

Hentzenwerke Whitepaper Series

# ***Remote Access Via SSH***

**By Whil Hentzen**

**SSH is one of those typical “Linux mysteries” for the uninitiated. SSH provides a secure mechanism to connect to another machine over a network. This allows you to control a remote computer (such as through the command window) over the Internet without exposing your connection to other people. Here's what SSH does, why you'd use it, how it works and, most importantly, how to use it.**

## 1. Preface

### 1.1 Copyright

Copyright 2006 Whil Hentzen. Some rights reserved. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License, which basically means that you can copy, distribute, and display only unaltered copies of this work, but in return, you must give the original author credit, you may not distribute the work for commercial gain, nor create derivative works based on it without first licensing those rights from the author. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/2.0/>.

### 1.2 Revisions

#### 1.2.1 History

Version	Date	Synopsis	Author
1.0.0	2004/8/19	Original	WH
1.1.0	2006/1/6	Added public/private key usage	WH

#### 1.2.2 New version

The newest version of this document will be found at [www.hentzenwerke.com](http://www.hentzenwerke.com).

#### 1.2.3 Feedback and corrections

If you have questions, comments, or corrections about this document, please feel free to email me at 'books@hentzenwerke.com'. I also welcome suggestions for passages you find unclear.

### 1.3 Acknowledgments

Thanks to MLUG member Joe Baker for a great talk on SSH in 2003, the MLUG regulars at the August '04 and December '05 meetings, Tom Francis, Aaron Schrab, and Chad Voelker for content and reviews, Ted Roche for his critical eyeballing and nudging me to include scp, and Steve Suehring for some background material.

### 1.4 Disclaimer

No warranty! This material is provided as is, with no warranty of fitness for any particular purpose. Use the concepts, examples and other content at your own risk. There may be errors and inaccuracies that in some configurations may be damaging to your system. The author(s) disavows all liability for the contents of this document.

Before making any changes to your system, ensure that you have backups and other resources to restore the system to its state before making those changes.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

### 1.5 Prerequisites

This document was written using Fedora Core 2.0 on the server and clients running Fedora Core 1.0 and SuSE 9.2, and assumes a beginner's familiarity with use of Linux via the GUI and the Command Window.

## 2. SSH definition - what is it and what does it do?

Connecting to another machine on the Internet is a risky proposition. Some tools, like ftp and telnet, that provide access to another machine transmit the username and password in clear text (meaning that they're not encrypted, and thus readable by anyone who captures them.) In some cases, this isn't a problem. For example, FTP transfers are fine for communication that doesn't have to be secure, such as if you're just downloading a set of RPMs whose authenticity can be verified separately. But this isn't acceptable if you are trying to transfer things securely.

On the other hand, telnetting over a public network into a machine to set a configuration, while a common practice years ago, is now disavowed as a practice by all savvy users. This is because there are all sorts of bad guys out there with tools to capture these clear text usernames and passwords for their own unscrupulous purposes. As a result, the safe way to connect is to use a program that encrypts the communications between the two computers.

SSH is one such program. It is a software utility included with every mainstream Linux distribution, that provides a method of using a command window (or shell) on a remote machine. It's name is an acronym for "Secure SHell". SSH has to run both on the machine you're using (known as the local machine or the client, running client software) and the machine you're connecting to (known as the remote machine, the host, or the server, running server software.) SSH runs on the remote machine just like any other Linux service, and is initiated by you on the client in order to connect to the remote machine.

The internal behavior of SSH is to establish a secure tunnel and then pass back and forth the contents of a shell session. In fact, it gets better. SSH also will also let you share that secure tunnel with other applications that would not be secure on their own.

### 3. The configuration used for this discussion

I used the following components for this document.

1. A Web server running Fedora Core 2.0 with an IP address of 169.207.151.113 that is tied to the domain of www.hidbigo.com. This Web server had port 22 open in order to be able to accept SSH connections.

2. A client running Fedora Core 1.0, running on a separate network and Internet connection (for the initial part), and a client running SuSE 9.2, also running on a separate network and Internet connection (for the public/private key part.)

### 4. Using SSH via Passwords

For this HOWTO, I'm going to assume you've got access to both the server for configuration purposes and then will use the client to connect to the server.

First, SSH (the server) must be running on the remote machine, and you need to know the IP address or URL of the remote machine. The remote machine needs to be able to accept requests on port 22, the port used by SSH.

#### 4.1 Configuring the server

On the server, the first thing you should do is change a setting in `/etc/ssh/sshd_config` the SSH config file. The default value for

```
PermitRootLogin
```

is (on some distributions)

```
yes
```

With this default, anyone can try to log in to the server as root, and once that's accomplished (either by knowing the root password or by guessing), they own the machine.

Change this setting to no, like so:

```
PermitRootLogin no
```

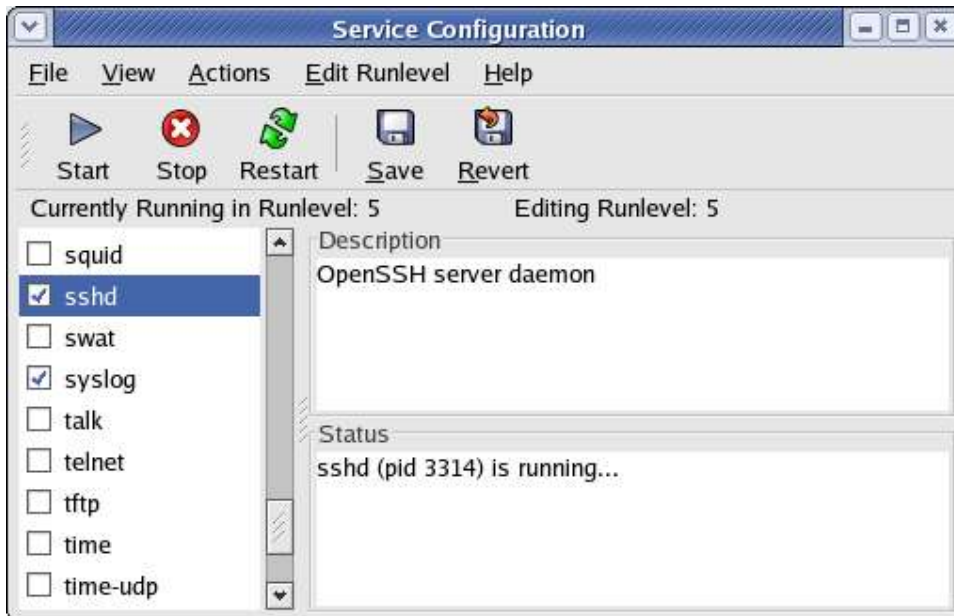
Some distributions have this setting commented out, which is equivalent to having the setting set to no. Nonetheless, it's probably a good idea to explicitly set it to no. This 'no' setting prevents someone from SSH'ing into the machine as root. Instead, they would have to SSH into the server using a different user account, and then, as that user, 'su' to become root. This means they have to know not only the root account's password, but also the username and password of another account on the machine for the initial access, so you're made it harder for them to break in and gain control.

Once you made the change, you'll need to restart the SSH server. This can be done via a command in the command window or through the GUI.

In the command window, switch to root, and then issue the command

```
/etc/rc.d/init.d/sshd restart
```

In the GUI, open the Services dialog via the System Settings | Server Settings | Services menu option (you'll need to enter the root password), select the sshd service, and then click the Restart button as shown in **Figure 1**.



**Figure 1.** The Service Configuration dialog allows you to restart the SSH service.

## 4.2 Using the client to connect to the server

Now let's go to the Linux client - the machine you're working on. Open a command window and issue the command

```
ssh
```

followed by the IP address or URL of the remote machine, like so

```
ssh 169.207.151.113
```

or

```
ssh www.hidbigo.com
```

Upon receipt of a connection attempt from a client, the two machines will undertake a handshaking process that includes the server sending its fingerprint to the client so that the client can verify that the server is indeed who it says it is. The client will compare its own copy of the server's key (the copy is stored on the client) with the copy that the server has sent.

If this is the first time you've connected to the remote machine, you'll see a response displaying the fingerprint of the remote machine, a warning that the client doesn't know if that's the right fingerprint, and you'll be prompted to verify that the fingerprint is correct, as shown in **Figure 2**.



**Figure 2.** Upon first connection, you're asked to confirm this is the machine you want to connect to.

The line

The authenticity of host 'www.hidbigo.com (169.207.151.113)' can't be established.

means that the client doesn't yet have a copy, and thus the client doesn't know if the fingerprint is valid or not. . The line

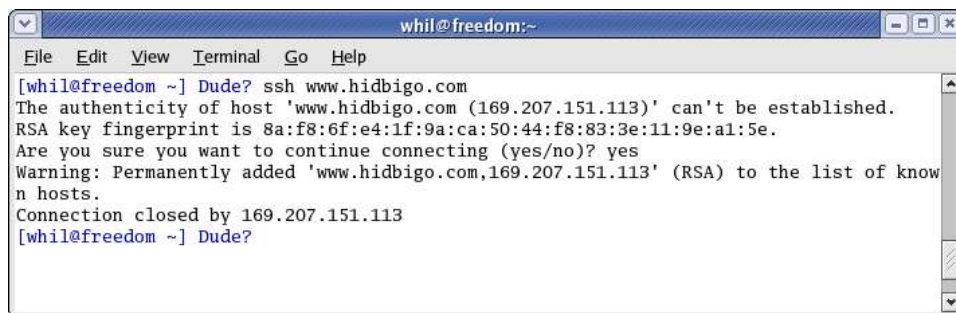
RSA key fingerprint is 8a:f8:6f:e4:1f:9a:ca:50:44:f8:83:3e:11:9e:a1:5e.

shows the key that the server purportedly sent to the client. It is up to the client user to confirm that this key is indeed the actual key of the server. I say 'purportedly' because it is technically possible that the fingerprint that the server sent was intercepted and modified en route before its display on the client's machine.

If this connection is between machines containing highly sensitive data, you may wish to verify the validity of the key through a second channel, such as the telephone. In other words, for this first contact, you (the person in front of the client) would communicate with the owner of the server via another mechanism to find out what the real fingerprint of the server is, and compare that value with the value that was displayed on your computer screen. If they're the same, you can be sure that the fingerprint was not intercepted and modified en route.

To be pragmatic, in most situations you'll simply assume that the initial value of the fingerprint received from the server is authentic.

In both cases, assuming that the the key is good, you'll enter 'yes' to the "Are you sure you want to continue connecting?" prompt, and a copy of the key will be saved on the client computer, as indicated by the "Permanently added 'www.hidbigo.com, 169.207.151.113' (RSA) to the list of known hosts." message in **Figure 3**.



```

whil@freedom:~
File Edit View Terminal Go Help
[whil@freedom ~] Dude? ssh www.hidbigo.com
The authenticity of host 'www.hidbigo.com (169.207.151.113)' can't be established.
RSA key fingerprint is 8a:f8:6f:e4:1f:9a:ca:50:44:f8:83:3e:11:9e:a1:5e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'www.hidbigo.com,169.207.151.113' (RSA) to the list of known hosts.
Connection closed by 169.207.151.113
[whil@freedom ~] Dude?

```

**Figure 3.** SSH asks you to verify that the key the server sent to you, the client, is correct; you'll either want to independently verify the authenticity of the key, or blindly trust on the first connection.

The list of known hosts referred to is in the file

`/home/{username}/.ssh/known_hosts`

and looks like this:

```

www.hidbigo.com,169.207.151.113 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEAnAUoD6Jhn4KbJyAiX3oX/nR+u4BIP03ZWx1l+PKLDju1n6mYAiUI6m2LK/I0fPDydWC1X/w13Ip+Y/udAN/KmFmPrzAuJIJgVn/81SBRHLY2AsW7gKXGNnwvPrQ7O3v6+tmSTeJrvnUwSr2022rncQ+gf1Tc1rE3L/d9G+5GnXM=

```

The `known_hosts` file will contain a separate entry for every host (server) that the client connects to.

Note that there is a `known_hosts` file for each user on the client, so that the hosts that Al connects to aren't confused with the hosts that Barb connects to.

Once the value has been added to the `known_hosts` file, during subsequent connections, the value received from the server will be compared with the value stored on the client. If the values differ, it could mean that the server has been modified, such as through an update of the ssh program, or that the server has been attacked and compromised. Regardless, in the event of the value has changed, you'll want to investigate why before continuing with the connection.

Upon successful entry of the password, the connection will be closed, as shown in Figure 3, but the client will now have a record of the server. Issue the 'ssh' command again with the IP address or URL, and you'll immediately be prompted for the password this time, as shown in **Figure 4**.



```
whil@www:~  
File Edit View Terminal Go Help  
[whil@freedom ~] Dude? ssh www.hidbigo.com  
whil@www.hidbigo.com's password:  
[whil@www whil]$
```

**Figure 4.** After successfully supplying the password to the remote user account, you'll see a command prompt that confirms you're connected remotely.

Upon successful entry of the password this time, you'll get a command prompt for the user on the server, and you can now issue commands to the server just as if you were sitting in front of the machine itself.

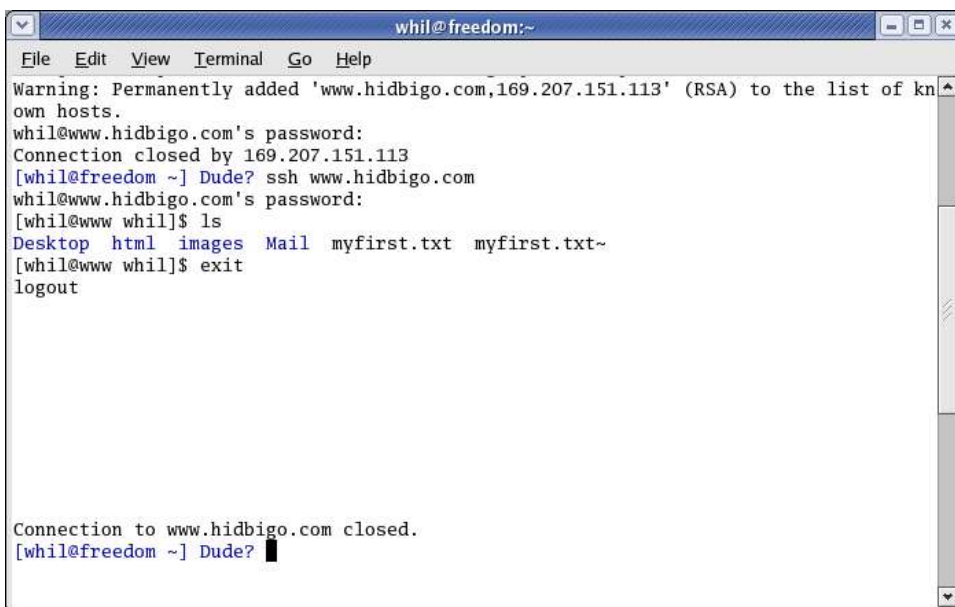
At this point, it's appropriate to discuss the user accounts and passwords more explicitly. When you're logging on to the remote box, you're logging onto an account on that machine, and thus, you're being prompted for the password of the client's currently logged on user – but on the server machine. In other words, if you're logged in as 'alvin' on the client machine, there will need to be an 'alvin' account on the server as well. Enter the password for the 'alvin' account on the server.

If you, the reader, try to log in to the sample domain here, [www.hidbigo.com](http://www.hidbigo.com), you'll be stymied unless you know a user account and the password associated with that account.

Once you're done with your session on the remote machine, type

```
exit
```

in the command window, as shown in **Figure 5**.



```
whil@freedom:~  
File Edit View Terminal Go Help  
Warning: Permanently added 'www.hidbigo.com,169.207.151.113' (RSA) to the list of known hosts.  
whil@www.hidbigo.com's password:  
Connection closed by 169.207.151.113  
[whil@freedom ~] Dude? ssh www.hidbigo.com  
whil@www.hidbigo.com's password:  
[whil@www whil]$ ls  
Desktop html images Mail myfirst.txt myfirst.txt~  
[whil@www whil]$ exit  
logout  
  
Connection to www.hidbigo.com closed.  
[whil@freedom ~] Dude? █
```

**Figure 5.** Logging out of the SSH session restores your command prompt to its previous value.

After a few moments, the server will respond with the logout command and then the client will confirm the closure of the connection as well. The command window on the client will be returned to the currently logged in user on the client.

## 5. Using SSH via Public Key Authentication

While password authentication solves the problem of mechanisms that transmit passwords in clear text, it's still susceptible to cracking. The year of 2005 saw a rapid increase in SSH attacks using brute force password guessing tools against common accounts. Hopefully your remote machine doesn't have an account named 'bob' with a password of 'god' or 'darthvader' or 'qwerty' (although if it does, the box has long been compromised.) As a result, it's an opportune time to update this paper with a discussion of an alternative to passwords - authentication using public and private keys.

This method of authentication has an added benefit - you can log into the remote machine without having to type a password. In fact, some folks espouse public key authentication more for this convenience than for the additional security benefits.

## 5.1 How public/private keys work

Public key authentication involves a pair of related keys that in this situation take the form of text files. The public key sits on the remote machine while the private key resides on the local (and presumably terribly, terribly safe) machine. When the local machine connects to the remote box, the two keys are matched up by the remote machine. If they match correctly, the authentication passes and you are granted access.

### 5.1.1 An amateur's discussion of public and private keys

You can skip this section if you're not interested in how the keys work. But since you're a Linux geek, you probably are interested.

(For a more in-depth discussion, see Simon Singh's excellent book, *The Code Book* (Anchor Books, ISBN 0-385-49532-3) that addresses this topic as part of complete coverage of cryptography from ancient Egypt to modern day methods.)

The idea behind public and private keys is that a user, Alice, can distribute her public key in many, many places, for everyone to use. Anyone, say, Bob, can use that public key to encrypt a message and send the result to Alice. However, since Alice is the only one who has the matching private key, she's the only person who can decrypt the message.

Singh uses the example of Alice having the only key to a padlock, and then distributing bazillions of copies of this padlock, say, to every post office in the world. Then, when Bob wanted to send a secret message to Alice, he'd go to his local post office, ask for an "Alice padlock", lock up his message, and send it off to her. When she got the locked-up message, she'd use her key to unlock it.

The padlock(s), of course, represent the public key, and Alice's key represents her private key. Anyone can encrypt a message for Alice, but only she, using her private key, can decrypt it. This way, two people can communicate securely without ever having to meet to exchange a mediating mechanism first.

In the realm of computers, the keys are files that contain data, and the locking mechanism is a mathematical algorithm. Alice publishes a public key,  $N$ , which is the sum of two prime numbers, and which only she knows the two prime factors.

Bob encrypts his message using a special function and Alice's public key, like so:

$$\text{EncrMsg} = f(\text{UnenMsg}, N)$$

This function is special because it is a 'one-way' function. You can calculate  $\text{EncrMsg}$  if you know  $\text{UnenMsg}$  and  $N$ , but you can't reverse engineer the function to determine  $\text{UnenMsg}$  if you know  $\text{EncrMsg}$  and  $N$ . This is in contrast to a 'two-way' function like multiplication, where, if you know any two of the pieces, you can determine the third.

A simple example of a one-way function is modulus arithmetic. If you know  $i$ , you can calculate  $3^i \bmod N$  like so:

$$x = 3^i / N$$

However, if you know  $x$  and  $N$ , and those values are large, it becomes highly laborious (even with a computer) to determine what  $i$  is.

Back to Alice and Bob. Since the special function can only be reverse-engineered using the factors, and only Alice knows the factors, it works. (This, of course, is why the search for very, very large prime numbers is so interesting to security folks.)

## 5.2 To implement

As with the password section, I'm going to assume you've got access to both machines - you'll use the server for configuration purposes and then will use the client to connect to the server.

First, SSH (the server) must be running on the remote machine, and you need to know the IP address or hostname of the remote machine. The remote machine needs to be able to accept requests on port 22, the port used by SSH.

### 5.2.1 Configuring the server

On the server, the first thing you will need to do is change several settings in `/etc/ssh/sshd_config` the SSH config file. First, change the default value for

```
PermitRootLogin
```

to

no

if it hasn't been already done (as described in section 4.1.) Next, make sure the following lines are set as shown:

```
PubkeyAuthentication yes
PasswordAuthentication no
ChallengeResponseAuthentication no
UsePAM no
```

The first enables public key authentication, the second and third disable other types of authentication, so that potential intruders can't use an alternative method to authenticate and gain access. The last turns off PAM (Pluggable Authentication Module.)

Once you made these four changes, you'll need to restart the SSH server via the command:

```
/etc/rc.d/init.d/sshd restart
```

And now you're done with configuring the server (although don't ship the box off to a bunker under the Rocky Mountains just quite yet, since we'll be working with it once more for a moment in order to move the public key file to the remote box.)

By the way, I'm assuming that you have a non-SSH mechanism for moving the public key file that you create on the local box to the remote box. If you were planning on using SCP (via password authentication) to copy the public key file to the remote box, you'll want to keep "PasswordAuthentication" set to 'yes' until after you get the file transferred and verify that you can SSH into the box via Public Key Authentication successfully. Don't want to lock yourself out of the box by throwing away the only working key.

### 5.2.2 Create the public and private keys

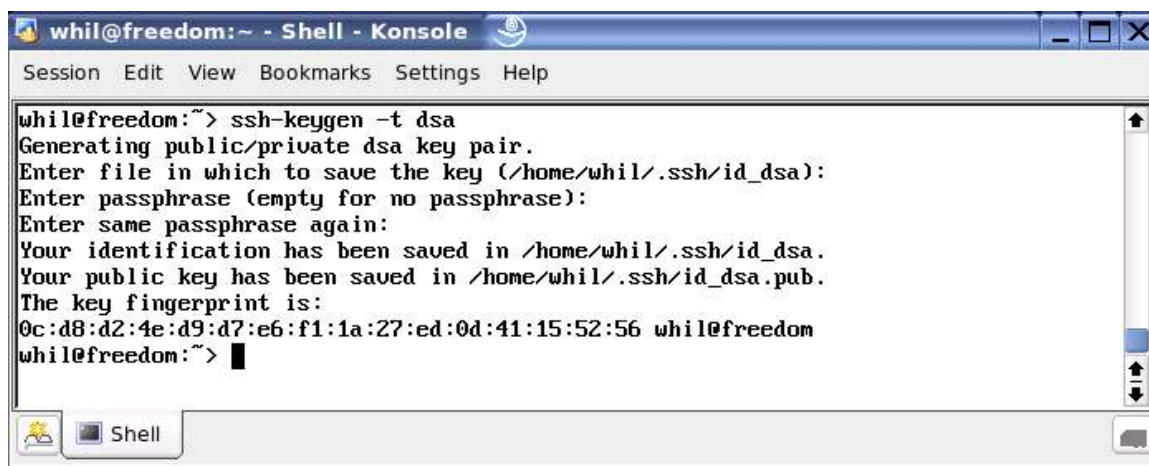
On the client, create the public and private keys via the `ssh-keygen` command. There are two variations of the command, one that uses an RSA public/private keypair, the other that uses DSA. OpenSSH recommends DSA, although it's not hard to find rabid fans for both options. Type

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

or

```
ssh-keygen -t dsa
```

You'll be prompted for a passphrase, and then prompted to verify the passphrase. as shown in **Figure 6**. Use something long but easy to remember (and easy to type!) - and don't use your password!



**Figure 6.** Generating public and private keys via `ssh-keygen`.

(You may be wondering what the difference between a password and a passphrase is. In the most limited sense, a password is a word, like 'mother', or 'zombiemare' or 'jl44RT\_9', while a passphrase is a full string of (preferably memorable) words, like "send in the clowns", "paul is dead long live the king", or "it was the best of times it was the worst of times". If



you're wondering why use a passphrase instead of a password, it's because you're going to be able to use the passphrase to keep your key in memory, while with password authentication, you'll end up typing it in over and over again.)

Once the keys have been generated in memory (it may take five or ten seconds, depending on the horsepower of the box you're on), you'll be prompted for a filename; if you simply hit return, ssh-keygen will use the default values of `id_dsa` and `id_dsa.pub` (or `id_rsa` and `id_rsa.pub`), and put them in `$HOME/.ssh`.

After the keys have been written, you'll be reminded what your files are named and where they're located (again, as shown in Figure 6.)

The first file is your private key, and you want to keep it secret, secret, secret. Theoretically, since it's on your local box, it's presumably as safe as anything else on the box. The private key is a simple text file, viewable through any text editor. If you were to open it up, you'd see something like that shown in Figure 7.

```

-----BEGIN DSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, FB87EA9C52772487

DXy0yC6IRDnyWr6ubYG5h9pMT17IGo75yzI1Y9iQW3YpF1kS45tr869FYCXn71Q9
wBOPjne9XKZ2twiQUjzSRiWUixr7Lr217Y8/VEs2JZ1+fdPBj10kaIXT89EjBs4UW
esAVIm0k7ZJWL/PbZbLUipjTV21ELdrynjQ1a5yQtHX23dQACHE800I2mT0uH1Ka
GGX9LAXsq8s2yHuLBUP73hcfdT3B1FTkwnkqCmeCsKuhHPR48P5IXUYBqo2kIj01
sNb5xVdgee5LnpqN1aar+y5z6R3hmWbwgWPiuHGD5gPXHz4K2DCNU2Dr3yU35e0E
+Myu11JFQjjQnXgKeUJPDUPNSJVjLXxkfd00NMQfHxQjX6tw0/CnHx2CwHHegDB
Gk5i9znlp+i+y6umTR1kWaXmwOCp7d1q2+jxwQ590Y1Lc1R7UuzmGTFYoBzMFepm0
UnzwMdb0RRKGxhmq+61kWG9Pn6qr0Ng4d0QasZzQZnsQQ1DaK7CkB/RsA1+1+H9oH
Na0UMCmgpt8F2b52nJC98n0WZo2wHT0uHaYKP00VzuK1sTKIKUkcf2b7eYAh2dKW
Uv5BkwbMk+AJZgZnpDf/Rw==
-----END DSA PRIVATE KEY-----

id_dsa lines 1-15/15 (END)

```

Figure 7. Contents of a typical private key.

The second file (with the `.pub` extension) is your public key. Just like the collection of Alice's locks, you can (and are encouraged to) distribute this key to anyone who might want to encrypt something before sending it to you. Some folks place their public key on a public Web page, and then provide a link to that page in their email signature. A quick peek at it through your favorite text editor will look something like Figure 8.

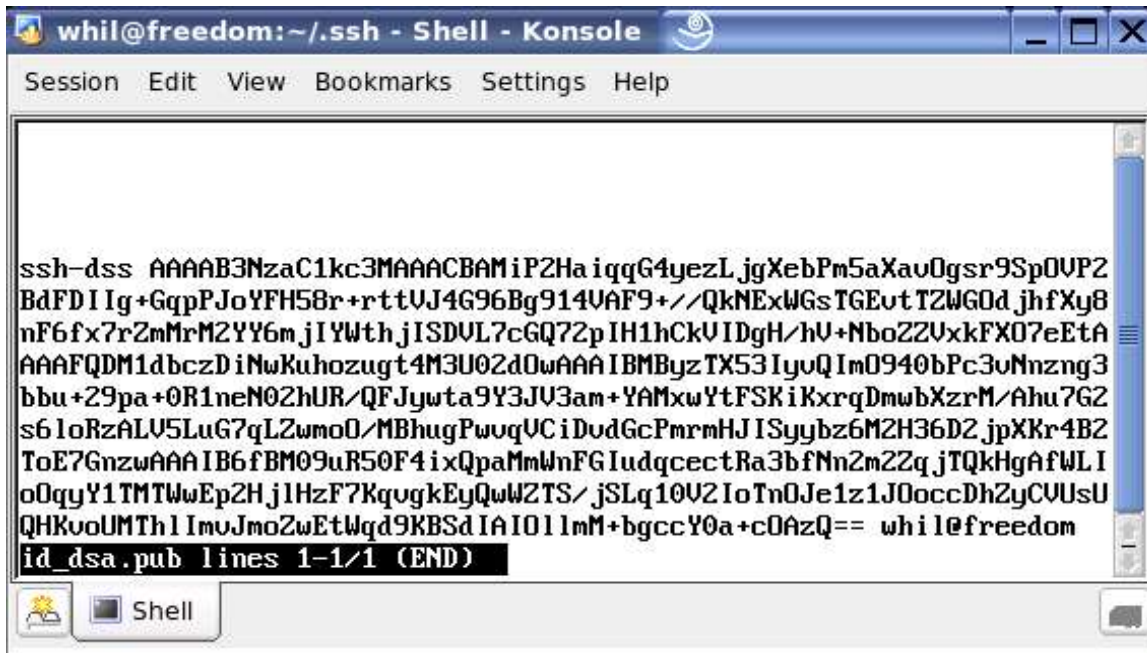


Figure 8. Contents of a typical public key file.

For this public key to be used with SSH on the remote box, it needs to be put on the remote system into the `ssh/authorized_keys` file in the home directory of the account that you'll be SSH-ing into. For example, if you're going to be doing a

```
ssh bob@www.example.com
```

command, then you need to put `id_dsa.pub` into `/home/bob/.ssh/authorized_keys` on the `www.example.com` box.

You can get the public key file over to the remote box via the SCP command (explained in more detail later in this whitepaper) like so:

```
local ~> scp .ssh/id_dsa.pub bob@www.example.com:/home/bob/.ssh
```

replacing `www.example.com` with the URL or IP address of your remote box. This will put a copy of the public key file in the `.ssh` directory of the `bob` user on the remote box.

Next, access the remote box (either via SSH - password authentication, or physical access). Stuff the public key into the `authorized_keys` file like so:

```
remote ~/.ssh> cat id_dsa.pub >> authorized_keys
```

(The `authorized_keys` file on the remote box can hold more than public key - one for each local system you would be connecting from - so the `'cat'` command adds this public key to the file instead of replacing the entire file, hence the use of the `'>>'` instead of just `'>'`.)

Finally, you'll want to make sure that the permissions on the `.ssh` directory and the `authorized_keys` file are correct.

```
remote ~> chmod 700 .ssh
```

will make sure the directory is not readable by anyone else, while

```
remote ~> chmod 644 .ssh/authorized_keys
```

will make sure the `authorized_keys` file is read-only by every one else. Now you're all set to give it a shot.

### 5.2.3 Using the client to connect to the server

From your local machine, issue the ssh command. Suppose you wanted to log in to the 'bob' account on the 'www.example.com' box, you'd enter:

```
local >ssh bob@example.com
```

You'll be prompted for the passphrase that you entered in Figure 6 earlier. Enter it and you'll be logged into the account on the remote machine, as shown in **Figure 9**.



**Figure 9.** After SSHing into a remote box, you'll be prompted for your passphrase.

Remember those settings in the sshd\_config file on the remote box that you changed section 5.2.1? If you didn't change the PasswordAuthentication setting to no, you can still gain access to the remote box using password authentication. If you just hit 'Enter' after being prompted for the passphrase, you'll be prompted for your password next, and if you enter the password for the account on the remote machine, you'll gain access as expected.

As a result, just because you turned "PubkeyAuthentication" to 'yes' doesn't mean you're safe from password crackers - you'll still need to turn PasswordAuthentication to 'no' as well.

### 5.2.4 Adding your passphrase to your ssh-agent key ring

One of the benefits to using a passphrase is that you don't have to enter it every time you SSH into the remote box. However, if you've already gone through section 5.2.3 and then played around a bit, you'll find that you're prompted for your passphrase every time. What a nuisance!

A program called 'ssh-agent', which is normally started when you log into your GUI, keeps your keys in memory, relieving you from having to type the passphrase in each time. Use the ssh-add program to add your private key to the ssh-agent keyring, using the passphrase to unlock it, like so:

```
local> ssh-add
```

and enter your passphrase when prompted, as shown in **Figure 10**.



```
whil@freedom:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

whil@freedom:~> ssh-add
Enter passphrase for /home/whil/.ssh/id_dsa:
Identity added: /home/whil/.ssh/id_dsa (/home/whil/.ssh/id_dsa)
whil@freedom:~>
```

**Figure 10.** Running `ssh-add` to add your passphrase to your keyring.

Now, run SSH - you should find that you're not prompted for your passphrase, as shown in **Figure 11**.



```
whil@freedom:~ - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help

whil@freedom:~> ssh www.hidbiqo.com
Last login: Fri Jan 6 16:16:41 2006 from 68.21.207.29
Have a lot of fun...
whil@indy:~>
```

**Figure 11.** Once you've added your key to your keyring, you won't be prompted for it any longer.

Your key will stay in memory as long as you stay logged in, until you explicitly remove the key from the keyring, or until you lock `ssh-agent` (to keep it from handing out keys) via `ssh-add -x`. (You can use `ssh-add -X` to unlock.)

If you don't have the `ssh-agent` process running, check out the man page for `ssh-agent`. It's surprisingly readable.

When you log out of the GUI, `ssh-agent` will terminate and free the memory used to store the keys.

### 5.2.5 Using a private key on a separate device

If you don't want to keep your private key on your local machine, say, because you bounce from machine to machine, you could put it on a removable device, like a USB drive. Suppose your USB flash drive was mounted on

```
/usb
```

and the private key was thus at

```
/usb/id_dsa
```

You could `ssh` into a remote box like so:

```
local> ssh -i /usb/id_dsa bob@www.example.com
```

## 6. Advanced Functions

After you've used SSH a bit, you'll want to do a few more things.

## 6.1 Become root on the remote machine

You can become root on the remote machine by issuing the 'su' command in the command window once you're connected to the remote machine. You'll need the password of the root user on the remote machine, of course. When you're done with your root work on the remote machine, issue the

```
exit
```

command and you'll be returned to the regular user who was logged onto the remote machine. For example, if alvin had logged onto the server, the prompt in the command window would say

```
[alvin@remote alvin]$
```

After alvin then su'd to become root, the prompt would say

```
[root@remote alvin]$
```

or if alvin did 'su -', the prompt would say

```
[root@remote ~]$
```

where the ~ denotes root's home directory, because the '-' incorporates the root user's environment and switches the current directory to the user's home directory, which would be /root in this case.

## 6.2 Logging in as a different account

If you want to log in as a specific user on the remote machine in a minimum number of keystrokes, you can do it in one of two ways. If you wanted to log in as the user bob on the remote machine, for example, you could

```
ssh -l bob www.hidbigo.com
```

or

```
ssh bob@www.hidbigo.com
```

where '-l' is the letter "ell", and where you could use the IP address instead of the hostname if you wished.

## 6.3 Testing SSH on a single machine

If you don't have a pair of machines to experiment with, you can log in to the local machine, using '127.0.0.1' or 'localhost' as the IP address or URL, like so:

```
ssh localhost
```

or

```
ssh 127.0.0.1
```

## 6.4 Restarting SSH remotely

You can restart SSH remotely by logging onto the remote machine via SSH, changing to the root account, and issuing the restart command. If you forget to switch to root first, you'll get a pair of errors, as shown in **Figure 12**.



```
whil@www:~  
File Edit View Terminal Go Help  
[whil@www whil]$ /etc/rc.d/init.d/ssh restart  
Stopping sshd:/etc/rc.d/init.d/ssh: line 212: kill: (10463) - Operation not permitted  
[FAILED]  
Starting sshd:/etc/ssh/ssh_config: Permission denied  
[FAILED]  
[whil@www whil]$
```

Figure 12. You can't restart SSH on the remote machine unless you're logged in as root.

Figure 13 shows a successful restart.



```
whil@www:/home/whil  
File Edit View Terminal Go Help  
[root@www whil]# /etc/rc.d/init.d/ssh restart  
Stopping sshd: [ OK ]  
Starting sshd: [ OK ]  
[root@www whil]#
```

Figure 13. When you are logged in as root on the remote machine, you can restart SSH.

At this point, it would probably be a good idea to try establishing another (brand new) SSH connection to the server before closing the current (original) connection. This way, if you messed up the configuration (indicated by the new connection attempt failing), you haven't locked yourself out, since you still have the original connection open and available to make changes.

## 6.5 Using scp (secure copy)

So you can make a secure connection to another machine, and then run programs on that remote machine from the comfort of your own easy chair. What else might you want to be able to do? Copy files back and forth, just as if that remote machine was just another locally accessible share. Why wouldn't you just use the 'cp' command? Because when you're on the remote box, 'cp' no longer knows anything about the client machine. When you're connected to a remote machine via ssh, the command window you're using is for all practical purposes a command window on that remote box, just as if you were sitting in front of it. Only you aren't. And so, if you were in front of the remote machine, you wouldn't be able to use the cp command to copy files back to the client box back on your desk, right?

Enter scp, the secure copy command. scp uses the SSH secure tunnel to transmit files in both directions.

scp, however, knows about both the client and the server – about both the local machine and the remote box. Here's how it works. Suppose you are on a local box, and you want to transfer a file from your local box, say, a new HTML page named itemlist.txt, to the remote machine, and keep the same name. Issue the scp command:

```
[whil@freedom ~] Dude? scp ./itemlist.txt whil@www.hidbigo.com:/home/somedir  
whil@www.hidbigo.com's password:  
itemlist.txt 100% 191 3.4MB/s 00:00  
[whil@freedom ~] Dude?
```

Note that the syntax of scp is similar to cp: command, source, and then target. When you are describing the remote machine, however, you need to include more than just the filename – you need to include the name of the host and the user you're connecting through (just like an ssh command) as well.

You'll be asked for whil's password on the remote machine, and assuming success in that area, voila, the file will be copied through the secure SSH tunnel.

## 7. Where to go for more information

The ssh command has both man and info pages (type "man ssh" and "info ssh" in a command window) as part of its inclusion in a Linux distribution. In addition, you may find the following Web sites useful.

<http://www.suso.org/linux/tutorials/ssh.phtml>

[http://wks.uts.ohio-state.edu/sysadm\\_course/html/sysadm-558.html](http://wks.uts.ohio-state.edu/sysadm_course/html/sysadm-558.html)

<http://www.openssh.com/>

## 8. About the author

Whil Hentzen started out life in the early '80's as a custom software developer using dBASE II (he still has the original 8 1/2 x 11 grey binder of documentation, much to the chagrin of his wife), and switched to FoxPro in 1990. Besides billing 15,000 hours in the 90's, he presented more than 70 papers at conferences throughout North America and Europe, edited FoxTalk, Pinnacle Publishing's high end technical journal for 7 years, hosted the Great Lakes Great Database Workshop since 1994. He's written 7 books and published 30 more on a variety of software development topics. He was a Microsoft Most Valuable Professional from 1995 through 2003 for his contributions to the FoxPro development community, and received the first Microsoft Lifetime Achievement Award for Visual FoxPro in 2001.

Whil began using Linux on the desktop when OpenOffice.org became a standard in the mainstream distributions, as it spelled potential for custom application development in the future, and has been a Linux user, developer, and evangelist ever since. His first book on Linux, *Linux Transfer for Windows Power Users*, was published in early 2004.

He is available for new and legacy Visual FoxPro application development as well as Web and desktop development on Linux.

## 9. A word from our sponsor

This free whitepaper is published and distributed by Hentzenwerke Publishing, Inc. We have the largest lists of "Moving to Linux", OpenOffice.org, and Visual FoxPro books on the planet.

We also have oodles of free whitepapers on our website and more are being added regularly. Our Preferred Customer mailing list gets bi-monthly announcements of new whitepapers (and gets discounts on our books, first crack at special deals, and other stuff as we think of it.)

Click on "Your Account" at [www.hentzenwerke.com](http://www.hentzenwerke.com) to get on our Preferred Customer list.

**If you found this whitepaper helpful, check out these Hentzenwerke Publishing books as well:**

**Linux Transfer for Windows® Network Admins:  
A roadmap for building a Linux file and print server  
Michael Jang**

**Linux Transfer for Windows® Power Users:  
Getting started with Linux for the desktop  
Whil Hentzen**