**Applied Cryptography**

**Individual Project: PKI-TLS**

**Student: Nicola Maiorana**

**Email:** [**nam10102@nyu.edu**](mailto:nam10102@nyu.edu)

**ID: nam10102**

**CS-GY 6903 2025 Spring**

## Objective

This is the final individual project for this class and is designed to help students understand the practical implementations of Public Key Infrastructure (PKI) and Transport Layer Security (TLS), including mutual TLS (mTLS). I will configure a server and a client to communicate securely using TLS and mTLS and discuss the implications. I will set up a server and client to securely communicate using TLS and mTLS and explain the practical use of certificates.

## Setup

For this project I will be using WSL2 on a Surface Pro 6 computer running Windows 11. In a Ubuntu 24.04 VM I will define 2 docker instances, one for the client and one for the server. To facilitate working with docker containers, I will create several alias definitions in the main VM. These will be added to the .bashrc file in my home directory:

A screen shot of a computer code

AI-generated content may be incorrect.

The following files will be used to define each docker container to be used in this lab:

#### docker-compose.yml

A screenshot of a computer

AI-generated content may be incorrect.

This configuration creates 2 hosts, one called csgy6903-client, and one called csgy6903-server. Each is assigned a shared folder and an IP address. Each host configuration is also provided a docker-file definition to create and configure each host.

#### docker-client (192.168.1.100)

A screen shot of a computer

AI-generated content may be incorrect.

The csgy6903-client will be constructed from the latest ubuntu image and have the following packages installed:

* Curl
* Vim
* Net-tools
* Iproute2
* Traceroute
* Iputils-ping
* tcpdump

#### docker-server (192.168.1.101)

A computer screen with white text

AI-generated content may be incorrect.

The csgy6903-server will be constructed from the latest ubuntu image and have the following packages installed:

* nginx
* EasyRSA
* curl
* vim
* net-tools
* iproute2
* traceroute
* iputils-ping
* wget
* unzip

### Building and starting the containers:

A computer screen with text and images

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

\*Note: *The container id will be different for every new build*

### Wireshark

I attempted to view the network traffic using Wireshark installed on the host Windows system but could not get it to see the network interface for the docker containers. I will generate pcap files using tcpdump so they can be viewed in Wireshark:



#### Testing the basic client/server communication:

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

You can see the HTTP network traffic between the client (192.168.1.100) and the server (192.168.1.101) in the “no\_security.pcap” file.

### Setup complete

With the docker containers built and configured, we can now proceed with the lab working to implement PKI and TLS between the client and the server.

## TLS Configuration using Self-Signed and ECC

The first part of configuring the server to use TLS was to generate self-signed certificates to be applied to the nginx server. Using EasyRSA, we configured the vars file to build ECC certificates. This was not capable of creating a self-signed certificate w/out creating a CA. So for this we decided to use openssl:

#### Openssl Self-signed certificate:

Generating a self-signed certificate using the prime256v1 ECC.

A screenshot of a computer program

AI-generated content may be incorrect.

#### Certificate/Signed Certificate:



#### Configure NGINX:

Here we configure nginx to use the self-signed certificate and key. We also tell it which ECC to use. After updating the configuration, we restarted the nginx server.

A computer screen shot of text

AI-generated content may be incorrect.

#### Capture TCP Traffic from Client to Server:

On the top screen we start the tcpdump capture while on the bottom screen we use curl to send a request to the server. Note the -k option used to allow curl to accept self-signed certificates. As expected, the HTML which includes the “Welcome to the CS-GY 6903 PKI/TLS Project” was returned.

A screenshot of a computer program

AI-generated content may be incorrect.

#### Wireshark:

A screenshot of a computer

AI-generated content may be incorrect.

Above you can clearly see the client request a TLS handshake and the server responding.

## mTLS Configuration using CA and ECC

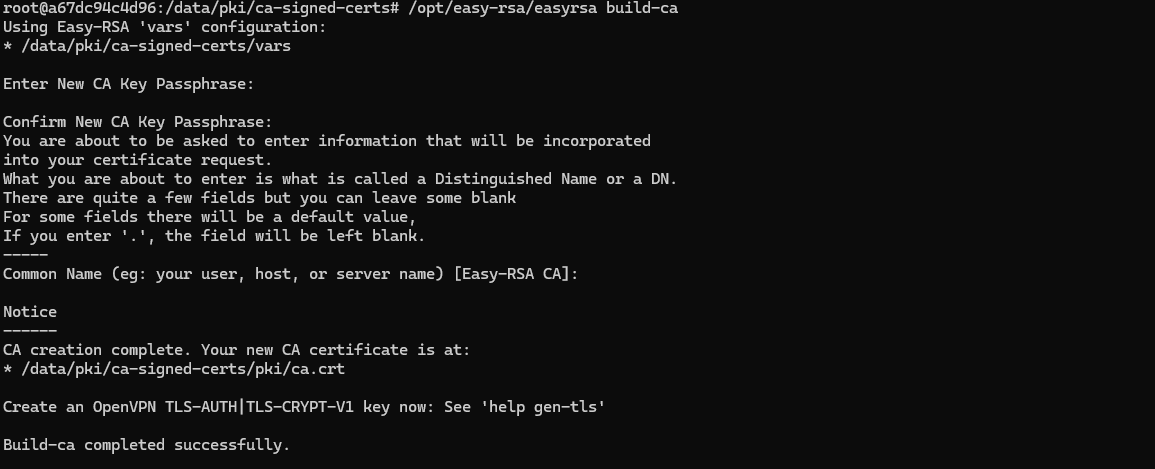
In this part of the lab, we will use EasyRSA to construct our server and client certificates. The first step in this process is to create a CA to sign our certificates:

#### Configure EasyRSA to use ECC:



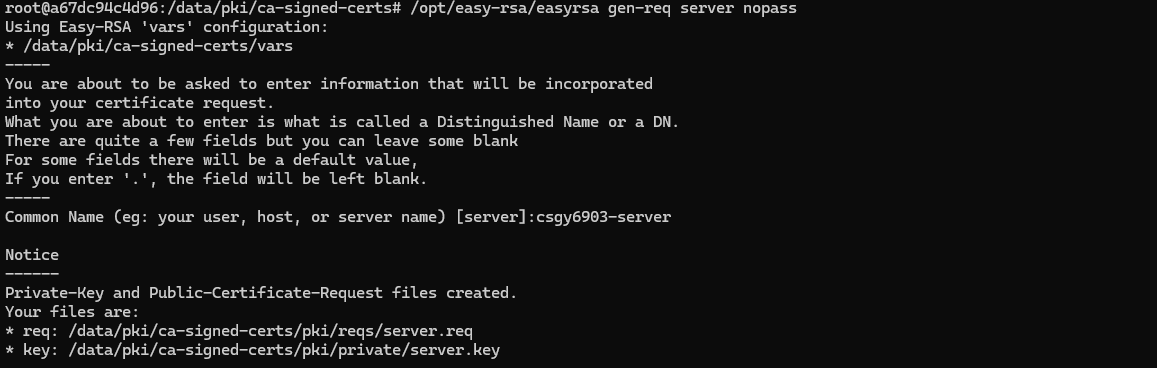
#### Create CA:

For this CA we used Easy-RSA CA as the certificate authority. We also used “tlsproject” for our passphrase.

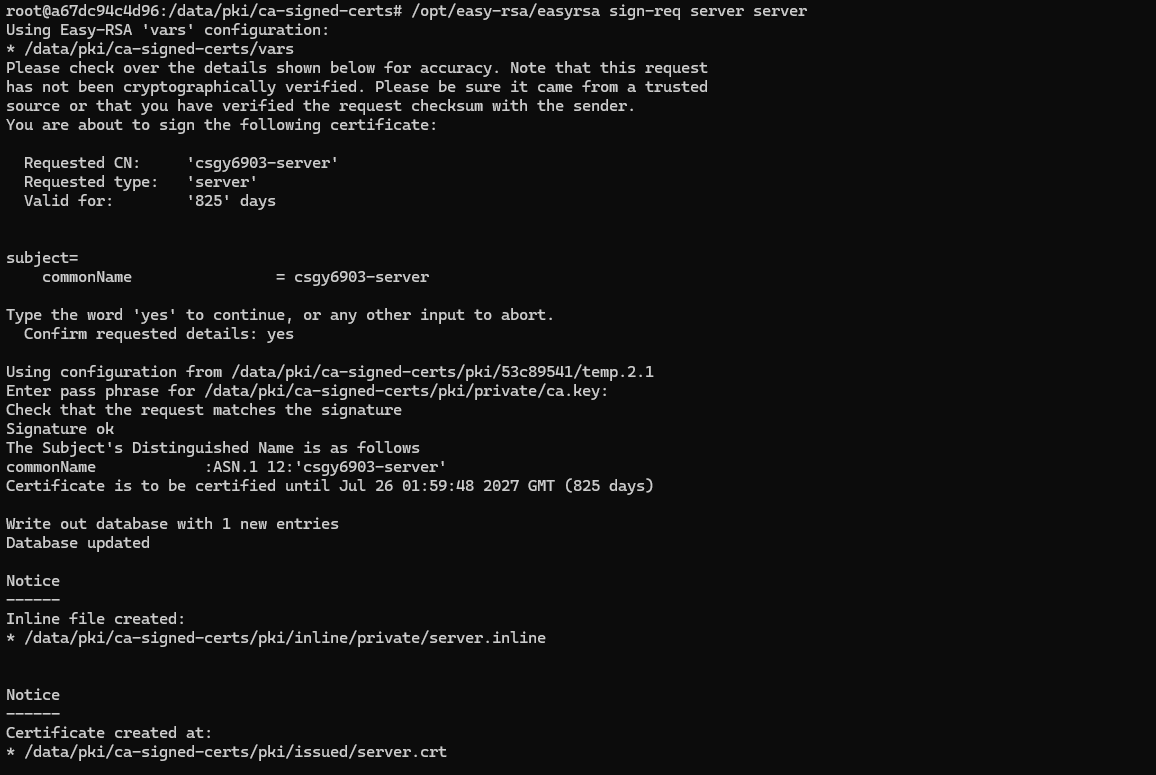


The next step will be to build the server and client certificates. First, we will start with the server certificate.

#### Create Server certificate:

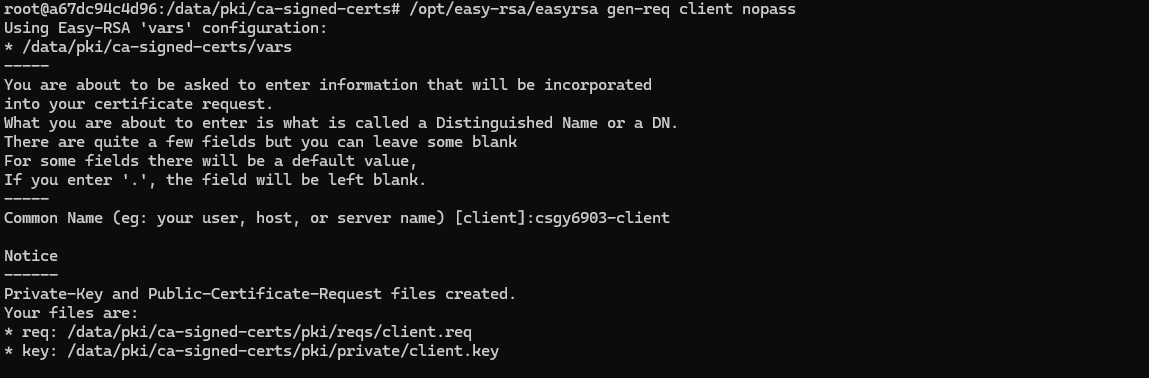


#### Sign the certificate using the CA:

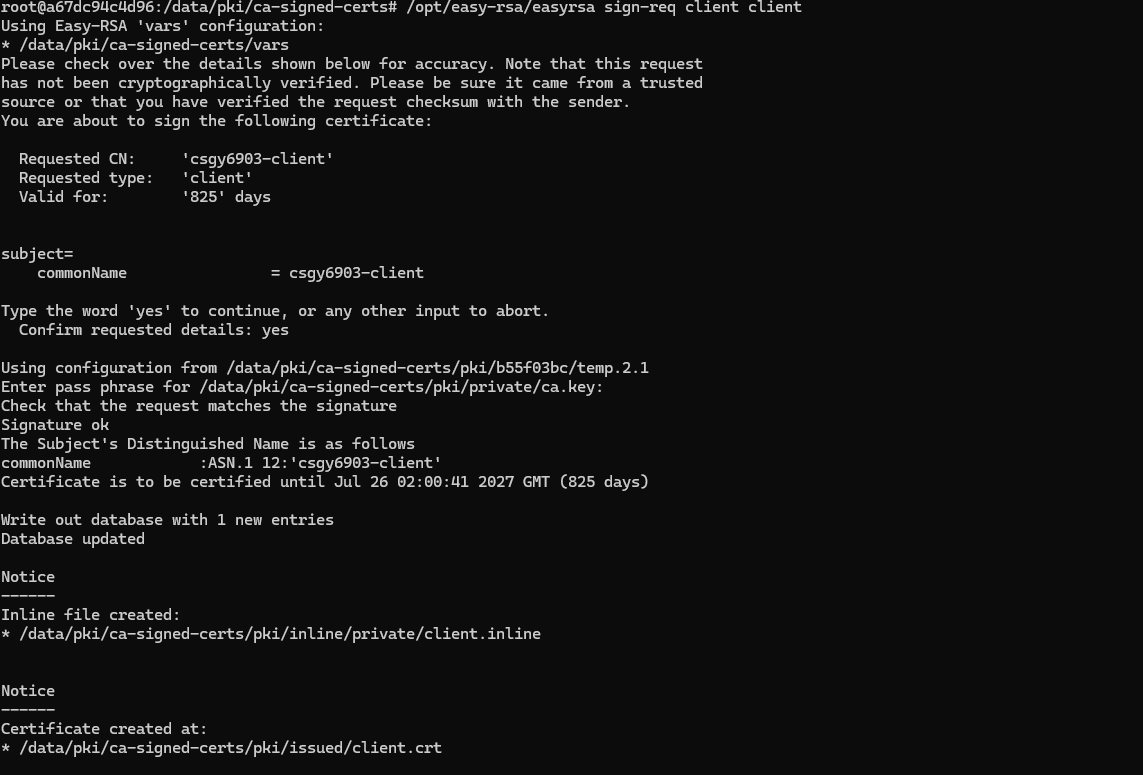


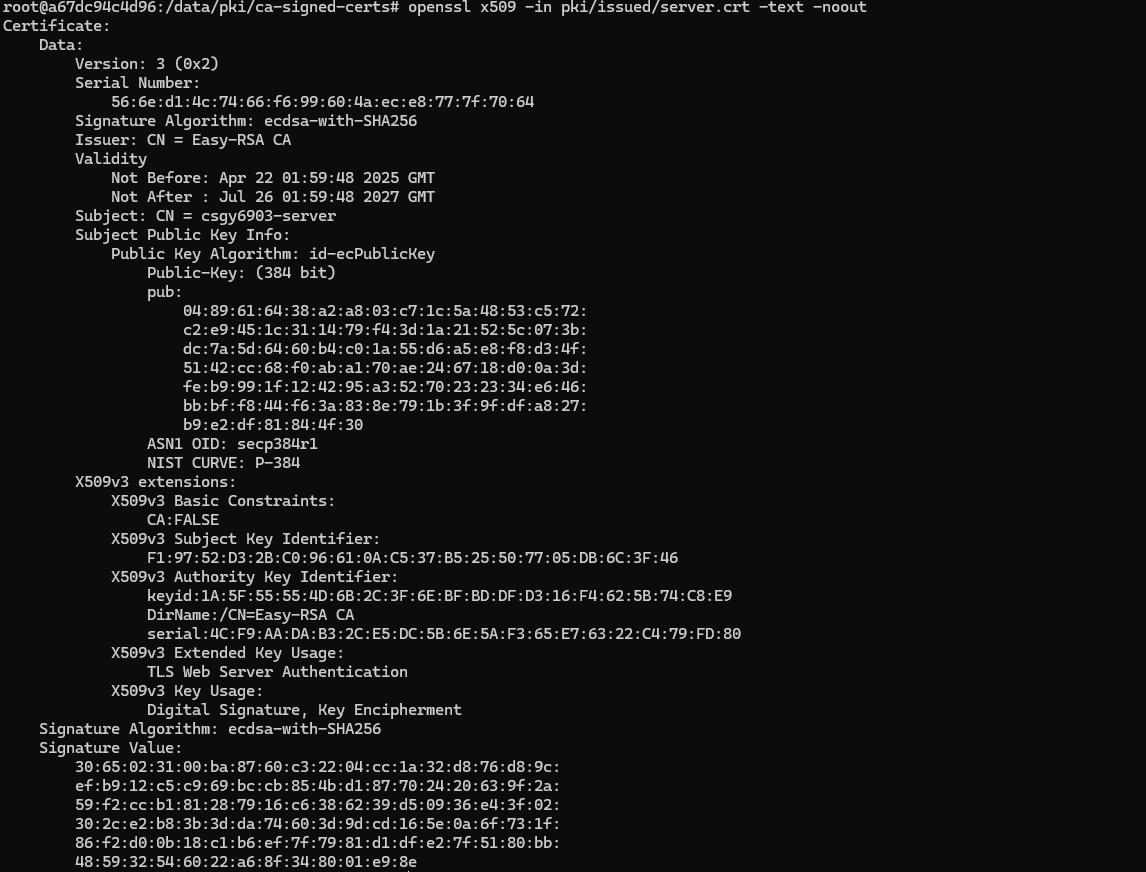
Next, we will create the client certificate.

#### Create Client certificate:

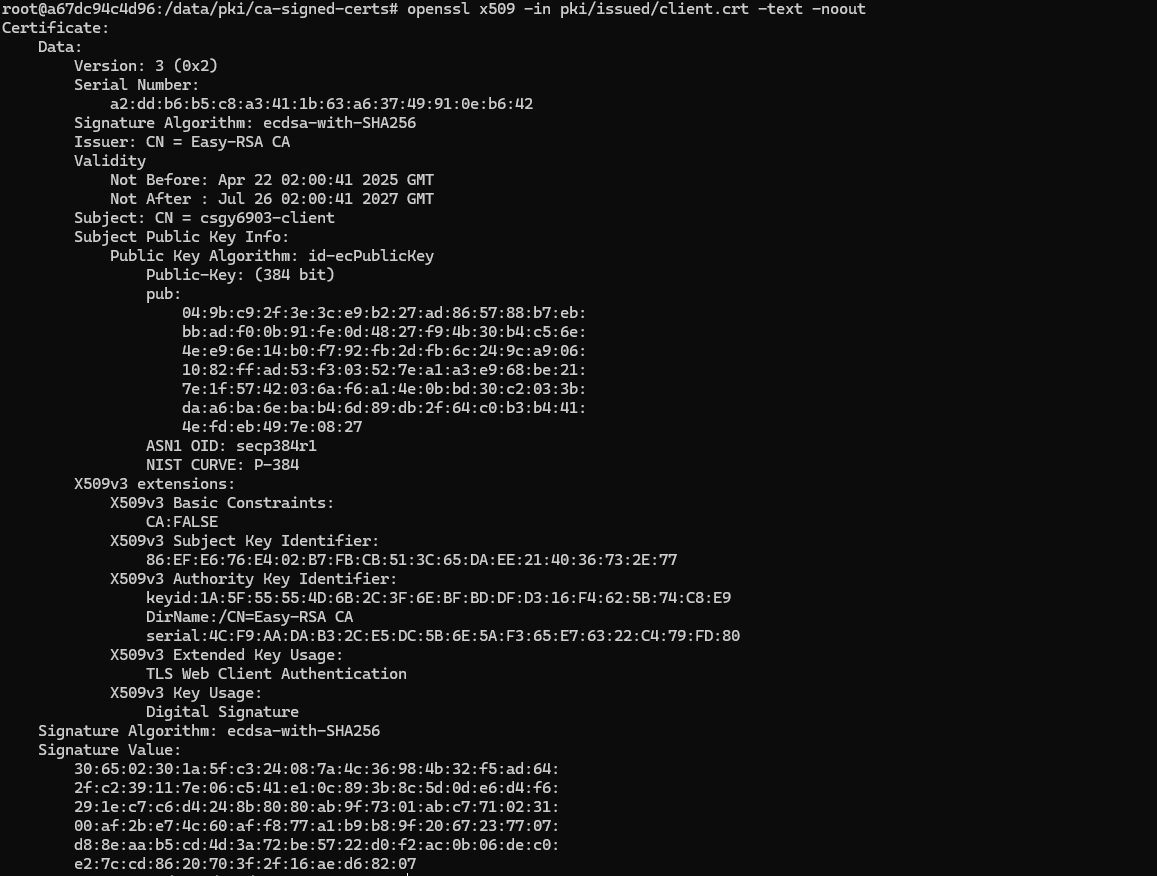


#### Sign the certificate using the CA:



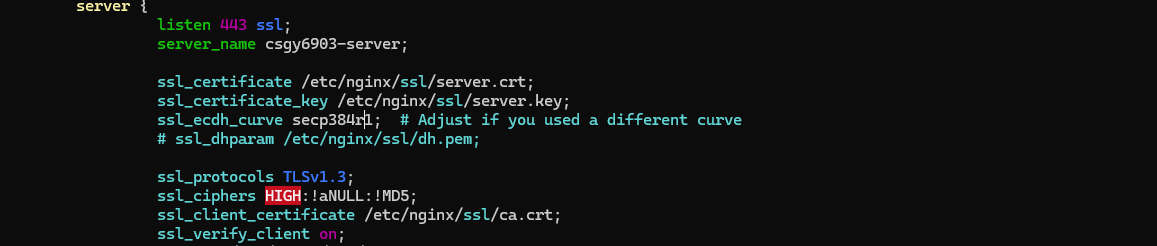
Now that we have both signed certificates, lets verify them. First the server:

And the client:



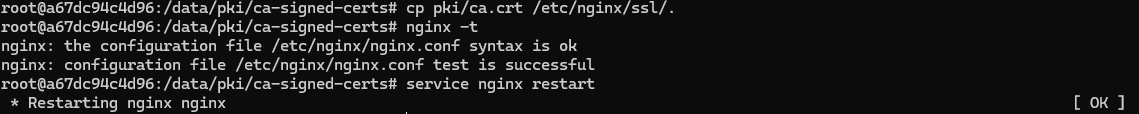
The next step is to configure NGINX to require a client certificate.

#### Configure NGINX to require a client certificate



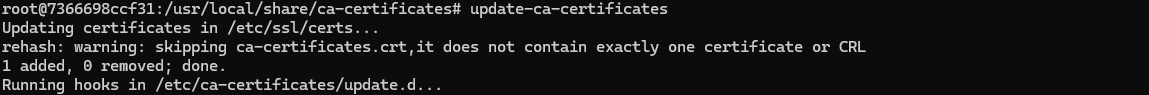
We will need to copy the CA certificate to the NGINX ssl directory and restart the server:

#### Configure and restart NGINX:



Now we evaluate the client to see if it can request a web page from the server. For this attempt we will not configure the client to use a certificate, and the request should be denied by the server. But first we will install the ca certificate on the client host so it trusts the server.

#### Install CA certificate on client:



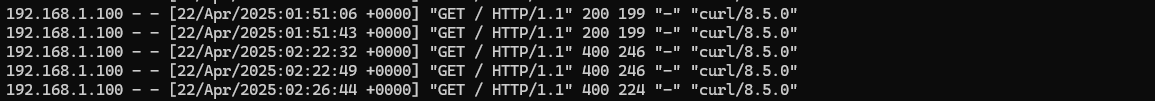
#### Evaluate the client with no client certificate:

A screenshot of a computer program

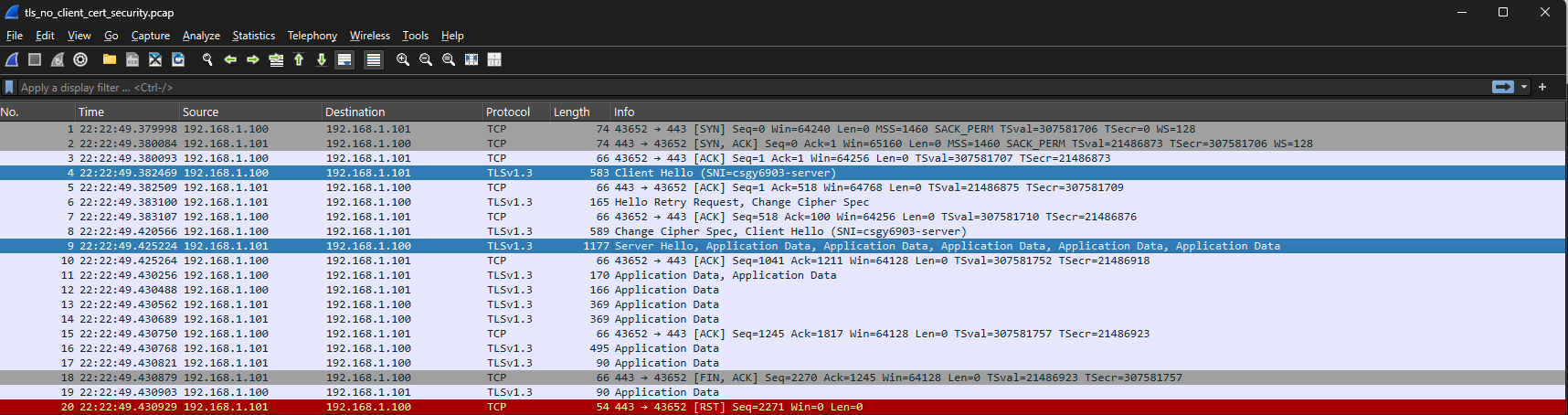
AI-generated content may be incorrect.

As you can see the server rejected the client’s request. Let us examine the server logs and the captured pcap file:

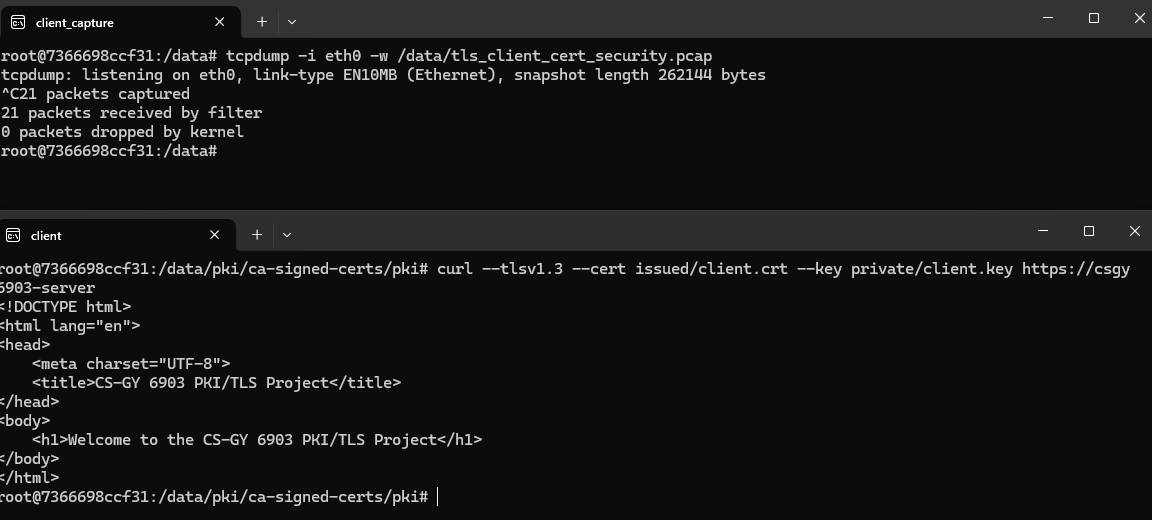
#### Access log for NGINX:



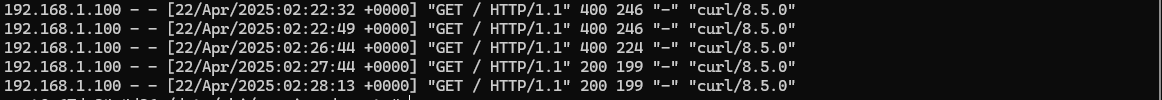
You can see the successful previous attempts with a response of 200, followed by the 400 bad requests after the client certificate required by the server.



Next, we will pass the client certificate in the request and see if we get a 200 web response back:

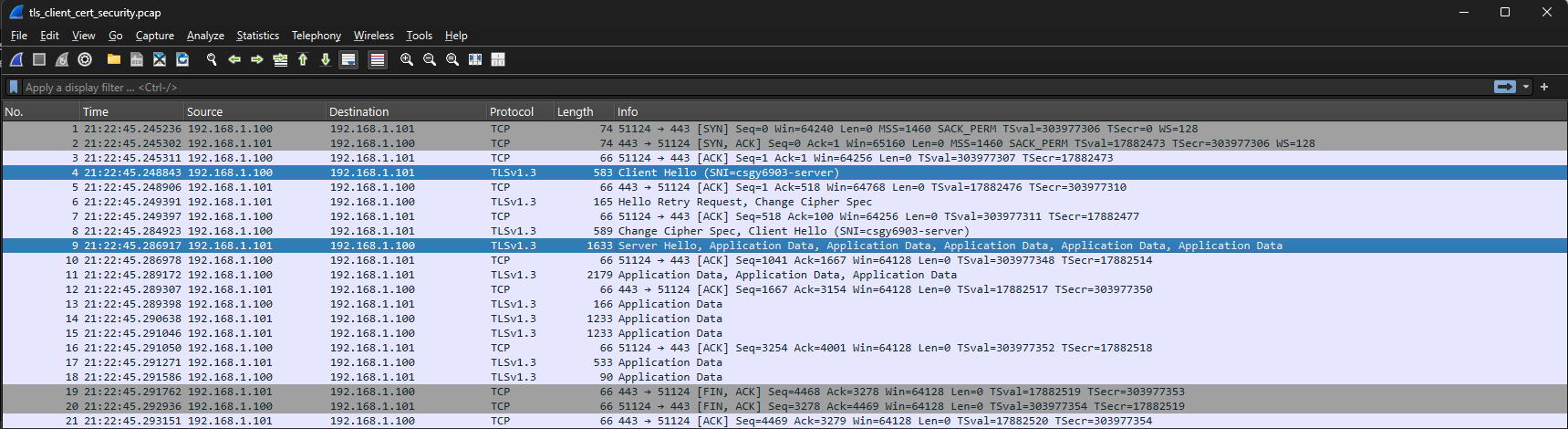


We can see that the server sent the web page back to the client. Here is the access log from NGINX:



Here we see the 200 responses back after the previous 400 responses.

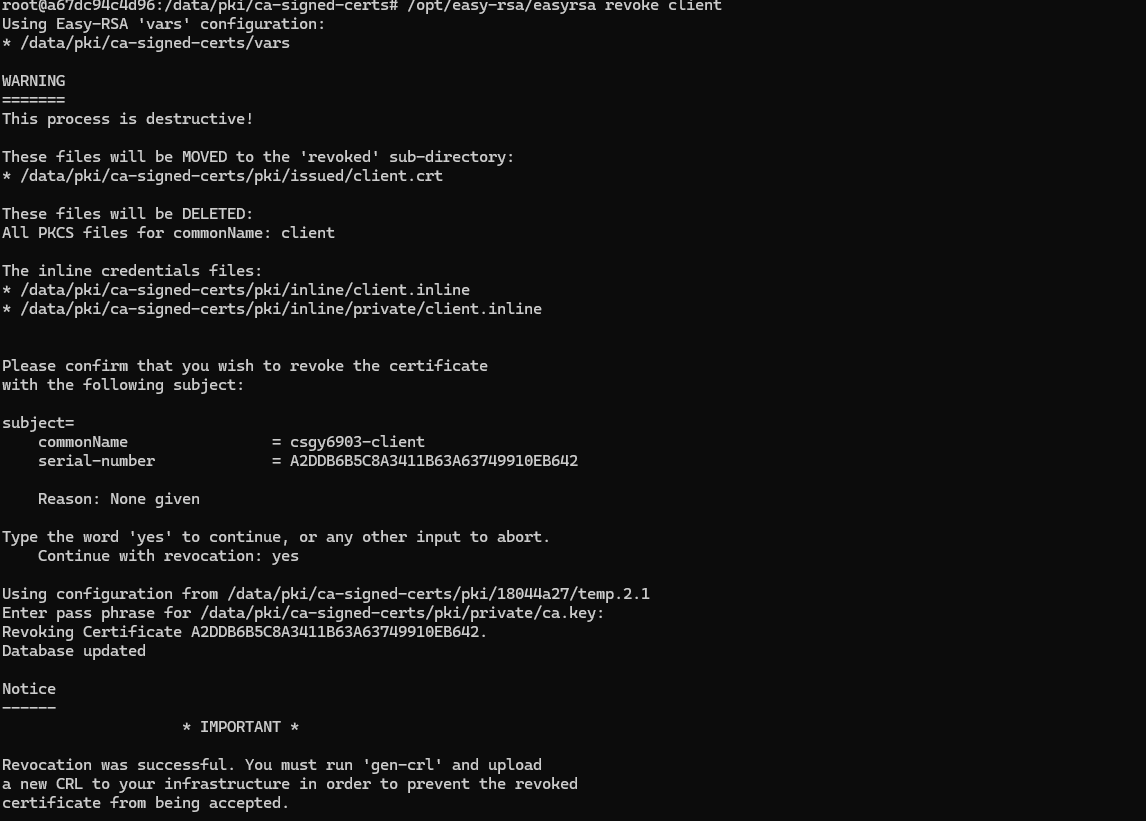
Here is the pcap file in Wireshark:



## Revoking Existing Client Certificates

In this part of the lab we will revoke the existing client certificate from the server. For this we need to configure NGINX. Using EasyRSA, we will revoke the current client certificate and generate a CRL:

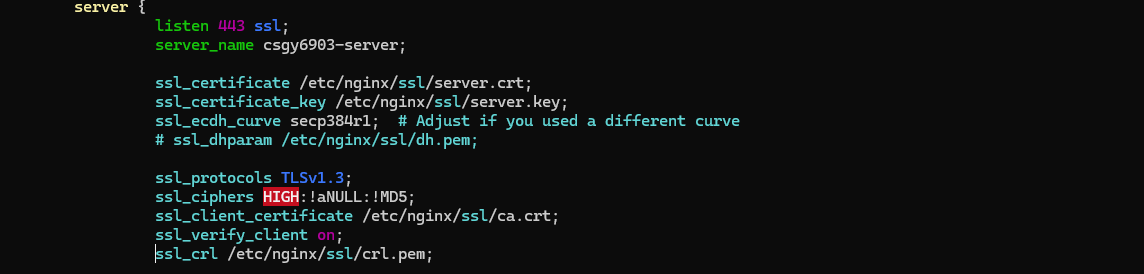
#### Revoke Client certificate and generate CRL:



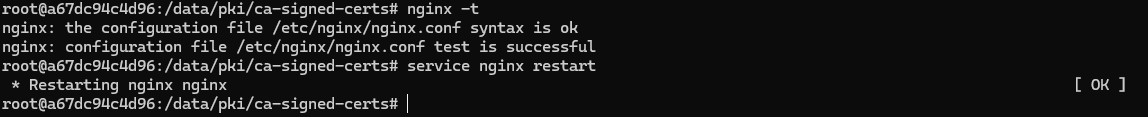
#### Generate CRL:



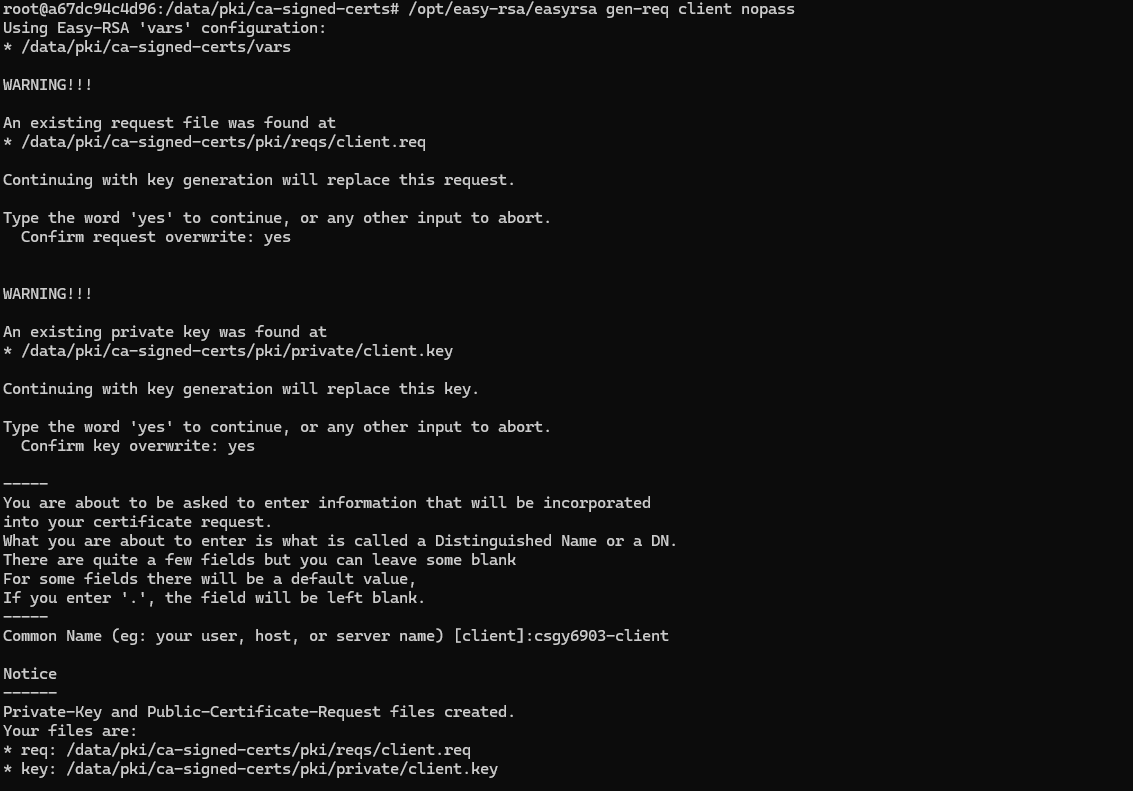
#### Now configure NGINX to use the CRL:



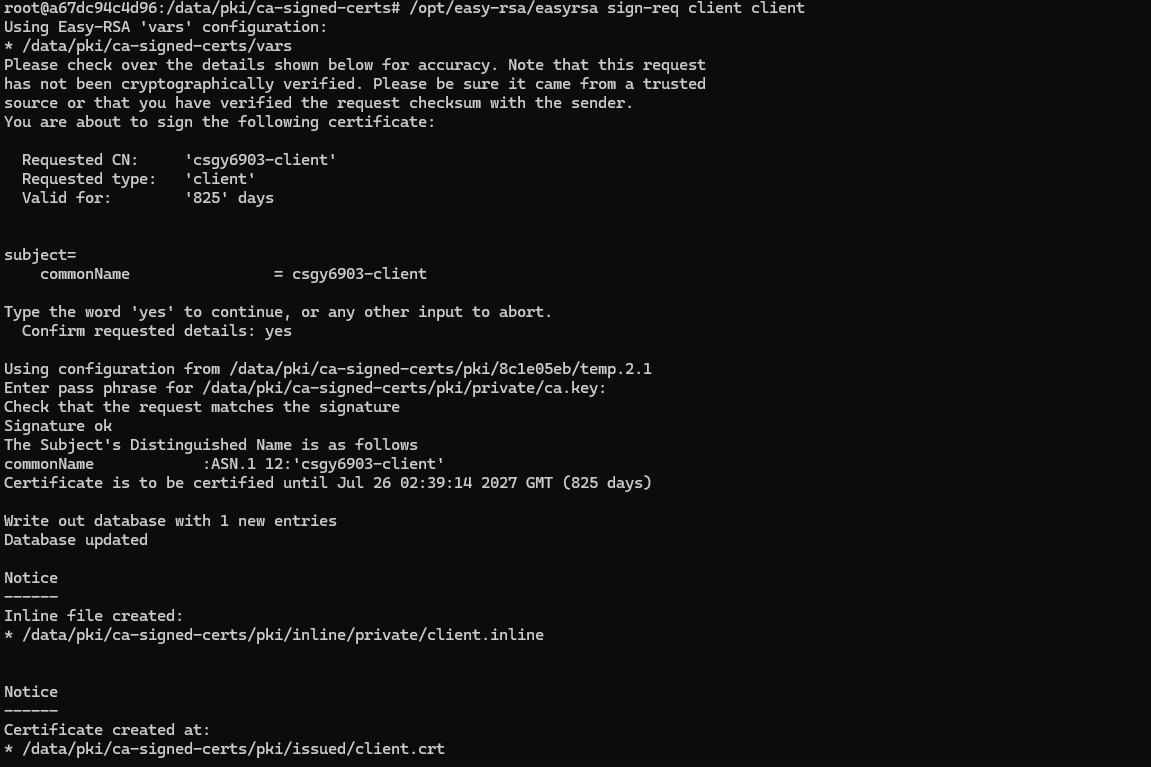
#### Verify the NGINX configuration and restart the server:



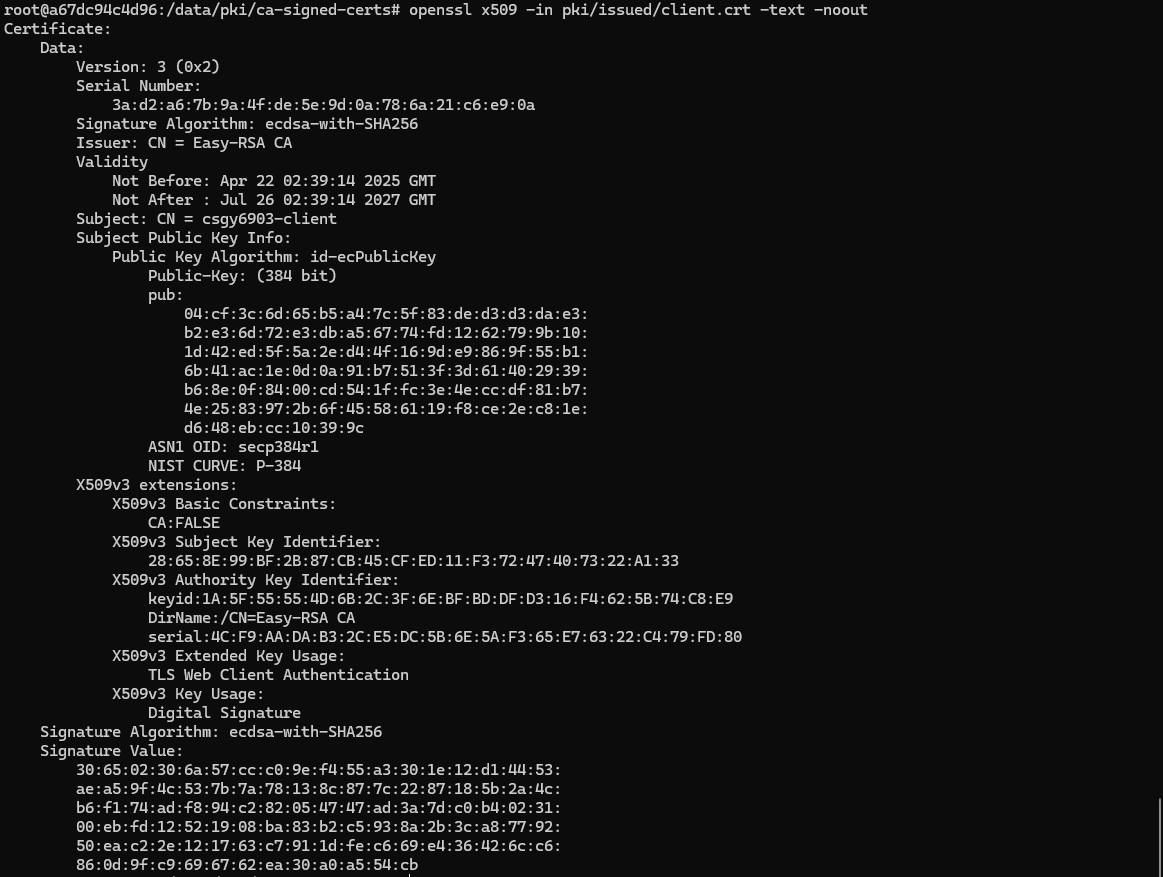
#### Create a new client certificate:



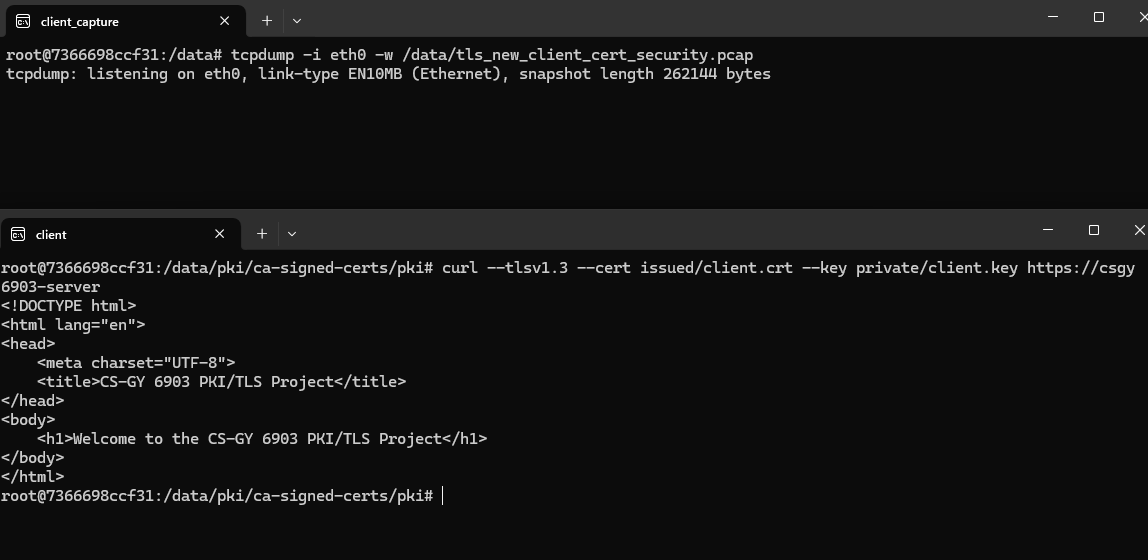
#### Sign new client certificate:



#### New Client Certificate:



Now let us see if the client can still request a web page from the server with the new certificate.



#### Wireshark pcap:

