CprE 308 Laboratory 9: Introduction to SSH and GnuPG

Department of Electrical and Computer Engineering
Iowa State University

1 Submission

Submit a report in DOC or PDF format on Canvas including the following items:

- **(10 pts)** A cohesive summary of what you learned through the various experiments performed in this lab. This should be no more than two paragraphs.
- **(80 pts)** A write-up of each experiment in the lab. Each experiment has a list of items you need to include. If output was required, you can use either screen capture or copy-and-paste text from the terminal

Attendance takes 10 points. **Due Date**: One week after your lab session.

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:

```
linuxremote1.engineering.iastate.edu (1,2,4,5 are all valid)
linux-1.ece.iastate.edu (1 through 6 are all valid)
```

3 Instructions

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

3.1 SSH (50 pts)

SSH, short for Secure Shell, is a protocol used to authenticate, encrypt, and digest data transmitted over a network. 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.

3.1.1 Logging in to a remote server (5 pts)

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 which is a SSH server. You try to access these files using SSH.

(1 pt) Open shell and run the command: `ssh linuxremote1.engineering.iastate.edu`. Copy the command line and output into the lab report. Run "ls" to check the directories. Which machine is the output information coming from, your local machine or the remote machine? (You can exit the remote shell by entering `exit`)

The above command tries to log in to the remote serving using the local machine's user name. It will not work if the local user name does not exist on the remote server, e.g., when you are using your personal PC.

(2 pts) Use command `ssh -l <username> <address>`" to log in the remote server. Include the command you used into the lab report.

(2 pts) Use command `"ssh <username>@<address>"` to log in the remote server. Include the command you used the lab report.

3.1.2 Secure file transfer (5 pts)

Suppose that while browsing your remote files you encounter a file you would like to grab. Another ssh-like client program, `scp`, is used to copy the file across the network via a secure channel.

(2 pts) From your local shell, use `scp` to fetch a file from the remote server to your local machine. Copy your command into the lab report.

(3 pts) Now use your partner's machine to fetch a file from the remote server to your partner's machine. Copy your command into lab report.

3.1.3 SSH escape sequences (5 pts)

In the above procedure, we log out the ssh session then we use scp to fetch a file. But what if we do not want to log out? SSH supports escape sequences. By default, the escape character is the tilde ~.

(5 pts) Log in to the remote server via ssh, 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 `linuxremote1`. Then type '`fg`' to return to the SSH session. Copy the three commands you used into lab report.

3.1.4 Known Hosts (5 pts)

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. Here is the question: how do you know after running command `ssh linuxremote1.engineering.iastate.edu` successfully, theastate.edu again. Do you need to input a pass- word this time? No. You need only input the passphrase. On the surface, the only difference is that you provide the passphrase to your private key, instead machine you logged in is really `linuxremote1.engineering.iastate.edu`?

Each machine has a unique identity. The identity is a host key pair. One is the public key; the other is the private key. The public key is used by the machine to identify itself to other machines. When a client connects to a server, both machines will send their public key to each other. Likewise, the question still exists: how does the client know that the machine which sends the public key is the one it wants to connect? It is a good idea to record the server's public key before connecting to it. Here comes a role for the administrator. He can maintain a known-host list for all the machines in the system. But what if the system is too big, say all the machines in the world? Some protocols such as secure DNS may be the answer. In this lab, we assume that we can trust the public key which the machine sends for the first time, i.e. we assume that a man-in-the-middle attack is not possible the first time we connect to a server.

The first time a client connects to the server, the public key sent by the server is stored in your local account on the machine you used. After that, SSH will check the stored public key with the one sent by the server. Meanwhile, the server will also store the public key of the client. In the future, SSH will check the public key stored with the one sent by the client.

(5 pts) Under your local home directory there is a directory called "`.ssh`". In this directory, you will find a file named **known_hosts**. Open the file and you will find that it records several machines' public keys. Copy the name of a machine and its IP address into lab report.

3.1.5 Cryptographic keys (10 pts)

As we stated above, each server and client has a key pair which is called a public/private key pair. The public key is used to identify the server or client itself. The private key is used to decode encrypted information or produce a digital signature. To use cryptographic authentication, you must first generate a key pair for yourself, consisting of a private key and a public key.

(4 pts) On your local machine, run command `ssh-keygen -t dsa` to generate an DSA key pair. Under the .ssh directory, which file contains your private key? Which file contains your public key?

(3 pts) 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 pts) Which file is the passphrase used to encrypt? Why isn't the other one encrypted with the passphrase?

3.1.6 Host authentication and User authentication

This part may seem confusing at first. Client host and server host have their own public keys and private keys. You generated another public/private key pairs by `ssh-keygen`. Is that necessary?

Until now there is only host authentication in your operation. The public and private keys you used are the keys of the host. These keys are generated when the SSH server is first started. The program `ssh-keygen` is used to generate the user key pairs which are used in user authentication. In the command `ssh linuxremote1.engineering.iastate.edu`, the user authentication is accomplished by password authentication.

3.1.7 Installing a user public key on an SSH Server Machine (15 pts)

When passwords are used for authentication, the host operating system maintains the association between the username and the password. For user keys, you must set up a similar association manually. 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.

The user public key must be placed in the file `~/.ssh/authorized_keys`. A typical authorized_keys file contains a list of **public-key** data, one key per line.

For this experiment, use `linux-1.ece.iastate.edu` as the remote host and preferably your own computer as client.

(5 pts) In the report, include all commands you used to copy your local public key into the remote file **authorized_keys** in remote server.

Try connecting to the remote SSH server again. If your public key was successfully added, you should only need to input the passphrase for the key instead of the password for the user name.

(10 pts) Briefly describe the whole authentication procedure after you entered the passphrase for the key.

3.1.8 The SSH agent (5 pts)

Every time you use ssh to log in to a server, you have to retype the passphrase. That is a terrible thing. Life should be simpler. Wouldn't you like the machine to remember your passphrase? Whenever you come, it knows you and lets you in directly. The SSH agent will achieve this. It will render your key to the SSH client on your behalf.

The SSH agent is a process that remembers your private keys in memory and provides authentication services to SSH clients. When the agent is running, the SSH client process will interact with the agent directly instead of you. The agent program for OpenSSH is ssh-agent. The agent will run until it is killed, which happens for example, when you log out.

To run the agent, enter: `ssh-agent $SHELL`, where SHELL is the environment variable containing the name of your login shell. Once the agent is running, it is time to load the private key into it using the `ssh-add` program.

(3 pts) Copy all the commands you used into the report.

(2 pts) Now log in to the remote server again. Copy the command and the output into the report.

3.2 GnuPG (30 pts)

In this part, we will learn the GNU Privacy Guard (GnuPG or GPG) which is widely used to encrypt, decrypt, sign and check messages on the Internet.

3.2.1 Generate a key pair (5 pts)

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

```
$gpg --gen-key
gpg (GnuPG) 1.4.5; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

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

There are three kinds of options. Option 1 creates two key pairs. The DSA key pair is the primary key pair used only for making signatures. Option 2 only creates a DSA key pair. Option 5 only creates a RSA key pair. For us the default option is fine.

```
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

You must also choose a key size. The longer the key the more secure it is. For most applications, the default key size is adequate. However, this will change with time. For example, the size of 1024 is enough today. But it is possible that 2048 will be required 10 years from now. But the smaller size of the key, the higher speed for encryption and decryption is. A larger key size may affect signature length. Once these keys are selected, the key size can never be changed.

Finally, you must choose an expiration date. If Option 1 was chosen, the expiration date will be used for both the ElGamal and DSA keypairs.

```
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 by changing keys, it will be more difficult to attack your key. For example, it will take the hacker 1 month to decrypt your key according to the speed of computer nowadays. For this reason you may choose to change you key each week.

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name:
```

You must provide a user ID in addition to the key parameters. The user ID is used to associate the key being created with a real person. After this, you are also required to input the email-address, comments and passphrase.

(5 pts) Use the GPG command-line option `--gen-key` to create a new key pair using the default selection. Copy the command and your selection into the lab report.

3.2.2 Exchanging keys (5 pts)

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. The key is exported in a binary format.

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

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 ASCIIarmored 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>
```

(5 pts) Exchange the public key with your partner. Include the command you used and the output of the `--list-keys`(after you have your partner's public key) into the report.(Hint: you can use `scp` or email to distribute your public key)

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>
```

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>
```

(5 pts) Encrypt your public key file and send it to your partner. Copy your command into the report.

(5 pts) Decrypt the public key file that your partner has sent to you. Copy your command into the report.

3.2.4 Making and verifying signatures (10 pts)

The command-line option `--sign` is used to make a digital signature. The document to sign is input, and the signed 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 recover the original document. To check the signature use the `--verify` option. To verify the signature and extract the document use the `--decrypt` option. The signed document to verify and recover is input and the recovered document is output.

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

A signed document has limited usefulness. Other users must recover the original document from the signed version, and even with clear-signed documents, the signed document must be edited to recover the original. Therefore, there is a third method for signing a document that creates a detached signature, which is a separate file. A detached signature is created using the `--detach-sig` option.

```
$ gpg --output <output file> --detach-sig <signed file>
```

(5 pts) Sign your public key file and send the signed file to your partner. Copy your command into the report.

(5 pts) Verify the signature of the public key file sent by your partner. Copy your command into the report.