Tutorial 3

Exercise 1

- To extend the exercise 2 from tutorial 2 to be able to use digital signatures, we need to add one function for creating hash of the message and then sign it using the private key of the sender, alice.

```
def sign_message(private_key, message):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message.encode())
    digest = message_hash.finalize()

    signature = private_key.sign(
        digest,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature
```

The function takes in private_key of alice and the plaintext message. It uses hashes to create a SHA256 hash of the plaintest and stores it in variables digest. Then signature is made using private_key and signed with the digest, some padding, this time with PSS – probabilistic signature scheme.

- Similarly a function for message verification is made.

```python
def verify_signature(public_key, message, signature):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message.encode())
    digest = message_hash.finalize()
    try:
        public_key.verify(
            signature,
            digest,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return True
    except InvalidSignature:
        return False
```

This function takes in the public_key of the sender(alice), the plaintext message and the signature. Verification works by computing the sha256 hash of the received plaintext message and then using public_key to verify if the signature received is same as the one now created in variable digest. If both are same True is returned if not then False.

Exercise 2

- The commands to set up a rootCA and self sign it are

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl genrsa -out rootCA.key 2048
(failed reverse-i-search)`openssl req': git commit -am "whoop used for lo^C"
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ ^C
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 3650 -out rootCA.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]:FI
State or Province Name (full name) [Some-State]:Tampere
Locality Name (eg, city) []:Tampere
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$
```

Created rootCA.key with rsa and 2048 bits then using that key to sign a crt for rootCA.crt with validity of 3650 days.

Now create key for user, alice and certificate signing request to be sent to rootCA

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl genrsa -out alice.key 2048
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl req -new -key alice.key -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]:FI
State or Province Name (full name) [Some-State]:Tampere
Locality Name (eg, city) []:Tampere
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$
```

Password is kept empty.

Sign the CSR with rootCA for 365 days validitiy

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl x509 -req -in alice.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out alice.cr
t -days 365 -sha256
Certificate request self-signature ok
subject=C = FI, ST = Tampere, L = Tampere, O = Internet Widgits Pty Ltd
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$
```

Verification

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$ openssl verify -CAfile rootCA.crt alice.crt
alice.crt: OK
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise2 (main)$
```

I found that if i put the same values for rootCA.crt and alice.crt in CountryName, Locality etc it assumed that alice.crt is self signed because Issuer and Subject field had the same entries.

Exercise 3

- First create new server key and certificate using openssl

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise3 (main)$ openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365
-out server.crt
^C
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise3 (main)$ ls
server.crt  server.key  server.py
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial3/exercise3 (main)$
```

For domain name i used localhost or 127.0.0.1 also works.

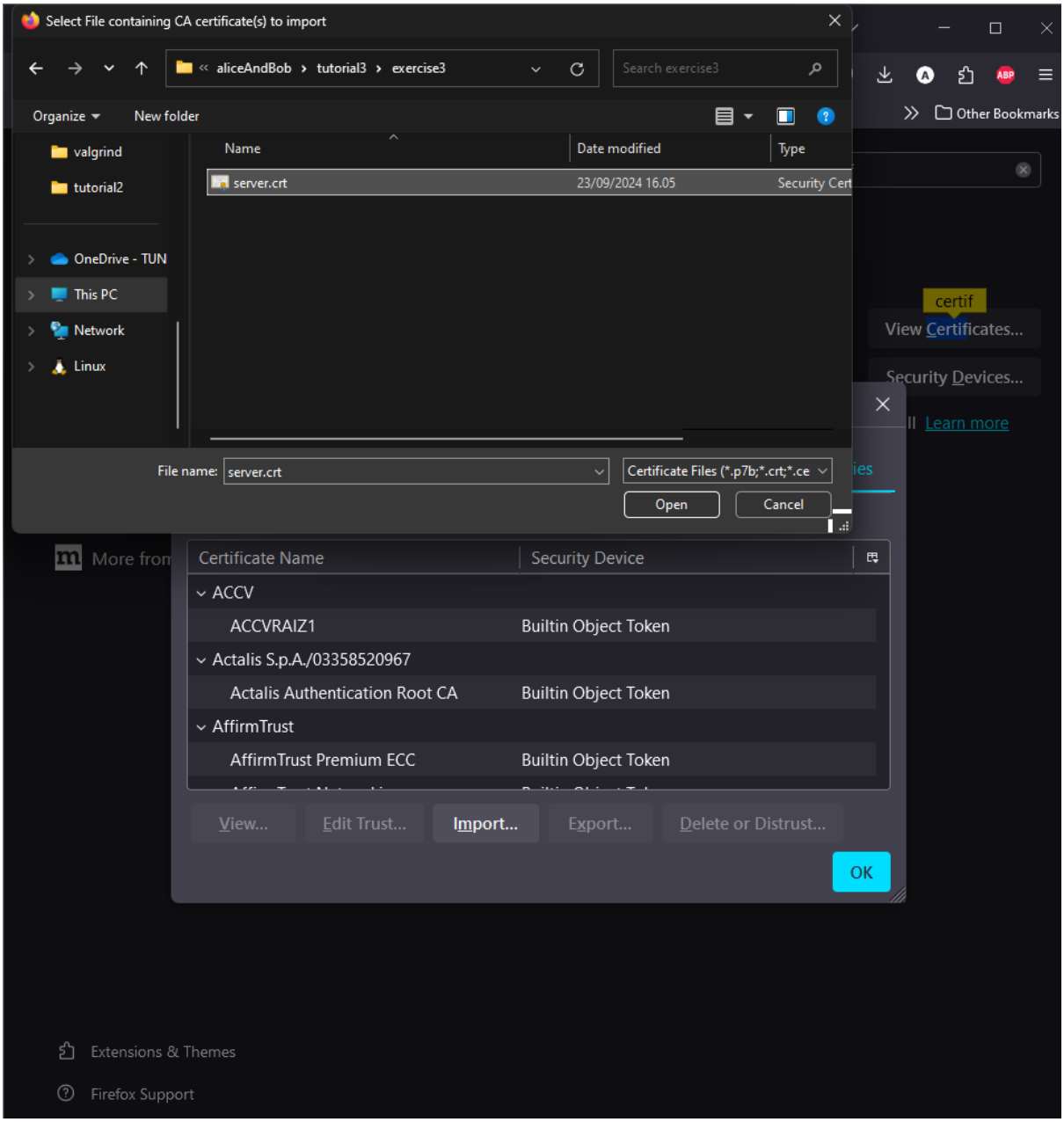
- The python server looks like this

tutorial3 > exercise3 > server.py > ...

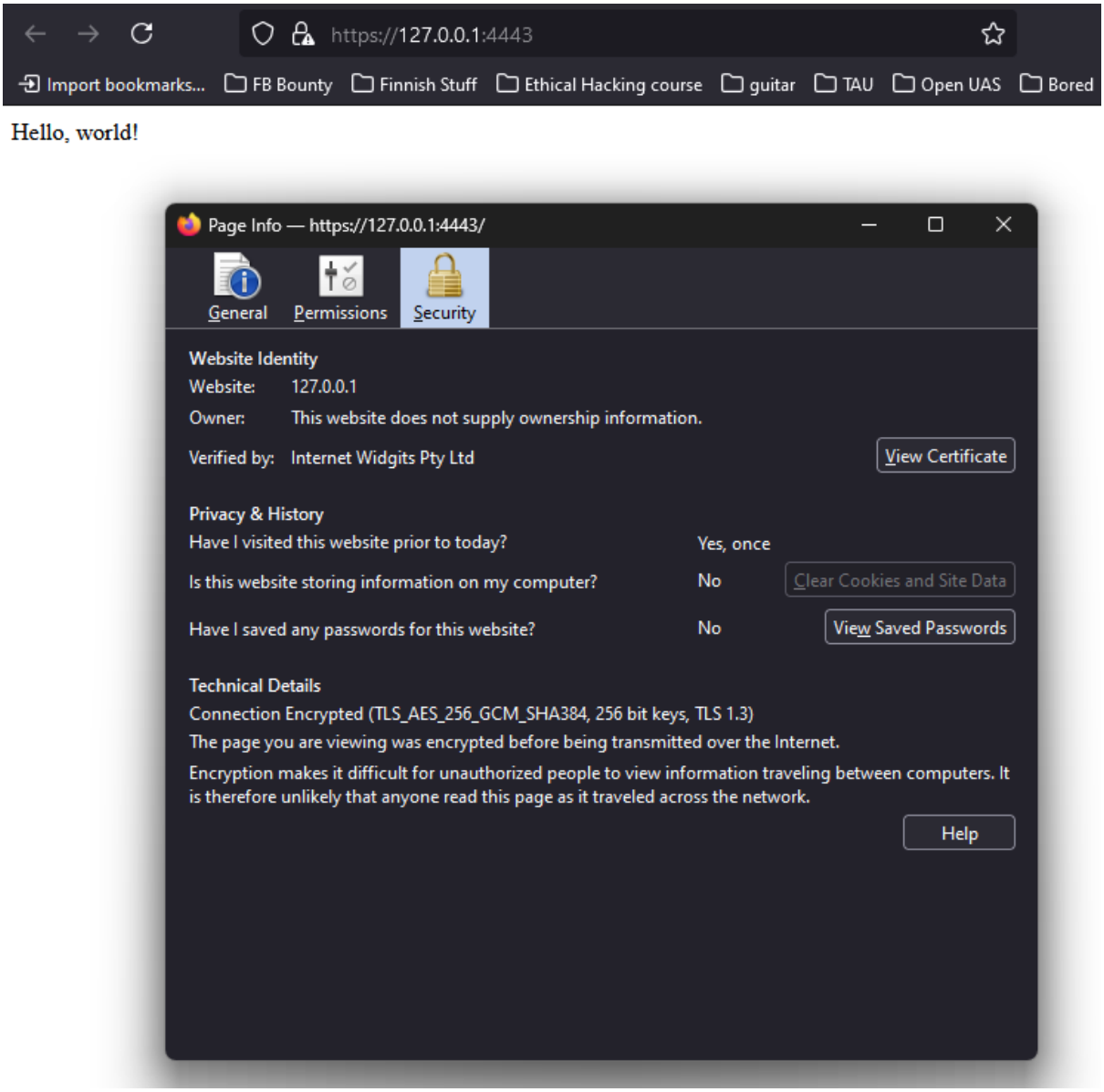
```python
from http.server import HTTPServer, BaseHTTPRequestHandler
import ssl

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(b'Hello, world!')

httpd = HTTPServer(('localhost', 4443), SimpleHTTPRequestHandler)

httpd.socket = ssl.wrap_socket(httpd.socket,
        keyfile="server.key",
        certfile='server.crt', server_side=True)

print("Serving on https://localhost:4443")
httpd.serve_forever()

```

It just displays Hello, world! When accessed.

- To configure my computer to associate domain name to localhost I didn't have to do anything since i kept domain name as localhost. But if it was anything else we could edit the /etc/hosts file via WSL2.
- To allow firefox to trust this certificate we need to add it in settings. That is done by going to Settings -> Search Certificate -> View Certificates -> Authorities -> Import -> server.crt

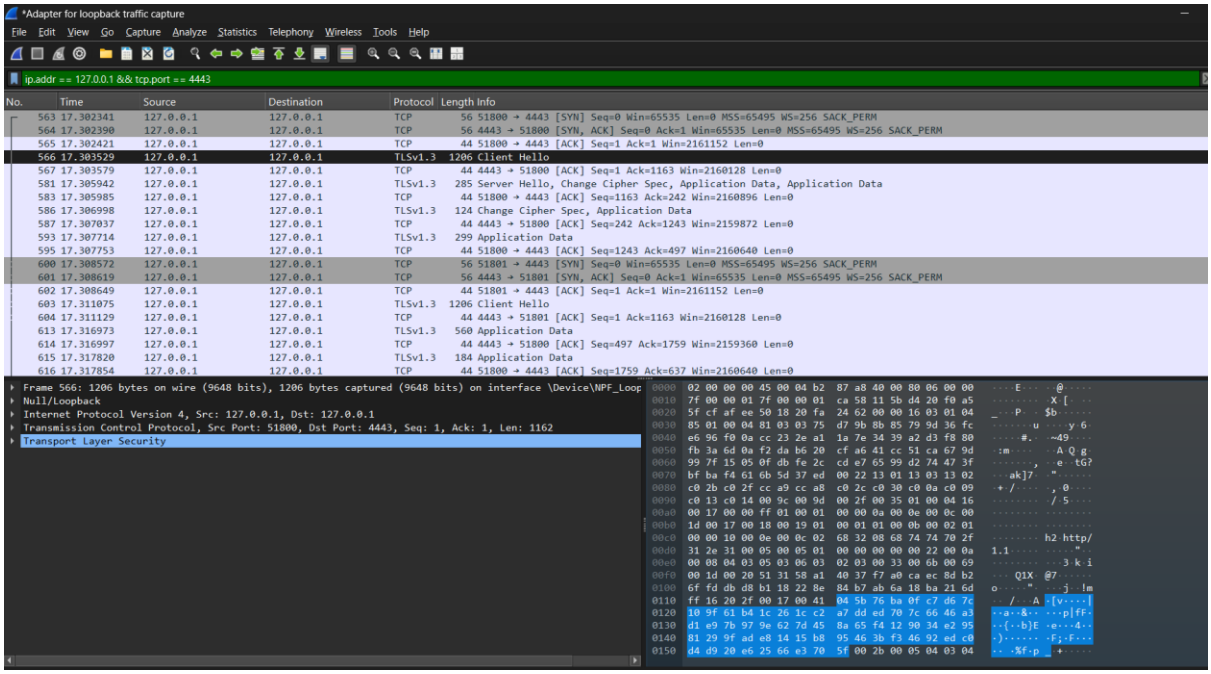

Select File containing CA certificate(s) to import
aliceAndBob › tutorial3 › exercise3
Search exercise3
Organize
New folder
valgrind
tutorial2
OneDrive - TUN
This PC
Network
Linux
Name
Date modified
Type
server.crt
23/09/2024 16.05
Security Cert
File name:
server.crt
Certificate Files (*.p7b;*.crt;*.ce
Open
Cancel
Other Bookmarks
certif
View Certificates...
Security Devices...
Learn more
Certificate Name
Security Device
ACCV
ACCVRAIZ1
Builtin Object Token
Actalis S.p.A./03358520967
Actalis Authentication Root CA
Builtin Object Token
AffirmTrust
AffirmTrust Premium ECC
Builtin Object Token
View...
Edit Trust...
Import...
Export...
Delete or Distrust...
OK
Extensions & Themes
Firefox Support

- Then go to https://127.0.0.1:4443



Firefox still says the connection is not secure because the website doesn't provide any information about the ownership but https works just fine.

- Output in wireshark



In wireshark we can see the TLS 3way handshake between the client and the server. And because of TLS we also cannot see the websites content ie ‘Hello, world!’ in cleartext which would’ve been visible if it was HTTP.

Exercise 4

I tried to do exercise 4 and created self signed certificate but getting that certificate into outlook was/is a challenge. I tried numerous things like adding that certificate via mmc console to my device and settings in Outlook trust centre but outlook would always complain that there is some issue with the certificate and didn’t let me send the email.

Getting the trust centre settings in outlook was a whole other ordeal, the new outlook doesn’t allow users to set those permissions and i had to force outlook to use older version to be able to find those settings.

I think this exercise needs some updating or atleast I have no ideas how to do it.

- In order to send signed email you need – personal certificate (private key)
- In order to send encrypted messages you need recipients public key. To decrypt an email you need your private key and the email that was encrypted using your public key.