished with exit code 1
```

Create a self-signed SSL Certificate

- The process of self-generating an SSL certificate for our local Python application has three steps:
  - Create the private RSA key. openssl genrsa -out www.key 4096
  - Generate a certificate signing request (CSR) using the private key.

  openssl req -new -key **www.key** -config www.cnf -out **www.csr**

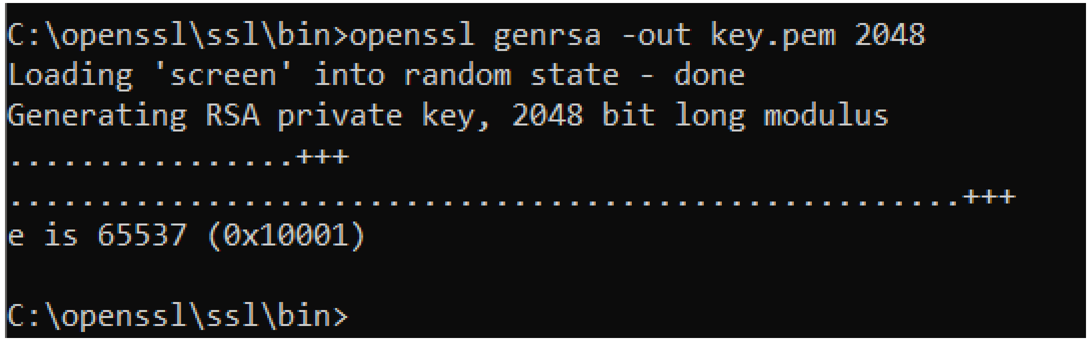
  - Sign the CSR request to create the certificate. Generate www.crt
  - cat www.crt www.key > www.pem
- **Prerequisite: Installing OpenSSL**

Creating a private and public key pair

- Once installed, run the OpenSSL command prompt. Type openssl to start the application.
- To generate a new RSA private key, type:
  - **genrsa -out {path_to_pem_file} 2048**
  - {path_to_pem_file} is the absolute path where the PEM file will be generated. Example: C:\Users\user\keyfile.pem.
- To generate a public key, type:
  - **rsa -pubout -in {path_private_pem} -out (path_public_pem)**
  - {path_private_pem} is the path to the private key PEM file. Example: C:\Users\user\privatekeyfile.pem.
  - (path_public_pem) is the path where the public key will be generated. Example: C:\Users\user\keyfile.pem.

Create the private RSA key

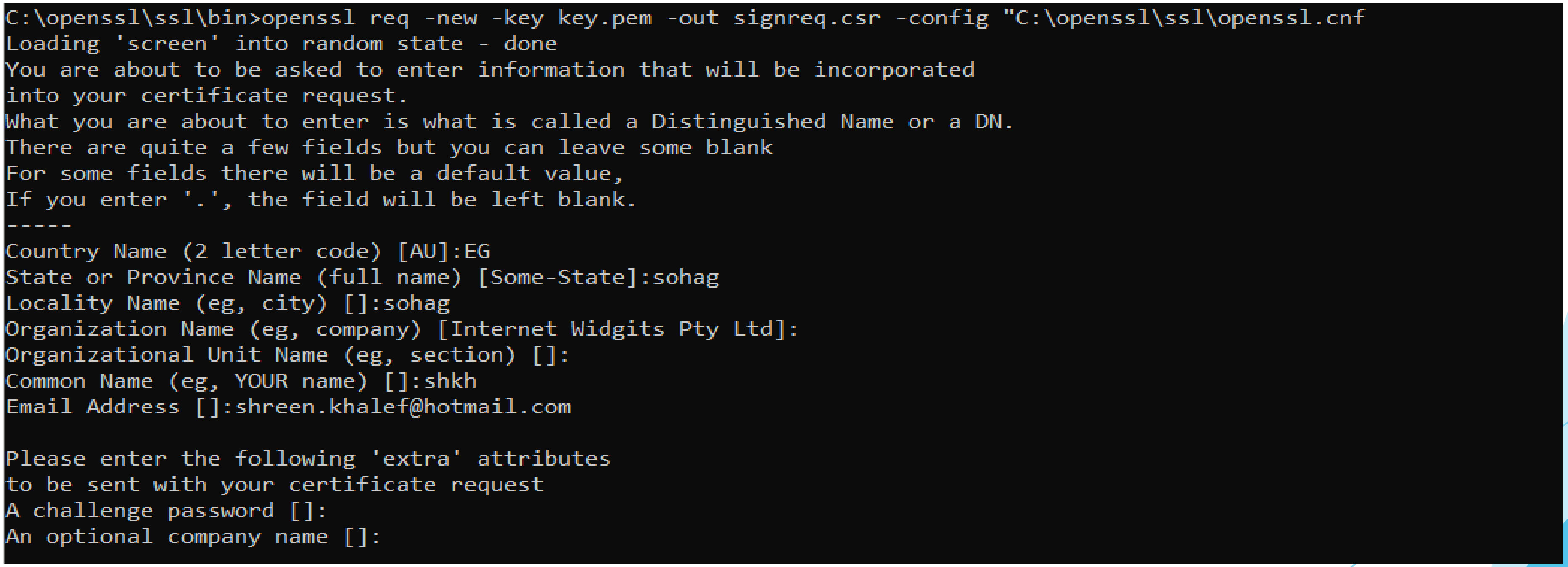
- openssl genrsa -out key.pem 2048



```
C:\openssl\ssl\bin>openssl genrsa -out key.pem 2048
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
................+++
.....................................................+++
e is 65537 (0x10001)

C:\openssl\ssl\bin>
```

Generate a certificate signing request (CSR) using the private key

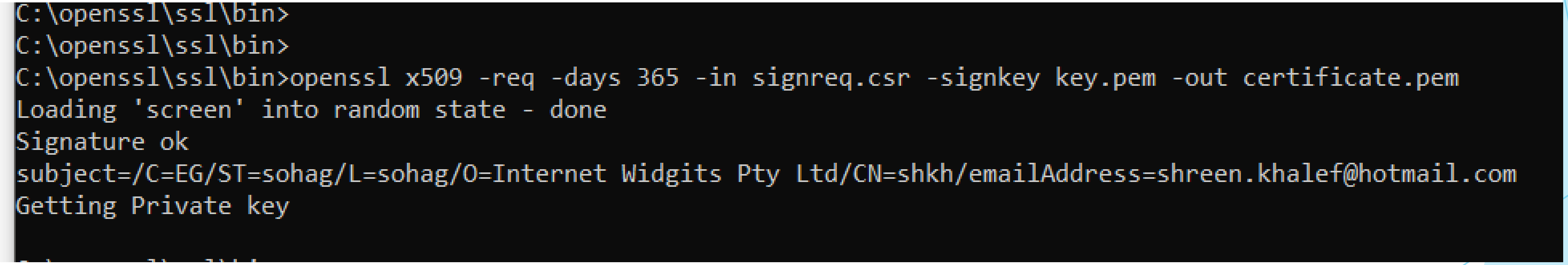
- openssl req -new -key key.pem -out signreq.csr -config "C:\openssl\ssl\openssl.cnf



```
C:\openssl\ssl\bin>openssl req -new -key key.pem -out signreq.csr -config "C:\openssl\ssl\openssl.cnf
Loading 'screen' into random state - done
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]:EG
State or Province Name (full name) [Some-State]:sohag
Locality Name (eg, city) []:sohag
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:shkh
Email Address []:shreen.khalef@hotmail.com

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

Sign the CSR request to create the certificate

- openssl x509 -req -days 365 -in signreq.csr -signkey key.pem -out certificate.pem
- To view file use
  - openssl x509 -text -noout -in certificate.pem



```
C:\openssl\ssl\bin>
C:\openssl\ssl\bin>
C:\openssl\ssl\bin>openssl x509 -req -days 365 -in signreq.csr -signkey key.pem -out certificate.pem
Loading 'screen' into random state - done
Signature ok
subject=/C=EG/ST=sohag/L=sohag/O=Internet Widgits Pty Ltd/CN=shkh/emailAddress=shreen.khalef@hotmail.com
Getting Private key
```

The Python SSL library

- We use the Python SSL library to provide TLS encryption in socket-based communication between Python clients and servers.
- It uses cryptography and message digests to secure data and detect alteration attempts in the network. Digital certificates provide authentication.

ssl — TLS/SSL wrapper for socket objects

- This module provides access to Transport Layer Security (often known as "Secure Sockets Layer") encryption and peer authentication facilities for network sockets, both client-side and server-side. This module uses the OpenSSL library.
- import ssl

Socket creation

- ssl.**create_default_context**(*purpose=Purpose.SERVER_AUTH*, *cafile=None*, *capath=None*, *cadata=None*)
- Return a new SSLContext object with default settings for the given *purpose*.

Load_cert_chain() Method Of SSLContext Class In Python

- Method Signature:
  - load_cert_chain(certfile, keyfile=None, password=None)
- Parameters:
  - certfile - Path of the X.509 certificate file in PEM(Privacy Enhanced Email) format.
  - keyfile - The private key of the certificate certfile='localhost.pem‘
  - password - Password for the private key if the private key is encrypted. The value to this parameter can be a string, bytes or byte array or a function returning string, bytes or byte array.
- Return value:
  - None

Wrap_socket() Method Of SSLContext Class In Python

- **Method Signature:**
  - **wrap_socket(sock, server_side=False, do_handshake_on_connect=True, server_hostname=None, session=None);**
- **Parameters:**
  - sock – The socket instance from which the SSLSocket needs to be created.
  - server_side – Denotes whether the SSLSocket being created is a server socket or a client socket.
  - server_hostname – Server hostname to which the client is connecting to. This parameter needs to be supplied a value only if the server_side = False.
- **Return Value:**
  - An object of type ssl.SSLSocket

Securing a Socket with TLS for Both Client and Server

- First, create a TLS *context* object that knows all of your preferences regarding certificate validation and choice of cipher.
- Second, use the context's wrap_socket() method to let the OpenSSL library take control of your TCP connection, exchange the necessary greetings with the other end, and set up an encrypted channel.
- Finally, perform all further communication with the ssl_sock
that has been returned to you so that the TLS layer always has the chance to encrypt your data before it actually hits the wire

Client

1. Create Context

   context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH, cafile=cafile)

2. use the context’s wrap_socket()

   ssl_sock = context.wrap_socket(sock, server_hostname=host)

3. Read data using ssl_sock

   data = ssl_sock.recv(1024)

Client

```
import  socket, ssl

def client(host, port, cafile=None):
    context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH, cafile=cafile)
    raw_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    raw_sock.connect((host, port))
    print('Connected to host {!r} and port {}'.format(host, port))
    ssl_sock = context.wrap_socket(raw_sock, server_hostname=host)
    while True:
        data = ssl_sock.recv(1024)
        if not data:
          break
        print(repr(data))

host='localhost'
port=12345
cafile='ca.crt'
client(host,port,cafile)
```

Server

1. **Create Context**

```
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
   context.load_cert_chain(certfile)
```

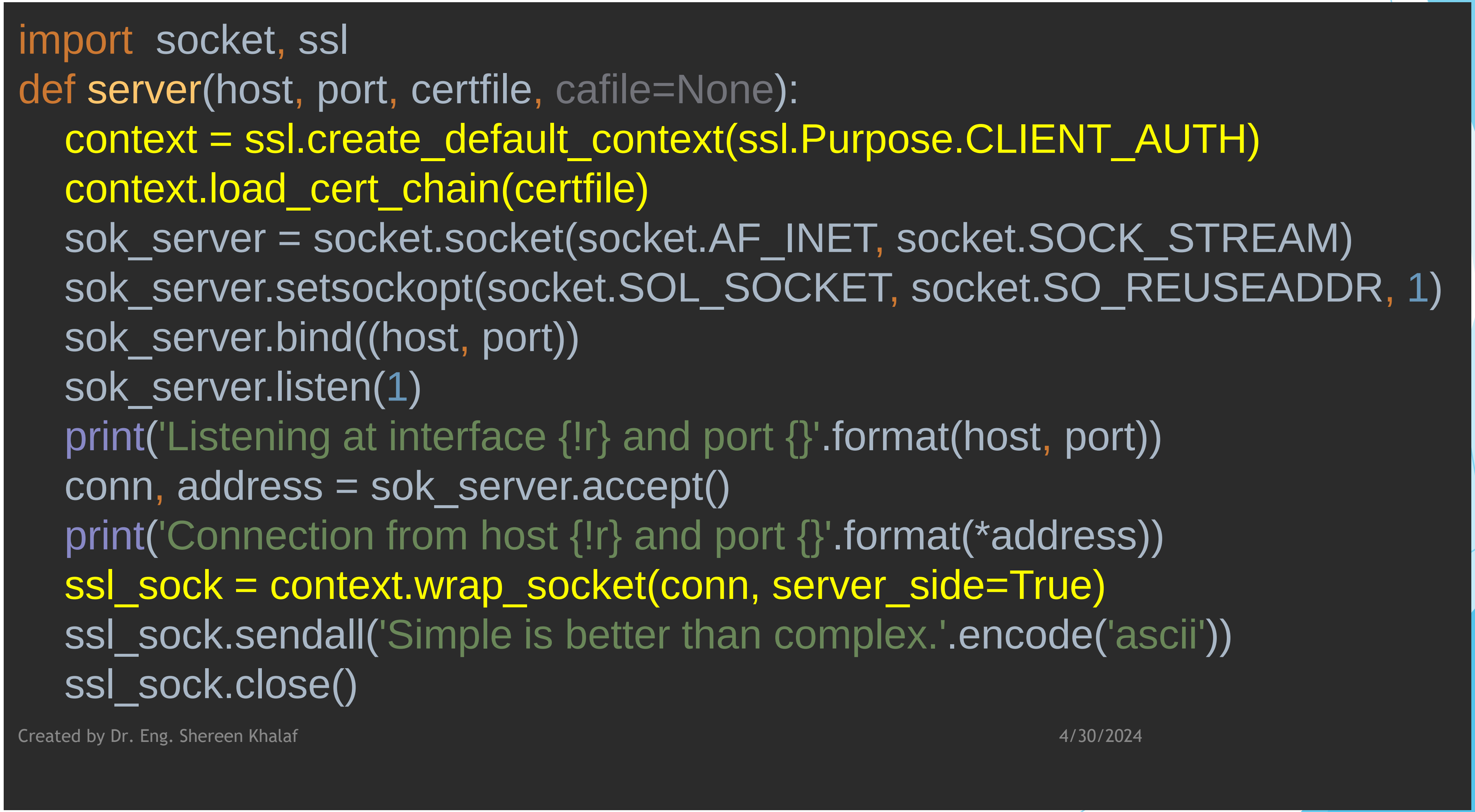
2. **use the context’s wrap_socket()**

```
conn, address = sok_server.accept()
ssl_sock = context.wrap_socket(conn, server_side=True)
```

3. **Read data using ssl_sock**

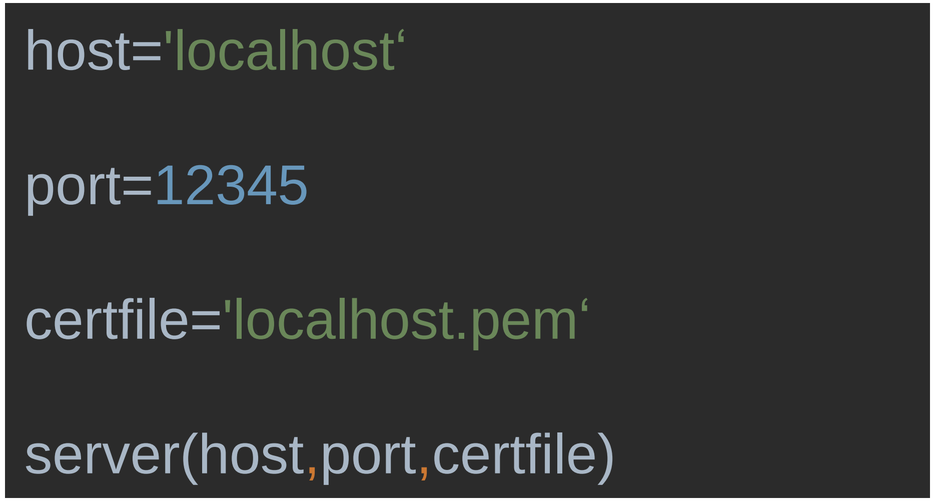
```
      ssl_sock.sendall('Simple is better than complex.'.encode('ascii'))
   ssl_sock.close()
```

Server



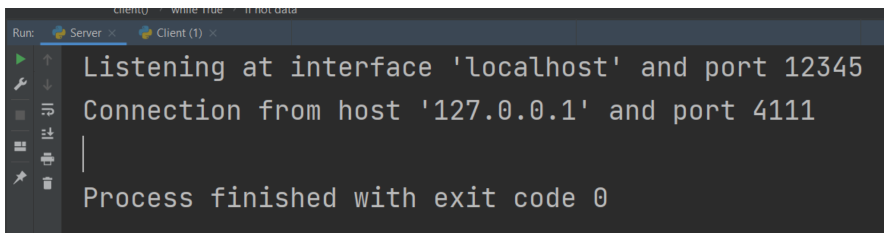
```
import  socket, ssl
def server(host, port, certfile, cafile=None):
    context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain(certfile)
    sok_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sok_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sok_server.bind((host, port))
    sok_server.listen(1)
    print('Listening at interface {!r} and port {}'.format(host, port))
    conn, address = sok_server.accept()
    print('Connection from host {!r} and port {}'.format(*address))
    ssl_sock = context.wrap_socket(conn, server_side=True)
    ssl_sock.sendall('Simple is better than complex.'.encode('ascii'))
    ssl_sock.close()
```

Run Server

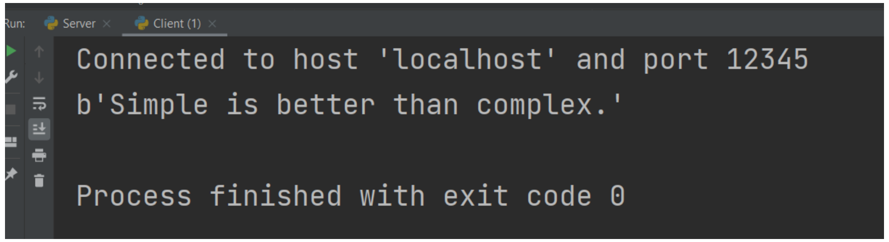


```
host='localhost‘

port=12345

certfile='localhost.pem‘

server(host,port,certfile)
```

Results



```
Listening at interface 'localhost' and port 12345
Connection from host '127.0.0.1' and port 4111

Process finished with exit code 0
```



```
Connected to host 'localhost' and port 12345
b'Simple is better than complex.'

Process finished with exit code 0
```

Thank you