**5.2.4 Using** **public key authentication with PSCP**

Like PuTTY, PSCP can authenticate using a public key instead of a password. There are three ways you can do this.

Firstly, PSCP can use PuTTY saved sessions in place of hostnames (The name of the remote server, or the name of an existing PuTTY saved session. In the latter case, the session's settings for hostname, port number, cipher type and username will be used.). So you would do this:

* Run PuTTY, and create a PuTTY saved session (see [section 4.1.2](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-saving)) which specifies your private key file (see [section 4.20.8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-ssh-privkey)). You will probably also want to specify a username to log in as (All three of the SSH, Telnet and Rlogin protocols allow you to specify what user name you want to log in as, without having to type it explicitly every time. In this box you can type that user name.).
* In PSCP, you can now use the name of the session instead of a hostname: type pscp sessionname:file localfile, where sessionname is replaced by the name of your saved session.

Secondly, you can supply the name of a private key file on the command line, with the -i option. (The -i option allows you to specify the name of a private key file in \*.PPK format which PuTTY will use to authenticate with the server. This option is only meaningful if you are using SSH. For general information on public-key authentication, see [chapter 8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey). This option is equivalent to the ‘Private key file for authentication’ box in the Auth panel of the PuTTY configuration box)

Thirdly, PSCP will attempt to authenticate using Pageant if Pageant is running (see [chapter 9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant)). So you would do this:

* Ensure Pageant is running, and has your private key stored in it.
* Specify a user and host name to PSCP as normal. PSCP will automatically detect Pageant and try to use the keys within it.

For more general information on public-key authentication, see [chapter 8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey).

# Chapter 8: Using public keys for SSH authentication

## 8.1 Public key authentication - an introduction

Public key authentication is an alternative means of identifying yourself to a login server, instead of typing a password. It is more secure and more flexible, but more difficult to set up.

In conventional password authentication, you prove you are who you claim to be by proving that you know the correct password. The only way to prove you know the password is to tell the server what you think the password is. This means that if the server has been hacked, or spoofed (see [section 2.2](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter2.html#gs-hostkey)), an attacker can learn your password.

Public key authentication solves this problem. You generate a key pair, consisting of a public key (which everybody is allowed to know) and a private key (which you keep secret and do not give to anybody). The private key is able to generate signatures. A signature created using your private key cannot be forged by anybody who does not have that key; but anybody who has your public key can verify that a particular signature is genuine.

So you generate a key pair on your own computer, and you copy the public key to the server. Then, when the server asks you to prove who you are, PuTTY can generate a signature using your private key. The server can verify that signature (since it has your public key) and allow you to log in. Now if the server is hacked or spoofed, the attacker does not gain your private key or password; they only gain one signature. And signatures cannot be re-used, so they have gained nothing.

There is a problem with this: if your private key is stored unprotected on your own computer, then anybody who gains access to that will be able to generate signatures as if they were you. So they will be able to log in to your server under your account. For this reason, your private key is usually encrypted when it is stored on your local machine, using a passphrase of your choice. In order to generate a signature, PuTTY must decrypt the key, so you have to type your passphrase.

This can make public-key authentication less convenient than password authentication: every time you log in to the server, instead of typing a short password, you have to type a longer passphrase. One solution to this is to use an authentication agent, a separate program which holds decrypted private keys and generates signatures on request. PuTTY's authentication agent is called Pageant. When you begin a Windows session, you start Pageant and load your private key into it (typing your passphrase once). For the rest of your session, you can start PuTTY any number of times and Pageant will automatically generate signatures without you having to do anything. When you close your Windows session, Pageant shuts down, without ever having stored your decrypted private key on disk. Many people feel this is a good compromise between security and convenience. See [chapter 9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant) for further details.

There is more than one public-key algorithm available. The most common is RSA, but others exist, notably DSA (otherwise known as DSS), the USA's federal Digital Signature Standard. The key types supported by PuTTY are described in [section 8.2.2](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-keytype).

## 8.2 Using PuTTYgen, the PuTTY key generator

PuTTYgen is a key generator. It generates pairs of public and private keys to be used with PuTTY, PSCP, and Plink, as well as the PuTTY authentication agent, Pageant (see [chapter 9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant)). PuTTYgen generates RSA and DSA keys.

When you run PuTTYgen you will see a window where you have two choices: ‘Generate’, to generate a new public/private key pair, or ‘Load’ to load in an existing private key.

### 8.2.1 Generating a new key

This is a general outline of the procedure for generating a new key pair. The following sections describe the process in more detail.

* First, you need to select which type of key you want to generate, and also select the strength of the key. This is described in more detail in [section 8.2.2](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-keytype) and [section 8.2.3](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-strength).
* Then press the ‘Generate’ button, to actually generate the key. [Section 8.2.4](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-generate) describes this step.
* Once you have generated the key, select a comment field ([section 8.2.6](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-comment)) and a passphrase ([section 8.2.7](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-passphrase)).
* Now you're ready to save the private key to disk; press the ‘Save private key’ button. (See [section 8.2.8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-savepriv)).

Your key pair is now ready for use. You may also want to copy the public key to your server, either by copying it out of the ‘Public key for pasting into authorized\_keys file’ box (see [section 8.2.10](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-pastekey)), or by using the ‘Save public key’ button ([section 8.2.9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-savepub)). However, you don't need to do this immediately; if you want, you can load the private key back into PuTTYgen later (see [section 8.2.11](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-load)) and the public key will be available for copying and pasting again.

[Section 8.3](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey-gettingready) describes the typical process of configuring PuTTY to attempt public-key authentication, and configuring your SSH server to accept it.

### 8.2.2 Selecting the type of key

Before generating a key pair using PuTTYgen, you need to select which type of key you need. PuTTYgen currently supports three types of key:

* An RSA key for use with the SSH-1 protocol.
* An RSA key for use with the SSH-2 protocol.
* A DSA key for use with the SSH-2 protocol.

The SSH-1 protocol only supports RSA keys; if you will be connecting using the SSH-1 protocol, you must select the first key type or your key will be completely useless.

The SSH-2 protocol supports more than one key type. The two types supported by PuTTY are RSA and DSA.

The PuTTY developers strongly recommend you use RSA. DSA has an intrinsic weakness which makes it very easy to create a signature which contains enough information to give away the private key! This would allow an attacker to pretend to be you for any number of future sessions. PuTTY's implementation has taken very careful precautions to avoid this weakness, but we cannot be 100% certain we have managed it, and if you have the choice we strongly recommend using RSA keys instead.

If you really need to connect to an SSH server which only supports DSA, then you probably have no choice but to use DSA. If you do use DSA, we recommend you do not use the same key to authenticate with more than one server.

### 8.2.3 Selecting the size (strength) of the key

The ‘Number of bits’ input box allows you to choose the strength of the key PuTTYgen will generate.

Currently 1024 bits should be sufficient for most purposes.

### 8.2.4 The ‘Generate’ button

Once you have chosen the type of key you want, and the strength of the key, press the ‘Generate’ button and PuTTYgen will begin the process of actually generating the key.

First, a progress bar will appear and PuTTYgen will ask you to move the mouse around to generate randomness. Wave the mouse in circles over the blank area in the PuTTYgen window, and the progress bar will gradually fill up as PuTTYgen collects enough randomness. You don't need to wave the mouse in particularly imaginative patterns (although it can't hurt); PuTTYgen will collect enough randomness just from the fine detail of exactly how far the mouse has moved each time Windows samples its position.

When the progress bar reaches the end, PuTTYgen will begin creating the key. The progress bar will reset to the start, and gradually move up again to track the progress of the key generation. It will not move evenly, and may occasionally slow down to a stop; this is unfortunately unavoidable, because key generation is a random process and it is impossible to reliably predict how long it will take.

When the key generation is complete, a new set of controls will appear in the window to indicate this.

### 8.2.5 The ‘Key fingerprint’ box

The ‘Key fingerprint’ box shows you a fingerprint value for the generated key. This is derived cryptographically from the public key value, so it doesn't need to be kept secret.

The fingerprint value is intended to be cryptographically secure, in the sense that it is computationally infeasible for someone to invent a second key with the same fingerprint, or to find a key with a particular fingerprint. So some utilities, such as the Pageant key list box (see [section 9.2.1](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant-mainwin-keylist)) and the Unix ssh-add utility, will list key fingerprints rather than the whole public key.

### 8.2.6 Setting a comment for your key

If you have more than one key and use them for different purposes, you don't need to memorise the key fingerprints in order to tell them apart. PuTTYgen allows you to enter a comment for your key, which will be displayed whenever PuTTY or Pageant asks you for the passphrase.

The default comment format, if you don't specify one, contains the key type and the date of generation, such as rsa-key-20011212. Another commonly used approach is to use your name and the name of the computer the key will be used on, such as simon@simons-pc.

To alter the key comment, just type your comment text into the ‘Key comment’ box before saving the private key. If you want to change the comment later, you can load the private key back into PuTTYgen, change the comment, and save it again.

### 8.2.7 Setting a passphrase for your key

The ‘Key passphrase’ and ‘Confirm passphrase’ boxes allow you to choose a passphrase for your key. The passphrase will be used to encrypt the key on disk, so you will not be able to use the key without first entering the passphrase.

When you save the key, PuTTYgen will check that the ‘Key passphrase’ and ‘Confirm passphrase’ boxes both contain exactly the same passphrase, and will refuse to save the key otherwise.

If you leave the passphrase fields blank, the key will be saved unencrypted. You should not do this without good reason; if you do, your private key file on disk will be all an attacker needs to gain access to any machine configured to accept that key. If you want to be able to log in without having to type a passphrase every time, you should consider using Pageant ([chapter 9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant)) so that your decrypted key is only held in memory rather than on disk.

Under special circumstances you may genuinely need to use a key with no passphrase; for example, if you need to run an automated batch script that needs to make an SSH connection, you can't be there to type the passphrase. In this case we recommend you generate a special key for each specific batch script (or whatever) that needs one, and on the server side you should arrange that each key is restricted so that it can only be used for that specific purpose. The documentation for your SSH server should explain how to do this (it will probably vary between servers).

Choosing a good passphrase is difficult. Just as you shouldn't use a dictionary word as a password because it's easy for an attacker to run through a whole dictionary, you should not use a song lyric, quotation or other well-known sentence as a passphrase. DiceWare ([www.diceware.com](http://www.diceware.com/)) recommends using at least five words each generated randomly by rolling five dice, which gives over 2^64 possible passphrases and is probably not a bad scheme. If you want your passphrase to make grammatical sense, this cuts down the possibilities a lot and you should use a longer one as a result.

Do not forget your passphrase. There is no way to recover it.

### 8.2.8 Saving your private key to a disk file

Once you have generated a key, set a comment field and set a passphrase, you are ready to save your private key to disk.

Press the ‘Save private key’ button. PuTTYgen will put up a dialog box asking you where to save the file. Select a directory, type in a file name, and press ‘Save’.

This file is in PuTTY's native format (\*.PPK); it is the one you will need to tell PuTTY to use for authentication (see [section 4.20.8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-ssh-privkey)) or tell Pageant to load (see [section 9.2.2](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant-mainwin-addkey)).

### 8.2.9 Saving your public key to a disk file

RFC 4716 specifies a standard format for storing SSH-2 public keys on disk. Some SSH servers (such as ssh.com's) require a public key in this format in order to accept authentication with the corresponding private key. (Others, such as OpenSSH, use a different format; see [section 8.2.10](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-pastekey).)

To save your public key in the SSH-2 standard format, press the ‘Save public key’ button in PuTTYgen. PuTTYgen will put up a dialog box asking you where to save the file. Select a directory, type in a file name, and press ‘Save’.

You will then probably want to copy the public key file to your SSH server machine. See [section 8.3](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey-gettingready) for general instructions on configuring public-key authentication once you have generated a key.

If you use this option with an SSH-1 key, the file PuTTYgen saves will contain exactly the same text that appears in the ‘Public key for pasting’ box. This is the only existing standard for SSH-1 public keys.

### 8.2.10 ‘Public key for pasting into authorized\_keys file’

All SSH-1 servers require your public key to be given to it in a one-line format before it will accept authentication with your private key. The OpenSSH server also requires this for SSH-2.

The ‘Public key for pasting into authorized\_keys file’ gives the public-key data in the correct one-line format. Typically you will want to select the entire contents of the box using the mouse, press Ctrl+C to copy it to the clipboard, and then paste the data into a PuTTY session which is already connected to the server.

See [section 8.3](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey-gettingready) for general instructions on configuring public-key authentication once you have generated a key.

### 8.2.11 Reloading a private key

PuTTYgen allows you to load an existing private key file into memory. If you do this, you can then change the passphrase and comment before saving it again; you can also make extra copies of the public key.

To load an existing key, press the ‘Load’ button. PuTTYgen will put up a dialog box where you can browse around the file system and find your key file. Once you select the file, PuTTYgen will ask you for a passphrase (if necessary) and will then display the key details in the same way as if it had just generated the key.

If you use the Load command to load a foreign key format, it will work, but you will see a message box warning you that the key you have loaded is not a PuTTY native key. See [section 8.2.12](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-conversions) for information about importing foreign key formats.

### 8.2.12 Dealing with private keys in other formats

Most SSH-1 clients use a standard format for storing private keys on disk. PuTTY uses this format as well; so if you have generated an SSH-1 private key using OpenSSH or ssh.com's client, you can use it with PuTTY, and vice versa.

However, SSH-2 private keys have no standard format. OpenSSH and ssh.com have different formats, and PuTTY's is different again. So a key generated with one client cannot immediately be used with another.

Using the ‘Import’ command from the ‘Conversions’ menu, PuTTYgen can load SSH-2 private keys in OpenSSH's format and ssh.com's format. Once you have loaded one of these key types, you can then save it back out as a PuTTY-format key (\*.PPK) so that you can use it with the PuTTY suite. The passphrase will be unchanged by this process (unless you deliberately change it). You may want to change the key comment before you save the key, since OpenSSH's SSH-2 key format contains no space for a comment and ssh.com's default comment format is long and verbose.

PuTTYgen can also export private keys in OpenSSH format and in ssh.com format. To do so, select one of the ‘Export’ options from the ‘Conversions’ menu. Exporting a key works exactly like saving it (see [section 8.2.8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-savepriv)) - you need to have typed your passphrase in beforehand, and you will be warned if you are about to save a key without a passphrase.

Note that since only SSH-2 keys come in different formats, the export options are not available if you have generated an SSH-1 key.

## 8.3 Getting ready for public key authentication

Connect to your SSH server using PuTTY with the SSH protocol. When the connection succeeds you will be prompted for your user name and password to login. Once logged in, you must configure the server to accept your public key for authentication:

* If your server is using the SSH-1 protocol, you should change into the .ssh directory and open the file authorized\_keys with your favourite editor. (You may have to create this file if this is the first key you have put in it). Then switch to the PuTTYgen window, select all of the text in the ‘Public key for pasting into authorized\_keys file’ box (see [section 8.2.10](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-pastekey)), and copy it to the clipboard (Ctrl+C). Then, switch back to the PuTTY window and insert the data into the open file, making sure it ends up all on one line. Save the file.
* If your server is OpenSSH and is using the SSH-2 protocol, you should follow the same instructions, except that in earlier versions of OpenSSH 2 the file might be called authorized\_keys2. (In modern versions the same authorized\_keys file is used for both SSH-1 and SSH-2 keys.)
* If your server is ssh.com's product and is using SSH-2, you need to save a public key file from PuTTYgen (see [section 8.2.9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#puttygen-savepub)), and copy that into the .ssh2 directory on the server. Then you should go into that .ssh2 directory, and edit (or create) a file called authorization. In this file you should put a line like Key mykey.pub, with mykey.pub replaced by the name of your key file.
* For other SSH server software, you should refer to the manual for that server.

You may also need to ensure that your home directory, your .ssh directory, and any other files involved (such as authorized\_keys, authorized\_keys2 or authorization) are not group-writable or world-writable. You can typically do this by using a command such as

chmod go-w $HOME $HOME/.ssh $HOME/.ssh/authorized\_keys

Your server should now be configured to accept authentication using your private key. Now you need to configure PuTTY to attempt authentication using your private key. You can do this in any of three ways:

* Select the private key in PuTTY's configuration. See [section 4.20.8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-ssh-privkey) for details.
* Specify the key file on the command line with the -i option. See [section 3.8.3.18](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter3.html#using-cmdline-identity) for details.
* Load the private key into Pageant (see [chapter 9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant)). In this case PuTTY will automatically try to use it for authentication if it can.

# Chapter 9: Using Pageant for authentication

Pageant is an SSH authentication agent. It holds your private keys in memory, already decoded, so that you can use them often without needing to type a passphrase.

## 9.1 Getting started with Pageant

Before you run Pageant, you need to have a private key in \*.PPK format. See [chapter 8](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter8.html#pubkey) to find out how to generate and use one.

When you run Pageant, it will put an icon of a computer wearing a hat into the System tray. It will then sit and do nothing, until you load a private key into it.

If you click the Pageant icon with the right mouse button, you will see a menu. Select ‘View Keys’ from this menu. The Pageant main window will appear. (You can also bring this window up by double-clicking on the Pageant icon.)

The Pageant window contains a list box. This shows the private keys Pageant is holding. When you start Pageant, it has no keys, so the list box will be empty. After you add one or more keys, they will show up in the list box.

To add a key to Pageant, press the ‘Add Key’ button. Pageant will bring up a file dialog, labelled ‘Select Private Key File’. Find your private key file in this dialog, and press ‘Open’.

Pageant will now load the private key. If the key is protected by a passphrase, Pageant will ask you to type the passphrase. When the key has been loaded, it will appear in the list in the Pageant window.

Now start PuTTY and open an SSH session to a site that accepts your key. PuTTY will notice that Pageant is running, retrieve the key automatically from Pageant, and use it to authenticate. You can now open as many PuTTY sessions as you like without having to type your passphrase again.

(PuTTY can be configured not to try to use Pageant, but it will try by default. See [section 4.20.3](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-ssh-tryagent) and [section 3.8.3.9](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter3.html#using-cmdline-agentauth) for more information.)

When you want to shut down Pageant, click the right button on the Pageant icon in the System tray, and select ‘Exit’ from the menu. Closing the Pageant main window does not shut down Pageant.

## 9.2 The Pageant main window

The Pageant main window appears when you left-click on the Pageant system tray icon, or alternatively right-click and select ‘View Keys’ from the menu. You can use it to keep track of what keys are currently loaded into Pageant, and to add new ones or remove the existing keys.

### 9.2.1 The key list box

The large list box in the Pageant main window lists the private keys that are currently loaded into Pageant. The list might look something like this:

ssh1 1024 22:c3:68:3b:09:41:36:c3:39:83:91:ae:71:b2:0f:04 k1

ssh-rsa 1023 74:63:08:82:95:75:e1:7c:33:31:bb:cb:00:c0:89:8b k2

For each key, the list box will tell you:

* The type of the key. Currently, this can be ssh1 (an RSA key for use with the SSH-1 protocol), ssh-rsa (an RSA key for use with the SSH-2 protocol), or ssh-dss (a DSA key for use with the SSH-2 protocol).
* The size (in bits) of the key.
* The fingerprint for the public key. This should be the same fingerprint given by PuTTYgen, and (hopefully) also the same fingerprint shown by remote utilities such as ssh-keygen when applied to your authorized\_keys file.
* The comment attached to the key.

### 9.2.2 The ‘Add Key’ button

To add a key to Pageant by reading it out of a local disk file, press the ‘Add Key’ button in the Pageant main window, or alternatively right-click on the Pageant icon in the system tray and select ‘Add Key’ from there.

Pageant will bring up a file dialog, labelled ‘Select Private Key File’. Find your private key file in this dialog, and press ‘Open’. If you want to add more than one key at once, you can select multiple files using Shift-click (to select several adjacent files) or Ctrl-click (to select non-adjacent files).

Pageant will now load the private key(s). If a key is protected by a passphrase, Pageant will ask you to type the passphrase.

(This is not the only way to add a private key to Pageant. You can also add one from a remote system by using agent forwarding; see [section 9.4](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant-forward) for details.)

### 9.2.3 The ‘Remove Key’ button

If you need to remove a key from Pageant, select that key in the list box, and press the ‘Remove Key’ button. Pageant will remove the key from its memory.

You can apply this to keys you added using the ‘Add Key’ button, or to keys you added remotely using agent forwarding (see [section 9.4](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter9.html#pageant-forward)); it makes no difference.

## 9.3 The Pageant command line

Pageant can be made to do things automatically when it starts up, by specifying instructions on its command line. If you're starting Pageant from the Windows GUI, you can arrange this by editing the properties of the Windows shortcut that it was started from.

If Pageant is already running, invoking it again with the options below causes actions to be performed with the existing instance, not a new one.

### 9.3.1 Making Pageant automatically load keys on startup

Pageant can automatically load one or more private keys when it starts up, if you provide them on the Pageant command line. Your command line might then look like:

C:\PuTTY\pageant.exe d:\main.ppk d:\secondary.ppk

If the keys are stored encrypted, Pageant will request the passphrases on startup.

If Pageant is already running, this syntax loads keys into the existing Pageant.

### 9.3.2 Making Pageant run another program

You can arrange for Pageant to start another program once it has initialised itself and loaded any keys specified on its command line. This program (perhaps a PuTTY, or a WinCVS making use of Plink, or whatever) will then be able to use the keys Pageant has loaded.

You do this by specifying the -c option followed by the command, like this:

C:\PuTTY\pageant.exe d:\main.ppk -c C:\PuTTY\putty.exe

## 9.4 Using agent forwarding

Agent forwarding is a mechanism that allows applications on your SSH server machine to talk to the agent on your client machine.

Note that at present, agent forwarding in SSH-2 is only available when your SSH server is OpenSSH. The ssh.com server uses a different agent protocol, which PuTTY does not yet support.

To enable agent forwarding, first start Pageant. Then set up a PuTTY SSH session in which ‘Allow agent forwarding’ is enabled (see [section 4.20.6](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter4.html#config-ssh-agentfwd)). Open the session as normal. (Alternatively, you can use the -A command line option; see [section 3.8.3.10](http://tartarus.org/%7Esimon/putty-snapshots/htmldoc/Chapter3.html#using-cmdline-agent) for details.)

If this has worked, your applications on the server should now have access to a Unix domain socket which the SSH server will forward back to PuTTY, and PuTTY will forward on to the agent. To check that this has actually happened, you can try this command on Unix server machines:

unixbox:~$ echo $SSH\_AUTH\_SOCK

/tmp/ssh-XXNP18Jz/agent.28794

unixbox:~$

If the result line comes up blank, agent forwarding has not been enabled at all.

Now if you run ssh on the server and use it to connect through to another server that accepts one of the keys in Pageant, you should be able to log in without a password:

unixbox:~$ ssh -v otherunixbox

[...]

debug: next auth method to try is publickey

debug: userauth\_pubkey\_agent: trying agent key my-putty-key

debug: ssh-userauth2 successful: method publickey

[...]

If you enable agent forwarding on that SSH connection as well (see the manual for your server-side SSH client to find out how to do this), your authentication keys will still be available on the next machine you connect to - two SSH connections away from where they're actually stored.

In addition, if you have a private key on one of the SSH servers, you can send it all the way back to Pageant using the local ssh-add command:

unixbox:~$ ssh-add ~/.ssh/id\_rsa

Need passphrase for /home/fred/.ssh/id\_rsa

Enter passphrase for /home/fred/.ssh/id\_rsa:

Identity added: /home/fred/.ssh/id\_rsa (/home/simon/.ssh/id\_rsa)

unixbox:~$

and then it's available to every machine that has agent forwarding available (not just the ones downstream of the place you added it).

## 9.5 Security considerations

Using Pageant for public-key authentication gives you the convenience of being able to open multiple SSH sessions without having to type a passphrase every time, but also gives you the security benefit of never storing a decrypted private key on disk. Many people feel this is a good compromise between security and convenience.

It is a compromise, however. Holding your decrypted private keys in Pageant is better than storing them in easy-to-find disk files, but still less secure than not storing them anywhere at all. This is for two reasons:

* Windows unfortunately provides no way to protect pieces of memory from being written to the system swap file. So if Pageant is holding your private keys for a long period of time, it's possible that decrypted private key data may be written to the system swap file, and an attacker who gained access to your hard disk later on might be able to recover that data. (However, if you stored an unencrypted key in a disk file they would certainly be able to recover it.)
* Although, like most modern operating systems, Windows prevents programs from accidentally accessing one another's memory space, it does allow programs to access one another's memory space deliberately, for special purposes such as debugging. This means that if you allow a virus, trojan, or other malicious program on to your Windows system while Pageant is running, it could access the memory of the Pageant process, extract your decrypted authentication keys, and send them back to its master.

Similarly, use of agent forwarding is a security improvement on other methods of one-touch authentication, but not perfect. Holding your keys in Pageant on your Windows box has a security advantage over holding them on the remote server machine itself (either in an agent or just unencrypted on disk), because if the server machine ever sees your unencrypted private key then the sysadmin or anyone who cracks the machine can steal the keys and pretend to be you for as long as they want.

However, the sysadmin of the server machine can always pretend to be you on that machine. So if you forward your agent to a server machine, then the sysadmin of that machine can access the forwarded agent connection and request signatures from your private keys, and can therefore log in to other machines as you. They can only do this to a limited extent - when the agent forwarding disappears they lose the ability - but using Pageant doesn't actually prevent the sysadmin (or hackers) on the server from doing this.

Therefore, if you don't trust the sysadmin of a server machine, you should never use agent forwarding to that machine. (Of course you also shouldn't store private keys on that machine, type passphrases into it, or log into other machines from it in any way at all; Pageant is hardly unique in this respect.)

**Install freedomotic on Raspberry Pi (first untested draft)**

**STEP 1: Download Raspberry Debian Wheezy image**

wget http://ftp.snt.utwente.nl/pub/software/rpi/images/debian/7/2012-08-08-wheezy-armel/2012-08-08-wheezy-armel.zip  
unzip 2012-08-08-wheezy-armel.zip

**STEP 2: Copy the image on the SD card**

Plug SD card (2GB or 4GB) and take note of the related device (eg: using dmesg command)

sudo dd if=2012-08-08-wheezy-armel.img of=SD\_DEVICE

**WARNING**: be sure the **of** (output file) is the SD card otherwise you can mess up your hard drive. This operation will take at least 10 minutes and will not print any output until the end.

**STEP 3: Download and install Freedomotic stable version**

wget http://freedomotic.googlecode.com/files/Freedomotic\_v5.3.0.zip

To be sure unplug and replug the SD card. Execute these commands

cp Freedomotic\_v5.3.0.zip /media/SD\_DEVICE\_LABEL/home/pi/Desktop  
cd /media/SD\_DEVICE\_LABEL/home/pi/Desktop  
unzip Freedomotic\_v5.3.0.zip

**NOTE**: say yes to all if it asks to replace files

**STEP 4: Running Raspberry Pi**

Umount the SD card, insert it into your raspberry pi and turn it on.

**STEP 5: Download and install Oracle Java SE Embedded**

Follow these guide <https://blogs.oracle.com/hinkmond/entry/quickie_guide_getting_java_embedded> <http://www.oracle.com/technetwork/java/embedded/downloads/javase/index.html>

**NOTE:** First click accept, then choose the third bundle in the list:

ARMv7 Linux - Headful EABI, VFP, SoftFP ABI, Little Endian ejre-7u6-fcs-b24-linux-arm-vfp-client\_headful-10\_aug\_2012.tar.gz

If you download the package on the pc you must before **scp** the bundle to your RPi:

   scp <ejre-bundle> pi@<ip\_addr\_rpi>:/home/pi

From Raspberry untar the bundle and rename (move) the **ejre1.7.0\_06** directory to **/usr/local/java**

   cd /home/pi  
   tar zxvf ejre-7u6-fcs-b24-linux-arm-vfp-client\_headful-10\_aug\_2012.tar.gz  
   sudo mv ejre1.7.0\_06 /usr/local/java  
   export PATH=$PATH:/usr/local/java/bin

To verify if all is ok digit

   java -version

You should see

     java version "1.7.0\_06"  
     Java(TM) SE Embedded Runtime Environment (build 1.7.0\_06-b24, headless)  
     Java HotSpot(TM) Embedded Client VM (build 23.2-b09, mixed mode)

**STEP 6: Start Freedomotic**

**From Raspberry:**

run the **/home/pi/Desktop/freeodmotic/start-freedom.sh** script

You can also use

java -jar freedomotic.jar

**From another pc using ssh:**

set the display variable on the Raspberry (needed for graphical interface)

export DISPLAY =:0.0

connect to the Raspberry by **ssh**

ssh -X pi@ip-address-raspberry

run the **start-freedom.sh** script in **/home/pi/Desktop/freeodmotic/start-freedom.sh**.

**NOTE: to follow and contribute to the development** [**join the group**](http://www.freedomotic.com/content/freedomotic-raspberry)













