**GENI - Public-Key Cryptography and PKI**

**Lab Environment Parameters**

**RSPEC file:**

<https://raw.githubusercontent.com/informationcomputerscience/EdGENI/master/Rspec-Files/PublicKeyLab-Rspec.txt>

**Lab Overview**

This lab will help you to get familiar with Public-Key encryption and Public-Key Infrastructure (PKI). We will use the GENI infrastructure to host a machine, on which you work as a root Certificate Authority (CA), issue a certificate, generate public/private key pairs, and gain experience with digital signatures and PKI based authentication. For this lab we will utilize the Firefox web browser and OpenSSL. Throughout and after completing the lab we will record documentation of our experience and answer questions listed in the lab.

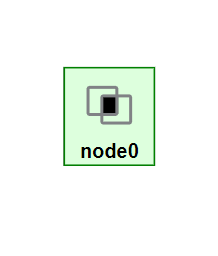
**Preparation**

**Before you start, you will need to complete the following setups (If you have completed some steps before, you can skip them):**

1. Setup user account on GENI and join a Project. (see the document *Setup-User*). See the Lab Environment for the GENI project name.
2. Setup Lab in GENI (see the document *Setup-Lab*). See the Lab Environment for the RSPEC file.
3. Establish SSH connections to each GENI virtual machine and obtain the GUI through VNC (see the document *Connect-to-VM*)

**NOTE:** This project requires the use of the Firefox browser, so you will need the GUI interface via VNC Server for this project. If you need to refresh on how to install VNC Server on your GENI machine, please see Step 2 and following on Connect-to-VM.

**Network Topology**



Our topology for this lab is very simple. We will be using a single host, node0, to execute all necessary tasks. From this host we will become a root CA, create and grant a certificate, and use that certificate in Firefox. OpenSSL is already installed on the GENI host by default. You will be responsible for installing VNC Server (if you haven’t already) and the Mozilla Firefox browser.

**Task 1 - Check OpenSSL and Install Mozilla Firefox**

1. On the VM, open a terminal, type in **openssl version** to verify if OpenSSL has already been installed. In most cases, OpenSSL has been installed on the VM. If it does not, type in the command **sudo apt-get install openssl** to install it.
2. Install Mozilla Firefox for use later on in the lab. Open a terminal window in your GUI and issue the command **sudo apt-get install firefox**. This command will download and install Firefox to your GENI machine.
3. After installing Firefox, type **firefox &** into your command line. *Make sure the command is issued in your VM GUI. Otherwise, you won’t be able to see the firefox*. This will start up a new Firefox instance, and it should appear within a few seconds after you issue the command (there is a slight start up delay, so do not be concerned if the window doesn’t appear instantly). Now that we have Firefox up and running, you can minimize the window or move it off to the side, we won’t need it for a little bit. You can also minimize the terminal window we used to open Firefox with (but don’t close it!).

**Task 2 - 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 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 certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on PKI. Root CA certificates are unconditionally trusted.

1. 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. We will need a copy of the OpenSSL configuration file to use for our project. To begin, we will copy the openssl.cnf file to the directory we plan to work in. The openssl.cnf file is located in the directory **/usr/lib/ssl/openssl.cnf**  
     
   **NOTE:** We recommend creating a directory to work in for the project, it makes keeping track of your files and what you are doing easier!  
     
   Navigate to your project directory. From your working directory type the command:

**cp /usr/lib/ssl/openssl.cnf openssl.cnf**.

This will create a copy of the OpenSSL configuration file in our project directory.

1. We now need to create a directory structure as specified in the openssl.cnf file (look at the [CA default] 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.

1. As we described before, we need to generate a self-signed certificate for our CA. This means that our CA is totally trusted, and its certificate will serve as the root certificate. To generate the self-signed certificate for our CA, type the command:   
     
   **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 and ca.crt. The file ca.key contains the CA’s private key, while ca.crt contains the public-key certificate.
2. **Open the two files (ca.key and ca.crt) and make a screenshot for the content of each file. Explain what is certified in ca.crt and who is signing this certificate.**

**Task 3 - Create a Certificate for PKILabServer.com**

Now that we have become a root CA, we can sign and issue digital certificates. Our first customer will be a company called PKILabServer.com. For this company to get a certificate from our root CA, it needs to go through 3 steps:

1. We, the PKILabServer.com, firstly need to generate a 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:   
   Type the command: **openssl genrsa -aes128 -out server.key 1024**   
     
   **NOTE:** The server.key is an encoded text file (and is 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 command **openssl rsa -in server.key -text**.
2. Next we will generate a Certificate Signing Request (CSR). Once the company has the key file, it should generates a Certificate Signing Request, 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 (server FQDN or YOUR name).  
     
   Type: **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.
3. 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 are using 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. To perform this, issue the command:  
     
   **openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf**
4. **Open server.crt and take a screenshot to display the content of your certificate.**

**NOTE:** If OpenSSL refuses to generate certificates, or the certificate is empty, 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"**

**Task 4 - Use PKI for Websites**

1. Next up, we’re going to look at how public-key certificates are used by websites to secure web browsing. First, we need to get our domain name. Let’s use **PKILabServer.com** as our domain name. To get our host to recognize this domain name, let’s add the following entry to /etc/hosts; this entry basically maps the domain name PKILabServer.com to our localhost (i.e., 127.0.0.1):  
     
    **127.0.0.1 PKILabServer.com**Next, we’ll launch a simple web server using the certificate we generated earlier in our lab. We can do this using the command **s\_server**. First, though, we need to complete another step. We need to combine our secret key and certificate into one file. Type in this series of commands:  
     
   **cp server.key server.pem  
   sudo -s  
   cat server.crt >> server.pem  
   exit**  
   **NOTE**: The above command **sudo -s** allows us to temporarily become root. We need to do this in order to execute the command **cat server.crt >> server.pem**, which is actually two commands in one. In order to append the output of server.key to the server.pem file, we need root permission. We can’t utilize the typical sudo here, however, since sudo will only work on the first command, in this case **cat server.crt**. It will not work to allow us to modify server.pem with the output of **cat server.crt**, so we need to use sudo -s. We then issue the exit command to return to our normal user state after performing what we need to as root.  
     
   Now that we’ve combined our files, we launch the web server with this command:  
    **openssl s\_server -cert server.pem -www**  
     
   By default, the server will listen on port 4433. You can alter that using the -accept option if you wish to. Now, you can access the server with firefox by using the following URL: **https://PKILabServer.com:4433/** (if you chose to change your port to something beside 4433, you’ll need to put it in place of 4433 after the .com:). Most likely, going to this URL won’t work. You may or may not get an error message like the following: “pkilabserver.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown.” Regardless, if you can’t access the site, don’t worry, that’s what’s supposed to happen.   
     
   Had this certificate been assigned by an established CA like VeriSign, we probably wouldn’t have this issue. Firefox tends to have the major CA’s certificates already in it’s repository. Unfortunately, the certificate of PKILabServer.com is signed by our own CA (i.e., using ca.crt), and our CA is not recognized by Firefox. We can use two ways to get Firefox to accept our 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 CA’s, 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 won’t go   
    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:  
     
    **1)** Click on the “hamburger menu” in the top right corner of the browser   
    window (the three horizontal bars stacked on top of each other).   
    **2)** Click on the “Preferences” or “Options” item (Above “Customize”).  
    **3)** Click on “Privacy & Security” on the left-hand side of the page.  
    **4)** Scroll down to the bottom of the page. You should see a section called   
    “Certificates.” Click on the “View Certificates” button.  
     
   Now you should see a list of certificates that are already accepted by Firefox. From here, we can “import” our own certificate. **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, point your browser to **https://PKILabServer.com:4433**. You should be able to see the server return a lot of information.
2. **Take a screenshot of the browser.**
3. **Based on the SSL handshake protocol we learned , explain the messages on your screenshot.**
4. After complete the above steps, you need to shut off the web server. Go back to the terminal that you executed the command

**openssl s\_server -cert server.pem -www**

, typing ctrl+c will shut off the web server for PKILabServer.com.

**Task 5 - Performance Comparison: RSA vs. AES**

Next up, we will perform public-key encryption/decryption by using OpenSSL. You may need to search on the Internet to learn the syntax for the OpenSSL commands necessary. The webpages below list a couple of examples:

<https://rietta.com/blog/2012/01/27/openssl-generating-rsa-key-from-command/>

<https://raymii.org/s/tutorials/Encrypt_and_decrypt_files_to_public_keys_via_the_OpenSSL_Command_Line.html>

<https://www.madboa.com/geek/openssl/#how-do-i-sign-a-digest>

.

Please complete the following steps:

1. Prepare a file (message.txt) that contains a 16-byte message.
2. **Generate a 1024-bit RSA public/private key pair. Take a screenshot to show your command.**
3. **Encrypt** message.txt **using the public key; save the output in** message\_enc.txt, **take a screenshot to show your command and the content of message\_enc.txt.**
4. **Decrypt** message\_enc.txt **using the private key, attaching a screenshot.**
5. **Encrypt** message.txt **using a 128-bit AES key, attaching a screenshot.**
6. **After you finish the above exercise, you can now 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   
   **Attach a screenshot to show your experiments and discuss your observation.**

**Task 6 - Create Digital Signatures**

Finally, we are going to use OpenSSL to generate digital signatures. Please prepare a file (example.txt) of any size. Please also prepare an RSA public/private key pair, and then do the following:

1. **Sign the SHA256 hash of** example.txt**; save the output in** example.sha256**, attaching a screenshot.**

1. **Verify the digital signature in** example.sha256**, attaching a screenshot.**
2. **Slightly modify** example.txt**, and verify the digital signature again, attaching a screenshot.**

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.