

Lab 6 - Public-Key Cryptography and PKI

Deadline: May 8, 2020

(Submissions allowed until May 10, 2020 with no penalty)

This lab is to be done individually.

Overview

The learning objective of this lab is for students to get familiar with the concepts in the Public-Key encryption and Public-Key Infrastructure (PKI). After finishing the lab, students should be able to gain first-hand experience on public-key encryption, digital signature, public-key certificates, certificate authority, authentication based on PKI. Moreover, students will be able to use tools and write programs to create secure channels using PKI.

Reading Material

Please go through Lec 18 (Key Management and PKI) videos and reading material.

Lab Setup

Note. This uses the same environment as the previous lab.

This lab requires the following tools:

1. Openssl
2. A hex editor (Ghex or Bless. You can also use an online hex editor)

You can either install these tools on your own or download the prepared VM from [here](#) or [here](#) or [here](#).

Note. The VM file you download is a virtual hard disk.

Note. VM username: seed , password: dees

Note. For those that cannot set up their VMs, ubuntu and macOS has openssl installed by default. You would only need to install a hex editor for the lab.

Lab Tasks

Task 1: Become 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 certificates 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. RootCA'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 manual page of openssl.cnf can be found using Google search. You can also get a copy of the configuration file from /usr/lib/ssl/openssl.cnf. After copying this file into your current directory, you need to create several subdirectories as specified in the configuration file (look at the [CAdefault] section):

```
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.

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 is stored in two files: `ca.key` `ca.crt`. The file **ca.key** contains the CA's private key, while **ca.crt** contains the public-key certificate.

Submit screenshots of your work.

Task 2: Create a Certificate for PKILabServer.com

Now, we have become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called PKILabServer.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 keypair. 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 PKILabServer.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 [policymatch] 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"

Submit screenshots of your work.

Task 3: Use PKI for Websites

In this lab, we will explore how public-key certificates are used by web sites to secure web browsing. First, we need to get our domain name. Let us use PKILabServer.com as our domain name. To get our computers to recognize this domain name, let us add the following entry to /etc/hosts; this entry basically maps the domain name PKILabServer.com to our localhost:

127.0.0.1 PKILabServer.com

Next, 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 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://PKILabServer.com:4433/>. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following:

"pkilabserver.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown"

Had this 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 PKILabServer.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.

1. 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.

2. 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 > Advanced > 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 websites"**. You will see that our CA's certificate is now in Firefox's list of the accepted certificates.

Now, point the browser to <https://PKILabServer.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 PKILabServer.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.

Task 4: Performance Comparison: RSA versus AES

In this task, we will study the performance of public-key algorithms. Please prepare a file (message.txt) that contains a 16-byte message. Please also generate a 1024-bit RSA public/private key pair. Then, do the following:

1. Encrypt message.txt using the public key; save the output in messageenc.txt.
2. Decrypt messageenc.txt using the private key.
3. Encrypt message.txt using a 128-bit AES key.
4. Compare the time spent on each of the above operations, and describe your observations. If an operation is too fast, you may want to repeat it for many times, and then take an average. Explain how you chose the method of testing as well.

After you finish the above exercise, you can now use OpenSSL's speed command to do such a benchmarking. Please describe whether your observations are similar to those from the outputs of the speed command. The following command shows examples of using speed to benchmark rsa and aes:

```
openssl speed rsa  
openssl speed aes
```

Task 6: Create a Digital Signature

In this task, we will use OpenSSL to generate digital signatures. Please prepare a file (example.txt) of any size. Please also prepare an RSA public/private key pair. Do the following:

1. Sign the SHA256 hash of example.txt; save the output in example.sha256.
2. Verify the digital signature in example.sha256.
3. Slightly modify example.txt, and verify the digital signature again.

Please describe how you did the above operations (e.g., what commands do you use, etc.). Explain your observations. Please also explain why digital signatures are useful.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanations to the observations that are interesting or surprising. In your report, you need to answer **all the questions** listed in this lab.

You are required to submit a **2x10xxx.pdf** file with all of your findings by May 8, 2020.