Create/inspect key pair, encrypt/decrypt and sign/verify using OpenSSL

1 hour 1 Credit

Rate Lab

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).

You'll access OpenSSL through SSH onto the Linux VM instance associated with your lab.

What you'll do

- **OpenSSL:** You'll explore what generating key pairs looks like using OpenSSL by using SSH to access the Linux instance.
- **Encrypt and decrypt:** You'll use the key pair to encrypt and decrypt some small amount of data.
- **Verify:** You'll use the key pair to sign and verify data to ensure its accuracy.

Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green "Start Lab" button at the top of the screen.

Note: For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

After you click the "Start Lab" button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:

Accessing the virtual machine

Please find one of the three relevant options below based on your device's operating system.

Note: Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you'll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM's External IP address to connect.

Download your PPK key file

You can download the VM's private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.

Connect to your VM using SSH and PuTTY

1. You can download Putty from here

2. In the **Host Name (or IP address)** box, enter username@external_ip_address.

Note: Replace **username** and **external_ip_address** with values provided in the lab.
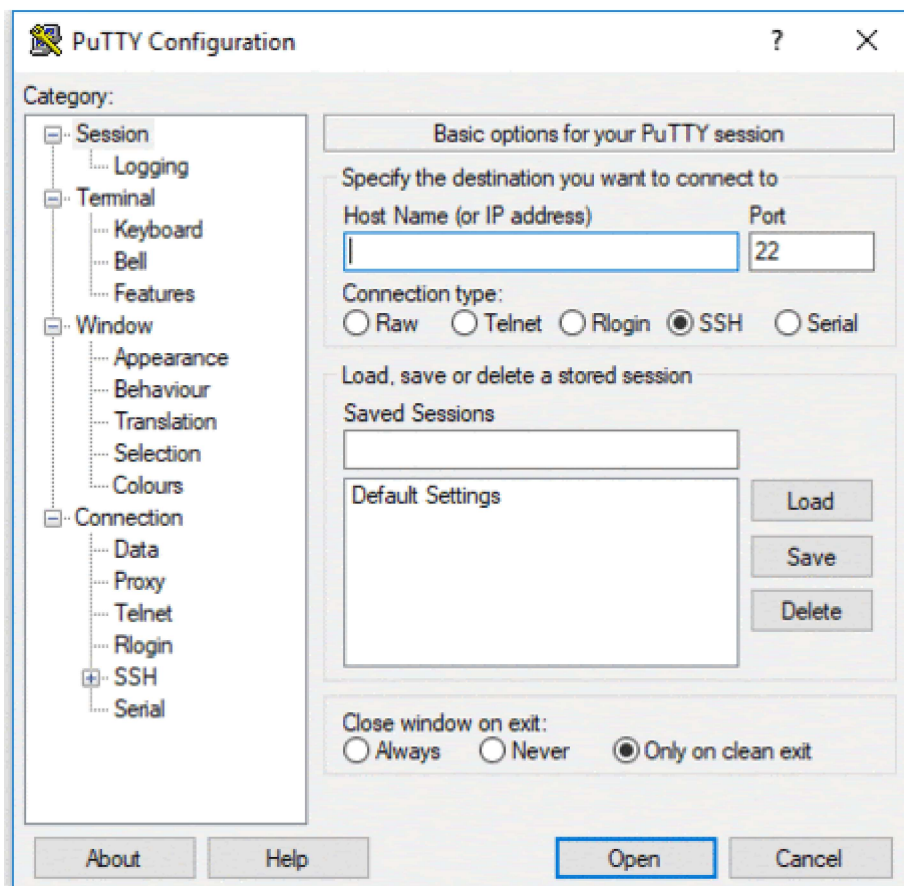
3. In the **Category** list, expand **SSH**.

4. Click **Auth** (don't expand it).

5. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.

6. Click on the **Open** button.

Note: PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.
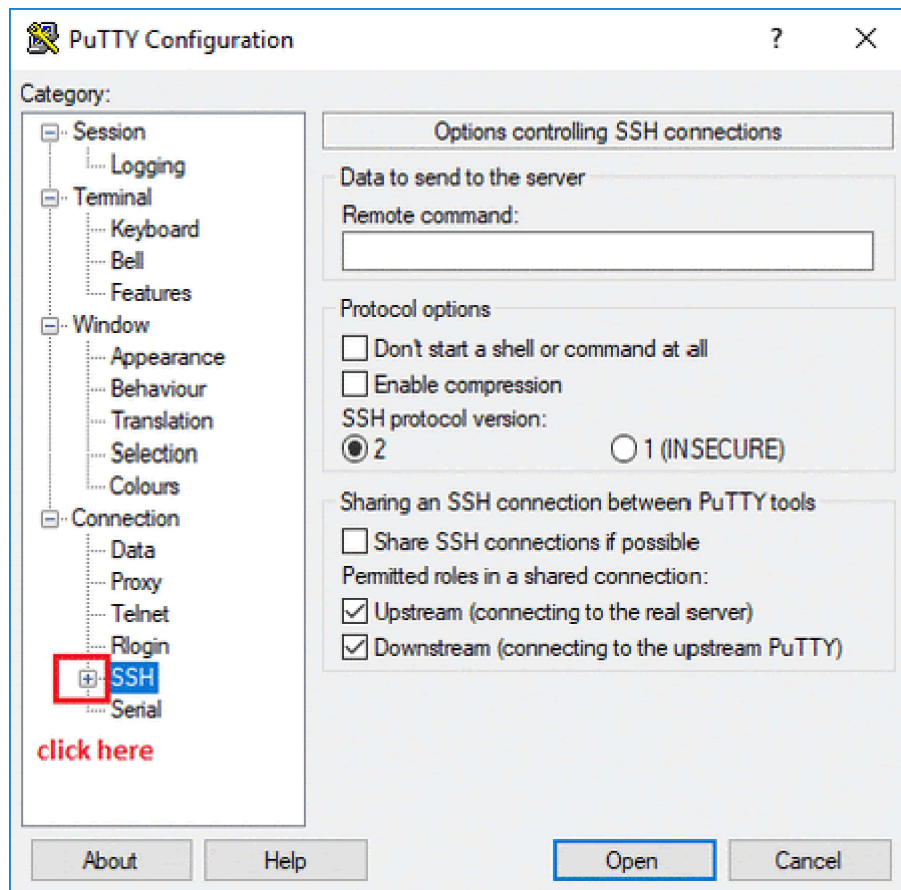
7. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

Common issues

If PuTTY fails to connect to your Linux VM, verify that:

- You entered **<username>@<external ip address>** in PuTTY.

- You downloaded the fresh new PPK file for this lab from Qwiklabs.

- You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

Download your VM's private key file.

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.
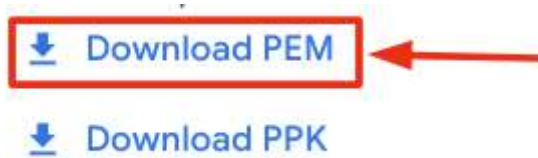
Connect to the VM using the local Terminal application

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.

   - To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.

   - To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.

2. Enter the following commands.

Note: Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.
You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

```
chmod 600 ~/Downloads/qwikLABS-XXXXX.pem

ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address
```


Option 3: Chrome OS users: Connecting to your VM via SSH

Note: Make sure you are not in **Incognito/Private mode** while launching the application.
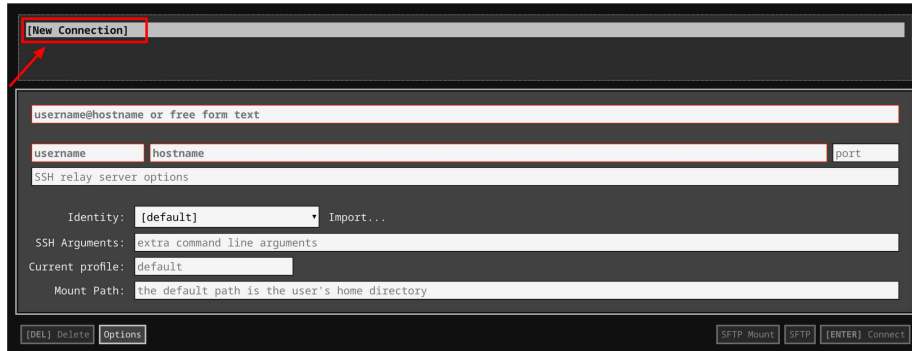**Download your VM's private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.
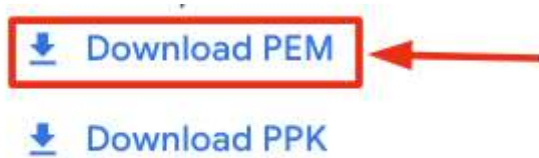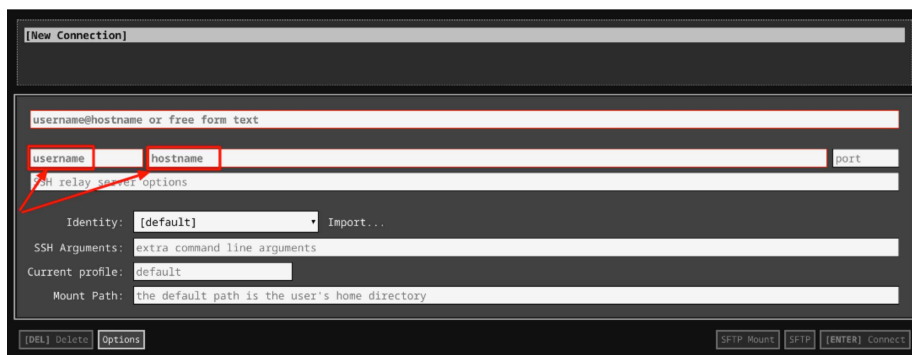
Connect to your VM

1. Add Secure Shell from here to your Chrome browser.

2. Open the Secure Shell app and click on **[New Connection]**.

3. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.

4. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

Note: If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

5. Once your key is uploaded, click on the **[ENTER] Connect** button below.

6. For any prompts, type **yes** to continue.

7. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

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
```

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

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
```

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:

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

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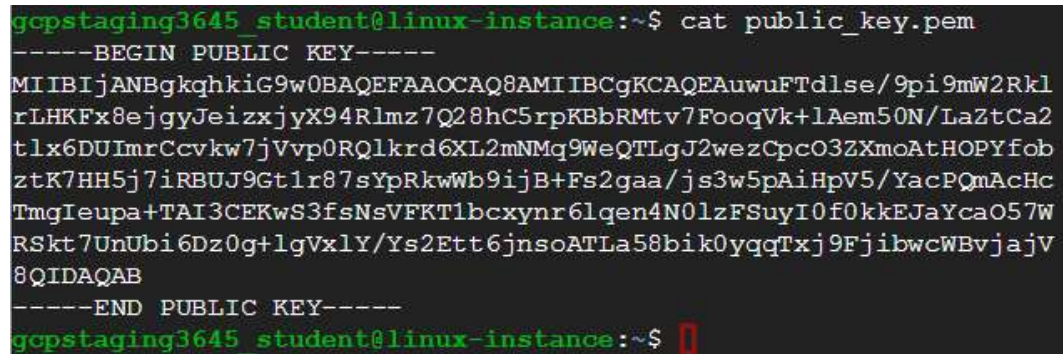
```
openssl rsa -in private_key.pem -outform PEM -pubout -out public_key.pem
```

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
```


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

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
```

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
```

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 may look like:

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
```

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

Click *Check my progress* to verify the objective.

Encrypting and decrypting

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
```

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
```

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

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

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.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.