Public-Key Cryptography and PKI

35 Points–Partner Lab
Due Date: Lesson 19

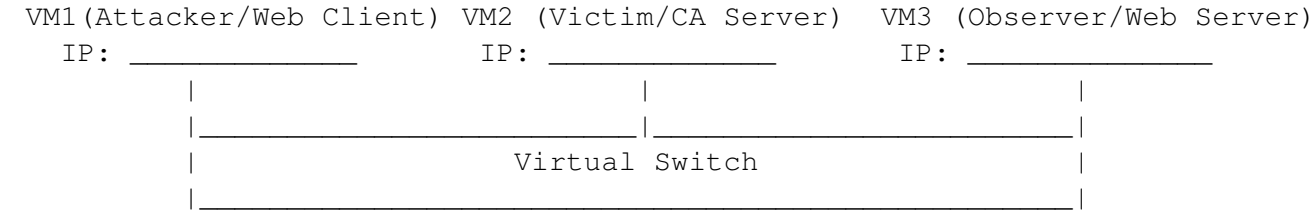
1 Overview

The learning objective of this lab is for students to familiarize themselves with concepts in the Public-Key encryption and Public-Key Infrastructure (PKI). After finishing the lab, students should be able to demonstrate application of public-key encryption, digital signatures, public-key certificates, certificate authorities, and authentication based on PKI.

2 Lab Environment

2.1 Network

You should start by loading up your three Ubuntu systems. Log in and verify the IP and MAC address of each. Write each system information below for future reference. Based on the set-up from previous labs, Attacker's IP should be 10.172.x.12, Victim should be 10.172.x.10, and Observer should be 10.172.x.11.



DOUBLE CHECK YOUR LAB SET-UP BEFORE YOU GO FORWARD!

2.2 Directory Creation for Lab Usage.

You will need to create a number of directories to maintain certificates, revocation lists and configuration files. Move to `/home/eecs/` directory on Victim (CA Server) and create the following directory structure using `mkdir`:

```
-home (exists)
  -eecs (exists)
    -PKI
      -CA
        -CS482_EECS_CA
          -certs
          -crl
          -newcerts
```

2.3 SSH Server

You will need ssh installed and running on ALL machines. To install:

```
sudo apt-get install openssh-server
```

To run:

```
sudo service ssh start
```

See Lab 5 for further details.

3 Lab Tasks

3.1 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; Symantec (VeriSign) is the largest CA at the time of writing. Users who want a digital certificate 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 this lab the CA Server will be VM2 Victim, go to that machine now. 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`. You should obtain a copy of the configuration file from `/usr/lib/ssl/openssl.cnf`. and place it in your `ca` directory you just created. Once you have copied the configuration file into your directory, use `cat openssl.cnf` and view the `[CA_default]` section. You will note that these point to the directories you just created in the lab setup. If the root directory is not `CS482_EECS_CA`, please update it accordingly in the configuration file. **NOTE: Beaware of the `./` relative file path in `dir`; the computer will assign `dir` the value of the location of `oppenssl.cnf` concatentated with `CS482_EECS_CA`.**

```
dir             = ./CS482_EECS_CA  # 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
```

There are two more files we need to create, `index.txt` and `serial`. For the `index.txt` file, simply create an empty file (make sure there is no spaces or empty lines in that file). The easiest way to do this action is using the `touch` command. For the `serial` file, put a single number in string format in the file. We will use `1000` in this lab. 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, please do this inside the `ca` directory:

```
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. Make sure you remember what values you use for these fields. You will need to enter the same values again later on. 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.

3.2 Task 2: Create a Certificate for `PKILabServer.com`

Now, that 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`, which will host a web server on VM3 Observer, go to that machine now. 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). Run the following command in your home, i.e. `/home/eecs/` directory. 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
```

Question 1: Provide a screenshot for evidence.

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generates a Certificate Signing Request (CSR), which 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. Again, do this in the your home directory on VM3 Observer Web Server.

```
openssl req -new -key server.key -out server.csr -config /usr/lib/ssl/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. The CSR files ashould always be sent to a trusted CA for their signature via secure means. In this lab, we will transfer our CSR via `scp` to our own trusted CA to generate certificates.

Question 2: What is `scp` and why is it safer than tftp or other means of transferring files?

Go to VM2 Victim CA Server and issue:

```
scp eecs@<IP of VM3 Observer Web Server>:~/server.csr ~/PKI/CA/
```

Go to the `ca` directory and run The following command to turn 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
```

Note: The above command is all one line, without the slash included.

If `OpenSSL` refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA, i.e. you used different countries, or other info between your server CSR and the certificate authority certificate. 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 (you must regenerate a new request with the correct info), 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".`

To verify that everything worked correctly, look in the `index` and `serial` files:

```
cat index.txt
cat serial
```

You should see your certificate information along with the serial number of "1000". If you look inside the serial file, you will see it has incremented by one, to "1001".

Troubleshooting: If you are still stuck and your certificate didn't compile correctly, first look over all the lab steps again to ensure you completed all in the right order/location. If you are using the same key but an issue raised during the signing of the server cert, you may get an issue where you are prevented from signing redundant instances of the same entity. To fix this you can modify the code in two places:

```
In openssl.cnf set 'unique_subject=no'
In index.txt set 'unique_subject=no'
```

Question 3: Provide a screencapture of your index.txt file showing the completed certificate.

3.3 Task 3: Use PKI for Web Sites

Go back to VM3 Observer Web Server so we can 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. First, use `scp` command to tranfer the signed `server.crt` file back to VM3 Observer Web Server. Next, we will generate the `pem` file used for our website by combining the secret key and certificate:

```
$ cp server.key server.pem
$ cat server.crt >> server.pem
```

Launch the web server using server.pem

```
sudo openssl s_server -cert server.pem -www -accept 443
```

Now we will explore how public-key certificates are used by web sites to secure web browsing. First, goto VM1 Attacker Web Client. We need to get VM3 Observer Web Server's domain name. Let us use `PKILabServer.com` as our domain name. Go to VM1 Attacker Web Client. To get our computer to recognize this domain name, add the following entry to `/etc/hosts`; this entry basically maps the domain name `PKILabServer.com` to VM3 Observer Web Server:

```
<VM3's Observer Web Server IP> PKILabServer.com
```

Now, you can access the server using the following URL: `https://PKILabServer.com:443/`. 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.

- 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:** First use `scp` to copy the `ca.crt` file from VM2 Victim CA Server to VM1 Attacker Web Client. On the client, 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. underneath the "Authorities" tab, 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.

Now, close firefox, start WireShark and point the browser to `https://PKILabServer.com:443`.

Question 4: Provide a screenshot of the successfully applied certificate browser session

Modify a single byte of `server.pem`, and restart the server, and reload the URL.

Question 5: 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.

Question 6: Provide a WireShark capture and provide a BRIEF description of what is happening .

3.4 Task 4: Performance Comparison: RSA versus AES

In this task, we will study the performance of public-key algorithms. You could prepare a file (`message.txt`) that contains a message and then time the encryption with your RSA or AES key. An easier option may be to use `OpenSSL`'s `speed` command to do such benchmarking. The following command shows examples of using `speed` to benchmark `rsa` and `aes`:

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

Question 7: Explain the results of your speedtest. You should support your argument with analysis of each algorithm's operation.

3.5 Task 5: OPTIONAL LAB TASK, Individual Submission (5pts): 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, then 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.

Question 8: (Optional, Individual) 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.

Note: there are many detailed online examples in which you can reference to help with this task.

4 Submission

Your group needs to submit a detailed lab report to describe what you have done and what you have observed. This includes

- explanations and supporting evidence for all specified questions and submission items

- time spent
- any challenging points or thoughts on what you found interesting during the lab
- a reflection on security principles
- a separate individual report for the optional lab task, if you chose to complete.

5 Submission requirements

5.1 Rubric

1. 3 pts per question
2. 5 pts bonus Task 5.
3. 10 pts feflection

5.2 Partner Submission

Provide one written lab report, answering each question properly labelled with the number and original question, per partner team. Be sure to include the time spent on the lab and document any external resources used. Again good documentation:

1. clearly enumerates tasks with a description of you did and evidence.
2. shows the progress you were able to achieve.
3. explains your troubleshooting attempts.
4. accurately describes an issue and the potential solution (if really good, I will give near full credit).

5.3 Individual Submission

Each member needs to submit a detailed lab reflection. This includes

- approximately one half page that describes what PKI is providing us in the lab. Use key terms from chapter 1, the lecture and assigned readings.
- any challenging points or thoughts on what you found interesting during the lab
- time spent you personally spent and how much effort you put forth
- time your partner spent, and how much effort they put forth
- be sure document any external resources used.