Part of SSH’s security involves *encryption keys*. Each server system and each user has  
a unique number, or key, for identification purposes. In fact, SSH uses a security system  
that involves two keys: a *public key* and a *private key*. These two keys are mathematically  
linked in such a way that data encrypted with a particular *public* key may be decrypted  
only with the matching *private* key.  
When establishing an SSH connection, each side sends its public key to the other.  
Thereafter, each side encrypts data with the other side’s public key, ensuring that only the  
intended recipient can decrypt the data. In practice, this is just the first step of the process,  
but it’s critical. What’s more, SSH clients typically retain the public keys of servers they’ve  
contacted. This enables them to spot changes to a public key. Such changes can be signs of  
tampering, so if a client detects such a change, it will warn its user of this fact.  
Most OpenSSH server startup scripts include code that looks for stored public and private keys and, if they’re not present, generates them. In total, four to six keys are needed:  
public and private keys for two or three encryption tools that SSH supports. These keys are  
normally stored in /etc/ssh and are called ssh\_host\_rsa\_key and ssh\_host\_dsa\_key  
for private keys, depending on the encryption algorithm used. For public keys, the same  
filenames are used, except a .pub filename extension is added to the file’s name. Some  
systems also add ssh\_host\_rsa1\_key and its associated public key.  
If your system doesn’t have these keys and you can’t get the SSH server to start up, you  
can try generating the keys with the following ssh-keygen commands:  
# **ssh-keygen -q -t rsa1 -f /etc/ssh/ssh\_host\_key -C '' -N ''**  
# **ssh-keygen -q -t rsa -f /etc/ssh/ssh\_host\_rsa\_key -C '' -N ''**  
# **ssh-keygen -q -t dsa -f /etc/ssh/ssh\_host\_dsa\_key -C '' -N ''**

-q: quiet

-t: type

-f: file

-C: comment

-N: new passphrase. If you don’t use this option, you’ll be asked for a passphrase

Each of these commands generates both a private key (named in the -f parameter) and a  
public key (with the same name but with .pub appended).  
Don’t run these ssh-keygen commands if the SSH key files already exist! Replacing the  
working files will cause clients who’ve already connected to the SSH server to complain  
about the changed keys and possibly refuse to establish a connection.

Be sure the *private* keys are suitably protected; if an intruder obtains one of  
these keys, the intruder can impersonate your system. Typically, these files  
should have 0600 (-rw-------) permissions and be owned by root. The  
*public* key files (with .pub filename extensions) should be readable by all  
users, though.

When you configure a client system, you may want to consider creating a global cache of  
host keys. As already noted, the ssh program records host keys for each individual user. (It  
stores these in the ~/.ssh/known\_hosts file.) When you set up the client, you can populate  
the global ssh\_known\_hosts file, which is normally stored in /etc or /etc/ssh. Doing so  
ensures that the public key list is as accurate as the sources you use to populate the global  
file. It also eliminates confirmation messages when users first connect to the hosts whose  
keys you’ve selected to include in the global file.  
How do you create this file? One simple way is to copy the file from a user account that’s  
been used to connect to the servers you want to include, as shown here:  
$ **sudo cp /home/Rich/.ssh/known\_hosts /etc/ssh/ssh\_known\_hosts**  
[sudo] password for Christine:  
$

In the past, you could review SSH’s known hosts file in a text editor since  
it’s a text-mode file. Now OpenSSH v4.0 and newer versions support hashing this file’s data. When this feature is enabled, the information is *hashed*  
and stored. The idea is that you’ll still be able to authenticate SSH servers  
to which you connect because a hash of the typed hostname will match  
a hash of the stored hostname. However, if an attacker steals your known  
hosts file, the attacker will be unable to determine the identities of the  
computers to which you’ve been connecting. An unfortunate side effect of  
this hashing is that you can’t tell what servers it describes yourself.