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:

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

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

Copy SSH key to clipboard:

### Mac

* Type: **pbcopy < ~/.ssh/id\_rsa.pub**

### Linux (Ubuntu)

* Type and copy output: **cat ~/.ssh/id\_rsa.pub**

### Windows (Git Bash)

* Type: **clip < ~/.ssh/id\_rsa.pub**

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

(i) 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 <username>@<host>

You should be prompted for the passphrase for your key:

Enter passphrase for key '/home/<user>/.ssh/id\_rsa':

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

Troubleshooting

Encrypted Home Directory

If you have an encrypted home directory, SSH cannot access your authorized\_keys file because it is inside your encrypted home directory and won't be available until after you are authenticated. Therefore, SSH will default to password authentication.

To solve this, create a folder outside your home named /etc/ssh/<username> (replace "<username>" with your actual username). This directory should have 755 permissions and be owned by the user. Move the authorized\_keys file into it. The authorized\_keys file should have 644 permissions and be owned by the user.

Then edit your /etc/ssh/sshd\_config and add:

AuthorizedKeysFile /etc/ssh/%u/authorized\_keys

Finally, restart ssh with:

sudo service ssh restart

The next time you connect with SSH you should not have to enter your password.

username@host's password:

If you are not prompted for the passphrase, and instead get just the

username@host's password:

prompt as usual with password logins, then read on. There are a few things which could prevent this from working as easily as demonstrated above. On default Ubuntu installs however, the above examples should work. If not, then check the following condition, as it is the most frequent cause:

On the host computer, ensure that the /etc/ssh/sshd\_config contains the following lines, and that they are uncommented;

PubkeyAuthentication yes

RSAAuthentication yes

If not, add them, or uncomment them, restart OpenSSH, and try logging in again. If you get the passphrase prompt now, then congratulations, you're logging in with a key!

Permission denied (publickey)

If you're sure you've correctly configured sshd\_config, copied your ID, and have your private key in the .ssh directory, and still getting this error:

Permission denied (publickey).

Chances are, your /home/<user> or ~/.ssh/authorized\_keys permissions are too open by OpenSSH standards. You can get rid of this problem by issuing the following commands:

chmod go-w ~/

chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized\_keys

Error: Agent admitted failure to sign using the key.

This error occurs when the ssh-agent on the client is not yet managing the key. Issue the following commands to fix:

ssh-add

This command should be entered after you have copied your public key to the host computer.

Debugging and sorting out further problems

The permissions of files and folders is crucial to this working. You can get debugging information from both the client and server.

if you think you have set it up correctly , yet still get asked for the password, try starting the server with debugging output to the terminal.

sudo /usr/sbin/sshd -d

To connect and send information to the client terminal

ssh -v ( or -vv) username@host's

Where to From Here?

No matter how your public key was generated, you can add it to your Ubuntu system by opening the file .ssh/authorized\_keys in your favourite text editor and adding the key to the bottom of the file. You can also limit the SSH features that the key can use, such as disallowing port-forwarding or only allowing a specific command to be run. This is done by adding "options" before the SSH key, on the same line in the authorized\_keys file. For example, if you maintain a CVS repository, you could add a line like this:

command="/usr/bin/cvs server",no-agent-forwarding,no-port-forwarding,no-X11-forwarding,no-user-rc ssh-dss <string of nonsense>...

When the user with the specified key logged in, the server would automatically run /usr/bin/cvs server, ignoring any requests from the client to run another command such as a shell. For more information, see the sshd man page. /755

Public and Private Keys

Public key authentication is more secure than password authentication. This is particularly important if the computer is visible on the internet. If you don't think it's important, try logging the login attempts you get for the next week. My computer - a perfectly ordinary desktop PC - had over 4,000 attempts to guess my password and almost 2,500 break-in attempts in the last week alone.

With public key authentication, the authenticating entity has a public key and a private key. Each key is a large number with special mathematical properties. The private key is kept on the computer you log in from, while the public key is stored on the .ssh/authorized\_keys file on all the computers you want to log in to. When you log in to a computer, the SSH server uses the public key to "lock" messages in a way that can only be "unlocked" by your private key - this means that even the most resourceful attacker can't snoop on, or interfere with, your session. As an extra security measure, most SSH programs store the private key in a passphrase-protected format, so that if your computer is stolen or broken in to, you should have enough time to disable your old public key before they break the passphrase and start using your key. Wikipedia has a more detailed explanation of how keys work.

Public key authentication is a much better solution than passwords for most people. In fact, if you don't mind leaving a private key unprotected on your hard disk, you can even use keys to do secure automatic log-ins - as part of a network backup, for example. Different SSH programs generate public keys in different ways, but they all generate public keys in a similar format:

<ssh-rsa or ssh-dss> <really long string of nonsense> <username>@<host>

Key-Based SSH Logins

Key-based authentication is the most secure of several modes of authentication usable with OpenSSH, such as plain password 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, one needs to create a key pair, securely store the private key on the computer one wants to log in from, and store the public key on the computer one wants to log in to.

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.

Choosing a good passphrase

You need to change all your locks if your RSA key is stolen. Otherwise the thief could impersonate you wherever you authenticate with that key.

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.

The decision to protect your key with a passphrase involves convenience x security. Note that if you protect your key with a passphrase, then when you type the passphrase to unlock it, your local computer will generally leave the key unlocked for a time. So if you use the key multiple times without logging out of your local account in the meantime, you will probably only have to type the passphrase once.

If you do adopt a passphrase, pick a strong one and store it securely in a password manager. You may also write it down on a piece of paper and keep it in a secure place. If you choose not to protect the key with a passphrase, then just press the return when ssh-keygen asks.