# Lab: PKI, Certificates, and Apache v1.1

*Portions of this lab were partially funded by the National Science Foundation under Award No. 1303306 and 1718086 awarded to Dr. Wenliang Du at Syracuse University. See Copyright at end of this Lab for information.*

# Contents

**Introduction**

1. Becoming a Certificate Authority (CA)
2. Create a Certificate for sra221.com
3. Deploying a Certificate in an Apache Website
4. Challenge – Fix the site

# Introduction

Symmetric encryption algorithms are fast but they are weak due to the fact that the sender and receiver need to share the same key. Public Key Cryptography such as Asymmetric encryption solves the shared key problem but is slow. The Diffie-Hellman key exchange algorithm allows us to securely exchange a shared key over public networks to allow for the secure exchange of symmetric keys. This approach is in use today when we communicate with websites where an encrypted channel for confidentiality is a concern.

One of the problems we encounter is the "Man in the Middle" (MITM) attack whereby an attacker can replace the intended recipients public key with the attackers. This could lead Alice, in this case to be sending confidential information to Mallory (Attacker) who has now spoofed Bob's address.



MITM attacks are defeated through Digital Certificates that are signed by a trusted authority and linked to an entity like example.com or Bob.



# Deliverables for Lab Reports

Your lab report should be completed on your local computer (laptop, desktop). Files saved to your Practice Labs server will not be saved. This lab DOES NOT use the screen shot buttons or save functions from the left panel of Practice Labs. We are only using the servers in Practice Labs.

Refer to the Canvas assignment for how your report should be formatted. The report should be submitted in a separate document with only your screen shots and answers, see Canvas for details.

# Learning Objectives

The Learning Objectives for this Lab are to broaden our understanding of the following;

* Public-key encryption
* Public-Key Infrastructure (PKI)
* Certificate Authority (CA) and root CA
* X.509 certificate and self-signed certificate
* Apache, HTTP, and HTTPS
* Man-in-the-middle attacks

.

# Lab Topology

For this lab we will utilize the following Servers;

* **PLABSEED** – (SEED version of Linux) - 192.168.0.3

# Setting Up the Lab Environment

Power on your SEED Linux server. In order to get the SEED server you may have to open one of the previous Practice Labs in the side panel to get a list of servers.

For this lab, you will clone a github repository holding the setup scripts and instructions to the lab for easier copy and paste functionality. The setup script will automate many of the time consuming setup tasks and free students up to focus on the Learning Objectives.

Open a terminal session on the SEED server and perform the following commands. Any text that is in the brackets {} is a comment provided to help you understand what we are trying to accomplish with a command.

**sudo git clone https://github.com/ndctdmh/pkilab**

This command should have downloaded scripts into the **pkilab** directory under /home/seed.

Go into that pkilab directory and change permissions and then run the **pkisetup.sh** script using sudo. Enter these commands to accomplish those tasks;

**cd pkilab** {Change to pkilab dir}

**sudo chmod 755 \*** {Add execute permissions to the scripts}

**sudo ./pkisetup.sh** {Run the setup script}

Press enter to run the script and watch for any errors. The script will first copy the lab instructions to your Lab server's desktop. Open this file and use it to make it easier to cut and paste some of the commands from the lab.

# Section 1: Becoming a Certificate Authority (CA)

A Certificate Authority (CA) is a trusted entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. A number of commercial CAs are treated as root CAs; VeriSign is the largest CA at the time of writing. Users who want to get digital certificates issued by the commercial CAs need to pay those CAs.

In this lab, we need to create digital certificates, but we are not going to pay any commercial CA. We will become a root CA ourselves, and then use this CA to issue certificate for others (e.g. servers). In this task, we will make ourselves a root CA, and generate a certificate for this CA. Unlike other certificates, which are usually signed by another CA, the root CA’s certificates are self-signed. Root CA’s certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on PKI. Root CA’s certificates are unconditionally trusted.

## TASK 1 – Generate a Self-Signed Certificate

As we described before, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

Make sure to run these commands from /home/seed

**cd /home/seed**

**openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf**

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The results of the command are stored in two files: **ca.key** and **ca.crt**. The file ca.key contains the CA’s private key, while ca.crt contains the public-key certificate.

* 1. **(Screen Shot ) -**Screen shot the outputs of your new CA private Key and the Public Key Cert. Your output should be a Linux cat or more command of both the cert and key file. We only need to see the top portion of the outputs (Lines 1- 10 are fine).
  2. **(Question and Observation) –** List the importance of the **ca.key** and the **ca.crt** files, be sure to include the purpose of each for a Certificate Authority.

## TASK 2 – Creating a Certificate for sra221.com

Now, that we are a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called **sra221.com**. For this company to get a digital certificate from a CA, it needs to go through three steps.

1. **Generate public/private key pair.** The company needs to first create its own public/private key pair. We can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to encrypt the private key (using the AES-128 encryption algorithm, as is specified in the command option). The keys will be stored in the file server.key:

**openssl genrsa -aes128 -out server.key 1024**

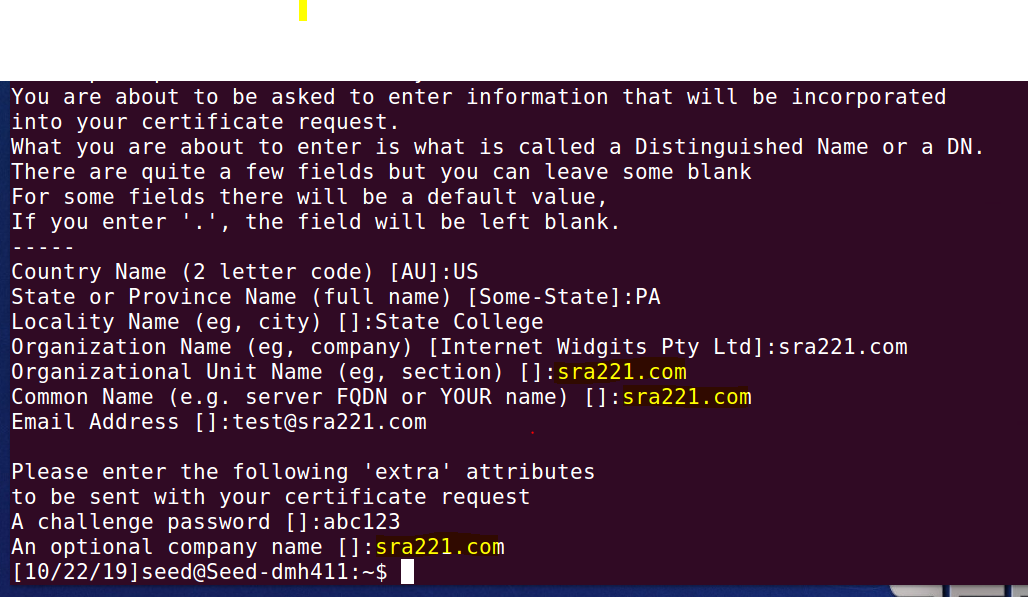
The server.key is an RSA Private KEY and encoded text file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc. To see those, you can run the following command:

**openssl rsa -in server.key –text | more**

1. **Generate a Certificate Signing Request (CSR).** Once the company has the key file, it should generates a Certificate Signing Request (CSR), which basically includes the company’s public key. The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server’s true identity). Please use **sra221.com** as the common name of the certificate request.

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

Keep in mind that this is the CSR for our company sra221.com so keep the names consistent when filling out the CSR, see the screen shot below;

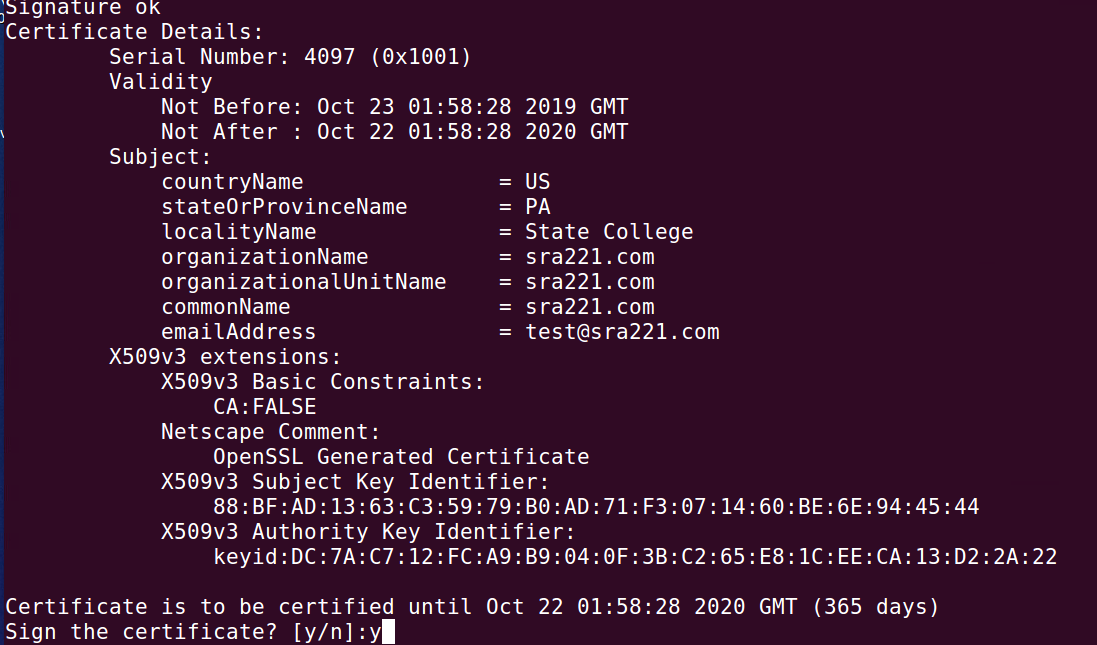


It should be noted that the above command is quite similar to the one we used in creating the self-signed certificate for the CA. The only difference is the -x509 option. Without it, the command generates a request; with it, the command generates a self-signed certificate.

1. **Generating Certificates**. The CSR file needs to have the CA’s signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, we will use our own trusted CA to generate certificates. The following command turns the certificate signing request (server.csr) into an X.509 certificate (server.crt), using the CA’s ca.crt and ca.key: (Put this command on ONE LINE)

**openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf**

**The openssl command** will generate the Certificate. It should prompt you to sign. Note our identifying information, like sra221.com in the CSR.



1. **View the Certificate.**

You can view the new certificate with the following command. Run from your HOME Directory.

**openssl x509 -in server.crt –text |more**

**{Note:** the above command may fail, backspace over –text and retype it}

If it continues to fail then it's possible your key never was generated. Did you see the Y/N prompt to sign the certificate like the screen shot above? If not redo the command.

* 1. (Screen Shot) Paste a copy of the first lines of your certificate that show

1. Signature algorithm
2. Issuer
3. Subject

## TASK 3 – Deploying Certificate in an Apache Web Server

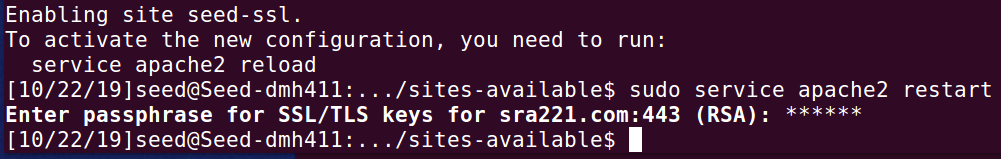
1. Now we need to run a series of commands to enable SSL. Apache will ask us to type the password used for encrypting the private key. Once everything is set up properly, we can browse the web site, and all the traffic between the browser and the server will be encrypted.

Run these commands; If you receive an error about the **clickjacking** site just keep going, this is normal.

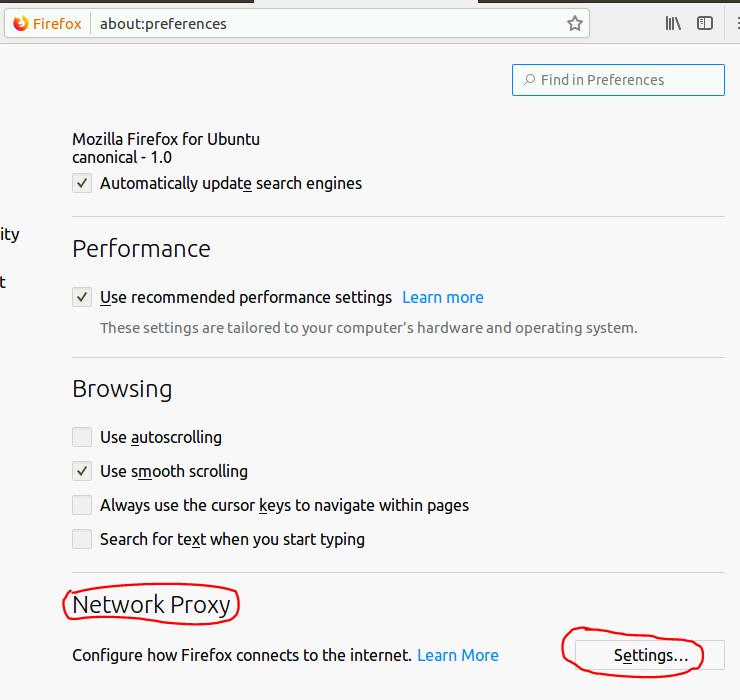
**sudo a2enmod ssl** {Enable the SSL module}

**sudo a2ensite seed-ssl** {Enable the site}

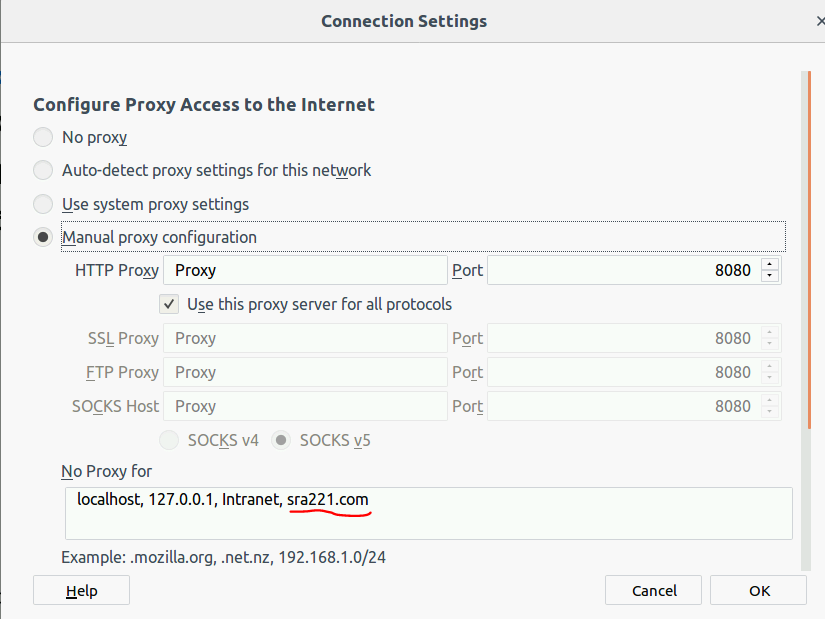
**sudo service apache2 restart** {Restart Apache}



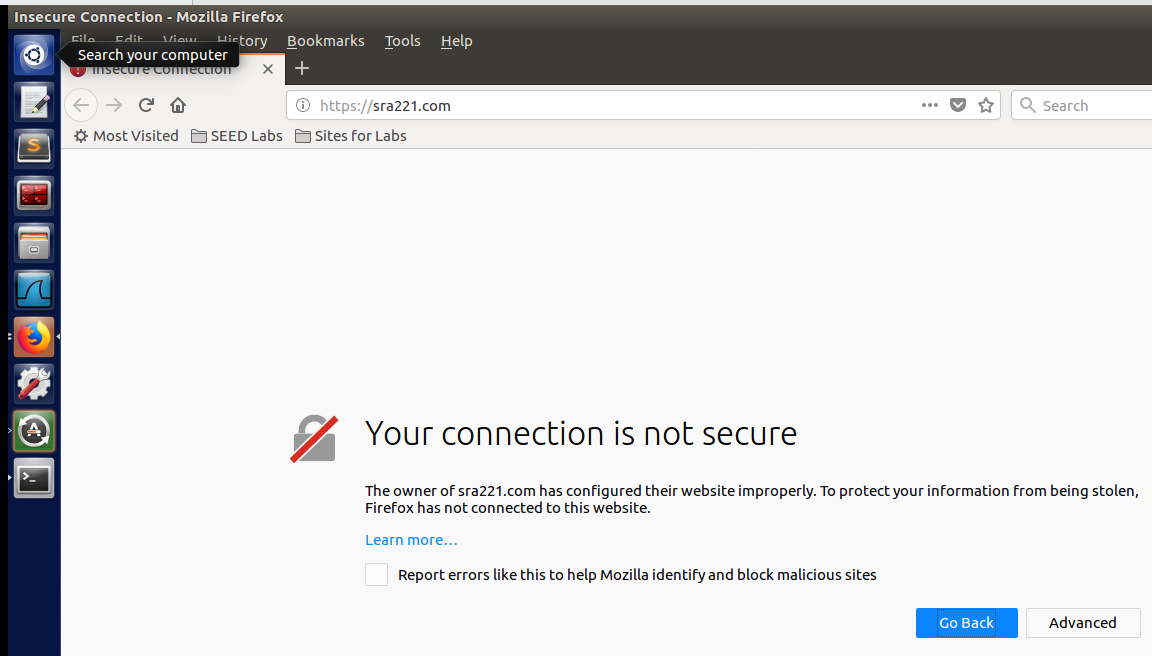
1. Change your Browsers Proxy to allow sra221.com
   1. Open preferences in the Firefox browser and find Network Proxy Settings.



Add your site "sra221.com" to the No Proxy list and save. We need to do this because the lab system is filtering through a proxy server and our site is not on the approved list.



1. Now, visit the site (https://sra221.com) from Firefox and observe.



1. Perform some analysis of how browsers use certificates. Fix the problem so that sra221.com is trusted and displays a green is good lock.
   1. (Screen Shot) Paste a screen shot showing that the sra221 site is trusted, this should include the Green Lock and your Welcome Message.
   2. (Screen Shot) Paste a screen shot after clicking on the Green lock showing the name of the Verified by; CA name. If you are using a snipping tool you might need to move the cursor off the screen so it does not move your output.
   3. (Question) Explain what you had to do to enable the browser to trust the entity SRA221.com.? Your response should include the role of the CA and what has been done with the certificate to enable trust by the browser.

Copyright © 2018 Wenliang Du, Syracuse University.

Portions of this document were partially funded by the National Science Foundation under Award No. 1303306 and 1718086. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. A human-readable summary of (and not a substitute for) the license isthe following: You are free to copy and redistribute the material in any medium or format. You must give appropriate credit. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. You may not use the material for commercial purposes.