Network Security Lab
Experiment#2
Public Key Cryptography

Dr. Ahmed Awad & Eng. Ibrahim Amryeh

February 13, 2021

1 Objectives

The purpose of this experiment is to gain the basic knowledge of public-key encryption, digital signature, and Public Key Infrastructure (PKI) certificates. This should be accomplished through using proper tools under Linux.

2 Pre-Lab

1. Read about Public-Key Cryptography
2. Read about Digital Signature Schemes.
3. Read about Public Key Infrastructure (PKI) certificates.

3 Procedure

3.1 Certificate Authority (CA)

a. Make sure that OpenSSL tool has been installed on your machine. How can you do that?

b. A Certificate Authority (CA) is a trusted entity that issues digital certificates to users. The users are then certified to their ownership of a public key by this certificate. **Briefly explain the concept of digital certificates.**

c. Provide an example of three commercial CAs.

d. What is a root CA? How does a root CA gain its own certificates?

e. Check the content of the configuration file of openssl under the directory **/usr/lib/ssl/**. The name of the file is **openssl.cnf**.

f. Create a copy of openssl.cnf file under your current directory. (Remember that you have to remove all your files when you finish your lab).

g. Create the sub-directory demoCA in your current directory.

h. Create a file named **index.txt** under demoCA . Leave it empty.

i. Create a file named **serial** under demoCA. Put a single number in string format in the file (e.g., 1000).

j. Create the following sub-directories under demoCA: certs, crl, newcerts.

k. Open the file openssl.cnf that you have just copied and check the values that correspond to the names of the files and directories you have created.

l. Now generate a self-signed certificate for our CA using the following command:
openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Fill all the needed information and keep the passphrase you enter in your mind.

m. You have two files generated **ca.key** contains the private key of our CA while **ca.crt** contains the public key certificate. Check the content of each file and explain it.

3.2 Giving Certificates for Clients

Now you are CA root. Assume that a company called **MyCompany.com** would like to get a digital certificate from you as CA.

3.2.1 Key-Pair Generation By the Client

a. The company has first to generate its public-private key pair. Let the encryption algorithm be RSA. To do so, issue the following command:
openssl genrsa -aes128 -out server.key 1024

b. What is the purpose of AES in the previous command?

c. What does 1024 represent in the previous command?

d. Open the file server.key using a text editor. Can you open it? Why?

e. Run the following command and **explain the content of the file server.key**:
openssl rsa -in server.key -text

3.2.2 Generation of Certificate Signing Request (CSR)

When the client generates the key pair, it has to generate a CSR which includes its public key. Once the CA receives the CSR, it confirms the identity information of the CSR.

a. Issue the following command and explain it:
openssl req -new -key server.key -out server.csr -config openssl.cnf

b. Check the content of the file server.csr.

3.2.3 Generating Certificates

The CSR needs to be converted into a certificate. This means, the CSR needs to be signed by the CA.

a. What are the needed files in your current directory to sign the CSR by the CA?

b. Change the following line in openssl.cnf file
policy=policy-match to **policy-anything**
What is the purpose of this modification?

c. Issue the following command and **Show the generated certificates of the client**:
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf

d. Explain the content of **index.txt** and **serial** files.

3.3 Public Key Infrastructure (PKI) for Domains

As mentioned before, let **MyCompany.com** be the domain of the client whose certificate will be used by web browsers to guarantee secure browsing.

a. Create manually an entry for the domain **MyCompany.com** in your Linux machine. Let the IP address mapped to this domain be the local host (127.0.0.1). How will you do that?

b. Combine the secret key and the certificate into one file named **server.pem**. To do so you need to issue the following commands:
cp server.key server.pem
cat server.crt >> server.pem
Carefully explain the above commands? What is the purpose of combining the certificate and the key into one file?

c. Start a web browser using the following command:
openssl s_server -cert server.pem -www.
You will prompted to enter a passphrase several times. Explain why?

d. The default used port number is 4433. Lunch the Firefox and use the following URL:
https://mycompany.com:4433/. What did you get? Explain why?

e. Usually, a CA requests browsing companies (such as Mozilla for Firefox) to include its own certificates. However, as this is not possible here, you will have to manually load your certificate (ca.cert) file into Firefox. **How can you do that?**

f. Now use the URL **https://mycompany.com:4433/** again in your Firefox and state your conclusions.

g. Modify a single byte of the file **server.pem** and restart the server. Reload the URL. What did you notice?

h. Now, restore the original **server.pem** file and restart the server again. Reload the URL and state your conclusions.

3.4 Digital Signatures Creation

a. What is the purpose of digital signatures?

b. Generate an RSA private key using the following command:
openssl genrsa -out key1.private 1024

c. Generate an RSA public key using the following command (from the private key):
openssl rsa -in key1.private -out key1.public -pubout -outform PEM

d. Create a file named **signfile.txt** and fill it with arbitrary text.

e. Generate a hash code using SHA256 for the file **signfile.txt**. How will you do that?

f. Save the generated hash code in a file named **signfile.sha**.

g. Encrypt the hash code using RSA private key using the following command:
openssl pkeyutl -encrypt -inkey key1.private -in signfile.sha > signfile.signature

h. Decrypt the encrypted hash code using the RSA public key.

i. Use the hash code to confirm the integrity of the file **signfile.txt**.

3.5 To DO

a. Prepare a file named **message.txt** that contains a 16-byte message.

b. Prepare a 1024-bit RSA public/private key pair.

c. Encrypt the message using the public key and save the output in **message-enc.txt**.

d. Decrypt the message using the private key.

e. Encrypt the message using 128-bit AES key.

f. Compare the time spent for each operation and state your conclusions.

References

https://seedsecuritylabs.org/