Google IT Support Certificate: Course 5: IT Security - Defense Against the Digital Dark Arts

Week 2: Pelcgbybtl (Cryptography)

Quiz: Creating/inspecting key pair, encrypting/decrypting and sign/verify using OpenSSL

Introduction

In this lab, you'll learn how to generate RSA private and public key pairs using the OpenSSL utility.
OpenSSL is a commercial-grade utility toolkit for Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It’s also a general-purpose cryptography library. OpenSSL is licensed under an Apache-style license, which means that you’re free to get it and use it for commercial and non-commercial purposes (subject to some simple license conditions).

What you’ll do

- **OpenSSL** -You'll explore what generating key pairs looks like using OpenSSL.
- **Encrypt and decrypt** -You’ll use the key pair to encrypt and decrypt some small amounts of data.
- **Verify** -You'll use the key pair to sign and verify data to ensure its accuracy.

Generating keys

Before you can encrypt or decrypt anything, you need a private and a public key, so let's generate those first!

Generating a private key

Remember, a key pair consists of a public key that you can make publicly available, and a private key that you need to keep secret. Shhhh. :) When someone wants to send you data and make sure that no one else can view it, they can encrypt it with your public key. Data that's encrypted with your public key can only be decrypted with your private key, to ensure that only you can view the original data. This is why it's important to keep private keys a secret! If someone else had a copy of your private key, they'd be able to decrypt data that's meant for you. Not good!

First, let's generate a 2048-bit RSA private key, and take a look at it. To generate the key, enter this command into the terminal:

```
openssl genrsa -out private_key.pem 2048
```

Copied!

content_copy

You should see the following output (or something very similar) :

```
Generating RSA private key, 2048 bit long modulus (2 primes)
................+++++
.........................................+++++
e is 65537 (0x010001)
```

This command creates a 2048-bit RSA key, called "private_key.pem". The name of the key is specified after the "-out" flag, and typically ends in ".pem". The number of bits is specified with the last argument. To view your new private key, use "cat" to print it to the screen, just like any other file:

```
cat private_key.pem
```

Copied!

content_copy

The contents of the private key file should look like a large jumble of random characters. This is actually correct, so don't worry about being able to read it:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA4kNMSmssCSYbOnq/UAHGH5xx9gjZaOiST3JQQtJO11L/YeBO
8DOHc7UawNADA/XDBAnGZih1M8T1PGc6Vk5SW2Lb8FMf9zG2XhYpCACFFPJAW00q
s4s1JesdugOprHZ8Jmm/QJl4KuCjlY/XdviCvcbxROIQ2mglR8nW1QWrhECQNBfo
dRSuTwmW3qBSW/Xd5pmTpP4GHCyUfRO9YCF/tZYtVMYg4FOqdGaTHRZbs6peMV4D
lSjZHDonnsGK0UJpxQNbtJEcG7vr7Vl8ziVWY5RUDND7nZYlQlbqxvvqbPPt+px3
4pAZ58eyOqeAmYBc8mwNoXp4YrC2deFng7zrKwIDAQABAoIBAB6SR0Ga33VQ/8bU
BPtzceidg8xhf7asDfDMGkodDmgLn9QCscfEvp2Er9uzf2TOlQ37oCH3f3aCOzxx
GjHFHV2Zquv630vQHLrztZGOOG0PGmD7uTRPL9wyu26BxjA2RioOibfZxKHOfmvb
5pn9k/S+Z6UOAobwIXFktTFNNdKFgalax813FlxFfmmoOC8kE30W6mP6iecP+ojm
xf577RhwR+PdE5zNNvm2F8j5ZWP39pboX7e3eYUCsEyPmVu1MSMTXrHHg6KNhCty
Qu1JfrAaisch+6vrAzfuP7t0WiILzieQgZzFDpI9HziwwOtCw+EKQhHCOPurWcO6
ByZUBzkCgYEA9aEprwqutbXB5H3QinxqXLInAH+wy8oTAMS6nV1sisIos6dD3CLO
u2fLRegv8PEUopASnzyv5PWU/iS+VJjdBCco59hmwW+7CVpaOJXlJ1qpznPVJmyx
pWsinM9Ug23GDd/jd61yKux22773RSGCYs9N7FVww5WYcDlWHLUFPk0CgYEA69DQ
h2iFuDSPonG8GPS6hf/KVRQaJZqGAINCk/2txTWmaz9VPdWT25+rxBzIoQOYAC4P
NjPHo/gJLrO/y6X6lAKBCje/Otb9E7GZwH0pFc7MxtQVR4ik6/7To3ancXNmawHe
owWZHDBRK+Ot33nZ+tYvAq48zE7rxNxsctZ9O1cCgYASsd12UR3S/q5vMZQ5thZy
```

```
T6zgQNe36v1fRZneeEnWlch7Q/PKQWvyn4e9Hlrnv7GOXeDM9dV9W6OnZCyIS8om
ksRuQO4xMsvNfm73d5ElWaUq7W3/qq4qpOjRfoY0Kpq0W6H4bd8OnUi+mN5BCLff
xV9s6WPXvv8HK5X+QVjQ0QKBgBrMqGY7IrdEge5cLpxHc8s2vq/ckPwlC4WTZUWc
VttKtZcKo41bcGpNQyAOhV6HIgcjNOdcCxw/XAvKsclbG5cmkbOvkjQFqs1KKccO
clTgI7WU9LYkeVm4pCS3n1/tVX5jwAGW6Uei1ha+0UvMdVFkdgM/+fjeHz1IL6r9
ZU4RAoGBALi33UjlJUYVMXPZc/JyFk8yyvRpYMRhmW7mQxR8gx0i1rNolPSccRkj
3NO+e1k86yyk3RsqBdixGKYDp2JqS+Aj7eHlxvUcrCAnpk9l96q8yuhQ4mJUWqs7
/hW6bxUPjDZ9BxprGZRL4ZLgPL+6C4Q4rE8TZu/5qQYDIy+ab03t
-----END RSA PRIVATE KEY-----
```

Head's up: Your private key will look similar to this, but it won't be the same. This is super important, because if openssl was generating the same keys over and over, we'd be in serious trouble!

Click *Check my progress* to verify the objective.

Generate private key

Check my progress

Generating a public key

Now, let's generate the public key from the private key, and inspect that one, too. Now that you have a private key, you need to generate a public key that goes along with it. You can give that to anyone who wants to send you encrypted data. When data is hashed using your public key, nobody will be able to decrypt it unless they have your private key. To create a public key based on a private key, enter the command below. You should see the following output:

```
openssl rsa -in private_key.pem -outform PEM -pubout -out public_key.pem
```

Copied!

content_copy

```
writing RSA key
```

You can view the public key in the same way that you viewed the private key. It should look like a bunch of random characters, like the private key, but different and slightly shorter:

```
cat public_key.pem
```

Copied!

content_copy

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4kNMSmssCSYbOnq/UAHG
H5xx9gjZaOiST3JQQtJO11L/YeBO8DOHc7UawNADA/XDBAnGZih1M8T1PGc6Vk5S
W2Lb8FMf9zG2XhYpCACFFPJAW00qs4s1JesdugOprHZ8Jmm/QJl4KuCjlY/XdviC
vcbxROIQ2mglR8nW1QWrhECQNBfodRSuTwmW3qBSW/Xd5pmTpP4GHCyUfRO9YCF/
tZYtVMYg4FOqdGaTHRZbs6peMV4DlSjZHDonnsGK0UJpxQNbtJEcG7vr7Vl8ziVW
Y5RUDND7nZYlQlbqxvvqbPPt+px34pAZ58eyOqeAmYBc8mwNoXp4YrC2deFng7zr
KwIDAQAB
-----END PUBLIC KEY-----
```

Head's up: Like your private key, your public key will look different than the one in this image.

Now that both of your keys have been created, and you can start using them to encrypt and decrypt data. Let's dive in!

Click *Check my progress* to verify the objective.

Generate public key

Check my progress

Encrypting and decrypting

You'll simulate someone encrypting a file using your public key and sending it to you, which allows you (and only you!) to decrypt it using your private key. Similarly, you can encrypt files using other people's public keys, knowing that only they will be able to decrypt them.

You'll create a text file that contains some information you want to protect by encrypting it. Then, you'll encrypt and inspect it. To create the file, enter the command below. It will create a new text file called "secret.txt" which just contains the text, "This is a secret message, for authorized parties only". Feel free to change this message to anything you'd like.

```
echo 'This is a secret message, for authorized parties only' > secret.txt
```

Copied!

content_copy

Then, to encrypt the file using your public key, enter this command:

```
openssl rsautl -encrypt -pubin -inkey public_key.pem -in secret.txt -out
secret.enc
```

Copied!

content_copy

This creates the file "secret.enc", which is an encrypted version of "secret.txt". Notice that if you try to view the contents of the encrypted file, the output is garbled. This is totally normal for encrypted messages because they're not meant to have their contents displayed visually.

Here's an example of what displaying the encrypted file "secret.enc" looks like in the nano editor using the following command below:

```
nano ~/secret.enc
```

Copied!

content_copy

Output:

```
^? <     e ^@vmD  ^B% r*M o^R ^O 8 X  {  ^\(^B  ^}= 1i T 9~ ^RT^\^Px ^T^l
n ^G ^O ^i  iN (W [ ^$
^a^d~m  , d Tq L   < J ^Q bdQ
=Q R[^kT  ^G iq   GG  ^T {  UZ^dV8^A ^~O#koj^N^^ K vT ^O3 ^Tn^oP^l^Pa
^u3^G^N^i0=c{ ^tR09  o@^d$
<br>
<br>
```

```
<br>
<br>
<br>
<br>
<br>
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur
Pos
^X Exit  ^R Read File  ^\ Replace  ^U Uncut Text  ^T To Spell  ^_ Go To
Line
```

To exit from the nano editor, use the command **Ctrl-X**.

The encrypted file will now be ready to send to whoever holds the matching private key. Since that's you, you can decrypt it and get the original contents back. Remember that we must use the private key to decrypt the message, since it was encrypted using the public key. Go ahead and decrypt the file, using this command:

```
openssl rsautl -decrypt -inkey private_key.pem -in secret.enc
```

Copied!

content_copy

This will print the contents of the decrypted file to the screen, which should match the contents of "secret.txt":

```
This is a secret message, for authorized parties only
```

Click *Check my progress* to verify the objective.

Encrypting and decrypting

Check my progress

Creating a hash digest

Now, you'll create a hash digest of the message, then create a digital signature of this digest. Once that's done, you'll verify the signature of the digest. This allows you to ensure that your message wasn't modified or forged. If the message was modified, the hash would be different from the signed one, and the verification would fail.

To create a hash digest of the message, enter this command:

```
openssl dgst -sha256 -sign private_key.pem -out secret.txt.sha256
secret.txt
```

Copied!

content_copy

This creates a file called "secret.txt.sha256" using your private key, which contains the hash digest of your secret text file.

With this file, anyone can use your public key and the hash digest to verify that the file hasn't been modified since you created and hashed it. To perform this verification, enter this command:

```
openssl dgst -sha256 -verify public_key.pem -signature secret.txt.sha256
secret.txt
```

Copied!

content_copy

This should show the following output, indicating that the verification was successful and the file hasn't been modified by a malicious third party:

```
Verified OK
```

If any other output was shown, it would indicate that the contents of the file had been changed, and it's likely no longer safe.

Click *Check my progress* to verify the objective.

Sign and verify

Check my progress

Conclusion

Wohoo! You've successfully used openssl to create both a public and a private key. You used them to practice file encryption and decryption, and to create and verify digital hashes.