SSH Key-Based Authentication

Typically, when someone uses a SSH client, that person needs to type a password for each new connection started. This can become bothersome if one is frequently making new connections or is in a situation where others may be physically present when the password is being typed. One alternative to typing the password so often is to use key-based authentication. There are several steps to setting up key-based authentication, but they are a one-time investment.

Generating SSH keypairs

A SSH keypair consists of a private key and a public key. Your private key is a secret in the same way that you password is a secret. And, your public key can be made publicly available in the same way that your name can be made publicly available. As with your password, you should not share your private key with anyone.

SSH tool suites usually provide a utility for generating these keypairs. On HPCC systems and most other Unix systems, there is a command named `ssh-keygen`. If you install the full PuTTY SSH suite on Windows, then you will have a utility program called `PuTTYgen`, which performs a similar function. When you use these utilities, you will be given an option for protecting your private key with a passphrase. Please do this; it will prevent your private key from being used by a malicious individual if it is ever stolen.

If you have not previously used `ssh-keygen`, then you can simply run the following command:

```
ssh-keygen -t rsa
```

After you have set a passphrase and it has generated the keys, you will find the key files in the `.ssh` directory under your home directory. By default, these are named `id_rsa` and `id_rsa.pub`. `id_rsa` is your private key and `id_rsa.pub` is your public key.

Use at least 2048 bits for an RSA key. This is usually the default key length, but can be specified with the '-b' option when generating your key, e.g. 'ssh-keygen -t rsa -b 2048'

Distributing SSH public keys

Some larger national HPC facilities, as well as sites which host free and open source software projects, only allow full access via key-based authentication. To access their machines, you need to first provide them with your public key. The MSU HPCC, of course, does not require key-based authentication, but we do provide it as an option. You can use it to connect from a home computer or personal workstation at work, or even from the computers of one facility to those of another. To get this to work, the machine that you are trying to login to must support key-based authentication (such as HPCC machines) and you must place your public key into what is known as the authorized keys file.

If you are seeking to login to HPCC with key-based authentication from your local computer, then you will need to perform the following:

- Create `~/.ssh` on `gateway.hpcc.msu.edu`, if you have not already done so.
- Copy the `id_rsa.pub` or equivalent file from your home or work machine over to `gateway.hpcc.msu.edu` as `~/.ssh/authorized_keys`, if that file does not already exist.
- If `~/.ssh/authorized_keys` does already exist on `gateway.hpcc.msu.edu`, then perform the following steps:
    - Copy the `id_rsa.pub` or equivalent file from your home or work machine over to `gateway.hpcc.msu.edu` as `~/work-pubkey`.
    - `cat ~/work-pubkey >> ~/.ssh/authorized_keys`
- Set the permissions appropriately by running "`chmod 700 ~/.ssh`" and "`chmod 600 ~/.ssh/authorized_keys`"
- Copy `id_rsa` from your local computer to ~/.ssh/ on the HPC gateway as well.

`~/.ssh/authorized_keys` is not a directory. It is a file. To store multiple authorized public keys in it, you will need to append the additional keys to the file.

Running the SSH agent

Although there are ways to use some private keys directly with your SSH client, you will likely want to run a SSH agent to manage any private keys that you have protected with passphrases. On HPCC and most other Unix systems, there is a program called `ssh-agent` for doing this. If you install the full PuTTY SSH suite on Windows, then you will have a similar utility called `Pageant`. If you are using Mac OSX, then see Adding a Private Key to Your Mac OSX Keychain.

What an SSH agent does is cache your private keys and allow your SSH clients to refer to this cache when attempting to establish new sessions. When you attempt a key-based connection to a remote SSH server, that server will look up the public keys in your authorized keys file on that remote machine and then challenge the connecting client to prove that it has a matching private key by decrypting a message encrypted with a public key. Your client will refer to the cache of private keys maintained by your SSH agent, for the purpose of decrypting this challenge message. If it finds the matching private key and is thus able to decrypt the challenge message from the remote SSH server, then you will be allowed to login once the client has proven this to the server. This all happens without you noticing anything different... except that you no longer need to type in your password during login.

To start `ssh-agent` on a Unix system using a Bourne shell-compatible shell (the default on HPCC), you can use the following command:

```
eval `ssh-agent`
```

This starts the agent and sets some Unix environment variables that tell the SSH client, `ssh`, where to talk to the running agent process. The equivalent command for those using a C shell-compatible shell is:

```
eval `ssh-agent -c`
```

Once it is running, you will want to load your private key(s) into the agent's cache. For example:

```
ssh-add
```

will prompt your for the passphrase on `~/.ssh/id_rsa` and load it into the agent's cache. If you are on some other system and have multiple keys pairs, then you may wish to load additional keys; for example:

```
ssh-add ~/.ssh/id_rsa_hpcc ~/.ssh/id_rsa_hpcc_vcs ~/.ssh/id_rsa_nersc
```

Once you have `ssh-agent` running and have some private keys loaded, then you should be able to make key-based connections to other systems where you have placed your corresponding public keys into the authorized keys files... no password necessary. Try it.

You don't want to leave stray `ssh-agent` processes running when you logout. To clean up after yourself, you can use:

```
eval `ssh-agent -k`
```

with Bourne shells and

```
eval `ssh-agent -c -k`
```

with C shells.

If you find yourself using the agent frequently, then you may wish to consider adding the `ssh-agent` startup and shutdown commands to your shell control files. For Bourne shells, you would want to add

```
eval `ssh-agent`
```

to `~/.bash_profile` and

```
eval `ssh-agent -k`
```

to `~/.bash_logout`. For C shells, you would want to add

```
eval `ssh-agent -c`
```

to your `~/.login` file and

```
eval `ssh-agent -c -k`
```

to your `~/.logout` file.

You should NOT add the `ssh-add` command to your shell initialization file. You should still plan on running this command by hand. As it issues prompts to your terminal, it is not a good idea to run it during shell initialization as the shell may not have fully configured your terminal yet.

Another way of managing `ssh-agent` is via a program called `keychain`. You can learn more at the Keychain web site.