Using SSH keys to authenticate users

Key-based authentication is the most secure of several modes of authentication usable with OpenSSH, such as plain password (the default with Ubuntu) and Kerberos tickets. Key-based authentication has several advantages over password authentication, for example the key values are significantly more difficult to brute-force, or guess than plain passwords, provided an ample key length. Other authentication methods are only used in very specific situations.

SSH can use either “RSA” (Rivest-Shamir-Adleman) or “DSA” (“Digital Signature Algorithm”) keys. Both of these were considered state-of-the-art algorithms when SSH was invented, but DSA has come to be seen as less secure in recent years. RSA is the only recommended choice for new keys, so this guide uses “RSA key” and “SSH key” interchangeably.

Key-based authentication uses two keys, one “public” key that anyone is allowed to see, and another “private” key that only the owner is allowed to see. To securely communicate using key-based authentication, you need to create a public key for the computer you're logging in from, and securely transmit it to the computer you're logging in to. Wikipedia has a good explanation of the theory.

Using key based logins with ssh is generally considered more secure than using plain password logins. This section of the guide will explain the process of generating a set of public/private RSA keys, and using them for logging into your Ubuntu computer(s) via OpenSSH.

Generating RSA Keys*

The first step involves creating a set of RSA keys for use in authentication.

This should be done on the client.

To create your public and private SSH keys on the command-line:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa
```

You will be prompted for a location to save the keys, and a passphrase for the keys. This passphrase will protect your private key while it's stored on the hard drive and be required to use the keys every time you need to login to a key-based system:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/b/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/b/.ssh/id_rsa.
Your public key has been saved in /home/b/.ssh/id_rsa.pub.
```

Your public key is now available as *.ssh/id_rsa.pub* in your home folder.

Congratulations! You now have a set of keys. Now it's time to make your systems allow you to login with them

Choosing a good passphrase

Just like with physical keys, you need to change all your locks if your RSA key is stolen. Otherwise, your thief will be able to get access to all your stuff.

An SSH key passphrase is a secondary form of security that gives you a little time when your keys are stolen. If your RSA key has a strong passphrase, it might take your attacker a few hours to guess by brute force. That extra time should be enough to log in to any computers you have an account on, delete your old key from the .ssh/authorized_keys file, and add a new key.

Your SSH key passphrase is only used to protect your private key from thieves. It's never transmitted over the Internet, and the strength of your key has nothing to do with the strength of your passphrase.

You have to choose for yourself whether to use a passphrase with your RSA key. Ultimately, it's a choice between cursing the difficulty every time you have to type it in, or cursing your glibness when someone logs in to all your accounts and changes your password so you can't get in any more.

If you choose to use a passphrase, pick something strong and write it down on a piece of paper that you keep in a safe place. If you choose not to use a password, just press the return key without typing a password - you'll never be asked for one again.

Key Encryption Level

Note: The default is a 2048 bit key. You can increase this to 4096 bits with the -b flag (Increasing the bits makes it harder to crack the key by brute force methods).

```
ssh-keygen -t rsa -b 4096
```

Password Authentication

The main problem with public key authentication is that you need a secure way of getting the public key onto a computer before you can log in with it. If you will only ever use an SSH key to log in to your own computer from a few other computers (such as logging in to your PC from your laptop), you should copy your SSH keys over on a memory stick, and disable password authentication altogether. If you would like to log in from other computers from time to time (such as a friend's PC), make sure you have a strong password.

Transfer Client Key to Host

The key you need to transfer to the host is the public one. If you can log in to a computer over SSH using a password, you can transfer your RSA key by doing the following from your own computer:

```
ssh-copy-id <username>@<host>
```

Where and should be replaced by your username and the name of the computer you're transferring your key to.

Due to this bug, you cannot specify a port other than the standard port 22. You can work around this by issuing the command like this:

```
ssh-copy-id "<username>@<host> -p <port_nr>"
```

If you are using the standard port 22, you can ignore this tip.

Another alternative is to copy the public key file to the server and concatenate it onto the authorized_keys file manually. It is wise to back that up first:

```
cp authorized_keys authorized_keys_Backup
cat id_rsa.pub >> authorized_keys
```

You can make sure this worked by doing:

```
ssh @
```

You should be prompted for the passphrase for your key:

```
Enter passphrase for key '/home//.ssh/id_rsa':
```

Enter your passphrase, and provided host is configured to allow key-based logins, you should then be logged in as usual.