📬 Submit this assignment by **Monday, March 6th at 11:59pm PST** using the *Submit* button 👉

## **Project 2: All About SSH Keys**

### **Overview**

In this week's project, you will generate SSH keys and learn how to use them for authentication, encryption and integrity purposes.

### **🎯 Goals**

By the end of this assignment you will be able to...

* Generate SSH keys and use them for authentication.
* Use SSH keys and openssl to encrypt and decrypt a textfile.
* Verify the integrity of a Git commit using SSH

### **📬 What You'll Turn In**

For this assignment, you'll be filling and submitting a copy of the [📄 **Project 2 Submission Template** (Google Doc)](https://docs.google.com/document/d/1yJ9Bc-gj_TKe2oiMSLEZ-r3L6XYmNofQFxvtW8xYLSM/copy)

* Before proceeding, we recommend you **open it up now** and read over the requirements in the document.
* It might be easier to "fill-as-you-go" than try to fill it all out after you complete the project.

In addition, you'll also be turning in **screenshots** of your kali terminal at various points in the task.

* To help you keep track of these, we'll put **📸 SCREENSHOT** tags throughout the instructions at the appropriate times.

Required Challenge

*(Detailed instructions for performing these tasks are provided below.)*

*To receive full credit, you must turn in..*

* **Screenshot showing openssl encrypt/decryption (Part 3), including:**
  + The appropriate encryption command
  + The appropriate decryption command
  + The contents of all 3 files: original, encrypted, decrypted
* **Screenshot showing Github commit signing (Part 4), including:**
  + The git commit command and its' output
  + The git show --show-signature command and its' output, showing a successful signature

You are **not** required to show:

* Installation of tools
* SSH key creation
* Process of obtaining access to Kali

Close Section

Stretch Challenge

*To receive bonus points, you can submit...*

* **A Screenshot showing an additional use of SSH keys**
  + Use the internet and see what else you can find!
* **A description showing an additional use of SSH keys**
  + What's the *wackiest* use of an SSH key out there?

Close Section

## **Project Instructions**

### **Part 1: Getting an SSH Key**

Your first goal is to create a new SSH key pair.

* Open a Kali terminal session using either ssh, or rdp then open a Console window in Kali

We'll use the ssh-keygen command. There are a lot of different options, but for today we'll generate 4096-bit SSH RSA public and private keys.

* Run ssh-keygen -t rsa -b 4096 -C "cyberstudent@kali.local"

**Commands breakdown:**

* ssh-keygen = the utility used to generate the key pair
* -t rsa = specifies the **t**ype of key to create (e.g., rsa)
* -C "cyberstudent@kali.local" = provides custom key comment (which will be appended at the end of the public key). This will be used for easy identification. Usually an email address is used as a comment, but you are free to use whatever works for you.
  + Think of it like a label for your key, so you don't mix it up with other keys!

After executing the command above, and the following message will appear:

┌──(kali㉿kali)-[~]

└─$ **ssh-keygen** -t rsa -b 4096 -C "cyberstudent@kali.local"

Generating public/private rsa key pair.

Afterwards, you will be prompted to enter the path to the file where you want the key saved. You can specify a specific file location or accept the default location by pressing Enter.

* Note: If you want your keyfile to have a different **name**, you can also specify it here. For example, if we wanted the key to be named id\_rsa\_cyb101:

Enter file in which to save the key (/home/kali/.ssh/id\_rsa): /home/kali/.ssh/id\_rsa\_cyb101

Once you press Enter, you will be ask to enter a passphrase.

* Adding a passphrase is *optional*.
* However, adding the passphrase prevents unauthorized use if someone gets access to your computer.
* For this project, you can just press Enter to proceed with no passphrase.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Now you should get a message confirming that your key pair has been created!

Your identification has been saved in /home/kali/.ssh/id\_rsa\_cyb101

Your public key has been saved in /home/kali/.ssh/id\_rsa\_cyb101.pub

🤔 Which key is which?

* An SSH key pair that consists of a public key and a private key.
  + The public key will have the .pub extension, and the private key will have no extension.
  + In the above example, the SSH private key is named id\_rsa\_cyb101, and the public key is named id\_rsa\_cyb101.pub.
* Sometimes, private keys get a .pem extension. We saw that in this week's lab with our ssh key!

You've created your keys! Go ahead and try previewing them with cat:

┌──**(**kali㉿kali**)**-[~]

└─$ cat ~/.ssh/id\_rsa\_cyb101

-----BEGIN OPENSSH PRIVATE KEY-----

b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAACFwAAAAdzc2gtcn

... truncated **for** space

┌──**(**kali㉿kali**)**-[~]

└─$ cat ~/.ssh/id\_rsa\_cyb101.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC5OSa7cKQQg4rWohWVYobfVUhilcojgydrxTOtzRDrYm/W+K1QrndFjxL6CH9JTFPapqkZDk92lDXvrn+qQKNaBkzNDjTyTKJp2cLd5eGIax2CWdSqgZhtjo+kAeB2wnEPU6C2rPqw5EVveATYoPfUBVTtfZt2PsxZGAVp

... shortened **for** space

QbFREW0tL05V8T2YHBtvoCeoSlQ1NjMLWvKQ**==** cyberstudent@kali.local

**Checkpoint 1**: You should have created a public/private key pair and be able to view them!

Next, we'll go over each of the three main uses for SSH keys: authentication, encryption and integrity.

### **Part 2: Authentication with SSH Keys**

You've actually already used an SSH key for authentication! That's how you used SSH to access the Kali terminal during the lab. Remember this command?

ssh -i ~/Downloads/azure\_kali\_key.pem kali@203.0.113.0

When we ran the above command, we used the azure\_kali\_key.pem private key to prove we were who we claimed to be!

**Checkpoint 2**: Look back at the lab and see if you can find where we authenticated with ssh!

### **Part 3: Encryption with SSH Keys**

We can use SSH keys for **asymmetric encryption**, where:

* The **public key** is used to encrypt a message
* The **private key** is used to decrypt it

This is powerful stuff! In the real world, asymmetric encryption lets us make sure we send our message to **the right person**. Consider the following example:

* You need to send your friend the Minecraft server password, but don't want to just send it unsecured -- anyone who accesses your network traffic might gain access!
* Instead, you use your friend's **public key** to encrypt the password. (Because it's a *public* key, your friend can safely share it with you.)
* Now, your friend is the only one who can decrypt that message -- because they're the only one with their **private key**!

#### **Trying it Out**

Open a Kali terminal, and let's generate a new pair of keys. (We're going to use openssl this time, because it comes with built-in cryptography functions.)

* Run the following commands:

openssl genrsa -out ~/.ssh/privatekey.pem 2048

openssl rsa -in ~/.ssh/privatekey.pem -out ~/.ssh/publickey.pem -pubout -outform PEM

Great. You can use ls ~/.ssh to view your .ssh folder's contents -- it should have two new keys now.

* Next, let's create a secret message file called secret.txt. You can use any message you like:
* Run echo "MY SECRET MESSAGE" > secret.txt, then cat secret.txt to view the result.

┌──(kali㉿kali)-[~]

└─$ **echo** "My secret code is CHEESE" > secret.txt

┌──(kali㉿kali)-[~]

└─$ **cat** secret.txt

My secret code is CHEESE

Okay, we've set everything up. Now let's use openssl to encrypt the file with publickey.pem.

* Run the following:

openssl pkeyutl -encrypt -pubin -inkey ~/.ssh/publickey.pem -in secret.txt -out secret.txt.encrypted

* Check out the results with cat secret.txt.encrypted -- it should print out a bunch of gibberish, because it's encrypted!

Now let's see if we can decrypt it using privatekey.pem.

* Run:

openssl pkeyutl -decrypt -inkey ~/.ssh/privatekey.pem -in secret.txt.encrypted -out secret.txt.decrypted

Did it work? Let's run cat secret.txt.decrypted to find out:

┌──(kali㉿kali)-[~]

└─$ **cat** secret.txt.decrypted

My secret code is CHEESE

**🎯 Checkpoint 3**: At this point, you've encrypted and decrypted a text file using an SSH key pair! Awesome!

Take a **📸 SCREENSHOT** of your terminal at this point -- you'll need to turn it in for this assignment!

### **Part 4: Integrity with SSH Keys**

The final use-case of SSH keys is **integrity** -- proving that a file hasn't been tampered with! For this case, we're going to try signing Git commits.

Git is an extremely common version control system that programmers across the world use to collect and share their code. As such, it's pretty important to make sure your code hasn't been illicitly modified! We can do that by a process called **signing**.

Signing is different from encryption because we don't hide the contents of the file. Instead, we add a **signature** to the file that's unique to that specific file, and **calculated based on a key**. If you try again with the same key and the same file, you should get the same result -- proving the file's validity. If even ONE byte of the file is modified, you'll get a different signature!

##### **Step 1: Setting up Git**

First, we'll need to install Git:

┌──(kali㉿kali)-[~]

└─$ **sudo** apt-get install git -y

Next, we'll need to set up a demo Git repository (project):

* First, make a folder for the project and change directory into that folder:

┌──(kali㉿kali)-[~]

└─$ **mkdir** ~/DemoProject

┌──(kali㉿kali)-[~]

└─$ **cd** ~/DemoProject

┌──(kali㉿kali)-[~/DemoProject]

└─$

The Kali prompt should update to show your new location. Now let's initialize an empty Git repository:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **git** init

Initialized empty Git repository in /home/kali/DemoProject/.git/

##### **Step 2: Configuring Git**

Awesome! Now it's time to **configure Git**!

* Run the following commands. You can use any name and (fake) email you like!

git config --global user.email "CYB101EMAIL"

git config --global user.name "CYB101NAME"

💡 TIP: In real-world development, configuring your Git username and email address is important as every Git commit will use these information to identify you as the author. Since you are using the --global option, Git will always use this information for anything you do on that system.

Next, we need to enable GPG signing and set the format to ssh.

* Execute the following commands:

git config --global commit.gpgsign true

git config --global gpg.format ssh

Great. We've told Git we want to sign commits... now let's give it our public key.

##### **Step 3: Registering our Key**

To help us register our key to Git, we'll use a tool called ssh-agent.

* Make sure it's active by running:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **eval** `ssh-agent -s`

Next, add your key pair to the agent using ssh-add:

* Run ssh-add ~/.ssh/id\_rsa\_cyb101 (or whatever you named your public key in the first part!)

You should see output like this:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **ssh-add** ~/.ssh/id\_rsa\_cyb101

Identity added: /home/kali/.ssh/id\_rsa\_cyb101 (cyberstudent@kali.local)

Note: If you created your SSH key with a passphrase, you'll have to enter it for the above command to work!

Once successful, you should see a confirmation that the identity has been added:

Identity added: /home/kali/.ssh/id\_rsa\_cyb101 (cyberstudent@kali.local)

Run the following command to retrieve your SSH public key string:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **ssh-add** -L

ssh-rsa AAAA...WvKQ== cyberstudent@kali.local

Copy the result -- your public key, including the "ssh-rsa" and your email at the end!

* Set your signing key to the copied value:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **git** config --global user.signingkey "ssh-rsa AAAA....WvKQ== cyberstudent@kali.local"

Whew! This is a lot! But you're almost done! We just need to tell Github that this is a valid key..

* Run the following command to create a file called ~/.ssh/git\_allowed\_signers

┌──(kali㉿kali)-[~/DemoProject]

└─$ **echo** -n "cyberstudent@kali.local " > ~/.ssh/git\_allowed\_signers && ssh-add -L >> ~/.ssh/git\_allowed\_signers

Then add it to Github's configuration:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **git** config --global gpg.ssh.allowedSignersFile ~/.ssh/git\_allowed\_signers

##### **Step 4: The Moment of Truth**

Now it's time to sign a commit!

* Run the following command with a message of your choice:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **git** commit --allow-empty --message="Did the SSH signing work?"

You should see a message like this after running the command above:

[master (root-commit) 691414d] Did the SSH signing work?

That looks promising!

* Verify that it worked by using --show-signature:

┌──(kali㉿kali)-[~/DemoProject]

└─$ **git** show --show-signature

The output should look something like this:

commit 691414d8f6ad836321812522085a1cdf3fd918b8 (HEAD -> master)

Good "git" signature with RSA key <KEY>

Author: CYB101NAME <CYB101EMAIL>

Date: Tue Dec 6 21:10:09 2022 +0000

Did the SSH signing work?

🎉 Congratulations, you now have SSH signing enabled for your Git commits! 🎉

Take a **📸 SCREENSHOT** of your terminal at this point -- you'll need to turn it in for this assignment!

💡 Hints

* I think I might have messed up Git! Help!
  + Use the git config --edit and git config --global --edit commands to see what settings you currently have for Git.
  + If you have to, clear these files out and start the section over.

Close Section

### **📬 Submitting Your Project**

* [📄 **Project 2 Submission Template** (Google Doc)](https://docs.google.com/document/d/1yJ9Bc-gj_TKe2oiMSLEZ-r3L6XYmNofQFxvtW8xYLSM/copy)

✔ Am I Ready to Submit?

Check if you're **ready to submit** with the following questions:

* Did you complete all of the **Required Challenges**?
* Did you copy and fill out the [**Project 2 Submission Template**](https://docs.google.com/document/d/1yJ9Bc-gj_TKe2oiMSLEZ-r3L6XYmNofQFxvtW8xYLSM/copy)?
  + It is important that you follow the same layout as the template so that we can easily access your work.
  + Be sure to check off each feature that is implemented in the **"Submission Checklist"** section
* Are any required images/GIFs correctly displaying in your document?
* Did you set your document to ***"Anyone with the link can Edit"***?

If you answered **yes** to **all** of these questions, you are ready to submit!

Look for the **"Submit"** button at the top of this page.

Close Section

📣 **Late Submissions**

* We highly encourage you submit your project in any state (even if it is not done) by the deadline: **Monday, March 6th at 11:59pm PST**.
* You can continue to work on your project with our **48-hour extension** in which your project will be graded once more once the extension deadline has passed.
* Don't forget to **resubmit through the course portal** with your **updated document link**!