CprE 308 Laboratory 8: Introduction to SSH and GnuPG

Department of Electrical and Computer Engineering
Iowa State University

1 Submission

Take the Lab 8 quiz on Canvas.

2 Resources

Read the manual pages about SSH and GnuPG by typing the following command in your terminal.

```
man ssh
man scp
man ssh-keygen
man ssh-agent
man gpg
```

ISU Linux Remote Servers:

```
linux-1.ece.iastate.edu
```

You'll need a terminal emulator.

- Windows: PuTTY (https://www.putty.org/)
- macOS: Terminal (built-in)
- Linux: Terminal (built-in)

3 Instructions

In this lab, you will become familiar with the use of ssh (secure shell) and gpg (GNU privacy guard).

3.1 SSH

SSH, short for Secure Shell, is a protocol used to authenticate, encrypt, and digest data transmitted over a network. SSH is used for a variety of tasks related to OSes. The SSH system is composed of two parts: the server and the client. The SSH server handles the incoming request from the SSH client. The SSH client sends the connection request to the server. SSH creates a secure channel between two computers. Any data transferred on the channel is encrypted and authenticated. SSH typically uses TCP port 22.

3.1.1 Logging in to a remote server

Suppose you are out of town on an interview in California. But you forget to bring the CprE 308 project documents which you want to show to the interviewer. You left the project documents on a machine that is running an SSH server. You try to access these files using SSH.

1. To access ISU servers, you will first need to connect to ISU's virtual private network (VPN). You do this by using a VPN client (Cisco AnyConnect) to connect to ISU's VPN server (vpn.iastate.edu). When you do this, your computer effectively joins ISU's internal network, and (at least some of) your network traffic is routed through ISU's network. If you don't have AnyConnect installed, visit vpn.iastate.edu with a web browser to download it. Otherwise, open AnyConnect and connect to vpn.iastate.edu.
2. If you're using macOS or Linux (or a Windows-based shell), open a shell and run the command: `ssh linux-1.ece.iastate.edu`.
3. If you're using PuTTY, open PuTTY and enter `linux-2.ece.iastate.edu` as the Host Name, and choose SSH as the connection type. Then from linux-2, you can ssh to linux-1 using the above command. If you do this, then for the remainder of the lab, your "local machine" will be linux-2, and the "remote machine" will be linux-1.
4. The above command tries to log in to the remote server using the local machine's user name. It will not work if the local user name does not exist on the remote machine, e.g., when you are using your personal PC. If you successfully connected above, use `exit` to disconnect (this ends the remote shell process). Now try using the command `ssh <username>@linux-1.ece.iastate.edu` to log in the remote server, where `<username>` is your user name on the *remote* machine.
5. Once you are connected, run `ls` to check the directories. Run `hostname` to view the hostname. Which machine is the output coming from, your local machine or the remote machine?
6. On the remote machine, run `ps -ef | grep sshd`. One of the results will be the SSH daemon process (e.g. /usr/sbin/sshd). This is the SSH server process, which listens for incoming SSH connections. This is an important point: **the SSH *server* is simply a process that runs on the machine you want to connect to**. Side note: on Linux systems that use systemd for system management, you can also run `systemctl status sshd` to view the status of the SSH server process.

3.1.2 Secure file transfer

Suppose that while browsing your remote files, you find a file you would like to grab. One option for copying the file from the remote machine to your local machine (or vice versa) is `scp`, a secure copy tool that uses the SSH protocol.

1. Read up on `scp` and try using it to copy a file from the remote machine to your local machine.
2. Now use `scp` to copy a file from your local machine to the remote machine.
3. Another tool that uses SSH to perform secure file copies is `rsync`. Read up on `rsync`. How is it different than `scp`?
4. Use `rsync -av` to copy a file from the remote machine to your local machine. Then run the exact same command to copy the file again. How many bytes were copied the first time? How many bytes were copied the second time?

3.1.3 SSH escape sequences

In the above procedure, we have to log out of the SSH session if we want to do something on the local machine. But what if we do not want to log out? SSH supports escape sequences to manipulate your SSH client process. By default, the escape character is the tilde ~.

Log in to the remote server via SSH, then press ~(tilde) and ?(question mark) to see all the supported escape sequences. Next type the escape character ~and `CTRL+Z` to suspend the connection. Now use `scp` to fetch a file from `linux-1`. Then type '`fg`' to return to the SSH session. Use `hostname` to verify which machine you're on.

3.1.4 Known Hosts

Two people who do not know each other generally will not trust each other. The same is true for two machines. When an SSH client connects to a server for the first time, it will realize that it has never connected to that machine before. In this case, it throws a warning: "I've never seen this machine before!" How does it know? Each SSH server has a *public key* that identifies it to clients. Typically, the first time you connect to a machine, the client warns you that the server's key is unrecognized. If you continue, the server's key is added to your "known hosts" file (`/home/<username>/.ssh/known_hosts`), and you will not receive any warnings in the future.

What's the point of this system? Assuming you actually reached the intended server the first time, then this ensures that you always reach the intended server, protecting against *man-in-the-middle attacks*. If you connected to `linux-1` yesterday with no problem, but then today it throws a unknown host warning, it's possible that another, untrustworthy machine is pretending to be `linux-1`! Of course, it's also possible that your `known_hosts` file got wiped out, or that the sysadmin changed the SSH public key for `linux-1`, or something equally boring happened.

Use `cat` to print out the contents of `/home/<username>/.ssh/known_hosts`. You should see an entry for `linux-1.ece.iastate.edu`. If you're feeling bold, edit the file (It's just an ASCII text file! Welcome to Linux!) and delete the entry. Then try SSHing to `linux-1` again. Do you get a warning?

3.1.5 Cryptographic Keys

Above we talked about how a *public key* uniquely identifies a machine. In fact, each server host has a pair of cryptographic keys—one public, and one private. Since the public key identifies a machine to other machines, then it is not secret—it's shared with a client when the client connects. In contrast, the *private*

key is used to decode encrypted information or produce a digital signature. Therefore, the private key is known only by its owner.

As a user, you can generate your own cryptographic keys and use them for authentication. You must first generate a public/private key pair for yourself.

1. On your local machine, run command `ssh-keygen` to generate a SHA-2 (Secure Hash Algorithm 2) key pair with 512-bit hash values. This command, from the OpenSSH package, can create keys using a variety of cryptographic algorithms, but this is currently the default setting. Under the .ssh directory, which file contains your new private key? Which file contains your public key?
2. When you create the key pair, you are asked to enter a passphrase. What is the difference between this passphrase and a password? What if you forget your passphrase? (HINT: read `man ssh-keygen`)
3. Which file is the passphrase used to encrypt? Why isn't the other one encrypted with the passphrase?

3.1.6 Key-based Authentication

The above section may seem confusing at first. You were able to log in before just fine—why did we generate a personal public/private key pair?

Until now, we've been using password-based authentication, meaning that you log in with a username/password combination. (On ISU servers, you may not need to enter a password due to the user of single sign-on, or SSO.) Now that you've used `ssh-keygen` to generate your own key pairs, you will be able to use key-based authentication.

After creating the user key pair on the local machine, you must install your public key to your account on the remote host. A remote account may have many public keys installed for accessing it in various ways.

- Use a secure copy command to copy your public key to your home directory on the remote machine. (If you're using linux-2 as the local host, it will already be there because your home directory is the same on both machines.)
- On the remote machine, concatenate your public key to the end of `~/.ssh/authorized_keys` using a command similar to the following: `cat mykey.pub >> ~/.ssh/authorized_keys`
- Disconnect from the remote machine and try connecting again. You should now be prompted to enter the passphrase for your private key.

What's the advantage of this method? Password-based authentication may be considered insecure because it allows anyone who knows a user's password to log in. With key-based authentication, the attacker would instead need to have the user's private key and its passphrase. Additionally, decrypting the key does not send the passphrase over the network, unlike entering a password.

However, it should be noted that if an attacker gains access to the user's private key, then the attacker can crack the key's password offline, which is much easier to do and harder to detect than cracking a password through repeated attempts. Therefore, it's very important to protect your private key file! Do an `ls -l ~/.ssh/`. You should see that only you, the owner, have read or write access to your private key file. If that's not the case, use `chmod 600` on the file to lock it down!

3.2 GnuPG

In this part, you will learn about GNU Privacy Guard (GnuPG or GPG), which can be used to encrypt and sign documents or messages on the Internet.

3.2.1 Generate a key pair

The first step to use GnuPG is to generate a key pair. For example:

```
bash-4.2$ gpg --gen-key
gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
```

There are several options. The default is fine.

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

You must also choose a key size. Follow your heart, but don't go less than the default size. The longer the key, the more secure it is. For most applications, the default key size is adequate. The larger the key, the harder it is to crack, but also the longer it takes to encrypt and decrypt using the key. A larger key size may also affect signature length. You cannot change the size of an existing key, and remember that it gets easier for attackers to crack longer keys over time.

Finally, you must choose an expiration date.

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
```

For most users, a key that does not expire is adequate. The expiration time should be chosen with care, however, since although it is possible to change the expiration date after the key is created, it may be difficult to communicate a change to users who have your public key. But if you change keys regularly, it will be more difficult to attack your key.

```
GnuPG needs to construct a user ID to identify your key.

Real name:
```

GPG constructs a user ID for your key using your name, your email address, and a comment provided by you. Follow the prompts. Note that this is in no way legally binding, but if you want to use it for real, you should probably use real values. Most of the point of the key is to have way to convince other people that you're really you.

Finally, you are asked to enter a passphrase. As with the SSH keys, this is used to encrypt the private key.

3.2.2 Exchanging keys

The remainder of this lab is intended to be completed with a partner; however, you can also simply generate another set of keys using GPG and perform the steps using your two different "identities."

Listing keys: To communicate with others you must exchange public keys. To list the keys on your public key ring use the command-line option `--list-keys`.

```
$ gpg --list-keys
```

Exporting a public key: To send your public key to a correspondent you must first export it. The command-line option `--export` is used to do this. It takes an additional argument identifying the public key to export—this can be the email address or part of the name. The key is exported in a binary format.

```
$ gpg --output <file> --export <identity>
```

It can be inconvenient when the binary key is to be sent through email or published on a web page. GnuPG therefore supports a command-line option `--armor` that causes output to be generated in an ASCII-armored format similar to unencoded documents. In general, any output from GnuPG, e.g., keys, encrypted documents, and signatures, can be ASCII-armored by adding the `--armor` option.

Importing a public key: Your partner's public key may be added to your public keyring with the `--import` option.

```
$ gpg --import <file>
```

3.2.3 Encrypting and decrypting documents (10 pts)

To encrypt a document, the option `--encrypt` is used. You must have the public keys of the intended recipients. The software expects the name of the document to encrypt as input; if omitted, it reads standard input. The encrypted result is placed on standard output or as specified using the option `--output`.

```
$ gpg --recipient <partner user id> --output <output file> --encrypt <input file>
```

If you're sending to yourself, you may also need to specify the secret key with `--local-user`:

```
$ gpg --local-user <your user id> --recipient <partner user id>
        --output <output file> --encrypt <input file>
```

The `--recipient` option is used once for each recipient and takes an extra argument specifying the public key to which the document should be encrypted.

To decrypt a message the option `--decrypt` is used. You need the private key to which the message was encrypted. Similar to the encryption process, the document to decrypt is input, and the decrypted result is output.

```
$ gpg --decrypt <encrypted file>
```

Test out encrypting a file, sending it your partner, and having your partner decrypt it.

3.2.4 Making and verifying signatures

The command-line option `--sign` is used to make a digital signature. The document to sign is input, and the signed (and encrypted) document is output.

```
$ gpg --output <output file> --sign <input file>
```

Given a signed document, you can either check the signature, or check the signature and decrypt the document. To check the signature use the `--verify` option.

```
$ gpg --verify <signed file>
```

You can also verify and extract the document at the same time using the `--decrypt` option as before.