

Public-Key Infrastructure (PKI) Lab

Copyright © 2018 by Wenliang Du.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.

1 Overview

Public key cryptography is the foundation of today's secure communication, but it is subject to man-in-the-middle attacks when one side of communication sends its public key to the other side. The fundamental problem is that there is no easy way to verify the ownership of a public key, i.e., given a public key and its claimed owner information, how do we ensure that the public key is indeed owned by the claimed owner? The Public Key Infrastructure (PKI) is a practical solution to this problem.

The learning objective of this lab is for students to gain the first-hand experience on PKI. SEED labs have a series of labs focusing on the public-key cryptography, and this one focuses on PKI. By doing the tasks in this lab, students should be able to gain a better understanding of how PKI works, how PKI is used to protect the Web, and how Man-in-the-middle attacks can be defeated by PKI. Moreover, students will be able to understand the root of the trust in the public-key infrastructure, and what problems will arise if the root trust is broken. This lab covers the following topics:

- 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

Readings. Detailed coverage of PKI can be found in the following:

- Chapter 23 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at <https://www.handsonsecurity.net>.

Related labs. A topic related to this lab is the Transport Layer Security (TLS), which is based on PKI. We have a separate lab for TLS. In addition, we have a lab called *RSA Public-Key Encryption and Signature Lab*, which focuses on the algorithm part of the public-key cryptography.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Lab Tasks

2.1 Task 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.

The Configuration File `openssl.cnf`. In order to use OpenSSL to create certificates, you have to have a configuration file. The configuration file usually has an extension `.cnf`. It is used by three OpenSSL commands: `ca`, `req` and `x509`.

The most important settings in this configuration file define the directory for storing certificate and key files, as well as records of the creation of certifications.

```
dir           = ./demoCA           # Where everything is kept
certs         = $dir/certs         # Where the issued certs are kept
crl_dir       = $dir/crl           # Where the issued crl are kept
new_certs_dir = $dir/newcerts      # default place for new certs.
database      = $dir/index.txt     # database index file.
serial        = $dir/serial        # The current serial number
```

For the `index.txt` file, simply create an empty file. For the `serial` file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file `openssl.cnf`, you can create and issue certificates.

The instructor will provide you with a configuration file to use for this lab and a shell script to create the required directory and files. These files will be provided in a ZIP archive. Unpack the ZIP archive in your `labs` directory. It will create a `pki` directory, which contains the provided files. Change into that directory, then run the `ca_setup.sh` script.

Certificate Authority (CA). 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:

```
$ 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 output 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.

2.2 Task 2: Creating a Certificate for SEEDPKILab2020.com

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called SEEDPKILab2020.com. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 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 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
```

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generate 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 `SEEDPKILab2020.com` as the common name of the certificate request.

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

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.

Step 3: 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 X509 certificate (`server.crt`), using the CA's `ca.crt` and `ca.key`:

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
    -config openssl.cnf
```

If OpenSSL refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA. The matching rules are specified in the configuration file (look at the `[policy_match]` section). You can change the names of your requests to comply with the policy, or you can change the policy. The configuration file also includes another policy (called `policy_anything`), which is less restrictive. You can choose that policy by changing the following line:

```
"policy = policy_match"  change to "policy = policy_anything".
```

2.3 Task 3: Deploying Certificate in an HTTPS Web Server

In this lab, we will explore how public-key certificates are used by websites to secure web browsing. We will set up an HTTPS website using `openssl`'s built-in web server.

Step 1: Configuring DNS. We choose `SEEDPKILab2020.com` as the name of our website. To get our computers recognize this name, let us add the following entry to `/etc/hosts`; this entry basically maps the hostname `SEEDPKILab2020.com` to our localhost (i.e., `127.0.0.1`):

```
127.0.0.1  SEEDPKILab2020.com
```

Step 2: Configuring the web server. Let us launch a simple web server with the certificate generated in the previous task. OpenSSL allows us to start a simple web server using the `s_server` command:

```
# Combine the secret key and certificate into one file
% cp server.key server.pem
% cat server.crt >> server.pem

# Launch the web server using server.pem
% openssl s_server -cert server.pem -www
```

By default, the server will listen on port 4433. You can alter that using the `-accept` option. Now, you can access the server using the following URL: `https://SEEDPKILab2020.com:4433/`. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: *“seedpkilab2020.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown”*.

Step 3: Getting the browser to accept our CA certificate. Had our certificate been assigned by VeriSign, we will not have such an error message, because VeriSign’s certificate is very likely preloaded into Firefox’s certificate repository already. Unfortunately, the certificate of `SEEDPKILab2020.com` is signed by our own CA (i.e., using `ca.crt`), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA’s self-signed certificate.

- We can request Mozilla to include our CA’s certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so we will not pursue this direction.
- **Load `ca.crt` into Firefox:** We can manually add our CA’s certificate to the Firefox browser by clicking the following menu sequence:

Edit -> Preference -> Privacy & Security -> View Certificates.

You will see a list of certificates that are already accepted by Firefox. From here, we can “import” our own certificate. Please import `ca.crt`, and select the following option: “Trust this CA to identify web sites”. You will see that our CA’s certificate is now in Firefox’s list of the accepted certificates.

Step 4. Testing our HTTPS website. Now, point the browser to `https://SEEDPKILab2020.com:4433`. Please describe and explain your observations. Please also do the following tasks:

1. Modify a single byte of `server.pem`, and restart the server, and reload the URL. What do you observe? Make sure you restore the original `server.pem` afterward. Note: the server may not be able to restart if certain places of `server.pem` is corrupted; in that case, choose another place to modify.
2. Since `SEEDPKILab2020.com` points to the localhost, if we use `https://localhost:4433` instead, we will be connecting to the same web server. Please do so, describe and explain your observations.

2.4 Task 4: Deploying Certificate in an Apache-Based HTTPS Website

The HTTPS server setup using openssl's `s_server` command is primarily for debugging and demonstration purposes. In this lab, we set up a real HTTPS web server based on Apache. The Apache server, which is already installed in our VM, supports the HTTPS protocol. To create an HTTPS website, we just need to configure the Apache server, so it knows where to get the private key and certificates. We give an example in the following to show how to enable HTTPS for a website `www.example.com`. Your task is to do the same for `SEEDPKILab2020.com` using the certificate generated from previous tasks.

An Apache server can simultaneously host multiple websites. It needs to know the directory where a website's files are stored. This is done via virtual host configuration files, located in the `/etc/apache2/sites-available` directory. To add an HTTP website, we create a new configuration file in this directory named `example_com.conf` in this directory with the following contents.

```
<VirtualHost *:80>
    ServerName www.example.com
    DocumentRoot /var/www/example_com
    DirectoryIndex index.html
</VirtualHost>
```

The `ServerName` entry specifies the name of the website, while the `DocumentRoot` entry specifies where the files for the website are stored. For the configuration example above, we would need to create the `DocumentRoot` directory `/var/www/example_com` and copy the files to be served into that directory. Note that this directory must be readable by the web server. Check that the permissions of the newly created directory match those of the default web site directory `/var/www/html`.

To make `www.example.com` an HTTPS website, we need to create a similar file with a name like `example_ssl.conf` with these contents:

```
<VirtualHost *:443>
    ServerName www.example.com
    DocumentRoot /var/www/example_com
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile      /etc/apache2/ssl/example.crt
    SSLCertificateKeyFile   /etc/apache2/ssl/example.key
</VirtualHost>
```

To make the SSL site work, the server certificate and key must be copied to the paths specified in the configuration file. You will need to remove the password from the key if one was set with the following command before copying it:

```
$ openssl rsa -in example.key -out example-nopw.key
$ mv example-nopw.key example.key
```

To enable a web site file, we use the `a2ensite` command, then restart the web server. We will check the configuration file syntax before attempting to restart the server to catch any typos or other mistakes in the new site configuration file. If the configuration test returns any errors, edit the configuration file to fix them before re-running the configuration test and restarting the web server.

```
$ sudo a2ensite example_ssl.conf
$ sudo apachectl configtest
$ sudo apachectl restart
```

Please use the above example as guidance to set up an HTTPS server for `SEEDPKILab2020.com`. In your lab report, describe the steps that you have taken, the contents of your configuration file, and provide a screenshot of the web browser window showing that you can successfully browse the HTTPS site.

2.5 Task 5: Launching a Man-In-The-Middle Attack

In this task, we will show how PKI can defeat Man-In-The-Middle (MITM) attacks. Figure 1 depicts how MITM attacks work. Assume Alice wants to visit `example.com` via the HTTPS protocol. She needs to get the public key from the `example.com` server; Alice will generate a secret, and encrypt the secret using the server's public key, and send it to the server. If an attacker can intercept the communication between Alice and the server, the attacker can replace the server's public key with its own public key. Therefore, Alice's secret is actually encrypted with the attacker's public key, so the attacker will be able to read the secret. The attacker can forward the secret to the server using the server's public key. The secret is used to encrypt the communication between Alice and server, so the attacker can decrypt the encrypted communication.

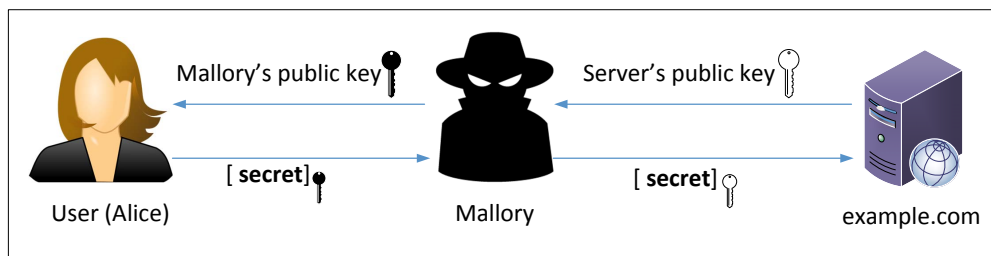


Figure 1: A Man-In-The-Middle (MITM) attack

The goal of this task is to help students understand how PKI can defeat such MITM attacks. In the task, we will emulate an MITM attack, and see how exactly PKI can defeat it. We will select a target website first. In this document, we use `example.com` as the target website, but in the task, to make it more meaningful, students should pick a popular website, such as a banking site and social network site.

Step 1: Setting up the malicious website. In Task 4, we have already set up an HTTPS website for `SEEDPKILab2020.com`. We will use the same Apache server to impersonate `example.com` (or the site chosen by students). To achieve this, students will need to

1. Create keys and certificates for impersonated site using our CA, as we did in Task 2.
2. Create an Apache virtual host configuration file with the `ServerName` of the impersonated site.
3. Make a copy of the home page of the legitimate site in the `DocumentRoot` specified in the configuration file. You can use your browser's ability to save web pages to do this.
4. Enable the site and restart Apache, as we did in Task 4.

Our goal is the following: when a user tries to visit `example.com`, we are going to get the user to land in our server, which hosts a fake website for `example.com`. If this were a social network or e-commerce site, the fake site could be used to carry out a phishing attack by displaying a login page similar to the one in the target website. If users cannot tell the difference, they may type their account credentials in the fake webpage, essentially disclosing the credentials to the attacker.

Step 2: Becoming the man in the middle There are several ways to get the user's HTTPS request to land in our web server. One way is to attack the routing, so the user's HTTPS request is routed to our web server. Another way is to attack DNS, so when the victim's machine tries to find out the IP address of the target web server, it gets the IP address of our web server. In this task, we use "attack" DNS. Instead of launching an actual DNS cache poisoning attack, we simply modify the victim's machine's `/etc/hosts` file to emulate the result of a DNS cache poisoning attack (the `IP_Address` in the following should be replaced by the actual IP address of the malicious server).

```
<IP_Address> example.com
```

Step 3: Browse the target website. With everything set up, now visit the target real website, and see what your browser would say. Please explain what you have observed.

3 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.