

PuTTY User Manual

PuTTY is a free (MIT-licensed) Windows Telnet and SSH client. This manual documents PuTTY, and its companion utilities PSCP, PSFTP, Plink, Pageant and PuTTYgen.

Note to Unix users: this manual currently primarily documents the Windows versions of the PuTTY utilities. Some options are therefore mentioned that are absent from the Unix version; the Unix version has features not described here; and the pterm and command-line puttygen and pageant utilities are not described at all. The only Unix-specific documentation that currently exists is the man pages.

This manual is copyright 1997-2024 Simon Tatham. All rights reserved. You may distribute this documentation under the MIT licence. See [appendix D](#) for the licence text in full.

- [Chapter 1: Introduction to PuTTY](#)
 - [1.1 What are SSH, Telnet, Rlogin, and SUPDUP?](#)
 - [1.2 How do SSH, Telnet, Rlogin, and SUPDUP differ?](#)
- [Chapter 2: Getting started with PuTTY](#)
 - [2.1 Starting a session](#)
 - [2.2 Verifying the host key \(SSH only\)](#)
 - [2.3 Logging in](#)
 - [2.4 After logging in](#)
 - [2.5 Logging out](#)
- [Chapter 3: Using PuTTY](#)
 - [3.1 During your session](#)
 - [3.2 Creating a log file of your session](#)
 - [3.3 Altering your character set configuration](#)
 - [3.4 Using X11 forwarding in SSH](#)
 - [3.5 Using port forwarding in SSH](#)
 - [3.6 Connecting to a local serial line](#)
 - [3.7 Making raw TCP connections](#)

- [3.8 Connecting using the Telnet protocol](#)
 - [3.9 Connecting using the Rlogin protocol](#)
 - [3.10 Connecting using the SUPDUP protocol](#)
 - [3.11 The PuTTY command line](#)
- [Chapter 4: Configuring PuTTY](#)
 - [4.1 The Session panel](#)
 - [4.2 The Logging panel](#)
 - [4.3 The Terminal panel](#)
 - [4.4 The Keyboard panel](#)
 - [4.5 The Bell panel](#)
 - [4.6 The Features panel](#)
 - [4.7 The Window panel](#)
 - [4.8 The Appearance panel](#)
 - [4.9 The Behaviour panel](#)
 - [4.10 The Translation panel](#)
 - [4.11 The Selection panel](#)
 - [4.12 The Copy panel](#)
 - [4.13 The Colours panel](#)
 - [4.14 The Connection panel](#)
 - [4.15 The Data panel](#)
 - [4.16 The Proxy panel](#)
 - [4.17 The SSH panel](#)
 - [4.18 The Kex panel](#)
 - [4.19 The Host Keys panel](#)
 - [4.20 The Cipher panel](#)
 - [4.21 The Auth panel](#)
 - [4.22 The Credentials panel](#)
 - [4.23 The GSSAPI panel](#)
 - [4.24 The TTY panel](#)
 - [4.25 The X11 panel](#)
 - [4.26 The Tunnels panel](#)
 - [4.27 The Bugs and More Bugs panels](#)
 - [4.28 The 'Bare ssh-connection' protocol](#)
 - [4.29 The Serial panel](#)
 - [4.30 The Telnet panel](#)
 - [4.31 The Rlogin panel](#)
 - [4.32 The SUPDUP panel](#)

- [4.33 Storing configuration in a file](#)
- [Chapter 5: Using PSCP to transfer files securely](#)
 - [5.1 Starting PSCP](#)
 - [5.2 PSCP Usage](#)
- [Chapter 6: Using PSFTP to transfer files securely](#)
 - [6.1 Starting PSFTP](#)
 - [6.2 Running PSFTP](#)
 - [6.3 Using public key authentication with PSFTP](#)
- [Chapter 7: Using the command-line connection tool Plink](#)
 - [7.1 Starting Plink](#)
 - [7.2 Using Plink](#)
 - [7.3 Using Plink in batch files and scripts](#)
 - [7.4 Using Plink with CVS](#)
 - [7.5 Using Plink with WinCVS](#)
- [Chapter 8: Using public keys for SSH authentication](#)
 - [8.1 Public key authentication - an introduction](#)
 - [8.2 Using PuTTYgen, the PuTTY key generator](#)
 - [8.3 Getting ready for public key authentication](#)
- [Chapter 9: Using Pageant for authentication](#)
 - [9.1 Getting started with Pageant](#)
 - [9.2 The Pageant main window](#)
 - [9.3 The Pageant command line](#)
 - [9.4 Using agent forwarding](#)
 - [9.5 Loading keys without decrypting them](#)
 - [9.6 Security considerations](#)
- [Chapter 10: Common error messages](#)
 - [10.1 'The host key is not cached for this server'](#)
 - [10.2 'WARNING - POTENTIAL SECURITY BREACH!'](#)
 - [10.3 'This server presented a certified host key which was signed by a different certification authority ...'](#)
 - [10.4 'SSH protocol version 2 required by our configuration but remote only provides \(old, insecure\) SSH-1'](#)
 - [10.5 'The first cipher supported by the server is ... below the configured warning threshold'](#)
 - [10.6 'Remote side sent disconnect message type 2 \(protocol error\): "Too many authentication failures for root"'](#)
 - [10.7 'Out of memory'](#)

- [10.8 ‘Internal error’, ‘Internal fault’, ‘Assertion failed’](#)
 - [10.9 ‘Unable to use key file’, ‘Couldn’t load private key’, ‘Couldn’t load this key’](#)
 - [10.10 ‘Server refused our key’, ‘Server refused our public key’, ‘Key refused’](#)
 - [10.11 ‘Access denied’, ‘Authentication refused’](#)
 - [10.12 ‘No supported authentication methods available’](#)
 - [10.13 ‘Incorrect MAC received on packet’ or ‘Incorrect CRC received on packet’](#)
 - [10.14 ‘Incoming packet was garbled on decryption’](#)
 - [10.15 ‘PuTTY X11 proxy: various errors’](#)
 - [10.16 ‘Network error: Software caused connection abort’](#)
 - [10.17 ‘Network error: Connection reset by peer’](#)
 - [10.18 ‘Network error: Connection refused’](#)
 - [10.19 ‘Network error: Connection timed out’](#)
 - [10.20 ‘Network error: Cannot assign requested address’](#)
- [Appendix A: PuTTY FAQ](#)
 - [A.1 Introduction](#)
 - [A.2 Features supported in PuTTY](#)
 - [A.3 Ports to other operating systems](#)
 - [A.4 Embedding PuTTY in other programs](#)
 - [A.5 Details of PuTTY’s operation](#)
 - [A.6 HOWTO questions](#)
 - [A.7 Troubleshooting](#)
 - [A.8 Security questions](#)
 - [A.9 Administrative questions](#)
 - [A.10 Miscellaneous questions](#)
- [Appendix B: Feedback and bug reporting](#)
 - [B.1 General guidelines](#)
 - [B.2 Reporting bugs](#)
 - [B.3 Reporting security vulnerabilities](#)
 - [B.4 Requesting extra features](#)
 - [B.5 Requesting features that have already been requested](#)
 - [B.6 Workarounds for SSH server bugs](#)
 - [B.7 Support requests](#)
 - [B.8 Web server administration](#)
 - [B.9 Asking permission for things](#)

- [B.10 Mirroring the PuTTY web site](#)
 - [B.11 Praise and compliments](#)
 - [B.12 E-mail address](#)
- [Appendix C: PPK file format](#)
 - [C.1 Overview](#)
 - [C.2 Outer layer](#)
 - [C.3 Private key encodings](#)
 - [C.4 Key derivation](#)
 - [C.5 Older versions of the PPK format](#)
- [Appendix D: PuTTY Licence](#)
- [Appendix E: PuTTY hacking guide](#)
 - [E.1 Cross-OS portability](#)
 - [E.2 Multiple backends treated equally](#)
 - [E.3 Multiple sessions per process on some platforms](#)
 - [E.4 C, not C++](#)
 - [E.5 Security-conscious coding](#)
 - [E.6 Independence of specific compiler](#)
 - [E.7 Small code size](#)
 - [E.8 Single-threaded code](#)
 - [E.9 Keystrokes sent to the server wherever possible](#)
 - [E.10 640×480 friendliness in configuration panels](#)
 - [E.11 Coroutines in protocol code](#)
 - [E.12 Explicit vtable structures to implement traits](#)
 - [E.13 Do as we say, not as we do](#)
- [Appendix F: PuTTY download keys and signatures](#)
 - [F.1 Public keys](#)
 - [F.2 Security details](#)
 - [F.3 Key rollover](#)
- [Appendix G: SSH-2 names specified for PuTTY](#)
 - [G.1 Connection protocol channel request names](#)
 - [G.2 Key exchange method names](#)
 - [G.3 Encryption algorithm names](#)
 - [G.4 Agent extension request names](#)
- [Appendix H: PuTTY authentication plugin protocol](#)
 - [H.1 Requirements](#)
 - [H.2 Transport and configuration](#)
 - [H.3 Data formats and marshalling](#)

- [H.4 Protocol versioning](#)
- [H.5 Overview and sequence of events](#)
- [H.6 Message formats](#)
- [H.7 References](#)

Chapter 1: Introduction to PuTTY

PuTTY is a free SSH, Telnet, Rlogin, and SUPDUP client for Windows systems.

- [1.1 What are SSH, Telnet, Rlogin, and SUPDUP?](#)
- [1.2 How do SSH, Telnet, Rlogin, and SUPDUP differ?](#)

1.1 What are SSH, Telnet, Rlogin, and SUPDUP?

If you already know what SSH, Telnet, Rlogin, and SUPDUP are, you can safely skip on to the next section.

SSH, Telnet, Rlogin, and SUPDUP are four ways of doing the same thing: logging in to a multi-user computer from another computer, over a network.

Multi-user operating systems, typically of the Unix family (such as Linux, MacOS, and the BSD family), usually present a command-line interface to the user, much like the ‘Command Prompt’ or ‘MS-DOS Prompt’ in Windows. The system prints a prompt, and you type commands which the system will obey.

Using this type of interface, there is no need for you to be sitting at the same machine you are typing commands to. The commands, and responses, can be sent over a network, so you can sit at one computer and give commands to another one, or even to more than one.

SSH, Telnet, Rlogin, and SUPDUP are *network protocols* that allow you to do this. On the computer you sit at, you run a *client*, which makes a network connection to the other computer (the *server*). The network connection carries your keystrokes and commands from the client to the server, and carries the server’s responses back to you.

These protocols can also be used for other types of keyboard-based interactive session. In particular, there are a lot of bulletin boards, talker systems and MUDs (Multi-User Dungeons) which support access using Telnet. There are even a few that support SSH.

You might want to use SSH, Telnet, Rlogin, or SUPDUP if:

- you have an account on a Unix system (or some other multi-user OS such as VMS or ITS) which you want to be able to access from somewhere else
- your Internet Service Provider provides you with a login account on a web server. (This might also be known as a *shell account*. A *shell* is the program that runs on the server and interprets your commands for you.)
- you want to use a bulletin board system, talker or MUD which can be accessed using Telnet.

You probably do *not* want to use SSH, Telnet, Rlogin, or SUPDUP if:

- you only use Windows. Windows computers have their own ways of networking between themselves, and unless you are doing something fairly unusual, you will not need to use any of these remote login protocols.

1.2 How do SSH, Telnet, Rlogin, and SUPDUP differ?

This list summarises some of the differences between SSH, Telnet, Rlogin, and SUPDUP.

- SSH (which stands for ‘secure shell’) is a recently designed, high-security protocol. It uses strong cryptography to protect your connection against eavesdropping, hijacking and other attacks. Telnet, Rlogin, and SUPDUP are all older protocols offering minimal security.
- SSH and Rlogin both allow you to log in to the server without having to type a password. (Rlogin’s method of doing this is insecure, and can allow an attacker to access your account on the server. SSH’s method is much more secure, and typically breaking the security requires the attacker to have gained access to your actual client machine.)
- SSH allows you to connect to the server and automatically send a command, so that the server will run that command and then disconnect. So you can use it in automated processing.

The Internet is a hostile environment and security is everybody’s responsibility. If you are connecting across the open Internet, then we recommend you use SSH. If the server you want to connect to doesn’t support SSH, it might be worth trying to persuade the administrator to install it.

If your client and server are both behind the same (good) firewall, it is more likely to be safe to use Telnet, Rlogin, or SUPDUP, but we still recommend you use SSH.

Chapter 2: Getting started with PuTTY

This chapter gives a quick guide to the simplest types of interactive login session using PuTTY.

- [2.1 Starting a session](#)
- [2.2 Verifying the host key \(SSH only\)](#)
- [2.3 Logging in](#)
- [2.4 After logging in](#)
- [2.5 Logging out](#)

2.1 Starting a session

When you start PuTTY, you will see a dialog box. This dialog box allows you to control everything PuTTY can do. See [chapter 4](#) for details of all the things you can control.

You don't usually need to change most of the configuration options. To start the simplest kind of session, all you need to do is to enter a few basic parameters.

In the 'Host Name' box, enter the Internet host name of the server you want to connect to. You should have been told this by the provider of your login account.

Now select a login protocol to use, from the 'Connection type' controls. For a login session, you should select SSH, Telnet, Rlogin, or SUPDUP. See [section 1.2](#) for a description of the differences between these protocols, and advice on which one to use. The *Raw* protocol is not used for interactive login sessions; you would usually use this for debugging other Internet services (see [section 3.7](#)). The *Serial* option is used for connecting to a local serial line, and works somewhat differently: see [section 3.6](#) for more information on this.

When you change the selected protocol, the number in the 'Port' box will change. This is normal: it happens because the various login services are usually provided on different network ports by the server machine. Most servers will use the standard port numbers, so you will not need to change the port setting. If your server provides login services on a non-standard port, your system administrator should have told you which one. (For example, many MUDs run Telnet service on a port other than 23.)

Once you have filled in the 'Host Name', 'Connection type', and possibly 'Port' settings, you are ready to connect. Press the 'Open' button at the bottom of the dialog box, and PuTTY will begin trying to connect you to the server.

2.2 Verifying the host key (SSH only)

If you are not using the SSH protocol, you can skip this section.

If you are using SSH to connect to a server for the first time, you will probably see a message looking something like this:

```
The host key is not cached for this server:  
ssh.example.com (port 22)  
You have no guarantee that the server is the computer you think  
it is.  
The server's ssh-ed25519 key fingerprint is:  
ssh-ed25519 255  
SHA256:TddlQk20DVs4LRcAsIfDN9pInKpY06D+h4kSHwWAj4w  
If you trust this host, press "Accept" to add the key to  
PuTTY's  
cache and carry on connecting.  
If you want to carry on connecting just once, without adding  
the key  
to the cache, press "Connect Once".  
If you do not trust this host, press "Cancel" to abandon the  
connection.
```

This is a feature of the SSH protocol. It is designed to protect you against a network attack known as *spoofing*: secretly redirecting your connection to a different computer, so that you send your password to the wrong machine. Using this technique, an attacker would be able to learn the password that guards your login account, and could then log in as if they were you and use the account for their own purposes.

To prevent this attack, each server has a unique identifying code, called a *host key*. These keys are created in a way that prevents one server from forging another server's key. So if you connect to a server and it sends you a different host key from the one you were expecting, PuTTY can warn you that the server may have been switched and that a spoofing attack might be in progress.

PuTTY records the host key for each server you connect to, in the Windows Registry. Every time you connect to a server, it checks that

the host key presented by the server is the same host key as it was the last time you connected. If it is not, you will see a stronger warning, and you will have the chance to abandon your connection before you type any private information (such as a password) into it. (See [section 10.2](#) for what that looks like.)

However, when you connect to a server you have not connected to before, PuTTY has no way of telling whether the host key is the right one or not. So it gives the warning shown above, and asks you whether you want to trust this host key or not.

Whether or not to trust the host key is your choice. If you are connecting within a company network, you might feel that all the network users are on the same side and spoofing attacks are unlikely, so you might choose to trust the key without checking it. If you are connecting across a hostile network (such as the Internet), you should check with your system administrator, perhaps by telephone or in person. (When verifying the fingerprint, be careful with letters and numbers that can be confused with each other: 0/o, 1/I/l, and so on.)

Many servers have more than one host key. If the system administrator sends you more than one fingerprint, you should make sure the one PuTTY shows you is on the list, but it doesn't matter which one it is.

If you don't have any fingerprints that look like the example (SHA256: followed by a long string of characters), but instead have pairs of characters separated by colons like a4:db:96:a7:..., try pressing the 'More info...' button and see if you have a fingerprint matching the 'MD5 fingerprint' there. This is an older and less secure way to summarise the same underlying host key; it's possible for an attacker to create their own host key with the same fingerprint; so you should avoid relying on this fingerprint format unless you have no choice. The 'More info...' dialog box also shows the full host public key, in case that is easier to compare than a fingerprint.

See [section 4.19](#) for advanced options for managing host keys.

2.3 Logging in

After you have connected, and perhaps verified the server's host key, you will be asked to log in, probably using a username and a password. Your system administrator should have provided you with these. (If, instead, your system administrator has asked you to provide, or provided you with, a 'public key' or 'key file', see [chapter 8](#).) PuTTY will display a text window (the 'terminal window' – it will have a black background unless you've changed the defaults), and prompt you to type your username and password into that window. (These prompts will include the PuTTY icon, to distinguish them from any text sent by the server in the same window.) Enter the username and the password, and the server should grant you access and begin your session. If you have mistyped your password, most servers will give you several chances to get it right.

While you are typing your password, you will not usually see the cursor moving in the window, but PuTTY *is* registering what you type, and will send it when you press Return. (It works this way to avoid revealing the length of your password to anyone watching your screen.) If you are using SSH, be careful not to type your username wrongly, because you will not have a chance to correct it after you press Return; many SSH servers do not permit you to make two login attempts using different usernames. If you type your username wrongly, you must close PuTTY and start again.

If your password is refused but you are sure you have typed it correctly, check that Caps Lock is not enabled. Many login servers, particularly Unix computers, treat upper case and lower case as different when checking your password; so if Caps Lock is on, your password will probably be refused.

2.4 After logging in

After you log in to the server, what happens next is up to the server! Most servers will print some sort of login message and then present a prompt, at which you can type commands which the server will carry out. Some servers will offer you on-line help; others might not. If you are in doubt about what to do next, consult your system administrator.

2.5 Logging out

When you have finished your session, you should log out by typing the server's own logout command. This might vary between servers; if in doubt, try `logout` or `exit`, or consult a manual or your system administrator. When the server processes your logout command, the PuTTY window should close itself automatically.

You can close a PuTTY session using the Close button in the window border, but this might confuse the server - a bit like hanging up a telephone unexpectedly in the middle of a conversation. We recommend you do not do this unless the server has stopped responding to you and you cannot close the window any other way.

Chapter 3: Using PuTTY

This chapter provides a general introduction to some more advanced features of PuTTY. For extreme detail and reference purposes, [chapter 4](#) is likely to contain more information.

- [3.1 During your session](#)
 - [3.1.1 Copying and pasting text](#)
 - [3.1.2 Scrolling the screen back](#)
 - [3.1.3 The System menu](#)
- [3.2 Creating a log file of your session](#)
- [3.3 Altering your character set configuration](#)
- [3.4 Using X11 forwarding in SSH](#)
- [3.5 Using port forwarding in SSH](#)
- [3.6 Connecting to a local serial line](#)
- [3.7 Making raw TCP connections](#)
- [3.8 Connecting using the Telnet protocol](#)
- [3.9 Connecting using the Rlogin protocol](#)
- [3.10 Connecting using the SUPDUP protocol](#)
- [3.11 The PuTTY command line](#)
 - [3.11.1 Starting a session from the command line](#)
 - [3.11.2 -cleanup](#)
 - [3.11.3 Standard command-line options](#)

3.1 During your session

A lot of PuTTY's complexity and features are in the configuration panel. Once you have worked your way through that and started a session, things should be reasonably simple after that. Nevertheless, there are a few more useful features available.

- [3.1.1 Copying and pasting text](#)
- [3.1.2 Scrolling the screen back](#)
- [3.1.3 The System menu](#)
 - [3.1.3.1 The PuTTY Event Log](#)
 - [3.1.3.2 Special commands](#)
 - [3.1.3.3 Starting new sessions](#)
 - [3.1.3.4 Changing your session settings](#)
 - [3.1.3.5 Copy All to Clipboard](#)
 - [3.1.3.6 Clearing and resetting the terminal](#)
 - [3.1.3.7 Full screen mode](#)

3.1.1 Copying and pasting text

Often in a PuTTY session you will find text on your terminal screen which you want to type in again. Like most other terminal emulators, PuTTY allows you to copy and paste the text rather than having to type it again. Also, copy and paste uses the Windows clipboard, so that you can paste (for example) URLs into a web browser, or paste from a word processor or spreadsheet into your terminal session.

By default, PuTTY's copy and paste works entirely with the mouse. (This will be familiar to people who have used `xterm` on Unix.) In order to copy text to the clipboard, you just click the left mouse button in the terminal window, and drag to select text. When you let go of the button, the text is *automatically* copied to the clipboard. You do not need to press `Ctrl-C` or `Ctrl-Ins`; in fact, if you do press `Ctrl-C`, PuTTY will send a `Ctrl-C` character down your session to the server where it will probably cause a process to be interrupted.

Pasting into PuTTY is done using the right button (or the middle mouse button, if you have a three-button mouse and have set it up; see [section 4.11.1](#)). (Pressing `Shift-Ins`, or selecting 'Paste' from the `Ctrl+right-click` context menu, have the same effect.) When you click the right mouse button, PuTTY will read whatever is in the Windows clipboard and paste it into your session. By default, this behaves *exactly* as if the clipboard contents had been typed at the keyboard; therefore, be careful of pasting formatted text into an editor that does automatic indenting, as you may find that the spaces pasted from the clipboard plus the spaces added by the editor add up to too many spaces and ruin the formatting. (Some remote applications can ask PuTTY to identify text that is being pasted, to avoid this sort of problem; but if your application does not, there is nothing PuTTY can do to avoid this.)

If you double-click the left mouse button, PuTTY will select a whole word. If you double-click, hold down the second click, and drag the mouse, PuTTY will select a sequence of whole words. (You can

adjust precisely what PuTTY considers to be part of a word; see [section 4.12.1](#).) If you *triple-click*, or triple-click and drag, then PuTTY will select a whole line or sequence of lines.

If you want to select a rectangular region instead of selecting to the end of each line, you can do this by holding down Alt when you make your selection. You can also configure rectangular selection to be the default, and then holding down Alt gives the normal behaviour instead: see [section 4.11.3](#) for details.

(In some Unix environments, Alt+drag is intercepted by the window manager. Shift+Alt+drag should work for rectangular selection as well, so you could try that instead.)

If you have a middle mouse button, then you can use it to adjust an existing selection if you selected something slightly wrong. (If you have configured the middle mouse button to paste, then the right mouse button does this instead.) Click the button on the screen, and you can pick up the nearest end of the selection and drag it to somewhere else.

If you are running PuTTY itself on Unix (not just using it to connect to a Unix system from Windows), by default you will likely have to use similar mouse actions in other applications to paste the text you copied from PuTTY, and to copy text for pasting into PuTTY; actions like Ctrl-C and Ctrl-V will likely not behave as you expect. [Section 4.11.4](#) explains why this is, and how you can change the behaviour. (On Windows there is only a single selection shared with other applications, so this confusion does not arise.)

It's possible for the server to ask to handle mouse clicks in the PuTTY window itself. If this happens, the mouse pointer will turn into an arrow, and using the mouse to copy and paste will only work if you hold down Shift. See [section 4.6.2](#) and [section 4.11.2](#) for details of this feature and how to configure it.

You can customise much of this behaviour, for instance to enable copy and paste from the keyboard; see [section 4.11](#).

3.1.2 Scrolling the screen back

PuTTY keeps track of text that has scrolled up off the top of the terminal. So if something appears on the screen that you want to read, but it scrolls too fast and it's gone by the time you try to look for it, you can use the scrollbar on the right side of the window to look back up the session history and find it again.

As well as using the scrollbar, you can also page the scrollback up and down by pressing Shift-PgUp and Shift-PgDn. You can scroll a line at a time using Ctrl-PgUp and Ctrl-PgDn, or to the top/bottom of the scrollback with Ctrl-Shift-PgUp and Ctrl-Shift-PgDn. These are still available if you configure the scrollbar to be invisible.

By default the last 2000 lines scrolled off the top are preserved for you to look at. You can increase (or decrease) this value using the configuration box; see [section 4.7.3](#).

3.1.3 The System menu

If you click the left mouse button on the icon in the top left corner of PuTTY's terminal window, or click the right mouse button on the title bar, you will see the standard Windows system menu containing items like Minimise, Move, Size and Close.

PuTTY's system menu contains extra program features in addition to the Windows standard options. These extra menu commands are described below.

(These options are also available in a context menu brought up by holding Ctrl and clicking with the right mouse button anywhere in the PuTTY window.)

- [3.1.3.1 The PuTTY Event Log](#)
- [3.1.3.2 Special commands](#)
- [3.1.3.3 Starting new sessions](#)
- [3.1.3.4 Changing your session settings](#)
- [3.1.3.5 Copy All to Clipboard](#)
- [3.1.3.6 Clearing and resetting the terminal](#)
- [3.1.3.7 Full screen mode](#)

3.1.3.1 The PuTTY Event Log

If you choose ‘Event Log’ from the system menu, a small window will pop up in which PuTTY logs significant events during the connection. Most of the events in the log will probably take place during session startup, but a few can occur at any point in the session, and one or two occur right at the end.

You can use the mouse to select one or more lines of the Event Log, and hit the Copy button to copy them to the clipboard. If you are reporting a bug, it’s often useful to paste the contents of the Event Log into your bug report.

(The Event Log is not the same as the facility to create a log file of your session; that’s described in [section 3.2](#).)

3.1.3.2 Special commands

Depending on the protocol used for the current session, there may be a submenu of ‘special commands’. These are protocol-specific tokens, such as a ‘break’ signal, that can be sent down a connection in addition to normal data. Their precise effect is usually up to the server. Currently only Telnet, SSH, and serial connections have special commands.

The ‘break’ signal can also be invoked from the keyboard with Ctrl-Break.

In an SSH connection, the following special commands are available:

- IGNORE message

Should have no effect.

- Repeat key exchange

Only available in SSH-2. Forces a repeat key exchange immediately (and resets associated timers and counters). For more information about repeat key exchanges, see [section 4.18.2](#).

- Cache new host key type

Only available in SSH-2. This submenu appears only if the server has host keys of a type that PuTTY doesn't already have cached, and so won't consider. Selecting a key here will allow PuTTY to use that key now and in future: PuTTY will do a fresh key-exchange with the selected key, and immediately add that key to its permanent cache (relying on the host key used at the start of the connection to cross-certify the new key). That key will be used for the rest of the current session; it may not

actually be used for future sessions, depending on your preferences (see [section 4.19.1](#)).

Normally, PuTTY will carry on using a host key it already knows, even if the server offers key formats that PuTTY would otherwise prefer, to avoid host key prompts. As a result, if you've been using a server for some years, you may still be using an older key than a new user would use, due to server upgrades in the meantime. The SSH protocol unfortunately does not have organised facilities for host key migration and rollover, but this allows you to manually upgrade.

- Break

Only available in SSH-2, and only during a session. Optional extension; may not be supported by server. PuTTY requests the server's default break length.

- Signals (SIGINT, SIGTERM etc)

Only available in SSH-2, and only during a session. Sends various POSIX signals. Not honoured by all servers.

The following special commands are available in Telnet:

- Are You There
- Break
- Synch
- Erase Character

PuTTY can also be configured to send this when the Backspace key is pressed; see [section 4.30.3](#).

- Erase Line
- Go Ahead
- No Operation

Should have no effect.

- Abort Process
- Abort Output
- Interrupt Process

PuTTY can also be configured to send this when Ctrl-C is typed;
see [section 4.30.3](#).

- Suspend Process

PuTTY can also be configured to send this when Ctrl-Z is typed;
see [section 4.30.3](#).

- End Of Record
- End Of File

With a serial connection, the only available special command is
'Break'.

3.1.3.3 Starting new sessions

PuTTY's system menu provides some shortcut ways to start new sessions:

- Selecting ‘New Session’ will start a completely new instance of PuTTY, and bring up the configuration box as normal.
- Selecting ‘Duplicate Session’ will start a session in a new window with precisely the same options as your current one - connecting to the same host using the same protocol, with all the same terminal settings and everything.
- In an inactive window, selecting ‘Restart Session’ will do the same as ‘Duplicate Session’, but in the current window.
- The ‘Saved Sessions’ submenu gives you quick access to any sets of stored session details you have previously saved. See [section 4.1.2](#) for details of how to create saved sessions.

3.1.3.4 Changing your session settings

If you select ‘Change Settings’ from the system menu, PuTTY will display a cut-down version of its initial configuration box. This allows you to adjust most properties of your current session. You can change the terminal size, the font, the actions of various keypresses, the colours, and so on.

Some of the options that are available in the main configuration box are not shown in the cut-down Change Settings box. These are usually options which don’t make sense to change in the middle of a session (for example, you can’t switch from SSH to Telnet in mid-session).

You can save the current settings to a saved session for future use from this dialog box. See [section 4.1.2](#) for more on saved sessions.

3.1.3.5 Copy All to Clipboard

This system menu option provides a convenient way to copy the whole contents of the terminal screen (up to the last nonempty line) and scrollback to the clipboard in one go.

3.1.3.6 Clearing and resetting the terminal

The ‘Clear Scrollback’ option on the system menu tells PuTTY to discard all the lines of text that have been kept after they scrolled off the top of the screen. This might be useful, for example, if you displayed sensitive information and wanted to make sure nobody could look over your shoulder and see it. (Note that this only prevents a casual user from using the scrollbar to view the information; the text is not guaranteed not to still be in PuTTY’s memory.) The ‘Reset Terminal’ option causes a full reset of the terminal emulation. A VT-series terminal is a complex piece of software and can easily get into a state where all the text printed becomes unreadable. (This can happen, for example, if you accidentally output a binary file to your terminal.) If this happens, selecting Reset Terminal should sort it out.

3.1.3.7 Full screen mode

If you find the title bar on a maximised window to be ugly or distracting, you can select Full Screen mode to maximise PuTTY ‘even more’. When you select this, PuTTY will expand to fill the whole screen and its borders, title bar and scrollbar will disappear. (You can configure the scrollbar not to disappear in full-screen mode if you want to keep it; see [section 4.7.3.](#)) When you are in full-screen mode, you can still access the system menu if you click the left mouse button in the *extreme* top left corner of the screen.

3.2 Creating a log file of your session

For some purposes you may find you want to log everything that appears on your screen. You can do this using the ‘Logging’ panel in the configuration box.

To begin a session log, select ‘Change Settings’ from the system menu and go to the Logging panel. Enter a log file name, and select a logging mode. (You can log all session output including the terminal control sequences, or you can just log the printable text. It depends what you want the log for.) Click ‘Apply’ and your log will be started. Later on, you can go back to the Logging panel and select ‘Logging turned off completely’ to stop logging; then PuTTY will close the log file and you can safely read it.

See [section 4.2](#) for more details and options.

3.3 Altering your character set configuration

If you find that special characters (accented characters, for example, or line-drawing characters) are not being displayed correctly in your PuTTY session, it may be that PuTTY is interpreting the characters sent by the server according to the wrong *character set*. There are a lot of different character sets available, and no good way for PuTTY to know which to use, so it's entirely possible for this to happen.

If you click ‘Change Settings’ and look at the ‘Translation’ panel, you should see a large number of character sets which you can select, and other related options. Now all you need is to find out which of them you want! (See [section 4.10](#) for more information.)

3.4 Using X11 forwarding in SSH

The SSH protocol has the ability to securely forward X Window System graphical applications over your encrypted SSH connection, so that you can run an application on the SSH server machine and have it put its windows up on your local machine without sending any X network traffic in the clear.

In order to use this feature, you will need an X display server for your Windows machine, such as Cygwin/X, X-Win32, or Exceed. This will probably install itself as display number 0 on your local machine; if it doesn't, the manual for the X server should tell you what it does do.

You should then tick the 'Enable X11 forwarding' box in the X11 panel (see [section 4.25](#)) before starting your SSH session. The 'X display location' box is blank by default, which means that PuTTY will try to use a sensible default such as :0, which is the usual display location where your X server will be installed. If that needs changing, then change it.

Now you should be able to log in to the SSH server as normal. To check that X forwarding has been successfully negotiated during connection startup, you can check the PuTTY Event Log (see [section 3.1.3.1](#)). It should say something like this:

```
2001-12-05 17:22:01 Requesting X11 forwarding
2001-12-05 17:22:02 X11 forwarding enabled
```

If the remote system is Unix or Unix-like, you should also be able to see that the `DISPLAY` environment variable has been set to point at display 10 or above on the SSH server machine itself:

```
fred@unixbox:~$ echo $DISPLAY
unixbox:10.0
```

If this works, you should then be able to run X applications in the remote session and have them display their windows on your PC.

For more options relating to X11 forwarding, see [section 4.25](#).

3.5 Using port forwarding in SSH

The SSH protocol has the ability to forward arbitrary network (TCP) connections over your encrypted SSH connection, to avoid the network traffic being sent in clear. For example, you could use this to connect from your home computer to a POP-3 server on a remote machine without your POP-3 password being visible to network sniffers.

In order to use port forwarding to connect from your local machine to a port on a remote server, you need to:

- Choose a port number on your local machine where PuTTY should listen for incoming connections. There are likely to be plenty of unused port numbers above 3000. (You can also use a local loopback address here; see below for more details.)
- Now, before you start your SSH connection, go to the Tunnels panel (see [section 4.26](#)). Make sure the ‘Local’ radio button is set. Enter the local port number into the ‘Source port’ box. Enter the destination host name and port number into the ‘Destination’ box, separated by a colon (for example, `popserver.example.com:110` to connect to a POP-3 server).
- Now click the ‘Add’ button. The details of your port forwarding should appear in the list box.

Now start your session and log in. (Port forwarding will not be enabled until after you have logged in; otherwise it would be easy to perform completely anonymous network attacks, and gain access to anyone's virtual private network.) To check that PuTTY has set up the port forwarding correctly, you can look at the PuTTY Event Log (see [section 3.1.3.1](#)). It should say something like this:

```
2001-12-05 17:22:10 Local port 3110 forwarding to  
popserver.example.com:110
```

Now if you connect to the source port number on your local PC, you should find that it answers you exactly as if it were the service

running on the destination machine. So in this example, you could then configure an e-mail client to use `localhost:3110` as a POP-3 server instead of `popserver.example.com:110`. (Of course, the forwarding will stop happening when your PuTTY session closes down.)

You can also forward ports in the other direction: arrange for a particular port number on the server machine to be forwarded back to your PC as a connection to a service on your PC or near it. To do this, just select the ‘Remote’ radio button instead of the ‘Local’ one. The ‘Source port’ box will now specify a port number on the server (note that most servers will not allow you to use port numbers under 1024 for this purpose).

An alternative way to forward local connections to remote hosts is to use dynamic SOCKS proxying. In this mode, PuTTY acts as a SOCKS server, which SOCKS-aware programs can connect to and open forwarded connections to the destination of their choice, so this can be an alternative to long lists of static forwardings. To use this mode, you will need to select the ‘Dynamic’ radio button instead of ‘Local’, and then you should not enter anything into the ‘Destination’ box (it will be ignored). PuTTY will then listen for SOCKS connections on the port you have specified. Most web browsers can be configured to connect to this SOCKS proxy service; also, you can forward other PuTTY connections through it by setting up the Proxy control panel (see [section 4.16](#) for details).

The source port for a forwarded connection usually does not accept connections from any machine except the SSH client or server machine itself (for local and remote forwardings respectively). There are controls in the Tunnels panel to change this:

- The ‘Local ports accept connections from other hosts’ option allows you to set up local-to-remote port forwardings (including dynamic port forwardings) in such a way that machines other than your client PC can connect to the forwarded port.
- The ‘Remote ports do the same’ option does the same thing for remote-to-local port forwardings (so that machines other than

the SSH server machine can connect to the forwarded port.) Note that this feature is only available in the SSH-2 protocol, and not all SSH-2 servers honour it (in OpenSSH, for example, it's usually disabled by default).

You can also specify an IP address to listen on. Typically a Windows machine can be asked to listen on any single IP address in the 127.*.*.* range, and all of these are loopback addresses available only to the local machine. So if you forward (for example) 127.0.0.5:79 to a remote machine's finger port, then you should be able to run commands such as finger fred@127.0.0.5. This can be useful if the program connecting to the forwarded port doesn't allow you to change the port number it uses. This feature is available for local-to-remote forwarded ports; SSH-1 is unable to support it for remote-to-local ports, while SSH-2 can support it in theory but servers will not necessarily cooperate.

(Note that if you're using Windows XP Service Pack 2, you may need to obtain a fix from Microsoft in order to use addresses like 127.0.0.5 - see [question A.7.17](#).)

For more options relating to port forwarding, see [section 4.26](#).

If the connection you are forwarding over SSH is itself a second SSH connection made by another copy of PuTTY, you might find the 'logical host name' configuration option useful to warn PuTTY of which host key it should be expecting. See [section 4.14.5](#) for details of this.

3.6 Connecting to a local serial line

PuTTY can connect directly to a local serial line as an alternative to making a network connection. In this mode, text typed into the PuTTY window will be sent straight out of your computer's serial port, and data received through that port will be displayed in the PuTTY window. You might use this mode, for example, if your serial port is connected to another computer which has a serial connection.

To make a connection of this type, simply select 'Serial' from the 'Connection type' radio buttons on the 'Session' configuration panel (see [section 4.1.1](#)). The 'Host Name' and 'Port' boxes will transform into 'Serial line' and 'Speed', allowing you to specify which serial line to use (if your computer has more than one) and what speed (baud rate) to use when transferring data. For further configuration options (data bits, stop bits, parity, flow control), you can use the 'Serial' configuration panel (see [section 4.29](#)).

After you start up PuTTY in serial mode, you might find that you have to make the first move, by sending some data out of the serial line in order to notify the device at the other end that someone is there for it to talk to. This probably depends on the device. If you start up a PuTTY serial session and nothing appears in the window, try pressing Return a few times and see if that helps.

A serial line provides no well defined means for one end of the connection to notify the other that the connection is finished. Therefore, PuTTY in serial mode will remain connected until you close the window using the close button.

3.7 Making raw TCP connections

A lot of Internet protocols are composed of commands and responses in plain text. For example, SMTP (the protocol used to transfer e-mail), NNTP (the protocol used to transfer Usenet news), and HTTP (the protocol used to serve Web pages) all consist of commands in readable plain text.

Sometimes it can be useful to connect directly to one of these services and speak the protocol ‘by hand’, by typing protocol commands and watching the responses. On Unix machines, you can do this using the system’s `telnet` command to connect to the right port number. For example, `telnet mailserver.example.com 25` might enable you to talk directly to the SMTP service running on a mail server.

Although the Unix `telnet` program provides this functionality, the protocol being used is not really Telnet. Really there is no actual protocol at all; the bytes sent down the connection are exactly the ones you type, and the bytes shown on the screen are exactly the ones sent by the server. Unix `telnet` will attempt to detect or guess whether the service it is talking to is a real Telnet service or not; PuTTY prefers to be told for certain.

In order to make a debugging connection to a service of this type, you simply select the fourth protocol name, ‘Raw’, from the ‘Protocol’ buttons in the ‘Session’ configuration panel. (See [section 4.1.1](#).) You can then enter a host name and a port number, and make the connection.

3.8 Connecting using the Telnet protocol

PuTTY can use the Telnet protocol to connect to a server.

Telnet was perhaps the most popular remote login protocol before SSH was introduced. It was general enough to be used by multiple server operating systems (Unix and VMS in particular), and supported many optional protocol extensions providing extra support for particular server features.

Unlike SSH, Telnet runs over an unsecured network connection, so it is a very bad idea to use it over the hostile Internet (though it is still used to some extent as of 2020).

3.9 Connecting using the Rlogin protocol

PuTTY can use the Rlogin protocol to connect to a server.

Rlogin was similar to Telnet in concept, but more focused on connections between Unix machines. It supported a feature for passwordless login, based on use of ‘privileged ports’ (ports with numbers below 1024, which Unix traditionally does not allow users other than `root` to allocate). Ultimately, based on the server trusting that the client's IP address was owned by the Unix machine it claimed to be, and that that machine would guard its privileged ports appropriately.

Like Telnet, Rlogin runs over an unsecured network connection.

3.10 Connecting using the SUPDUP protocol

PuTTY can use the SUPDUP protocol to connect to a server.

SUPDUP is a login protocol used mainly by PDP-10 and Lisp machines during the period 1975-1990. Like Telnet and Rlogin, it is unsecured, so modern systems almost never support it.

To make a connection of this type, select ‘SUPDUP’ from the ‘Connection type’ radio buttons on the ‘Session’ panel (see [section 4.1.1](#)). For further configuration options (character set, more processing, scrolling), you can use the ‘SUPDUP’ configuration panel (see [section 4.32](#)).

In SUPDUP, terminal emulation is more integrated with the network protocol than in other protocols such as SSH. The SUPDUP protocol can thus only be used with PuTTY proper, not with the command-line tool Plink.

The SUPDUP protocol does not support changing the terminal dimensions, so this capability is disabled during a SUPDUP session.

SUPDUP provides no well defined means for one end of the connection to notify the other that the connection is finished. Therefore, PuTTY in SUPDUP mode will remain connected until you close the window using the close button.

3.11 The PuTTY command line

PuTTY can be made to do various things without user intervention by supplying command-line arguments (e.g., from a command prompt window, or a Windows shortcut).

- [3.11.1 Starting a session from the command line](#)
- [3.11.2 -cleanup](#)
- [3.11.3 Standard command-line options](#)
 - [3.11.3.1 -load: load a saved session](#)
 - [3.11.3.2 Selecting a protocol: -ssh, -ssh-connection, -telnet, -rlogin, -supdup, -raw, -serial](#)
 - [3.11.3.3 -v: increase verbosity](#)
 - [3.11.3.4 -l: specify a login name](#)
 - [3.11.3.5 -L, -R and -D: set up port forwardings](#)
 - [3.11.3.6 -m: read a remote command or script from a file](#)
 - [3.11.3.7 -P: specify a port number](#)
 - [3.11.3.8 -pwfile and -pw: specify a password](#)
 - [3.11.3.9 -agent and -noagent: control use of Pageant for authentication](#)
 - [3.11.3.10 -A and -a: control agent forwarding](#)
 - [3.11.3.11 -x and -X: control X11 forwarding](#)
 - [3.11.3.12 -t and -T: control pseudo-terminal allocation](#)
 - [3.11.3.13 -N: suppress starting a shell or command](#)
 - [3.11.3.14 -nc: make a remote network connection in place of a remote shell or command](#)
 - [3.11.3.15 -c: enable compression](#)
 - [3.11.3.16 -1 and -2: specify an SSH protocol version](#)
 - [3.11.3.17 -4 and -6: specify an Internet protocol version](#)
 - [3.11.3.18 -i: specify an SSH private key](#)
 - [3.11.3.19 -cert: specify an SSH certificate](#)
 - [3.11.3.20 -no-trivial-auth: disconnect if SSH authentication succeeds trivially](#)
 - [3.11.3.21 -loghost: specify a logical host name](#)

- [3.11.3.22 -hostkey: manually specify an expected host key](#)
- [3.11.3.23 -pgpfp: display PGP key fingerprints](#)
- [3.11.3.24 -sercfg: specify serial port configuration](#)
- [3.11.3.25 -sessionlog, -sshlog, -sshrawlog: enable session logging](#)
- [3.11.3.26 -logoverwrite, -logappend: control behaviour with existing log file](#)
- [3.11.3.27 -proxycmd: specify a local proxy command](#)
- [3.11.3.28 -restrict-acl: restrict the Windows process ACL](#)
- [3.11.3.29 -host-ca: launch the host CA configuration](#)

3.11.1 Starting a session from the command line

These options allow you to bypass the configuration window and launch straight into a session.

To start a connection to a server called host:

```
putty.exe [-ssh | -ssh-connection | -telnet | -rlogin | -supdup  
| -raw] [user@]host
```

If this syntax is used, settings are taken from the Default Settings (see [section 4.1.2](#)); user overrides these settings if supplied. Also, you can specify a protocol, which will override the default protocol (see [section 3.11.3.2](#)).

For telnet sessions, the following alternative syntax is supported (this makes PuTTY suitable for use as a URL handler for telnet URLs in web browsers):

```
putty.exe telnet://host[:port]/
```

To start a connection to a serial port, e.g. COM1:

```
putty.exe -serial com1
```

In order to start an existing saved session called sessionname, use the `-load` option (described in [section 3.11.3.1](#)).

```
putty.exe -load "session name"
```

3.11.2 -cleanup

If invoked with the `-cleanup` option, rather than running as normal, PuTTY will remove its registry entries and random seed file from the local machine (after confirming with the user). It will also attempt to remove information about recently launched sessions stored in the 'jump list' on Windows 7 and up.

Note that on multi-user systems, `-cleanup` only removes registry entries and files associated with the currently logged-in user.

3.11.3 Standard command-line options

PuTTY and its associated tools support a range of command-line options, most of which are consistent across all the tools. This section lists the available options in all tools. Options which are specific to a particular tool are covered in the chapter about that tool.

- [3.11.3.1 -load: load a saved session](#)
- [3.11.3.2 Selecting a protocol: -ssh, -ssh-connection, -telnet, -rlogin, -supdup, -raw, -serial](#)
- [3.11.3.3 -v: increase verbosity](#)
- [3.11.3.4 -l: specify a login name](#)
- [3.11.3.5 -L, -R and -D: set up port forwardings](#)
- [3.11.3.6 -m: read a remote command or script from a file](#)
- [3.11.3.7 -P: specify a port number](#)
- [3.11.3.8 -pwfile and -pw: specify a password](#)
- [3.11.3.9 -agent and -noagent: control use of Pageant for authentication](#)
- [3.11.3.10 -A and -a: control agent forwarding](#)
- [3.11.3.11 -x and -X: control X11 forwarding](#)
- [3.11.3.12 -t and -T: control pseudo-terminal allocation](#)
- [3.11.3.13 -N: suppress starting a shell or command](#)
- [3.11.3.14 -nc: make a remote network connection in place of a remote shell or command](#)
- [3.11.3.15 -c: enable compression](#)
- [3.11.3.16 -1 and -2: specify an SSH protocol version](#)
- [3.11.3.17 -4 and -6: specify an Internet protocol version](#)
- [3.11.3.18 -i: specify an SSH private key](#)
- [3.11.3.19 -cert: specify an SSH certificate](#)
- [3.11.3.20 -no-trivial-auth: disconnect if SSH authentication succeeds trivially](#)
- [3.11.3.21 -loghost: specify a logical host name](#)
- [3.11.3.22 -hostkey: manually specify an expected host key](#)
- [3.11.3.23 -pgpfp: display PGP key fingerprints](#)

- [3.11.3.24 -sercfg: specify serial port configuration](#)
- [3.11.3.25 -sessionlog, -sshlog, -sshrawlog: enable session logging](#)
- [3.11.3.26 -logoverwrite, -logappend: control behaviour with existing log file](#)
- [3.11.3.27 -proxycmd: specify a local proxy command](#)
- [3.11.3.28 -restrict-acl: restrict the Windows process ACL](#)
- [3.11.3.29 -host-ca: launch the host CA configuration](#)

3.11.3.1 -load: load a saved session

The `-load` option causes PuTTY to load configuration details out of a saved session. If these details include a host name, then this option is all you need to make PuTTY start a session.

You need double quotes around the session name if it contains spaces.

If you want to create a Windows shortcut to start a PuTTY saved session, this is the option you should use: your shortcut should call something like

```
d:\path\to\putty.exe -load "my session"
```

(Note that PuTTY itself supports an alternative form of this option, for backwards compatibility. If you execute `putty @sessionname` it will have the same effect as `putty -load "sessionname"`. With the `@` form, no double quotes are required, and the `@` sign must be the very first thing on the command line. This form of the option is deprecated.)

3.11.3.2 Selecting a protocol: -ssh, -ssh-connection, -telnet, -rlogin, -supdup, -raw, -serial

To choose which protocol you want to connect with, you can use one of these options:

- `-ssh` selects the SSH protocol.
- `-ssh-connection` selects the bare ssh-connection protocol. (This is only useful in specialised circumstances; see [section 4.28](#) for more information.)
- `-telnet` selects the Telnet protocol.
- `-rlogin` selects the Rlogin protocol.
- `-supdup` selects the SUPDUP protocol.
- `-raw` selects the raw protocol.
- `-serial` selects a serial connection.

Most of these options are not available in the file transfer tools PSCP and PSFTP (which only work with the SSH protocol and the bare ssh-connection protocol).

These options are equivalent to the protocol selection buttons in the Session panel of the PuTTY configuration box (see [section 4.1.1](#)).

3.11.3.3 -v: increase verbosity

Most of the PuTTY tools can be made to tell you more about what they are doing by supplying the `-v` option. If you are having trouble when making a connection, or you're simply curious, you can turn this switch on and hope to find out more about what is happening.

3.11.3.4 -l: specify a login name

You can specify the user name to log in as on the remote server using the `-l` option. For example, `plink login.example.com -l fred`.

These options are equivalent to the username selection box in the Connection panel of the PuTTY configuration box (see [section 4.15.1](#)).

3.11.3.5 -L, -R and -D: set up port forwardings

As well as setting up port forwardings in the PuTTY configuration (see [section 4.26](#)), you can also set up forwardings on the command line. The command-line options work just like the ones in Unix ssh programs.

To forward a local port (say 5110) to a remote destination (say popserver.example.com port 110), you can write something like one of these: `putty -L 5110:popserver.example.com:110 -load mysession`
`plink mysession -L 5110:popserver.example.com:110`

To forward a remote port to a local destination, just use the `-R` option instead of `-L`:

```
putty -R 5023:mytelnetserver.myhouse.org:23 -load mysession  
plink mysession -R 5023:mytelnetserver.myhouse.org:23
```

To specify an IP address for the listening end of the tunnel, prepend it to the argument:

```
plink -L 127.0.0.5:23:localhost:23 myhost
```

To set up SOCKS-based dynamic port forwarding on a local port, use the `-D` option. For this one you only have to pass the port number: `putty -D 4096 -load mysession`

For general information on port forwarding, see [section 3.5](#).

These options are not available in the file transfer tools PSCP and PSFTP.

3.11.3.6 -m: read a remote command or script from a file

The `-m` option performs a similar function to the ‘Remote command’ box in the SSH panel of the PuTTY configuration box (see [section 4.17.1](#)). However, the `-m` option expects to be given a local file name, and it will read a command from that file.

With some servers (particularly Unix systems), you can even put multiple lines in this file and execute more than one command in sequence, or a whole shell script; but this is arguably an abuse, and cannot be expected to work on all servers. In particular, it is known *not* to work with certain ‘embedded’ servers, such as Cisco routers.

This option is not available in the file transfer tools PSCP and PSFTP.

3.11.3.7 -P: specify a port number

The **-P** option is used to specify the port number to connect to. If you have a Telnet server running on port 9696 of a machine instead of port 23, for example: `putty -telnet -P 9696 host.name plink - telnet -P 9696 host.name`

(Note that this option is more useful in Plink than in PuTTY, because in PuTTY you can write `putty -telnet host.name 9696` in any case.)

This option is equivalent to the port number control in the Session panel of the PuTTY configuration box (see [section 4.1.1](#)).

3.11.3.8 -pwfile and -pw: specify a password

A simple way to automate a remote login is to supply your password on the command line.

The `-pwfile` option takes a file name as an argument. The first line of text in that file will be used as your password.

The `-pw` option takes the password itself as an argument. This is **NOT SECURE** if anybody else uses the same computer, because the whole command line (including the password) is likely to show up if another user lists the running processes. `-pw` is retained for backwards compatibility only; you should use `-pwfile` instead.

Note that these options only work when you are using the SSH protocol. Due to fundamental limitations of Telnet, Rlogin, and SUPDUP, these protocols do not support automated password authentication.

3.11.3.9 -agent and -noagent: control use of Pageant for authentication

The `-agent` option turns on SSH authentication using Pageant, and `-noagent` turns it off. These options are only meaningful if you are using SSH.

See [chapter 9](#) for general information on Pageant.

These options are equivalent to the agent authentication checkbox in the Auth panel of the PuTTY configuration box (see [section 4.21.4](#)).

3.11.3.10 -A and -a: control agent forwarding

The -A option turns on SSH agent forwarding, and -a turns it off. These options are only meaningful if you are using SSH.

See [chapter 9](#) for general information on Pageant, and [section 9.4](#) for information on agent forwarding. Note that there is a security risk involved with enabling this option; see [section 9.6](#) for details.

These options are equivalent to the agent forwarding checkbox in the Auth panel of the PuTTY configuration box (see [section 4.21.7](#)).

These options are not available in the file transfer tools PSCP and PSFTP.

3.11.3.11 -x and -X: control X11 forwarding

The `-x` option turns on X11 forwarding in SSH, and `-X` turns it off. These options are only meaningful if you are using SSH.

For information on X11 forwarding, see [section 3.4](#).

These options are equivalent to the X11 forwarding checkbox in the X11 panel of the PuTTY configuration box (see [section 4.25](#)).

These options are not available in the file transfer tools PSCP and PSFTP.

3.11.3.12 -t and -T: control pseudo-terminal allocation

The `-t` option ensures PuTTY attempts to allocate a pseudo-terminal at the server, and `-T` stops it from allocating one. These options are only meaningful if you are using SSH.

These options are equivalent to the ‘Don’t allocate a pseudo-terminal’ checkbox in the SSH panel of the PuTTY configuration box (see [section 4.24.1](#)).

These options are not available in the file transfer tools PSCP and PSFTP.

3.11.3.13 -N: suppress starting a shell or command

The -N option prevents PuTTY from attempting to start a shell or command on the remote server. You might want to use this option if you are only using the SSH connection for port forwarding, and your user account on the server does not have the ability to run a shell.

This feature is only available in SSH protocol version 2 (since the version 1 protocol assumes you will always want to run a shell).

This option is equivalent to the 'Don't start a shell or command at all' checkbox in the SSH panel of the PuTTY configuration box (see [section 4.17.2](#)).

This option is not available in the file transfer tools PSCP and PSFTP.

3.11.3.14 -nc: make a remote network connection in place of a remote shell or command

The `-nc` option prevents Plink (or PuTTY) from attempting to start a shell or command on the remote server. Instead, it will instruct the remote server to open a network connection to a host name and port number specified by you, and treat that network connection as if it were the main session.

You specify a host and port as an argument to the `-nc` option, with a colon separating the host name from the port number, like this:

```
plink host1.example.com -nc host2.example.com:1234
```

This can be useful if you're trying to make a connection to a target host which you can only reach by SSH forwarding through a proxy host. One way to do this would be to have an existing SSH connection to the proxy host, with a port forwarding, but if you prefer to have the connection started on demand as needed, then this approach can also work.

However, this does depend on the program *using* the proxy being able to run a subprocess in place of making a network connection. PuTTY itself can do this using the 'Local' proxy type, but there's a built-in more flexible way using the 'SSH' proxy type. (See [section 4.16.1](#) for a description of both.) So this feature is probably most useful with another client program as the end user.

This feature is only available in SSH protocol version 2 (since the version 1 protocol assumes you will always want to run a shell). It is not available in the file transfer tools PSCP and PSFTP. It is available in PuTTY itself, although it is unlikely to be very useful in any tool other than Plink. Also, `-nc` uses the same server functionality as port forwarding, so it will not work if your server administrator has disabled port forwarding.

(The option is named -nc after the Unix program [nc](#), short for ‘netcat’. The command ‘plink host1 -nc host2:port’ is very similar in functionality to ‘plink host1 nc host2 port’, which invokes nc on the server and tells it to connect to the specified destination. However, Plink's built-in -nc option does not depend on the nc program being installed on the server.)

3.11.3.15 -c: enable compression

The -c option enables compression of the data sent across the network. This option is only meaningful if you are using SSH.

This option is equivalent to the ‘Enable compression’ checkbox in the SSH panel of the PuTTY configuration box (see [section 4.17.3](#)).

3.11.3.16 -1 and -2: specify an SSH protocol version

The -1 and -2 options force PuTTY to use version 1 or version 2 of the SSH protocol. These options are only meaningful if you are using SSH.

These options are equivalent to selecting the SSH protocol version in the SSH panel of the PuTTY configuration box (see [section 4.17.4](#)).

3.11.3.17 -4 and -6: specify an Internet protocol version

The -4 and -6 options force PuTTY to use the older Internet protocol IPv4 or the newer IPv6 for most outgoing connections.

These options are equivalent to selecting your preferred Internet protocol version as ‘IPv4’ or ‘IPv6’ in the Connection panel of the PuTTY configuration box (see [section 4.14.4](#)).

3.11.3.18 -i: specify an SSH private key

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.

If you are using Pageant, you can also specify a *public* key file (in RFC 4716 or OpenSSH format) to identify a specific key file to use. (This won't work if you're not running Pageant, of course.) For general information on public-key authentication, see [chapter 8](#).

This option is equivalent to the ‘Private key file for authentication’ box in the Auth panel of the PuTTY configuration box (see [section 4.22.1](#)).

3.11.3.19 -cert: specify an SSH certificate

The `-cert` option allows you to specify the name of a certificate file containing a signed version of your public key. If you specify this option, PuTTY will present that certificate in place of the plain public key, whenever it tries to authenticate with a key that matches. (This applies whether the key is stored in Pageant or loaded directly from a file by PuTTY.) This option is equivalent to the ‘Certificate to use with the private key’ box in the Auth panel of the PuTTY configuration box (see [section 4.22.2](#)).

3.11.3.20 -no-trivial-auth: disconnect if SSH authentication succeeds trivially

This option causes PuTTY to abandon an SSH session if the server accepts authentication without ever having asked for any kind of password or signature or token.

See [section 4.21.3](#) for why you might want this.

3.11.3.21 - `loghost`: specify a logical host name

This option overrides PuTTY's normal SSH host key caching policy by telling it the name of the host you expect your connection to end up at (in cases where this differs from the location PuTTY thinks it's connecting to). It can be a plain host name, or a host name followed by a colon and a port number. See [section 4.14.5](#) for more detail on this.

3.11.3.22 -hostkey: manually specify an expected host key

This option overrides PuTTY's normal SSH host key caching policy by telling it exactly what host key to expect, which can be useful if the normal automatic host key store in the Registry is unavailable. The argument to this option should be either a host key fingerprint, or an SSH-2 public key blob. See [section 4.19.3](#) for more information.

You can specify this option more than once if you want to configure more than one key to be accepted.

3.11.3.23 -pgpfp: display PGP key fingerprints

This option causes the PuTTY tools not to run as normal, but instead to display the fingerprints of the PuTTY PGP Master Keys, in order to aid with verifying new versions. See [appendix F](#) for more information.

3.11.3.24 -sercfg: specify serial port configuration

This option specifies the configuration parameters for the serial port (baud rate, stop bits etc). Its argument is interpreted as a comma-separated list of configuration options, which can be as follows:

- Any single digit from 5 to 9 sets the number of data bits.
- ‘1’, ‘1.5’ or ‘2’ sets the number of stop bits.
- Any other numeric string is interpreted as a baud rate.
- A single lower-case letter specifies the parity: ‘n’ for none, ‘o’ for odd, ‘e’ for even, ‘m’ for mark and ‘s’ for space.
- A single upper-case letter specifies the flow control: ‘N’ for none, ‘X’ for XON/XOFF, ‘R’ for RTS/CTS and ‘D’ for DSR/DTR.

For example, ‘-sercfg 19200,8,n,1,N’ denotes a baud rate of 19200, 8 data bits, no parity, 1 stop bit and no flow control.

3.11.3.25 -sessionlog, -sshlog, -sshrawlog: enable session logging

These options cause the PuTTY network tools to write out a log file. Each of them expects a file name as an argument, e.g. '-sshlog putty.log' causes an SSH packet log to be written to a file called 'putty.log'. The three different options select different logging modes, all available from the GUI too:

- -sessionlog selects 'All session output' logging mode.
- -sshlog selects 'SSH packets' logging mode.
- -sshrawlog selects 'SSH packets and raw data' logging mode.

For more information on logging configuration, see [section 4.2](#).

3.11.3.26 -logoverwrite, -logappend: control behaviour with existing log file

If logging has been enabled (in the saved configuration, or by another command-line option), and the specified log file already exists, these options tell the PuTTY network tools what to do so that they don't have to ask the user. See [Section 4.2.2](#) for details.

3.11.3.27 -proxycmd: specify a local proxy command

This option enables PuTTY's mode for running a command on the local machine and using it as a proxy for the network connection. It expects a shell command string as an argument.

See [section 4.16.1](#) for more information on this, and on other proxy settings. In particular, note that since the special sequences described there are understood in the argument string, literal backslashes must be doubled (if you want \ in your command, you must put \\ on the command line).

3.11.3.28 -restrict-acl: restrict the Windows process ACL

This option (on Windows only) causes PuTTY (or another PuTTY tool) to try to lock down the operating system's access control on its own process. If this succeeds, it should present an extra obstacle to malware that has managed to run under the same user id as the PuTTY process, by preventing it from attaching to PuTTY using the same interfaces debuggers use and either reading sensitive information out of its memory or hijacking its network session.

This option is not enabled by default, because this form of interaction between Windows programs has many legitimate uses, including accessibility software such as screen readers. Also, it cannot provide full security against this class of attack in any case, because PuTTY can only lock down its own ACL *after* it has started up, and malware could still get in if it attacks the process between startup and lockdown. So it trades away noticeable convenience, and delivers less real security than you might want. However, if you do want to make that tradeoff anyway, the option is available.

A PuTTY process started with `-restrict-acl` will pass that on to any processes started with Duplicate Session, New Session etc. (However, if you're invoking PuTTY tools explicitly, for instance as a proxy command, you'll need to arrange to pass them the `-restrict-acl` option yourself, if that's what you want.)

If Pageant is started with the `-restrict-acl` option, and you use it to launch a PuTTY session from its System Tray submenu, then Pageant will *not* default to starting the PuTTY subprocess with a restricted ACL. This is because PuTTY is more likely to suffer reduced functionality as a result of restricted ACLs (e.g. screen reader software will have a greater need to interact with it), whereas Pageant stores the more critical information (hence benefits more from the extra protection), so it's reasonable to want to run Pageant but not PuTTY with the ACL restrictions. You can force Pageant to

start subsidiary PuTTY processes with a restricted ACL if you also pass the `-restrict-putty-acl` option.

3.11.3.29 -host-ca: launch the host CA configuration

If you start PuTTY with the `-host-ca` option, it will not launch a session at all. Instead, it will just display the configuration dialog box for host certification authorities, as described in [section 4.19.4](#). When you dismiss that dialog box, PuTTY will terminate.

Chapter 4: Configuring PuTTY

This chapter describes all the configuration options in PuTTY.

PuTTY is configured using the control panel that comes up before you start a session. Some options can also be changed in the middle of a session, by selecting ‘Change Settings’ from the window menu.

- [4.1 The Session panel](#)
 - [4.1.1 The host name section](#)
 - [4.1.2 Loading and storing saved sessions](#)
 - [4.1.3 ‘Close window on exit’](#)
- [4.2 The Logging panel](#)
 - [4.2.1 ‘Log file name’](#)
 - [4.2.2 ‘What to do if the log file already exists’](#)
 - [4.2.3 ‘Flush log file frequently’](#)
 - [4.2.4 ‘Include header’](#)
 - [4.2.5 Options specific to SSH packet logging](#)
- [4.3 The Terminal panel](#)
 - [4.3.1 ‘Auto wrap mode initially on’](#)
 - [4.3.2 ‘DEC Origin Mode initially on’](#)
 - [4.3.3 ‘Implicit CR in every LF’](#)
 - [4.3.4 ‘Implicit LF in every CR’](#)
 - [4.3.5 ‘Use background colour to erase screen’](#)
 - [4.3.6 ‘Enable blinking text’](#)
 - [4.3.7 ‘Answerback to ^E’](#)
 - [4.3.8 ‘Local echo’](#)
 - [4.3.9 ‘Local line editing’](#)
 - [4.3.10 Remote-controlled printing](#)
- [4.4 The Keyboard panel](#)
 - [4.4.1 Changing the action of the Backspace key](#)
 - [4.4.2 Changing the action of the Home and End keys](#)
 - [4.4.3 Changing the action of the function keys and keypad](#)
 - [4.4.4 Changing the action of the shifted arrow keys](#)

- [4.4.5 Controlling Application Cursor Keys mode](#)
- [4.4.6 Controlling Application Keypad mode](#)
- [4.4.7 Using NetHack keypad mode](#)
- [4.4.8 Enabling a DEC-like Compose key](#)
- [4.4.9 ‘Control-Alt is different from AltGr’](#)
- [4.5 The Bell panel](#)
 - [4.5.1 ‘Set the style of bell’](#)
 - [4.5.2 ‘Taskbar/caption indication on bell’](#)
 - [4.5.3 ‘Control the bell overload behaviour’](#)
- [4.6 The Features panel](#)
 - [4.6.1 Disabling application keypad and cursor keys](#)
 - [4.6.2 Disabling xterm-style mouse reporting](#)
 - [4.6.3 Disabling remote terminal resizing](#)
 - [4.6.4 Disabling switching to the alternate screen](#)
 - [4.6.5 Disabling remote window title changing](#)
 - [4.6.6 Response to remote window title querying](#)
 - [4.6.7 Disabling remote scrollback clearing](#)
 - [4.6.8 Disabling destructive backspace](#)
 - [4.6.9 Disabling remote character set configuration](#)
 - [4.6.10 Disabling Arabic text shaping](#)
 - [4.6.11 Disabling bidirectional text display](#)
- [4.7 The Window panel](#)
 - [4.7.1 Setting the size of the PuTTY window](#)
 - [4.7.2 What to do when the window is resized](#)
 - [4.7.3 Controlling scrollback](#)
 - [4.7.4 ‘Push erased text into scrollback’](#)
- [4.8 The Appearance panel](#)
 - [4.8.1 Controlling the appearance of the cursor](#)
 - [4.8.2 Controlling the font used in the terminal window](#)
 - [4.8.3 ‘Hide mouse pointer when typing in window’](#)
 - [4.8.4 Controlling the window border](#)
- [4.9 The Behaviour panel](#)
 - [4.9.1 Controlling the window title](#)
 - [4.9.2 ‘Warn before closing window’](#)
 - [4.9.3 ‘Window closes on ALT-F4’](#)
 - [4.9.4 ‘System menu appears on ALT-Space’](#)
 - [4.9.5 ‘System menu appears on Alt alone’](#)

- [4.9.6 ‘Ensure window is always on top’](#)
 - [4.9.7 ‘Full screen on Alt-Enter’](#)
- [4.10 The Translation panel](#)
 - [4.10.1 Controlling character set translation](#)
 - [4.10.2 ‘Treat CJK ambiguous characters as wide’](#)
 - [4.10.3 ‘Caps Lock acts as Cyrillic switch’](#)
 - [4.10.4 Controlling display of line-drawing characters](#)
 - [4.10.5 Controlling copy and paste of line drawing characters](#)
 - [4.10.6 Combining VT100 line-drawing with UTF-8](#)
- [4.11 The Selection panel](#)
 - [4.11.1 Changing the actions of the mouse buttons](#)
 - [4.11.2 ‘Shift overrides application’s use of mouse’](#)
 - [4.11.3 Default selection mode](#)
 - [4.11.4 Assigning copy and paste actions to clipboards](#)
 - [4.11.5 ‘Permit control characters in pasted text’](#)
- [4.12 The Copy panel](#)
 - [4.12.1 Character classes](#)
 - [4.12.2 Copying in Rich Text Format](#)
- [4.13 The Colours panel](#)
 - [4.13.1 ‘Allow terminal to specify ANSI colours’](#)
 - [4.13.2 ‘Allow terminal to use xterm 256-colour mode’](#)
 - [4.13.3 ‘Allow terminal to use 24-bit colour’](#)
 - [4.13.4 ‘Indicate bolded text by changing...’](#)
 - [4.13.5 ‘Attempt to use logical palettes’](#)
 - [4.13.6 ‘Use system colours’](#)
 - [4.13.7 Adjusting the colours in the terminal window](#)
- [4.14 The Connection panel](#)
 - [4.14.1 Using keepalives to prevent disconnection](#)
 - [4.14.2 ‘Disable Nagle’s algorithm’](#)
 - [4.14.3 ‘Enable TCP keepalives’](#)
 - [4.14.4 ‘Internet protocol version’](#)
 - [4.14.5 ‘Logical name of remote host’](#)
- [4.15 The Data panel](#)
 - [4.15.1 ‘Auto-login username’](#)
 - [4.15.2 Use of system username](#)
 - [4.15.3 ‘Terminal-type string’](#)

- [4.15.4 ‘Terminal speeds’](#)
 - [4.15.5 Setting environment variables on the server](#)
- [4.16 The Proxy panel](#)
 - [4.16.1 Setting the proxy type](#)
 - [4.16.2 Excluding parts of the network from proxying](#)
 - [4.16.3 Name resolution when using a proxy.](#)
 - [4.16.4 Username and password](#)
 - [4.16.5 Specifying the Telnet, SSH, or Local proxy command](#)
 - [4.16.6 Controlling proxy logging](#)
- [4.17 The SSH panel](#)
 - [4.17.1 Executing a specific command on the server](#)
 - [4.17.2 ‘Don’t start a shell or command at all’](#)
 - [4.17.3 ‘Enable compression’](#)
 - [4.17.4 ‘SSH protocol version’](#)
 - [4.17.5 Sharing an SSH connection between PuTTY tools](#)
- [4.18 The Kex panel](#)
 - [4.18.1 Key exchange algorithm selection](#)
 - [4.18.2 Repeat key exchange](#)
- [4.19 The Host Keys panel](#)
 - [4.19.1 Host key type selection](#)
 - [4.19.2 Preferring known host keys](#)
 - [4.19.3 Manually configuring host keys](#)
 - [4.19.4 Configuring PuTTY to accept host certificates](#)
- [4.20 The Cipher panel](#)
- [4.21 The Auth panel](#)
 - [4.21.1 ‘Display pre-authentication banner’](#)
 - [4.21.2 ‘Bypass authentication entirely’](#)
 - [4.21.3 ‘Disconnect if authentication succeeds trivially’](#)
 - [4.21.4 ‘Attempt authentication using Pageant’](#)
 - [4.21.5 ‘Attempt TIS or CryptoCard authentication’](#)
 - [4.21.6 ‘Attempt keyboard-interactive authentication’](#)
 - [4.21.7 ‘Allow agent forwarding’](#)
 - [4.21.8 ‘Allow attempted changes of username in SSH-2’](#)
- [4.22 The Credentials panel](#)
 - [4.22.1 ‘Private key file for authentication’](#)
 - [4.22.2 ‘Certificate to use with the private key’](#)
 - [4.22.3 ‘Plugin to provide authentication responses’](#)

- [4.23 The GSSAPI panel](#)
 - [4.23.1 ‘Allow GSSAPI credential delegation’](#)
 - [4.23.2 Preference order for GSSAPI libraries](#)
- [4.24 The TTY panel](#)
 - [4.24.1 ‘Don’t allocate a pseudo-terminal’](#)
 - [4.24.2 Sending terminal modes](#)
- [4.25 The X11 panel](#)
 - [4.25.1 Remote X11 authentication](#)
 - [4.25.2 X authority file for local display](#)
- [4.26 The Tunnels panel](#)
 - [4.26.1 Controlling the visibility of forwarded ports](#)
 - [4.26.2 Selecting Internet protocol version for forwarded ports](#)
- [4.27 The Bugs and More Bugs panels](#)
 - [4.27.1 ‘Chokes on SSH-2 ignore messages’](#)
 - [4.27.2 ‘Handles SSH-2 key re-exchange badly’](#)
 - [4.27.3 ‘Chokes on PuTTY’s SSH-2 ‘winadj’ requests’](#)
 - [4.27.4 ‘Replies to requests on closed channels’](#)
 - [4.27.5 ‘Ignores SSH-2 maximum packet size’](#)
 - [4.27.6 ‘Discards data sent before its greeting’](#)
 - [4.27.7 ‘Chokes on PuTTY’s full KEXINIT’](#)
 - [4.27.8 ‘Old RSA/SHA2 cert algorithm naming’](#)
 - [4.27.9 ‘Requires padding on SSH-2 RSA signatures’](#)
 - [4.27.10 ‘Only supports pre-RFC4419 SSH-2 DH GEX’](#)
 - [4.27.11 ‘Miscomputes SSH-2 HMAC keys’](#)
 - [4.27.12 ‘Misuses the session ID in SSH-2 PK auth’](#)
 - [4.27.13 ‘Miscomputes SSH-2 encryption keys’](#)
 - [4.27.14 ‘Chokes on SSH-1 ignore messages’](#)
 - [4.27.15 ‘Refuses all SSH-1 password camouflage’](#)
 - [4.27.16 ‘Chokes on SSH-1 RSA authentication’](#)
- [4.28 The ‘Bare ssh-connection’ protocol](#)
- [4.29 The Serial panel](#)
 - [4.29.1 Selecting a serial line to connect to](#)
 - [4.29.2 Selecting the speed of your serial line](#)
 - [4.29.3 Selecting the number of data bits](#)
 - [4.29.4 Selecting the number of stop bits](#)
 - [4.29.5 Selecting the serial parity checking scheme](#)

- [4.29.6 Selecting the serial flow control scheme](#)
- [4.30 The Telnet panel](#)
 - [4.30.1 'Handling of OLD_ENVIRON ambiguity'](#)
 - [4.30.2 Passive and active Telnet negotiation modes](#)
 - [4.30.3 'Keyboard sends Telnet special commands'](#)
 - [4.30.4 'Return key sends Telnet New Line instead of ^M'](#)
- [4.31 The Rlogin panel](#)
 - [4.31.1 'Local username'](#)
- [4.32 The SUPDUP panel](#)
 - [4.32.1 'Location string'](#)
 - [4.32.2 'Extended ASCII Character set'](#)
 - [4.32.3 '**MORE** processing'](#)
 - [4.32.4 'Terminal scrolling'](#)
- [4.33 Storing configuration in a file](#)

4.1 The Session panel

The Session configuration panel contains the basic options you need to specify in order to open a session at all, and also allows you to save your settings to be reloaded later.

- [4.1.1 The host name section](#)
- [4.1.2 Loading and storing saved sessions](#)
- [4.1.3 'Close window on exit'](#)

4.1.1 The host name section

The top box on the Session panel, labelled ‘Specify the destination you want to connect to’, contains the details that need to be filled in before PuTTY can open a session at all.

- The ‘Host Name’ box is where you type the name, or the IP address, of the server you want to connect to.
- The ‘Connection type’ controls let you choose what type of connection you want to make: an SSH network connection, a connection to a local serial line, or various other kinds of network connection.
 - See [section 1.2](#) for a summary of the differences between the network remote login protocols SSH, Telnet, Rlogin, and SUPDUP.
 - See [section 3.6](#) for information about using a serial line.
 - See [section 3.7](#) for an explanation of ‘raw’ connections.
 - See [section 3.8](#) for a little information about Telnet.
 - See [section 3.9](#) for information about using Rlogin.
 - See [section 3.10](#) for information about using SUPDUP.
 - The ‘Bare ssh-connection’ option in the ‘Connection type’ control is intended for specialist uses not involving network connections. See [section 4.28](#) for some information about it.
- The ‘Port’ box lets you specify which port number on the server to connect to. If you select Telnet, Rlogin, SUPDUP, or SSH, this box will be filled in automatically to the usual value, and you will only need to change it if you have an unusual server. If you

select Raw mode, you will almost certainly need to fill in the ‘Port’ box yourself.

If you select ‘Serial’ from the ‘Connection type’ radio buttons, the ‘Host Name’ and ‘Port’ boxes are replaced by ‘Serial line’ and ‘Speed’; see [section 4.29](#) for more details of these.

4.1.2 Loading and storing saved sessions

The next part of the Session configuration panel allows you to save your preferred PuTTY options so they will appear automatically the next time you start PuTTY. It also allows you to create *saved sessions*, which contain a full set of configuration options plus a host name and protocol. A saved session contains all the information PuTTY needs to start exactly the session you want.

- To save your default settings: first set up the settings the way you want them saved. Then come back to the Session panel. Select the ‘Default Settings’ entry in the saved sessions list, with a single click. Then press the ‘Save’ button.

If there is a specific host you want to store the details of how to connect to, you should create a saved session, which will be separate from the Default Settings.

- To save a session: first go through the rest of the configuration box setting up all the options you want. Then come back to the Session panel. Enter a name for the saved session in the ‘Saved Sessions’ input box. (The server name is often a good choice for a saved session name.) Then press the ‘Save’ button. Your saved session name should now appear in the list box.

You can also save settings in mid-session, from the ‘Change Settings’ dialog. Settings changed since the start of the session will be saved with their current values; as well as settings changed through the dialog, this includes changes in window size, window title changes sent by the server, and so on.

- To reload a saved session: single-click to select the session name in the list box, and then press the ‘Load’ button. Your saved settings should all appear in the configuration panel.
- To modify a saved session: first load it as described above. Then make the changes you want. Come back to the Session

panel, and press the ‘Save’ button. The new settings will be saved over the top of the old ones.

To save the new settings under a different name, you can enter the new name in the ‘Saved Sessions’ box, or single-click to select a session name in the list box to overwrite that session. To save ‘Default Settings’, you must single-click the name before saving.

- To start a saved session immediately: double-click on the session name in the list box.
- To delete a saved session: single-click to select the session name in the list box, and then press the ‘Delete’ button.

Each saved session is independent of the Default Settings configuration. If you change your preferences and update Default Settings, you must also update every saved session separately.

Saved sessions are stored in the Registry, at the location

HKEY_CURRENT_USER\Software\SimonTatham\PutTY\Sessions

If you need to store them in a file, you could try the method described in [section 4.33](#).

4.1.3 ‘Close window on exit’

Finally in the Session panel, there is an option labelled ‘Close window on exit’. This controls whether the PuTTY terminal window disappears as soon as the session inside it terminates. If you are likely to want to copy and paste text out of the session after it has terminated, or restart the session, you should arrange for this option to be off.

‘Close window on exit’ has three settings. ‘Always’ means always close the window on exit; ‘Never’ means never close on exit (always leave the window open, but inactive). The third setting, and the default one, is ‘Only on clean exit’. In this mode, a session which terminates normally will cause its window to close, but one which is aborted unexpectedly by network trouble or a confusing message from the server will leave the window up.

4.2 The Logging panel

The Logging configuration panel allows you to save log files of your PuTTY sessions, for debugging, analysis or future reference.

The main option is a radio-button set that specifies whether PuTTY will log anything at all. The options are:

- ‘None’. This is the default option; in this mode PuTTY will not create a log file at all.
- ‘Printable output’. In this mode, a log file will be created and written to, but only printable text will be saved into it. The various terminal control codes that are typically sent down an interactive session alongside the printable text will be omitted. This might be a useful mode if you want to read a log file in a text editor and hope to be able to make sense of it.
- ‘All session output’. In this mode, *everything* sent by the server into your terminal session is logged. If you view the log file in a text editor, therefore, you may well find it full of strange control characters. This is a particularly useful mode if you are experiencing problems with PuTTY’s terminal handling: you can record everything that went to the terminal, so that someone else can replay the session later in slow motion and watch to see what went wrong.
- ‘SSH packets’. In this mode (which is only used by SSH connections), the SSH message packets sent over the encrypted connection are written to the log file (as well as Event Log entries). You might need this to debug a network-level problem, or more likely to send to the PuTTY authors as part of a bug report. *BE WARNED* that if you log in using a password, the password can appear in the log file; see [section 4.2.5](#) for options that may help to remove sensitive material from the log file before you send it to anyone else.

- ‘SSH packets and raw data’. In this mode, as well as the decrypted packets (as in the previous mode), the *raw* (encrypted, compressed, etc) packets are *also* logged. This could be useful to diagnose corruption in transit. (The same caveats as the previous mode apply, of course.)

Note that the non-SSH logging options (‘Printable output’ and ‘All session output’) only work with PuTTY proper; in programs without terminal emulation (such as Plink), they will have no effect, even if enabled via saved settings.

- [4.2.1 ‘Log file name’](#)
- [4.2.2 ‘What to do if the log file already exists’](#)
- [4.2.3 ‘Flush log file frequently’](#)
- [4.2.4 ‘Include header’](#)
- [4.2.5 Options specific to SSH packet logging](#)
 - [4.2.5.1 ‘Omit known password fields’](#)
 - [4.2.5.2 ‘Omit session data’](#)

```
<code>log-server1.example.com-20010528-  
110859.dat log-unixbox.somewhere.org-20010611-  
221001.dat </code>
```

4.2.2 ‘What to do if the log file already exists’

This control allows you to specify what PuTTY should do if it tries to start writing to a log file and it finds the file already exists. You might want to automatically destroy the existing log file and start a new one with the same name. Alternatively, you might want to open the existing log file and add data to the *end* of it. Finally (the default option), you might not want to have any automatic behaviour, but to ask the user every time the problem comes up.

4.2.3 ‘Flush log file frequently’

This option allows you to control how frequently logged data is flushed to disc. By default, PuTTY will flush data as soon as it is displayed, so that if you view the log file while a session is still open, it will be up to date; and if the client system crashes, there's a greater chance that the data will be preserved.

However, this can incur a performance penalty. If PuTTY is running slowly with logging enabled, you could try unchecking this option. Be warned that the log file may not always be up to date as a result (although it will of course be flushed when it is closed, for instance at the end of a session).

4.2.4 ‘Include header’

This option allows you to choose whether to include a header line with the date and time when the log file is opened. It may be useful to disable this if the log file is being used as realtime input to other programs that don't expect the header line.

4.2.5 Options specific to SSH packet logging

These options only apply if SSH packet data is being logged.

The following options allow particularly sensitive portions of unencrypted packets to be automatically left out of the log file. They are only intended to deter casual nosiness; an attacker could glean a lot of useful information from even these obfuscated logs (e.g., length of password).

- [4.2.5.1 'Omit known password fields'](#)
- [4.2.5.2 'Omit session data'](#)

4.2.5.1 ‘Omit known password fields’

When checked, decrypted password fields are removed from the log of transmitted packets. (This includes any user responses to challenge-response authentication methods such as ‘keyboard-interactive’.) This does not include X11 authentication data if using X11 forwarding.

Note that this will only omit data that PuTTY *knows* to be a password. However, if you start another login session within your PuTTY session, for instance, any password used will appear in the clear in the packet log. The next option may be of use to protect against this.

This option is enabled by default.

4.2.5.2 ‘Omit session data’

When checked, all decrypted ‘session data’ is omitted; this is defined as data in terminal sessions and in forwarded channels (TCP, X11, and authentication agent). This will usually substantially reduce the size of the resulting log file.

This option is disabled by default.

4.3 The Terminal panel

The Terminal configuration panel allows you to control the behaviour of PuTTY's terminal emulation.

- [4.3.1 ‘Auto wrap mode initially on’](#)
- [4.3.2 ‘DEC Origin Mode initially on’](#)
- [4.3.3 ‘Implicit CR in every LF’](#)
- [4.3.4 ‘Implicit LF in every CR’](#)
- [4.3.5 ‘Use background colour to erase screen’](#)
- [4.3.6 ‘Enable blinking text’](#)
- [4.3.7 ‘Answerback to ^E’](#)
- [4.3.8 ‘Local echo’](#)
- [4.3.9 ‘Local line editing’](#)
- [4.3.10 Remote-controlled printing](#)

4.3.1 ‘Auto wrap mode initially on’

Auto wrap mode controls what happens when text printed in a PuTTY window reaches the right-hand edge of the window.

With auto wrap mode on, if a long line of text reaches the right-hand edge, it will wrap over on to the next line so you can still see all the text. With auto wrap mode off, the cursor will stay at the right-hand edge of the screen, and all the characters in the line will be printed on top of each other.

If you are running a full-screen application and you occasionally find the screen scrolling up when it looks as if it shouldn't, you could try turning this option off.

Auto wrap mode can be turned on and off by control sequences sent by the server. This configuration option controls the *default* state, which will be restored when you reset the terminal (see [section 3.1.3.6](#)). However, if you modify this option in mid-session using ‘Change Settings’, it will take effect immediately.

4.3.2 ‘DEC Origin Mode initially on’

DEC Origin Mode is a minor option which controls how PuTTY interprets cursor-position control sequences sent by the server.

The server can send a control sequence that restricts the scrolling region of the display. For example, in an editor, the server might reserve a line at the top of the screen and a line at the bottom, and might send a control sequence that causes scrolling operations to affect only the remaining lines.

With DEC Origin Mode on, cursor coordinates are counted from the top of the scrolling region. With it turned off, cursor coordinates are counted from the top of the whole screen regardless of the scrolling region.

It is unlikely you would need to change this option, but if you find a full-screen application is displaying pieces of text in what looks like the wrong part of the screen, you could try turning DEC Origin Mode on to see whether that helps.

DEC Origin Mode can be turned on and off by control sequences sent by the server. This configuration option controls the *default* state, which will be restored when you reset the terminal (see [section 3.1.3.6](#)). However, if you modify this option in mid-session using ‘Change Settings’, it will take effect immediately.

4.3.3 ‘Implicit CR in every LF’

Most servers send two control characters, CR and LF, to start a new line of the screen. The CR character makes the cursor return to the left-hand side of the screen. The LF character makes the cursor move one line down (and might make the screen scroll).

Some servers only send LF, and expect the terminal to move the cursor over to the left automatically. If you come across a server that does this, you will see a stepped effect on the screen, like this:

```
First line of text
Second line
Third line
```

If this happens to you, try enabling the ‘Implicit CR in every LF’ option, and things might go back to normal:

```
First line of text
Second line
Third line
```

4.3.4 ‘Implicit LF in every CR’

Most servers send two control characters, CR and LF, to start a new line of the screen. The CR character makes the cursor return to the left-hand side of the screen. The LF character makes the cursor move one line down (and might make the screen scroll).

Some servers only send CR, and so the newly written line is overwritten by the following line. This option causes a line feed so that all lines are displayed.

4.3.5 ‘Use background colour to erase screen’

Not all terminals agree on what colour to turn the screen when the server sends a ‘clear screen’ sequence. Some terminals believe the screen should always be cleared to the *default* background colour. Others believe the screen should be cleared to whatever the server has selected as a background colour.

There exist applications that expect both kinds of behaviour. Therefore, PuTTY can be configured to do either.

With this option disabled, screen clearing is always done in the default background colour. With this option enabled, it is done in the *current* background colour.

Background-colour erase can be turned on and off by control sequences sent by the server. This configuration option controls the *default* state, which will be restored when you reset the terminal (see [section 3.1.3.6](#)). However, if you modify this option in mid-session using ‘Change Settings’, it will take effect immediately.

4.3.6 ‘Enable blinking text’

The server can ask PuTTY to display text that blinks on and off. This is very distracting, so PuTTY allows you to turn blinking text off completely.

When blinking text is disabled and the server attempts to make some text blink, PuTTY will instead display the text with a bolded background colour.

Blinking text can be turned on and off by control sequences sent by the server. This configuration option controls the *default* state, which will be restored when you reset the terminal (see [section 3.1.3.6](#)). However, if you modify this option in mid-session using ‘Change Settings’, it will take effect immediately.

4.3.7 ‘Answerback to ^E’

This option controls what PuTTY will send back to the server if the server sends it the ^E enquiry character. Normally it just sends the string ‘PuTTY’.

If you accidentally write the contents of a binary file to your terminal, you will probably find that it contains more than one ^E character, and as a result your next command line will probably read ‘PuTTYPuTTYPuTTY...’ as if you had typed the answerback string multiple times at the keyboard. If you set the answerback string to be empty, this problem should go away, but doing so might cause other problems.

Note that this is *not* the feature of PuTTY which the server will typically use to determine your terminal type. That feature is the ‘Terminal-type string’ in the Connection panel; see [section 4.15.3](#) for details.

You can include control characters in the answerback string using ^c notation. (Use ^~ to get a literal ^.)

4.3.8 ‘Local echo’

With local echo disabled, characters you type into the PuTTY window are not echoed in the window *by PuTTY*. They are simply sent to the server. (The *server* might choose to echo them back to you; this can't be controlled from the PuTTY control panel.)

Some types of session need local echo, and many do not. In its default mode, PuTTY will automatically attempt to deduce whether or not local echo is appropriate for the session you are working in. If you find it has made the wrong decision, you can use this configuration option to override its choice: you can force local echo to be turned on, or force it to be turned off, instead of relying on the automatic detection.

4.3.9 ‘Local line editing’

Normally, every character you type into the PuTTY window is sent immediately to the server the moment you type it.

If you enable local line editing, this changes. PuTTY will let you edit a whole line at a time locally, and the line will only be sent to the server when you press Return. If you make a mistake, you can use the Backspace key to correct it before you press Return, and the server will never see the mistake.

Since it is hard to edit a line locally without being able to see it, local line editing is mostly used in conjunction with local echo ([section 4.3.8](#)). This makes it ideal for use in raw mode or when connecting to MUDs or talkers. (Although some more advanced MUDs do occasionally turn local line editing on and turn local echo off, in order to accept a password from the user.) Some types of session need local line editing, and many do not. In its default mode, PuTTY will automatically attempt to deduce whether or not local line editing is appropriate for the session you are working in. If you find it has made the wrong decision, you can use this configuration option to override its choice: you can force local line editing to be turned on, or force it to be turned off, instead of relying on the automatic detection.

4.3.10 Remote-controlled printing

A lot of VT100-compatible terminals support printing under control of the remote server (sometimes called ‘passthrough printing’). PuTTY supports this feature as well, but it is turned off by default.

To enable remote-controlled printing, choose a printer from the ‘Printer to send ANSI printer output to’ drop-down list box. This should allow you to select from all the printers you have installed drivers for on your computer. Alternatively, you can type the network name of a networked printer (for example, \\printserver\printer1) even if you haven’t already installed a driver for it on your own machine.

When the remote server attempts to print some data, PuTTY will send that data to the printer *raw* - without translating it, attempting to format it, or doing anything else to it. It is up to you to ensure your remote server knows what type of printer it is talking to.

Since PuTTY sends data to the printer raw, it cannot offer options such as portrait versus landscape, print quality, or paper tray selection. All these things would be done by your PC printer driver (which PuTTY bypasses); if you need them done, you will have to find a way to configure your remote server to do them.

To disable remote printing again, choose ‘None (printing disabled)’ from the printer selection list. This is the default state.

4.4 The Keyboard panel

The Keyboard configuration panel allows you to control the behaviour of the keyboard in PuTTY. The correct state for many of these settings depends on what the server to which PuTTY is connecting expects. With a Unix server, this is likely to depend on the `termcap` or `terminfo` entry it uses, which in turn is likely to be controlled by the ‘Terminal-type string’ setting in the Connection panel; see [section 4.15.3](#) for details. If none of the settings here seems to help, you may find [question A.7.13](#) to be useful.

- [4.4.1 Changing the action of the Backspace key](#)
- [4.4.2 Changing the action of the Home and End keys](#)
- [4.4.3 Changing the action of the function keys and keypad](#)
- [4.4.4 Changing the action of the shifted arrow keys](#)
- [4.4.5 Controlling Application Cursor Keys mode](#)
- [4.4.6 Controlling Application Keypad mode](#)
- [4.4.7 Using NetHack keypad mode](#)
- [4.4.8 Enabling a DEC-like Compose key](#)
- [4.4.9 ‘Control-Alt is different from AltGr’](#)

4.4.1 Changing the action of the Backspace key

Some terminals believe that the Backspace key should send the same thing to the server as Control-H (ASCII code 8). Other terminals believe that the Backspace key should send ASCII code 127 (usually known as Control-?) so that it can be distinguished from Control-H. This option allows you to choose which code PuTTY generates when you press Backspace.

If you are connecting over SSH, PuTTY by default tells the server the value of this option (see [section 4.24.2](#)), so you may find that the Backspace key does the right thing either way. Similarly, if you are connecting to a Unix system, you will probably find that the Unix stty command lets you configure which the server expects to see, so again you might not need to change which one PuTTY generates. On other systems, the server's expectation might be fixed and you might have no choice but to configure PuTTY.

If you do have the choice, we recommend configuring PuTTY to generate Control-? and configuring the server to expect it, because that allows applications such as emacs to use Control-H for help.

(Typing Shift-Backspace will cause PuTTY to send whichever code isn't configured here as the default.)

4.4.2 Changing the action of the Home and End keys

The Unix terminal emulator rxvt disagrees with the rest of the world about what character sequences should be sent to the server by the Home and End keys.

xterm, and other terminals, send `ESC [1~` for the Home key, and `ESC [4~` for the End key. rxvt sends `ESC [H` for the Home key and `ESC [0w` for the End key.

If you find an application on which the Home and End keys aren't working, you could try switching this option to see if it helps.

4.4.3 Changing the action of the function keys and keypad

This option affects the function keys (F1 to F12) and the top row of the numeric keypad.

- In the default mode, labelled `ESC [n~`, the function keys generate sequences like `ESC [11~`, `ESC [12~` and so on. This matches the general behaviour of Digital's terminals.
- In Linux mode, F6 to F12 behave just like the default mode, but F1 to F5 generate `ESC [[A` through to `ESC [[E`. This mimics the Linux virtual console.
- In Xterm R6 mode, F5 to F12 behave like the default mode, but F1 to F4 generate `ESC OP` through to `ESC OS`, which are the sequences produced by the top row of the *keypad* on Digital's terminals.
- In VT400 mode, all the function keys behave like the default mode, but the actual top row of the numeric keypad generates `ESC OP` through to `ESC OS`.
- In VT100+ mode, the function keys generate `ESC OP` through to `ESC O[`
- In SCO mode, the function keys F1 to F12 generate `ESC [M` through to `ESC [x`. Together with shift, they generate `ESC [Y` through to `ESC [j`. With control they generate `ESC [k` through to `ESC [v`, and with shift and control together they generate `ESC [w` through to `ESC [{`.
- In Xterm 216 mode, the unshifted function keys behave the same as Xterm R6 mode. But pressing a function key together with Shift or Alt or Ctrl generates a different sequence containing an extra numeric parameter of the form (1 for Shift) + (2 for Alt) + (4 for Ctrl) + 1. For F1-F4, the basic sequences like

`ESC O P` become `ESC [1;bitmapP` and similar; for F5 and above,
`ESC[index~` becomes `ESC[index;bitmap~`.

If you don't know what any of this means, you probably don't need to fiddle with it.

4.4.4 Changing the action of the shifted arrow keys

This option affects the arrow keys, if you press one with any of the modifier keys Shift, Ctrl or Alt held down.

- In the default mode, labelled `ctrl` toggles app mode, the `Ctrl` key toggles between the default arrow-key sequences like `ESC [A` and `ESC [B`, and the sequences Digital's terminals generate in ‘application cursor keys’ mode, i.e. `ESC o A` and so on. Shift and Alt have no effect.
- In the ‘xterm-style bitmap’ mode, Shift, Ctrl and Alt all generate different sequences, with a number indicating which set of modifiers is active.

If you don't know what any of this means, you probably don't need to fiddle with it.

4.4.5 Controlling Application Cursor Keys mode

Application Cursor Keys mode is a way for the server to change the control sequences sent by the arrow keys. In normal mode, the arrow keys send `ESC [A` through to `ESC [D`. In application mode, they send `ESC OA` through to `ESC OD`.

Application Cursor Keys mode can be turned on and off by the server, depending on the application. PuTTY allows you to configure the initial state.

You can also disable application cursor keys mode completely, using the ‘Features’ configuration panel; see [section 4.6.1](#).

4.4.6 Controlling Application Keypad mode

Application Keypad mode is a way for the server to change the behaviour of the numeric keypad.

In normal mode, the keypad behaves like a normal Windows keypad: with NumLock on, the number keys generate numbers, and with NumLock off they act like the arrow keys and Home, End etc.

In application mode, all the keypad keys send special control sequences, *including* Num Lock. Num Lock stops behaving like Num Lock and becomes another function key.

Depending on which version of Windows you run, you may find the Num Lock light still flashes on and off every time you press Num Lock, even when application mode is active and Num Lock is acting like a function key. This is unavoidable.

Application keypad mode can be turned on and off by the server, depending on the application. PUTTY allows you to configure the initial state.

You can also disable application keypad mode completely, using the 'Features' configuration panel; see [section 4.6.1](#).

4.4.7 Using NetHack keypad mode

PuTTY has a special mode for playing NetHack. You can enable it by selecting ‘NetHack’ in the ‘Initial state of numeric keypad’ control.

In this mode, the numeric keypad keys 1-9 generate the NetHack movement commands (hjklyubn). The 5 key generates the . command (do nothing).

In addition, pressing Shift or Ctrl with the keypad keys generate the Shift- or Ctrl-keys you would expect (e.g. keypad-7 generates ‘y’, so Shift-keypad-7 generates ‘Y’ and Ctrl-keypad-7 generates Ctrl-Y); these commands tell NetHack to keep moving you in the same direction until you encounter something interesting.

For some reason, this feature only works properly when Num Lock is on. We don't know why.

4.4.8 Enabling a DEC-like Compose key

DEC terminals have a Compose key, which provides an easy-to-remember way of typing accented characters. You press Compose and then type two more characters. The two characters are ‘combined’ to produce an accented character. The choices of character are designed to be easy to remember; for example, composing ‘e’ and ‘`’ produces the ‘è’ character.

If your keyboard has a Windows Application key, it acts as a Compose key in PuTTY. Alternatively, if you enable the ‘AltGr acts as Compose key’ option, the AltGr key will become a Compose key.

4.4.9 ‘Control-Alt is different from AltGr’

Some old keyboards do not have an AltGr key, which can make it difficult to type some characters. PuTTY can be configured to treat the key combination Ctrl + Left Alt the same way as the AltGr key.

By default, this checkbox is checked, and the key combination Ctrl + Left Alt does something completely different. PuTTY's usual handling of the left Alt key is to prefix the Escape (Control-[) character to whatever character sequence the rest of the keypress would generate. For example, Alt-A generates Escape followed by a. So Alt-Ctrl-A would generate Escape, followed by Control-A.

If you uncheck this box, Ctrl-Alt will become a synonym for AltGr, so you can use it to type extra graphic characters if your keyboard has any.

(However, Ctrl-Alt will never act as a Compose key, regardless of the setting of ‘AltGr acts as Compose key’ described in [section 4.4.8](#).)

4.5 The Bell panel

The Bell panel controls the terminal bell feature: the server's ability to cause PuTTY to beep at you.

In the default configuration, when the server sends the character with ASCII code 7 (Control-G), PuTTY will play the Windows Default Beep sound. This is not always what you want the terminal bell feature to do; the Bell panel allows you to configure alternative actions.

- [4.5.1 ‘Set the style of bell’](#)
- [4.5.2 ‘Taskbar/caption indication on bell’](#)
- [4.5.3 ‘Control the bell overload behaviour’](#)

4.5.1 ‘Set the style of bell’

This control allows you to select various different actions to occur on a terminal bell:

- Selecting ‘None’ disables the bell completely. In this mode, the server can send as many Control-G characters as it likes and nothing at all will happen.
- ‘Make default system alert sound’ is the default setting. It causes the Windows ‘Default Beep’ sound to be played. To change what this sound is, or to test it if nothing seems to be happening, use the Sound configurer in the Windows Control Panel.
- ‘Visual bell’ is a silent alternative to a beeping computer. In this mode, when the server sends a Control-G, the whole PuTTY window will flash white for a fraction of a second.
- ‘Beep using the PC speaker’ is self-explanatory.
- ‘Play a custom sound file’ allows you to specify a particular sound file to be used by PuTTY alone, or even by a particular individual PuTTY session. This allows you to distinguish your PuTTY beeps from any other beeps on the system. If you select this option, you will also need to enter the name of your sound file in the edit control ‘Custom sound file to play as a bell’.

4.5.2 ‘Taskbar/caption indication on bell’

This feature controls what happens to the PuTTY window's entry in the Windows Taskbar if a bell occurs while the window does not have the input focus.

In the default state ('Disabled') nothing unusual happens.

If you select 'Steady', then when a bell occurs and the window is not in focus, the window's Taskbar entry and its title bar will change colour to let you know that PuTTY session is asking for your attention. The change of colour will persist until you select the window, so you can leave several PuTTY windows minimised in your terminal, go away from your keyboard, and be sure not to have missed any important beeps when you get back.

'Flashing' is even more eye-catching: the Taskbar entry will continuously flash on and off until you select the window.

4.5.3 ‘Control the bell overload behaviour’

A common user error in a terminal session is to accidentally run the Unix command `cat` (or equivalent) on an inappropriate file type, such as an executable, image file, or ZIP file. This produces a huge stream of non-text characters sent to the terminal, which typically includes a lot of bell characters. As a result of this the terminal often doesn't stop beeping for ten minutes, and everybody else in the office gets annoyed.

To try to avoid this behaviour, or any other cause of excessive beeping, PuTTY includes a bell overload management feature. In the default configuration, receiving more than five bell characters in a two-second period will cause the overload feature to activate. Once the overload feature is active, further bells will have no effect at all, so the rest of your binary file will be sent to the screen in silence. After a period of five seconds during which no further bells are received, the overload feature will turn itself off again and bells will be re-enabled.

If you want this feature completely disabled, you can turn it off using the checkbox ‘Bell is temporarily disabled when over-used’.

Alternatively, if you like the bell overload feature but don't agree with the settings, you can configure the details: how many bells constitute an overload, how short a time period they have to arrive in to do so, and how much silent time is required before the overload feature will deactivate itself.

Bell overload mode is always deactivated by any keypress in the terminal. This means it can respond to large unexpected streams of data, but does not interfere with ordinary command-line activities that generate beeps (such as filename completion).

4.6 The Features panel

PuTTY's terminal emulation is very highly featured, and can do a lot of things under remote server control. Some of these features can cause problems due to buggy or strangely configured server applications.

The Features configuration panel allows you to disable some of PuTTY's more advanced terminal features, in case they cause trouble.

- [4.6.1 Disabling application keypad and cursor keys](#)
- [4.6.2 Disabling xterm-style mouse reporting](#)
- [4.6.3 Disabling remote terminal resizing](#)
- [4.6.4 Disabling switching to the alternate screen](#)
- [4.6.5 Disabling remote window title changing](#)
- [4.6.6 Response to remote window title querying](#)
- [4.6.7 Disabling remote scrollback clearing](#)
- [4.6.8 Disabling destructive backspace](#)
- [4.6.9 Disabling remote character set configuration](#)
- [4.6.10 Disabling Arabic text shaping](#)
- [4.6.11 Disabling bidirectional text display](#)

4.6.1 Disabling application keypad and cursor keys

Application keypad mode (see [section 4.4.6](#)) and application cursor keys mode (see [section 4.4.5](#)) alter the behaviour of the keypad and cursor keys. Some applications enable these modes but then do not deal correctly with the modified keys. You can force these modes to be permanently disabled no matter what the server tries to do.

4.6.2 Disabling xterm-style mouse reporting

PuTTY allows the server to send control codes that let it take over the mouse and use it for purposes other than copy and paste. Applications which use this feature include the text-mode web browser `links`, the Usenet newsreader `trn` version 4, and the file manager `mc` (Midnight Commander).

If you find this feature inconvenient, you can disable it using the ‘Disable xterm-style mouse reporting’ control. With this box ticked, the mouse will *always* do copy and paste in the normal way.

Note that even if the application takes over the mouse, you can still manage PuTTY’s copy and paste by holding down the Shift key while you select and paste, unless you have deliberately turned this feature off (see [section 4.11.2](#)).

4.6.3 Disabling remote terminal resizing

PuTTY has the ability to change the terminal's size and position in response to commands from the server. If you find PuTTY is doing this unexpectedly or inconveniently, you can tell PuTTY not to respond to those server commands.

4.6.4 Disabling switching to the alternate screen

Many terminals, including PuTTY, support an ‘alternate screen’. This is the same size as the ordinary terminal screen, but separate.

Typically a screen-based program such as a text editor might switch the terminal to the alternate screen before starting up. Then at the end of the run, it switches back to the primary screen, and you see the screen contents just as they were before starting the editor.

Some people prefer this not to happen. If you want your editor to run in the same screen as the rest of your terminal activity, you can disable the alternate screen feature completely.

4.6.5 Disabling remote window title changing

PuTTY has the ability to change the window title in response to commands from the server. If you find PuTTY is doing this unexpectedly or inconveniently, you can tell PuTTY not to respond to those server commands.

4.6.6 Response to remote window title querying

PuTTY can optionally provide the xterm service of allowing server applications to find out the local window title. This feature is disabled by default, but you can turn it on if you really want it.

NOTE that this feature is a *potential security hazard*. If a malicious application can write data to your terminal (for example, if you merely `cat` a file owned by someone else on the server machine), it can change your window title (unless you have disabled this as mentioned in [section 4.6.5](#)) and then use this service to have the new window title sent back to the server as if typed at the keyboard. This allows an attacker to fake keypresses and potentially cause your server-side applications to do things you didn't want. Therefore this feature is disabled by default, and we recommend you do not set it to 'Window title' unless you *really* know what you are doing.

There are three settings for this option:

'None'

PuTTY makes no response whatsoever to the relevant escape sequence. This may upset server-side software that is expecting some sort of response.

'Empty string'

PuTTY makes a well-formed response, but leaves it blank. Thus, server-side software that expects a response is kept happy, but an attacker cannot influence the response string. This is probably the setting you want if you have no better ideas.

'Window title'

PuTTY responds with the actual window title. This is dangerous for the reasons described above.

4.6.7 Disabling remote scrollback clearing

PuTTY has the ability to clear the terminal's scrollback buffer in response to a command from the server. If you find PuTTY is doing this unexpectedly or inconveniently, you can tell PuTTY not to respond to that server command.

4.6.8 Disabling destructive backspace

Normally, when PuTTY receives character 127 (^?) from the server, it will perform a ‘destructive backspace’: move the cursor one space left and delete the character under it. This can apparently cause problems in some applications, so PuTTY provides the ability to configure character 127 to perform a normal backspace (without deleting a character) instead.

4.6.9 Disabling remote character set configuration

PuTTY has the ability to change its character set configuration in response to commands from the server. Some programs send these commands unexpectedly or inconveniently. In particular, BitchX (an IRC client) seems to have a habit of reconfiguring the character set to something other than the user intended.

If you find that accented characters are not showing up the way you expect them to, particularly if you're running BitchX, you could try disabling the remote character set configuration commands.

4.6.10 Disabling Arabic text shaping

PuTTY supports shaping of Arabic text, which means that if your server sends text written in the basic Unicode Arabic alphabet then it will convert it to the correct display forms before printing it on the screen.

If you are using full-screen software which was not expecting this to happen (especially if you are not an Arabic speaker and you unexpectedly find yourself dealing with Arabic text files in applications which are not Arabic-aware), you might find that the display becomes corrupted. By ticking this box, you can disable Arabic text shaping so that PuTTY displays precisely the characters it is told to display.

You may also find you need to disable bidirectional text display; see [section 4.6.11](#).

4.6.11 Disabling bidirectional text display

PuTTY supports bidirectional text display, which means that if your server sends text written in a language which is usually displayed from right to left (such as Arabic or Hebrew) then PuTTY will automatically flip it round so that it is displayed in the right direction on the screen.

If you are using full-screen software which was not expecting this to happen (especially if you are not an Arabic speaker and you unexpectedly find yourself dealing with Arabic text files in applications which are not Arabic-aware), you might find that the display becomes corrupted. By ticking this box, you can disable bidirectional text display, so that PuTTY displays text from left to right in all situations.

You may also find you need to disable Arabic text shaping; see [section 4.6.10](#).

4.7 The Window panel

The Window configuration panel allows you to control aspects of the PuTTY window.

- [4.7.1 Setting the size of the PuTTY window](#)
- [4.7.2 What to do when the window is resized](#)
- [4.7.3 Controlling scrollback](#)
- [4.7.4 ‘Push erased text into scrollback’](#)

4.7.1 Setting the size of the PuTTY window

The ‘Columns’ and ‘Rows’ boxes let you set the PuTTY window to a precise size. Of course you can also drag the window to a new size while a session is running.

4.7.2 What to do when the window is resized

These options allow you to control what happens when the user tries to resize the PuTTY window using its window furniture.

There are four options here:

- ‘Change the number of rows and columns’: the font size will not change. (This is the default.)
- ‘Change the size of the font’: the number of rows and columns in the terminal will stay the same, and the font size will change.
- ‘Change font size when maximised’: when the window is resized, the number of rows and columns will change, except when the window is maximised (or restored), when the font size will change. (In this mode, holding down the Alt key while resizing will also cause the font size to change.)
- ‘Forbid resizing completely’: the terminal will refuse to be resized at all.

4.7.3 Controlling scrollback

These options let you configure the way PuTTY keeps text after it scrolls off the top of the screen (see [section 3.1.2](#)).

The ‘Lines of scrollback’ box lets you configure how many lines of text PuTTY keeps. The ‘Display scrollbar’ options allow you to hide the scrollbar (although you can still view the scrollback using the keyboard as described in [section 3.1.2](#)). You can separately configure whether the scrollbar is shown in full-screen mode and in normal modes.

If you are viewing part of the scrollback when the server sends more text to PuTTY, the screen will revert to showing the current terminal contents. You can disable this behaviour by turning off ‘Reset scrollback on display activity’. You can also make the screen revert when you press a key, by turning on ‘Reset scrollback on keypress’.

4.7.4 ‘Push erased text into scrollback’

When this option is enabled, the contents of the terminal screen will be pushed into the scrollback when a server-side application clears the screen, so that your scrollback will contain a better record of what was on your screen in the past.

If the application switches to the alternate screen (see [section 4.6.4](#) for more about this), then the contents of the primary screen will be visible in the scrollback until the application switches back again.

This option is enabled by default.

4.8 The Appearance panel

The Appearance configuration panel allows you to control aspects of the appearance of PuTTY's window.

- [4.8.1 Controlling the appearance of the cursor](#)
- [4.8.2 Controlling the font used in the terminal window](#)
- [4.8.3 ‘Hide mouse pointer when typing in window’](#)
- [4.8.4 Controlling the window border](#)

4.8.1 Controlling the appearance of the cursor

The ‘Cursor appearance’ option lets you configure the cursor to be a block, an underline, or a vertical line. A block cursor becomes an empty box when the window loses focus; an underline or a vertical line becomes dotted.

The ‘Cursor blinks’ option makes the cursor blink on and off. This works in any of the cursor modes.

4.8.2 Controlling the font used in the terminal window

This option allows you to choose what font, in what size, the PuTTY terminal window uses to display the text in the session.

By default, you will be offered a choice from all the fixed-width fonts installed on the system, since VT100-style terminal handling expects a fixed-width font. If you tick the box marked ‘Allow selection of variable-pitch fonts’, however, PuTTY will offer variable-width fonts as well: if you select one of these, the font will be coerced into fixed-size character cells, which will probably not look very good (but can work OK with some fonts).

4.8.3 ‘Hide mouse pointer when typing in window’

If you enable this option, the mouse pointer will disappear if the PuTTY window is selected and you press a key. This way, it will not obscure any of the text in the window while you work in your session. As soon as you move the mouse, the pointer will reappear.

This option is disabled by default, so the mouse pointer remains visible at all times.

4.8.4 Controlling the window border

PuTTY allows you to configure the appearance of the window border to some extent.

The checkbox marked ‘Sunken-edge border’ changes the appearance of the window border to something more like a DOS box: the inside edge of the border is highlighted as if it sank down to meet the surface inside the window. This makes the border a little bit thicker as well. It’s hard to describe well. Try it and see if you like it.

You can also configure a completely blank gap between the text in the window and the border, using the ‘Gap between text and window edge’ control. By default this is set at one pixel. You can reduce it to zero, or increase it further.

4.9 The Behaviour panel

The Behaviour configuration panel allows you to control aspects of the behaviour of PuTTY's window.

- [4.9.1 Controlling the window title](#)
- [4.9.2 'Warn before closing window'](#)
- [4.9.3 'Window closes on ALT-F4'](#)
- [4.9.4 'System menu appears on ALT-Space'](#)
- [4.9.5 'System menu appears on Alt alone'](#)
- [4.9.6 'Ensure window is always on top'](#)
- [4.9.7 'Full screen on Alt-Enter'](#)

4.9.1 Controlling the window title

The ‘Window title’ edit box allows you to set the title of the PuTTY window. By default the window title will contain the host name followed by ‘PuTTY’, for example `server1.example.com - PuTTY`. If you want a different window title, this is where to set it.

PuTTY allows the server to send `xterm` control sequences which modify the title of the window in mid-session (unless this is disabled - see [section 4.6.5](#)); the title string set here is therefore only the *initial* window title.

As well as the *window* title, there is also an `xterm` sequence to modify the title of the window’s *icon*. This makes sense in a windowing system where the window becomes an icon when minimised, such as Windows 3.1 or most X Window System setups; but in the Windows 95-like user interface it isn’t as applicable.

By default, PuTTY only uses the server-supplied *window* title, and ignores the icon title entirely. If for some reason you want to see both titles, check the box marked ‘Separate window and icon titles’. If you do this, PuTTY’s window title and Taskbar caption will change into the server-supplied icon title if you minimise the PuTTY window, and change back to the server-supplied window title if you restore it. (If the server has not bothered to supply a window or icon title, none of this will happen.)

4.9.2 ‘Warn before closing window’

If you press the Close button in a PuTTY window that contains a running session, PuTTY will put up a warning window asking if you really meant to close the window. A window whose session has already terminated can always be closed without a warning.

If you want to be able to close a window quickly, you can disable the ‘Warn before closing window’ option.

4.9.3 ‘Window closes on ALT-F4’

By default, pressing ALT-F4 causes the window to close (or a warning box to appear; see [section 4.9.2](#)). If you disable the ‘Window closes on ALT-F4’ option, then pressing ALT-F4 will simply send a key sequence to the server.

4.9.4 ‘System menu appears on ALT-Space’

If this option is enabled, then pressing ALT-Space will bring up the PuTTY window's menu, like clicking on the top left corner. If it is disabled, then pressing ALT-Space will just send `ESC SPACE` to the server.

Some accessibility programs for Windows may need this option enabling to be able to control PuTTY's window successfully. For instance, Dragon NaturallySpeaking requires it both to open the system menu via voice, and to close, minimise, maximise and restore the window.

4.9.5 ‘System menu appears on Alt alone’

If this option is enabled, then pressing and releasing ALT will bring up the PuTTY window's menu, like clicking on the top left corner. If it is disabled, then pressing and releasing ALT will have no effect.

4.9.6 ‘Ensure window is always on top’

If this option is enabled, the PuTTY window will stay on top of all other windows.

4.9.7 ‘Full screen on Alt-Enter’

If this option is enabled, then pressing Alt-Enter will cause the PuTTY window to become full-screen. Pressing Alt-Enter again will restore the previous window size.

The full-screen feature is also available from the System menu, even when it is configured not to be available on the Alt-Enter key. See [section 3.1.3.7](#).

4.10 The Translation panel

The Translation configuration panel allows you to control the translation between the character set understood by the server and the character set understood by PuTTY.

- [4.10.1 Controlling character set translation](#)
- [4.10.2 ‘Treat CJK ambiguous characters as wide’](#)
- [4.10.3 ‘Caps Lock acts as Cyrillic switch’](#)
- [4.10.4 Controlling display of line-drawing characters](#)
- [4.10.5 Controlling copy and paste of line drawing characters](#)
- [4.10.6 Combining VT100 line-drawing with UTF-8](#)

4.10.1 Controlling character set translation

During an interactive session, PuTTY receives a stream of 8-bit bytes from the server, and in order to display them on the screen it needs to know what character set to interpret them in. Similarly, PuTTY needs to know how to translate your keystrokes into the encoding the server expects. Unfortunately, there is no satisfactory mechanism for PuTTY and the server to communicate this information, so it must usually be manually configured.

There are a lot of character sets to choose from. The ‘Remote character set’ option lets you select one.

By default PuTTY will use the UTF-8 encoding of Unicode, which can represent pretty much any character; data coming from the server is interpreted as UTF-8, and keystrokes are sent UTF-8 encoded. This is what most modern distributions of Linux will expect by default. However, if this is wrong for your server, you can select a different character set using this control.

A few other notable character sets are:

- The ISO-8859 series are all standard character sets that include various accented characters appropriate for different sets of languages.
- The Win125x series are defined by Microsoft, for similar purposes. In particular Win1252 is almost equivalent to ISO-8859-1, but contains a few extra characters such as matched quotes and the Euro symbol.
- If you want the old IBM PC character set with block graphics and line-drawing characters, you can select ‘CP437’.

If you need support for a numeric code page which is not listed in the drop-down list, such as code page 866, then you can try entering its name manually (cp866 for example) in the list box. If the underlying version of Windows has the appropriate translation table installed, PuTTY will use it.

4.10.2 ‘Treat CJK ambiguous characters as wide’

There are some Unicode characters whose width is not well-defined. In most contexts, such characters should be treated as single-width for the purposes of wrapping and so on; however, in some CJK contexts, they are better treated as double-width for historical reasons, and some server-side applications may expect them to be displayed as such. Setting this option will cause PuTTY to take the double-width interpretation.

If you use legacy CJK applications, and you find your lines are wrapping in the wrong places, or you are having other display problems, you might want to play with this setting.

This option only has any effect in UTF-8 mode (see [section 4.10.1](#)).

4.10.3 ‘Caps Lock acts as Cyrillic switch’

This feature allows you to switch between a US/UK keyboard layout and a Cyrillic keyboard layout by using the Caps Lock key, if you need to type (for example) Russian and English side by side in the same document.

Currently this feature is not expected to work properly if your native keyboard layout is not US or UK.

4.10.4 Controlling display of line-drawing characters

VT100-series terminals allow the server to send control sequences that shift temporarily into a separate character set for drawing simple lines and boxes. However, there are a variety of ways in which PuTTY can attempt to find appropriate characters, and the right one to use depends on the locally configured font. In general you should probably try lots of options until you find one that your particular font supports.

- ‘Use Unicode line drawing code points’ tries to use the box characters that are present in Unicode. For good Unicode-supporting fonts this is probably the most reliable and functional option.
- ‘Poor man’s line drawing’ assumes that the font *cannot* generate the line and box characters at all, so it will use the +, - and | characters to draw approximations to boxes. You should use this option if none of the other options works.
- ‘Font has XWindows encoding’ is for use with fonts that have a special encoding, where the lowest 32 character positions (below the ASCII printable range) contain the line-drawing characters. This is unlikely to be the case with any standard Windows font; it will probably only apply to custom-built fonts or fonts that have been automatically converted from the X Window System.
- ‘Use font in both ANSI and OEM modes’ tries to use the same font in two different character sets, to obtain a wider range of characters. This doesn’t always work; some fonts claim to be a different size depending on which character set you try to use.
- ‘Use font in OEM mode only’ is more reliable than that, but can miss out other characters from the main character set.

4.10.5 Controlling copy and paste of line drawing characters

By default, when you copy and paste a piece of the PuTTY screen that contains VT100 line and box drawing characters, PuTTY will paste them in the form they appear on the screen: either Unicode line drawing code points, or the ‘poor man’s’ line-drawing characters +, - and |. The checkbox ‘Copy and paste VT100 line drawing chars as lqqqk’ disables this feature, so line-drawing characters will be pasted as the ASCII characters that were printed to produce them. This will typically mean they come out mostly as q and x, with a scattering of jklmntuvw at the corners. This might be useful if you were trying to recreate the same box layout in another program, for example.

Note that this option only applies to line-drawing characters which were printed by using the VT100 mechanism. Line-drawing characters that were received as Unicode code points will paste as Unicode always.

4.10.6 Combining VT100 line-drawing with UTF-8

If PuTTY is configured to treat data from the server as encoded in UTF-8, then by default it disables the older VT100-style system of control sequences that cause the lower-case letters to be temporarily replaced by line drawing characters.

The rationale is that in UTF-8 mode you don't need those control sequences anyway, because all the line-drawing characters they access are available as Unicode characters already, so there's no need for applications to put the terminal into a special state to get at them.

Also, it removes a risk of the terminal *accidentally* getting into that state: if you accidentally write uncontrolled binary data to a non-UTF-8 terminal, it can be surprisingly common to find that your next shell prompt appears as a sequence of line-drawing characters and then you have to remember or look up how to get out of that mode. So by default, UTF-8 mode simply doesn't *have* a confusing mode like that to get into, accidentally or on purpose.

However, not all applications will see it that way. Even UTF-8 terminal users will still sometimes have to run software that tries to print line-drawing characters in the old-fashioned way. So the configuration option 'Enable VT100 line drawing even in UTF-8 mode' puts PuTTY into a hybrid mode in which it understands the VT100-style control sequences that change the meaning of the ASCII lower case letters, *and* understands UTF-8.

4.11 The Selection panel

The Selection panel allows you to control the way copy and paste work in the PuTTY window.

- [4.11.1 Changing the actions of the mouse buttons](#)
- [4.11.2 'Shift overrides application's use of mouse'](#)
- [4.11.3 Default selection mode](#)
- [4.11.4 Assigning copy and paste actions to clipboards](#)
 - [4.11.4.1 'Auto-copy selected text'](#)
 - [4.11.4.2 Choosing a clipboard for UI actions](#)
- [4.11.5 'Permit control characters in pasted text'](#)

4.11.1 Changing the actions of the mouse buttons

PuTTY's copy and paste mechanism is by default modelled on the Unix `xterm` application. The X Window System uses a three-button mouse, and the convention in that system is that the left button selects, the right button extends an existing selection, and the middle button pastes.

Windows often only has two mouse buttons, so when run on Windows, PuTTY is configurable. In PuTTY's default configuration ('Compromise'), the *right* button pastes, and the *middle* button (if you have one) extends a selection.

If you have a three-button mouse and you are already used to the `xterm` arrangement, you can select it using the 'Action of mouse buttons' control.

Alternatively, with the 'Windows' option selected, the middle button extends, and the right button brings up a context menu (on which one of the options is 'Paste'). (This context menu is always available by holding down Ctrl and right-clicking, regardless of the setting of this option.)

(When PuTTY itself is running on Unix, it follows the X Window System convention.)

4.11.2 ‘Shift overrides application's use of mouse’

PuTTY allows the server to send control codes that let it take over the mouse and use it for purposes other than copy and paste. Applications which use this feature include the text-mode web browser `links`, the Usenet newsreader `trn` version 4, and the file manager `mc` (Midnight Commander).

When running one of these applications, pressing the mouse buttons no longer performs copy and paste. If you do need to copy and paste, you can still do so if you hold down Shift while you do your mouse clicks.

However, it is possible in theory for applications to even detect and make use of Shift + mouse clicks. We don't know of any applications that do this, but in case someone ever writes one, unchecking the ‘Shift overrides application's use of mouse’ checkbox will cause Shift + mouse clicks to go to the server as well (so that mouse-driven copy and paste will be completely disabled).

If you want to prevent the application from taking over the mouse at all, you can do this using the Features control panel; see [section 4.6.2](#).

4.11.3 Default selection mode

As described in [section 3.1.1](#), PuTTY has two modes of selecting text to be copied to the clipboard. In the default mode ('Normal'), dragging the mouse from point A to point B selects to the end of the line containing A, all the lines in between, and from the very beginning of the line containing B. In the other mode ('Rectangular block'), dragging the mouse between two points defines a rectangle, and everything within that rectangle is copied.

Normally, you have to hold down Alt while dragging the mouse to select a rectangular block. Using the 'Default selection mode' control, you can set rectangular selection as the default, and then you have to hold down Alt to get the *normal* behaviour.

4.11.4 Assigning copy and paste actions to clipboards

Here you can configure which clipboard(s) are written or read by PuTTY's various copy and paste actions.

Most platforms, including Windows, have a single system clipboard. On these platforms, PuTTY provides a second clipboard-like facility by permitting you to paste the text you last selected in *this window*, whether or not it is currently also in the system clipboard. This is not enabled by default.

The X Window System (which underlies most Unix graphical interfaces) provides multiple clipboards (or 'selections'), and many applications support more than one of them by a different user interface mechanism. When PuTTY itself is running on Unix, it has more configurability relating to these selections.

The two most commonly used selections are called 'PRIMARY' and 'CLIPBOARD'; in applications supporting both, the usual behaviour is that PRIMARY is used by mouse-only actions (selecting text automatically copies it to PRIMARY, and middle-clicking pastes from PRIMARY), whereas CLIPBOARD is used by explicit Copy and Paste menu items or keypresses such as Ctrl-C and Ctrl-V.

- [4.11.4.1 'Auto-copy selected text'](#)
- [4.11.4.2 Choosing a clipboard for UI actions](#)

4.11.4.1 ‘Auto-copy selected text’

The checkbox ‘Auto-copy selected text to system clipboard’ controls whether or not selecting text in the PuTTY terminal window automatically has the side effect of copying it to the system clipboard, without requiring a separate user interface action.

On X, the wording of this option is changed slightly so that ‘CLIPBOARD’ is mentioned in place of the ‘system clipboard’. Text selected in the terminal window will *always* be automatically placed in the PRIMARY selection, as is conventional, but if you tick this box, it will *also* be placed in ‘CLIPBOARD’ at the same time.

4.11.4.2 Choosing a clipboard for UI actions

PuTTY has three user-interface actions which can be configured to paste into the terminal (not counting menu items). You can click whichever mouse button (if any) is configured to paste (see [section 4.11.1](#)); you can press Shift-Ins; or you can press Ctrl-Shift-V, although that action is not enabled by default.

You can configure which of the available clipboards each of these actions pastes from (including turning the paste action off completely). On platforms with a single system clipboard (such as Windows), the available options are to paste from that clipboard or to paste from PuTTY's internal memory of the last selected text within that window. On X, the standard options are CLIPBOARD OR PRIMARY.

(PRIMARY is conceptually similar in that it *also* refers to the last selected text – just across all applications instead of just this window.)

The two keyboard options each come with a corresponding key to copy to the same clipboard. Whatever you configure Shift-Ins to paste from, Ctrl-Ins will copy to the same location; similarly, Ctrl-Shift-C will copy to whatever Ctrl-Shift-V pastes from.

On X, you can also enter a selection name of your choice. For example, there is a rarely-used standard selection called 'SECONDARY', which Emacs (for example) can work with if you hold down the Meta key while dragging to select or clicking to paste; if you configure a PuTTY keyboard action to access this clipboard, then you can interoperate with other applications' use of it. Another thing you could do would be to invent a clipboard name yourself, to create a special clipboard shared *only* between instances of PuTTY, or between just instances configured in that particular way.

4.11.5 ‘Permit control characters in pasted text’

It is possible for the clipboard to contain not just text (with newlines and tabs) but also control characters such as ESC which could have surprising effects if pasted into a terminal session, depending on what program is running on the server side. Copying text from a mischievous web page could put such characters onto the clipboard.

By default, PuTTY filters out the more unusual control characters, only letting through the more obvious text-formatting characters (newlines, tab, backspace, and DEL).

Setting this option stops this filtering; on paste, any character on the clipboard is sent to the session uncensored. This might be useful if you are deliberately using control character pasting as a simple form of scripting, for instance.

4.12 The Copy panel

The Copy configuration panel controls behaviour specifically related to copying from the terminal window to the clipboard.

- [4.12.1 Character classes](#)
- [4.12.2 Copying in Rich Text Format](#)

4.12.1 Character classes

PuTTY will select a word at a time in the terminal window if you double-click to begin the drag. This section allows you to control precisely what is considered to be a word.

Each character is given a *class*, which is a small number (typically 0, 1 or 2). PuTTY considers a single word to be any number of adjacent characters in the same class. So by modifying the assignment of characters to classes, you can modify the word-by-word selection behaviour.

In the default configuration, the character classes are:

- Class 0 contains white space and control characters.
- Class 1 contains most punctuation.
- Class 2 contains letters, numbers and a few pieces of punctuation (the double quote, minus sign, period, forward slash and underscore).

So, for example, if you assign the @ symbol into character class 2, you will be able to select an e-mail address with just a double click.

In order to adjust these assignments, you start by selecting a group of characters in the list box. Then enter a class number in the edit box below, and press the ‘Set’ button.

This mechanism currently only covers ASCII characters, because it isn't feasible to expand the list to cover the whole of Unicode.

Character class definitions can be modified by control sequences sent by the server. This configuration option controls the *default* state, which will be restored when you reset the terminal (see [section 3.1.3.6](#)). However, if you modify this option in mid-session using ‘Change Settings’, it will take effect immediately.

4.12.2 Copying in Rich Text Format

If you enable ‘Copy to clipboard in RTF as well as plain text’, PuTTY will write formatting information to the clipboard as well as the actual text you copy. The effect of this is that if you paste into (say) a word processor, the text will appear in the word processor in the same font, colour, and style (e.g. bold, underline) PuTTY was using to display it.

This option can easily be inconvenient, so by default it is disabled.

4.13 The Colours panel

The Colours panel allows you to control PuTTY's use of colour.

- [4.13.1 'Allow terminal to specify ANSI colours'](#)
- [4.13.2 'Allow terminal to use xterm 256-colour mode'](#)
- [4.13.3 'Allow terminal to use 24-bit colour'](#)
- [4.13.4 'Indicate bolded text by changing...'](#)
- [4.13.5 'Attempt to use logical palettes'](#)
- [4.13.6 'Use system colours'](#)
- [4.13.7 Adjusting the colours in the terminal window](#)

4.13.1 ‘Allow terminal to specify ANSI colours’

This option is enabled by default. If it is disabled, PuTTY will ignore any control sequences sent by the server to request coloured text.

If you have a particularly garish application, you might want to turn this option off and make PuTTY only use the default foreground and background colours.

4.13.2 ‘Allow terminal to use xterm 256-colour mode’

This option is enabled by default. If it is disabled, PuTTY will ignore any control sequences sent by the server which use the extended 256-colour mode supported by recent versions of xterm.

If you have an application which is supposed to use 256-colour mode and it isn't working, you may find you need to tell your server that your terminal supports 256 colours. On Unix, you do this by ensuring that the setting of TERM describes a 256-colour-capable terminal. You can check this using a command such as `infocmp`:
`$ infocmp | grep colors` **colors#256**, cols#80, it#8, lines#24, pairs#256,

If you do not see ‘colors#256’ in the output, you may need to change your terminal setting. On modern Linux machines, you could try ‘xterm-256color’.

4.13.3 ‘Allow terminal to use 24-bit colour’

This option is enabled by default. If it is disabled, PuTTY will ignore any control sequences sent by the server which use the control sequences supported by modern terminals to specify arbitrary 24-bit RGB colour value.

4.13.4 ‘Indicate bolded text by changing...’

When the server sends a control sequence indicating that some text should be displayed in bold, PuTTY can handle this in several ways. It can either change the font for a bold version, or use the same font in a brighter colour, or it can do both (brighten the colour *and* embolden the font). This control lets you choose which.

By default bold is indicated by colour, so non-bold text is displayed in light grey and bold text is displayed in bright white (and similarly in other colours). If you change the setting to ‘The font’ box, bold and non-bold text will be displayed in the same colour, and instead the font will change to indicate the difference. If you select ‘Both’, the font and the colour will both change.

Some applications rely on ‘bold black’ being distinguishable from a black background; if you choose ‘The font’, their text may become invisible.

4.13.5 ‘Attempt to use logical palettes’

Logical palettes are a mechanism by which a Windows application running on an 8-bit colour display can select precisely the colours it wants instead of going with the Windows standard defaults.

If you are not getting the colours you ask for on an 8-bit display, you can try enabling this option. However, be warned that it's never worked very well.

4.13.6 ‘Use system colours’

Enabling this option will cause PuTTY to ignore the configured colours for ‘Default Background/Foreground’ and ‘Cursor Colour/Text’ (see [section 4.13.7](#)), instead going with the system-wide defaults.

Note that non-bold and bold text will be the same colour if this option is enabled. You might want to change to indicating bold text by font changes (see [section 4.13.4](#)).

4.13.7 Adjusting the colours in the terminal window

The main colour control allows you to specify exactly what colours things should be displayed in. To modify one of the PuTTY colours, use the list box to select which colour you want to modify. The RGB values for that colour will appear on the right-hand side of the list box. Now, if you press the ‘Modify’ button, you will be presented with a colour selector, in which you can choose a new colour to go in place of the old one. (You may also edit the RGB values directly in the edit boxes, if you wish; each value is an integer from 0 to 255.)

PuTTY allows you to set the cursor colour, the default foreground and background, and the precise shades of all the ANSI configurable colours (black, red, green, yellow, blue, magenta, cyan, and white). You can also modify the precise shades used for the bold versions of these colours; these are used to display bold text if you have chosen to indicate that by colour (see [section 4.13.4](#)), and can also be used if the server asks specifically to use them. (Note that ‘Default Bold Background’ is *not* the background colour used for bold text; it is only used if the server specifically asks for a bold background.)

4.14 The Connection panel

The Connection panel allows you to configure options that apply to more than one type of connection.

- [4.14.1 Using keepalives to prevent disconnection](#)
- [4.14.2 'Disable Nagle's algorithm'](#)
- [4.14.3 'Enable TCP keepalives'](#)
- [4.14.4 'Internet protocol version'](#)
- [4.14.5 'Logical name of remote host'](#)

4.14.1 Using keepalives to prevent disconnection

If you find your sessions are closing unexpectedly (most often with 'Connection reset by peer') after they have been idle for a while, you might want to try using this option.

Some network routers and firewalls need to keep track of all connections through them. Usually, these firewalls will assume a connection is dead if no data is transferred in either direction after a certain time interval. This can cause PuTTY sessions to be unexpectedly closed by the firewall if no traffic is seen in the session for some time.

The keepalive option ('Seconds between keepalives') allows you to configure PuTTY to send data through the session at regular intervals, in a way that does not disrupt the actual terminal session. If you find your firewall is cutting idle connections off, you can try entering a non-zero value in this field. The value is measured in seconds; so, for example, if your firewall cuts connections off after ten minutes then you might want to enter 300 seconds (5 minutes) in the box.

Note that keepalives are not always helpful. They help if you have a firewall which drops your connection after an idle period; but if the network between you and the server suffers from breaks in connectivity then keepalives can actually make things worse. If a session is idle, and connectivity is temporarily lost between the endpoints, but the connectivity is restored before either side tries to send anything, then there will be no problem - neither endpoint will notice that anything was wrong. However, if one side does send something during the break, it will repeatedly try to re-send, and eventually give up and abandon the connection. Then when connectivity is restored, the other side will find that the first side doesn't believe there is an open connection any more. Keepalives can make this sort of problem worse, because they increase the

probability that PuTTY will attempt to send data during a break in connectivity. (Other types of periodic network activity can cause this behaviour; in particular, SSH-2 re-keys can have this effect. See [section 4.18.2](#).)

Therefore, you might find that keepalives help connection loss, or you might find they make it worse, depending on what *kind* of network problems you have between you and the server.

Keepalives are only supported in Telnet and SSH; the Rlogin, SUPDUP, and Raw protocols offer no way of implementing them. (For an alternative, see [section 4.14.3](#).)

Note that if you are using SSH-1 and the server has a bug that makes it unable to deal with SSH-1 ignore messages (see [section 4.27.14](#)), enabling keepalives will have no effect.

4.14.2 ‘Disable Nagle’s algorithm’

Nagle's algorithm is a detail of TCP/IP implementations that tries to minimise the number of small data packets sent down a network connection. With Nagle's algorithm enabled, PuTTY's bandwidth usage will be slightly more efficient; with it disabled, you may find you get a faster response to your keystrokes when connecting to some types of server.

The Nagle algorithm is disabled by default for interactive connections.

4.14.3 ‘Enable TCP keepalives’

NOTE: TCP keepalives should not be confused with the application-level keepalives described in [section 4.14.1](#). If in doubt, you probably want application-level keepalives; TCP keepalives are provided for completeness.

The idea of TCP keepalives is similar to application-level keepalives, and the same caveats apply. The main differences are:

- TCP keepalives are available on *all* network connection types, including Raw, Rlogin, and SUPDUP.
- The interval between TCP keepalives is usually much longer, typically two hours; this is set by the operating system, and cannot be configured within PuTTY.
- If the operating system does not receive a response to a keepalive, it may send out more in quick succession and terminate the connection if no response is received.

TCP keepalives may be more useful for ensuring that half-open connections are terminated than for keeping a connection alive.

TCP keepalives are disabled by default.

4.14.4 ‘Internet protocol version’

This option allows the user to select between the old and new Internet protocols and addressing schemes (IPv4 and IPv6). The selected protocol will be used for most outgoing network connections (including connections to proxies); however, tunnels have their own configuration, for which see [section 4.26.2](#).

The default setting is ‘Auto’, which means PuTTY will do something sensible and try to guess which protocol you wanted. (If you specify a literal Internet address, it will use whichever protocol that address implies. If you provide a hostname, it will see what kinds of address exist for that hostname; it will use IPv6 if there is an IPv6 address available, and fall back to IPv4 if not.) If you need to force PuTTY to use a particular protocol, you can explicitly set this to ‘IPv4’ or ‘IPv6’.

4.14.5 ‘Logical name of remote host’

This allows you to tell PuTTY that the host it will really end up connecting to is different from where it thinks it is making a network connection.

You might use this, for instance, if you had set up an SSH port forwarding in one PuTTY session so that connections to some arbitrary port (say, `localhost` port 10022) were forwarded to a second machine's SSH port (say, `foovax` port 22), and then started a second PuTTY connecting to the forwarded port.

In normal usage, the second PuTTY will access the host key cache under the host name and port it actually connected to (i.e. `localhost` port 10022 in this example). Using the logical host name option, however, you can configure the second PuTTY to cache the host key under the name of the host *you* know that it's *really* going to end up talking to (here `foovax`).

This can be useful if you expect to connect to the same actual server through many different channels (perhaps because your port forwarding arrangements keep changing): by consistently setting the logical host name, you can arrange that PuTTY will not keep asking you to reconfirm its host key. Conversely, if you expect to use the same local port number for port forwardings to lots of different servers, you probably didn't want any particular server's host key cached under that local port number. (For this latter case, you could instead explicitly configure host keys in the relevant sessions; see [section 4.19.3](#).)

If you just enter a host name for this option, PuTTY will cache the SSH host key under the default SSH port for that host, irrespective of the port you really connected to (since the typical scenario is like the above example: you connect to a silly real port number and your connection ends up forwarded to the normal port-22 SSH server of some other machine). To override this, you can append a port

number to the logical host name, separated by a colon. E.g. entering ‘foovax:2200’ as the logical host name will cause the host key to be cached as if you had connected to port 2200 of foovax.

If you provide a host name using this option, it is also displayed in other locations which contain the remote host name, such as the default window title and the default SSH password prompt. This reflects the fact that this is the host you're *really* connecting to, which is more important than the mere means you happen to be using to contact that host. (This applies even if you're using a protocol other than SSH.)

4.15 The Data panel

The Data panel allows you to configure various pieces of data which can be sent to the server to affect your connection at the far end.

Each option on this panel applies to more than one protocol. Options which apply to only one protocol appear on that protocol's configuration panels.

- [4.15.1 'Auto-login username'](#)
- [4.15.2 Use of system username](#)
- [4.15.3 'Terminal-type string'](#)
- [4.15.4 'Terminal speeds'](#)
- [4.15.5 Setting environment variables on the server](#)

4.15.1 ‘Auto-login username’

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. (Some Telnet servers don't support this.)

In this box you can type that user name.

4.15.2 Use of system username

When the previous box ([section 4.15.1](#)) is left blank, by default, PuTTY will prompt for a username at the time you make a connection.

In some environments, such as the networks of large organisations implementing single sign-on, a more sensible default may be to use the name of the user logged in to the local operating system (if any); this is particularly likely to be useful with GSSAPI key exchange and user authentication (see [section 4.23](#) and [section 4.18.1.1](#)). This control allows you to change the default behaviour.

The current system username is displayed in the dialog as a convenience. It is not saved in the configuration; if a saved session is later used by a different user, that user's name will be used.

4.15.3 ‘Terminal-type string’

Most servers you might connect to with PuTTY are designed to be connected to from lots of different types of terminal. In order to send the right control sequences to each one, the server will need to know what type of terminal it is dealing with. Therefore, each of the SSH, Telnet, and Rlogin protocols allow a text string to be sent down the connection describing the terminal. On a Unix server, this selects an entry from the `termcap` or `terminfo` database that tells applications what control sequences to send to the terminal, and what character sequences to expect the keyboard to generate.

PuTTY attempts to emulate the Unix `xterm` program, and by default it reflects this by sending `xterm` as a terminal-type string. If you find this is not doing what you want - perhaps the remote system reports ‘Unknown terminal type’ - you could try setting this to something different, such as `vt220`.

If you're not sure whether a problem is due to the terminal type setting or not, you probably need to consult the manual for your application or your server.

4.15.4 ‘Terminal speeds’

The Telnet, Rlogin, and SSH protocols allow the client to specify terminal speeds to the server.

This parameter does *not* affect the actual speed of the connection, which is always ‘as fast as possible’; it is just a hint that is sometimes used by server software to modify its behaviour. For instance, if a slow speed is indicated, the server may switch to a less bandwidth-hungry display mode.

The value is usually meaningless in a network environment, but PuTTY lets you configure it, in case you find the server is reacting badly to the default value.

The format is a pair of numbers separated by a comma, for instance, 38400, 38400. The first number represents the output speed (*from* the server) in bits per second, and the second is the input speed (*to* the server). (Only the first is used in the Rlogin protocol.)

This option has no effect on Raw connections.

4.15.5 Setting environment variables on the server

The Telnet protocol provides a means for the client to pass environment variables to the server. Many Telnet servers have stopped supporting this feature due to security flaws, but PuTTY still supports it for the benefit of any servers which have found other ways around the security problems than just disabling the whole mechanism.

Version 2 of the SSH protocol also provides a similar mechanism, which is easier to implement without security flaws. Newer SSH-2 servers are more likely to support it than older ones.

This configuration data is not used in the SSH-1, rlogin or raw protocols.

To add an environment variable to the list transmitted down the connection, you enter the variable name in the ‘Variable’ box, enter its value in the ‘Value’ box, and press the ‘Add’ button. To remove one from the list, select it in the list box and press ‘Remove’.

4.16 The Proxy panel

The Proxy panel allows you to configure PuTTY to use various types of proxy in order to make its network connections. The settings in this panel affect the primary network connection forming your PuTTY session, and also any extra connections made as a result of SSH port forwarding (see [section 3.5](#)).

Note that unlike some software (such as web browsers), PuTTY does not attempt to automatically determine whether to use a proxy and (if so) which one to use for a given destination. If you need to use a proxy, it must always be explicitly configured.

- [4.16.1 Setting the proxy type](#)
- [4.16.2 Excluding parts of the network from proxying](#)
- [4.16.3 Name resolution when using a proxy](#)
- [4.16.4 Username and password](#)
- [4.16.5 Specifying the Telnet, SSH, or Local proxy command](#)
- [4.16.6 Controlling proxy logging](#)

4.16.1 Setting the proxy type

The ‘Proxy type’ drop-down allows you to configure what type of proxy you want PuTTY to use for its network connections. The default setting is ‘None’; in this mode no proxy is used for any connection.

- Selecting ‘HTTP CONNECT’ allows you to proxy your connections through a web server supporting the HTTP CONNECT command, as documented in [RFC 2817](#).
- Selecting ‘SOCKS 4’ or ‘SOCKS 5’ allows you to proxy your connections through a SOCKS server.
- Many firewalls implement a less formal type of proxy in which a user can make a Telnet or TCP connection directly to the firewall machine and enter a command such as `connect myhost.com 22` to connect through to an external host. Selecting ‘Telnet’ allows you to tell PuTTY to use this type of proxy, with the precise command specified as described in [section 4.16.5](#).
- There are several ways to use a SSH server as a proxy. All of these cause PuTTY to make a secondary SSH connection to the proxy host (sometimes called a ‘jump host’ in this context).

The ‘Proxy hostname’ field will be interpreted as the name of a PuTTY saved session if one exists, or a hostname if not. This allows multi-hop jump paths, if the referenced saved session is itself configured to use an SSH proxy; and it allows combining SSH and non-SSH proxying.

- ‘SSH to proxy and use port forwarding’ causes PuTTY to use the secondary SSH connection to open a port-forwarding channel to the final destination host (similar to OpenSSH’s `-J` option).

- ‘SSH to proxy and execute a command’ causes PuTTY to run an arbitrary remote command on the proxy SSH server and use that command’s standard input and output streams to run the primary connection over. The remote command line is specified as described in [section 4.16.5](#).
 - ‘SSH to proxy and invoke a subsystem’ is similar but causes PuTTY to start an SSH ‘subsystem’ rather than an ordinary command line. This might be useful with a specially set up SSH proxy server.
- Selecting ‘Local’ allows you to specify an arbitrary command on the local machine to act as a proxy. When the session is started, instead of creating a TCP connection, PuTTY runs the command (specified in [section 4.16.5](#)), and uses its standard input and output streams.

This could be used, for instance, to talk to some kind of network proxy that PuTTY does not natively support; or you could tunnel a connection over something other than TCP/IP entirely.

You can also enable this mode on the command line; see [section 3.11.3.27](#).

4.16.2 Excluding parts of the network from proxying

Typically you will only need to use a proxy to connect to non-local parts of your network; for example, your proxy might be required for connections outside your company's internal network. In the 'Exclude Hosts/IPs' box you can enter ranges of IP addresses, or ranges of DNS names, for which PuTTY will avoid using the proxy and make a direct connection instead.

The 'Exclude Hosts/IPs' box may contain more than one exclusion range, separated by commas. Each range can be an IP address or a DNS name, with a * character allowing wildcards. For example:

*.example.com

This excludes any host with a name ending in .example.com from proxying.

192.168.88.*

This excludes any host with an IP address starting with 192.168.88 from proxying.

192.168.88.* , *.example.com

This excludes both of the above ranges at once.

Connections to the local host (the host name `localhost`, and any loopback IP address) are never proxied, even if the proxy exclude list does not explicitly contain them. It is very unlikely that this behaviour would ever cause problems, but if it does you can change it by enabling 'Consider proxying local host connections'.

Note that if you are doing DNS at the proxy (see [section 4.16.3](#)), you should make sure that your proxy exclusion settings do not depend on knowing the IP address of a host. If the name is passed on to the

proxy without PuTTY looking it up, it will never know the IP address and cannot check it against your list.

4.16.3 Name resolution when using a proxy

If you are using a proxy to access a private network, it can make a difference whether DNS name resolution is performed by PuTTY itself (on the client machine) or performed by the proxy.

The ‘Do DNS name lookup at proxy end’ configuration option allows you to control this. If you set it to ‘No’, PuTTY will always do its own DNS, and will always pass an IP address to the proxy. If you set it to ‘Yes’, PuTTY will always pass host names straight to the proxy without trying to look them up first.

If you set this option to ‘Auto’ (the default), PuTTY will do something it considers appropriate for each type of proxy. Most types of proxy (HTTP, SOCK5, SSH, Telnet, and local) will have host names passed straight to them; SOCKS4 proxies will not.

Note that if you are doing DNS at the proxy, you should make sure that your proxy exclusion settings (see [section 4.16.2](#)) do not depend on knowing the IP address of a host. If the name is passed on to the proxy without PuTTY looking it up, it will never know the IP address and cannot check it against your list.

The original SOCKS 4 protocol does not support proxy-side DNS. There is a protocol extension (SOCKS 4A) which does support it, but not all SOCKS 4 servers provide this extension. If you enable proxy DNS and your SOCKS 4 server cannot deal with it, this might be why.

If you want to avoid PuTTY making *any* DNS query related to your destination host name (for example, because your local DNS resolver is very slow to return a negative response in that situation), then as well as setting this control to ‘Yes’, you may also need to turn off GSSAPI authentication and GSSAPI key exchange in SSH (see [Section 4.23](#) and [section 4.18.1.1](#) respectively). This is because GSSAPI setup also involves a DNS query for the destination host

name, and that query is performed by the separate GSSAPI library, so PuTTY can't override or reconfigure it.

4.16.4 Username and password

You can enter a username and a password in the ‘Username’ and ‘Password’ boxes, which will be used if your proxy requires authentication.

Note that if you save your session, the proxy password will be saved in plain text, so anyone who can access your PuTTY configuration data will be able to discover it.

If PuTTY discovers that it needs a proxy username or password and you have not specified one here, PuTTY will prompt for it interactively in the terminal window.

Authentication is not fully supported for all forms of proxy:

- Username and password authentication is supported for HTTP proxies and SOCKS 5 proxies.
 - With SOCKS 5, authentication is via CHAP if the proxy supports it (this is not supported in PuTTYtel); otherwise the password is sent to the proxy in plain text.
 - With HTTP proxying, authentication is via ‘HTTP Digest’ if possible (again, not supported in PuTTYtel), or ‘HTTP Basic’. In the latter case, the password is sent to the proxy in plain text.
- SOCKS 4 can use the ‘Username’ field, but does not support passwords.
- SSH proxying can use all the same forms of SSH authentication supported by PuTTY for its main connection. If the SSH server requests password authentication, any configured proxy password will be used, but other authentication methods such as public keys and GSSAPI will be tried first, just as for a

primary SSH connection, and if they require credentials such as a key passphrase, PuTTY will interactively prompt for these.

- You can specify a way to include a username and password in the Telnet/Local proxy command (see [section 4.16.5](#)). If you do so, and don't also specify the actual username and/or password in the configuration, PuTTY will interactively prompt for them.

4.16.5 Specifying the Telnet, SSH, or Local proxy command

If you are using the Telnet proxy type, the usual command required by the firewall's Telnet server is connect, followed by a host name and a port number. If your proxy needs a different command, you can enter an alternative in the 'Command to send to proxy' box.

If you are using the Local proxy type, the local command to run is specified here.

If you are using the 'SSH to proxy and execute a command' type, the command to run on the SSH proxy server is specified here. Similarly, if you are using 'SSH to proxy and invoke a subsystem', the subsystem name is constructed as specified here.

In this string, you can use \n to represent a new-line, \r to represent a carriage return, \t to represent a tab character, and \x followed by two hex digits to represent any other character. \\ is used to encode the \ character itself.

Also, the special strings %host and %port will be replaced by the host name and port number you want to connect to. For Telnet and Local proxy types, the strings %user and %pass will be replaced by the proxy username and password (which, if not specified in the configuration, will be prompted for) – this does not happen with SSH proxy types (because the proxy username/password are used for SSH authentication). The strings %proxyhost and %proxyport will be replaced by the host details specified on the *Proxy* panel, if any (this is most likely to be useful for proxy types using a local or remote command). To get a literal % sign, enter %%.

If a Telnet proxy server prompts for a username and password before commands can be sent, you can use a command such as:

```
%user\n%pass\nconnect %host %port\n
```

This will send your username and password as the first two lines to the proxy, followed by a command to connect to the desired host and port. Note that if you do not include the %user or %pass tokens in the Telnet command, then anything specified in 'Username' and 'Password' configuration fields will be ignored.

4.16.6 Controlling proxy logging

Often the proxy interaction has its own diagnostic output; this is particularly the case for local proxy commands.

The setting ‘Print proxy diagnostics in the terminal window’ lets you control how much of the proxy’s diagnostics are printed to the main terminal window, along with output from your main session.

By default (‘No’), proxy diagnostics are only sent to the Event Log; with ‘Yes’ they are also printed to the terminal, where they may get mixed up with your main session. ‘Only until session starts’ is a compromise; proxy messages will go to the terminal window until the main session is deemed to have started (in a protocol-dependent way), which is when they’re most likely to be interesting; any further proxy-related messages during the session will only go to the Event Log.

4.17 The SSH panel

The SSH panel allows you to configure options that only apply to SSH sessions.

- [4.17.1 Executing a specific command on the server](#)
- [4.17.2 'Don't start a shell or command at all'](#)
- [4.17.3 'Enable compression'](#)
- [4.17.4 'SSH protocol version'](#)
- [4.17.5 Sharing an SSH connection between PuTTY tools](#)

4.17.1 Executing a specific command on the server

In SSH, you don't have to run a general shell session on the server. Instead, you can choose to run a single specific command (such as a mail user agent, for example). If you want to do this, enter the command in the 'Remote command' box.

Note that most servers will close the session after executing the command.

4.17.2 ‘Don’t start a shell or command at all’

If you tick this box, PuTTY will not attempt to run a shell or command after connecting to the remote server. You might want to use this option if you are only using the SSH connection for port forwarding, and your user account on the server does not have the ability to run a shell.

This feature is only available in SSH protocol version 2 (since the version 1 protocol assumes you will always want to run a shell).

This feature can also be enabled using the `-N` command-line option; see [section 3.11.3.13](#).

If you use this feature in Plink, you will not be able to terminate the Plink process by any graceful means; the only way to kill it will be by pressing Control-C or sending a kill signal from another program.

4.17.3 ‘Enable compression’

This enables data compression in the SSH connection: data sent by the server is compressed before sending, and decompressed at the client end. Likewise, data sent by PuTTY to the server is compressed first and the server decompresses it at the other end. This can help make the most of a low-bandwidth connection.

4.17.4 ‘SSH protocol version’

This allows you to select whether to use SSH protocol version 2 or the older version 1.

You should normally leave this at the default of ‘2’. As well as having fewer features, the older SSH-1 protocol is no longer developed, has many known cryptographic weaknesses, and is generally not considered to be secure. PuTTY’s protocol 1 implementation is provided mainly for compatibility, and is no longer being enhanced.

If a server offers both versions, prefer ‘2’. If you have some server or piece of equipment that only talks SSH-1, select ‘1’ here, and do not treat the resulting connection as secure.

PuTTY will not automatically fall back to the other version of the protocol if the server turns out not to match your selection here; instead, it will put up an error message and abort the connection. This prevents an active attacker downgrading an intended SSH-2 connection to SSH-1.

4.17.5 Sharing an SSH connection between PuTTY tools

The controls in this box allow you to configure PuTTY to reuse an existing SSH connection, where possible.

The SSH-2 protocol permits you to run multiple data channels over the same SSH connection, so that you can log in just once (and do the expensive encryption setup just once) and then have more than one terminal window open.

Each instance of PuTTY can still run at most one terminal session, but using the controls in this box, you can configure PuTTY to check if another instance of itself has already connected to the target host, and if so, share that instance's SSH connection instead of starting a separate new one.

To enable this feature, just tick the box 'Share SSH connections if possible'. Then, whenever you start up a PuTTY session connecting to a particular host, it will try to reuse an existing SSH connection if one is available. For example, selecting 'Duplicate Session' from the system menu will launch another session on the same host, and if sharing is enabled then it will reuse the existing SSH connection.

When this mode is in use, the first PuTTY that connected to a given server becomes the 'upstream', which means that it is the one managing the real SSH connection. All subsequent PuTTYS which reuse the connection are referred to as 'downstreams': they do not connect to the real server at all, but instead connect to the upstream PuTTY via local inter-process communication methods.

For this system to be activated, *both* the upstream and downstream instances of PuTTY must have the sharing option enabled.

The upstream PuTTY can therefore not terminate until all its downstreams have closed. This is similar to the effect you get with

port forwarding or X11 forwarding, in which a PuTTY whose terminal session has already finished will still remain open so as to keep serving forwarded connections.

In case you need to configure this system in more detail, there are two additional checkboxes which allow you to specify whether a particular PuTTY can act as an upstream or a downstream or both. (These boxes only take effect if the main ‘Share SSH connections if possible’ box is also ticked.) By default both of these boxes are ticked, so that multiple PuTTYS started from the same configuration will designate one of themselves as the upstream and share a single connection; but if for some reason you need a particular PuTTY configuration *not* to be an upstream (e.g. because you definitely need it to close promptly) or not to be a downstream (e.g. because it needs to do its own authentication using a special private key) then you can untick one or the other of these boxes.

I have referred to ‘PuTTY’ throughout the above discussion, but all the other PuTTY tools which make SSH connections can use this mechanism too. For example, if PSCP or PSFTP loads a configuration with sharing enabled, then it can act as a downstream and use an existing SSH connection set up by an instance of GUI PuTTY. The one special case is that PSCP and PSFTP will *never* act as upstreams.

It is possible to test programmatically for the existence of a live upstream using Plink. See [section 7.2.3.4](#).

4.18 The Kex panel

The Kex panel (short for ‘key exchange’) allows you to configure options related to SSH-2 key exchange.

Key exchange occurs at the start of an SSH connection (and occasionally thereafter); it establishes a shared secret that is used as the basis for all of SSH's security features. It is therefore very important for the security of the connection that the key exchange is secure.

Key exchange is a cryptographically intensive process; if either the client or the server is a relatively slow machine, the slower methods may take several tens of seconds to complete.

If connection startup is too slow, or the connection hangs periodically, you may want to try changing these settings.

If you don't understand what any of this means, it's safe to leave these settings alone.

This entire panel is only relevant to SSH protocol version 2; none of these settings affect SSH-1 at all.

- [4.18.1 Key exchange algorithm selection](#)
 - [4.18.1.1 GSSAPI-based key exchange](#)
- [4.18.2 Repeat key exchange](#)

4.18.1 Key exchange algorithm selection

PuTTY supports a variety of SSH-2 key exchange methods, and allows you to choose which one you prefer to use; configuration is similar to cipher selection (see [section 4.20](#)).

PuTTY currently supports the following key exchange methods:

- ‘NTRU Prime / Curve25519 hybrid’: ‘Streamlined NTRU Prime’ is a lattice-based algorithm intended to resist quantum attacks. In this key exchange method, it is run in parallel with a conventional Curve25519-based method (one of those included in ‘ECDH’), in such a way that it should be no *less* secure than that commonly-used method, and hopefully also resistant to a new class of attacks.
- ‘ECDH’: elliptic curve Diffie-Hellman key exchange, with a variety of standard curves and hash algorithms.
- The original form of Diffie-Hellman key exchange, with a variety of well-known groups and hashes:
 - ‘Group 18’, a well-known 8192-bit group, used with the SHA-512 hash function.
 - ‘Group 17’, a well-known 6144-bit group, used with the SHA-512 hash function.
 - ‘Group 16’, a well-known 4096-bit group, used with the SHA-512 hash function.
 - ‘Group 15’, a well-known 3072-bit group, used with the SHA-512 hash function.
 - ‘Group 14’: a well-known 2048-bit group, used with the SHA-256 hash function or, if the server doesn’t support that, SHA-1.

- ‘Group 1’: a well-known 1024-bit group, used with the SHA-1 hash function. Neither we nor current SSH standards recommend using this method any longer, and it’s not used by default in new installations; however, it may be the only method supported by very old server software.
- ‘Diffie-Hellman group exchange’: with this method, instead of using a fixed group, PuTTY requests that the server suggest a group to use for a subsequent Diffie-Hellman key exchange; the server can avoid groups known to be weak, and possibly invent new ones over time, without any changes required to PuTTY’s configuration. This key exchange method uses the SHA-256 hash or, if the server doesn’t support that, SHA-1.
- ‘RSA-based key exchange’: this requires much less computational effort on the part of the client, and somewhat less on the part of the server, than Diffie-Hellman key exchange.
- ‘GSSAPI key exchange’: see [section 4.18.1.1](#).

If the first algorithm PuTTY finds is below the ‘warn below here’ line, you will see a warning box when you make the connection, similar to that for cipher selection (see [section 4.20](#)).

- [4.18.1.1 GSSAPI-based key exchange](#)

4.18.1.1 GSSAPI-based key exchange

PuTTY supports a set of key exchange methods that also incorporates GSSAPI-based authentication. They are enabled with the ‘Attempt GSSAPI key exchange’ checkbox (which also appears on the ‘GSSAPI’ panel).

PuTTY can only perform the GSSAPI-authenticated key exchange methods when using Kerberos V5, and not other GSSAPI mechanisms. If the user running PuTTY has current Kerberos V5 credentials, then PuTTY will select the GSSAPI key exchange methods in preference to any of the ordinary SSH key exchange methods configured in the preference list. There's a GSSAPI-based equivalent to most of the ordinary methods listed in [section 4.18.1](#); server support determines which one will be used. (PuTTY's preference order for GSSAPI-authenticated key exchange methods is fixed, not controlled by the preference list.)

The advantage of doing GSSAPI authentication as part of the SSH key exchange is apparent when you are using credential delegation (see [section 4.23.1](#)). The SSH key exchange can be repeated later in the session, and this allows your Kerberos V5 credentials (which are typically short-lived) to be automatically re-delegated to the server when they are refreshed on the client. (This feature is commonly referred to as ‘cascading credentials’.)

If your server doesn't support GSSAPI key exchange, it may still support GSSAPI in the SSH user authentication phase. This will still let you log in using your Kerberos credentials, but will only allow you to delegate the credentials that are active at the beginning of the session; they can't be refreshed automatically later, in a long-running session. See [section 4.23](#) for how to control GSSAPI user authentication in PuTTY.

Another effect of GSSAPI key exchange is that it replaces the usual SSH mechanism of permanent host keys described in [section 2.2](#).

So if you use this method, then you won't be asked any interactive questions about whether to accept the server's host key. Instead, the Kerberos exchange will verify the identity of the host you connect to, at the same time as verifying your identity to it.

4.18.2 Repeat key exchange

If the session key negotiated at connection startup is used too much or for too long, it may become feasible to mount attacks against the SSH connection. Therefore, the SSH-2 protocol specifies that a new key exchange should take place every so often; this can be initiated by either the client or the server.

While this renegotiation is taking place, no data can pass through the SSH connection, so it may appear to ‘freeze’. (The occurrence of repeat key exchange is noted in the Event Log; see [section 3.1.3.1](#).) Usually the same algorithm is used as at the start of the connection, with a similar overhead.

These options control how often PuTTY will initiate a repeat key exchange (‘rekey’). You can also force a key exchange at any time from the Special Commands menu (see [section 3.1.3.2](#)).

- ‘Max minutes before rekey’ specifies the amount of time that is allowed to elapse before a rekey is initiated. If this is set to zero, PuTTY will not rekey due to elapsed time. The SSH-2 protocol specification recommends a timeout of at most 60 minutes.

You might have a need to disable time-based rekeys completely for the same reasons that keepalives aren’t always helpful. If you anticipate suffering a network dropout of several hours in the middle of an SSH connection, but were not actually planning to send *data* down that connection during those hours, then an attempted rekey in the middle of the dropout will probably cause the connection to be abandoned, whereas if rekeys are disabled then the connection should in principle survive (in the absence of interfering firewalls). See [section 4.14.1](#) for more discussion of these issues; for these purposes, rekeys have much the same properties as keepalives. (Except that rekeys have cryptographic value in themselves, so you should bear that in mind when deciding whether to turn them off.) Note, however, that the SSH server can still initiate rekeys.

- ‘Minutes between GSSAPI checks’, if you’re using GSSAPI key exchange, specifies how often the GSSAPI credential cache is checked to see whether new tickets are available for delegation, or current ones are near expiration. If forwarding of GSSAPI credentials is enabled, PuTTY will try to rekey as necessary to keep the delegated credentials from expiring. Frequent checks are recommended; rekeying only happens when needed.
- ‘Max data before rekey’ specifies the amount of data (in bytes) that is permitted to flow in either direction before a rekey is initiated. If this is set to zero, PuTTY will not rekey due to transferred data. The SSH-2 protocol specification recommends a limit of at most 1 gigabyte.

As well as specifying a value in bytes, the following shorthand can be used:

- ‘1k’ specifies 1 kilobyte (1024 bytes).
- ‘1M’ specifies 1 megabyte (1024 kilobytes).
- ‘1G’ specifies 1 gigabyte (1024 megabytes).

Disabling data-based rekeys entirely is a bad idea. The integrity, and to a lesser extent, confidentiality of the SSH-2 protocol depend in part on rekeys occurring before a 32-bit packet sequence number wraps around. Unlike time-based rekeys, data-based rekeys won’t occur when the SSH connection is idle, so they shouldn’t cause the same problems. The SSH-1 protocol, incidentally, has even weaker integrity protection than SSH-2 without rekeys.

4.19 The Host Keys panel

The Host Keys panel allows you to configure options related to host key management.

Host keys are used to prove the server's identity, and assure you that the server is not being spoofed (either by a man-in-the-middle attack or by completely replacing it on the network). See [section 2.2](#) for a basic introduction to host keys.

Much of this panel is only relevant to SSH protocol version 2; SSH-1 only supports one type of host key.

- [4.19.1 Host key type selection](#)
- [4.19.2 Preferring known host keys](#)
- [4.19.3 Manually configuring host keys](#)
- [4.19.4 Configuring PuTTY to accept host certificates](#)
 - [4.19.4.1 Expressions you can enter in 'Valid hosts'](#)
 - [4.19.4.2 RSA signature types in certificates](#)

4.19.1 Host key type selection

PuTTY supports a variety of SSH-2 host key types, and allows you to choose which one you prefer to use to identify the server. Configuration is similar to cipher selection (see [section 4.20](#)).

PuTTY currently supports the following host key types:

- ‘Ed25519’: Edwards-curve DSA using a twisted Edwards curve with modulus 2^{255-19} .
- ‘Ed448’: another Edwards-curve DSA type, using a larger elliptic curve with a 448-bit instead of 255-bit modulus (so it has a higher security level than Ed25519).
- ‘ECDSA’: elliptic curve DSA using one of the NIST-standardised elliptic curves.
- ‘DSA’: straightforward DSA using modular exponentiation.
- ‘RSA’: the ordinary RSA algorithm.

If PuTTY already has one or more host keys stored for the server, it will by default prefer to use one of those, even if the server has a key type that is higher in the preference order. You can add such a key to PuTTY’s cache from within an existing session using the ‘Special Commands’ menu; see [section 3.1.3.2](#).

Otherwise, PuTTY will choose a key type based purely on the preference order you specify in the configuration.

If the first key type PuTTY finds is below the ‘warn below here’ line, you will see a warning box when you make the connection, similar to that for cipher selection (see [section 4.20](#)).

4.19.2 Preferring known host keys

By default, PuTTY will adjust the preference order for SSH-2 host key algorithms so that any host keys it already knows are moved to the top of the list.

This prevents you from having to check and confirm a new host key for a server you already had one for (e.g. because the server has generated an alternative key of a type higher in PuTTY's preference order, or because you changed the preference order itself).

However, on the other hand, it can leak information to a listener in the network about *whether* you already know a host key for this server.

For this reason, this policy is configurable. By turning this checkbox off, you can reset PuTTY to always use the exact order of host key algorithms configured in the preference list described in [section 4.19.1](#), so that a listener will find out nothing about what keys you had stored.

4.19.3 Manually configuring host keys

In some situations, if PuTTY's automated host key management is not doing what you need, you might need to manually configure PuTTY to accept a specific host key, or one of a specific set of host keys.

One reason why you might want to do this is because the host name PuTTY is connecting to is using round-robin DNS to return one of multiple actual servers, and they all have different host keys. In that situation, you might need to configure PuTTY to accept any of a list of host keys for the possible servers, while still rejecting any key not in that list.

Another reason is if PuTTY's automated host key management is completely unavailable, e.g. because PuTTY (or Plink or PSFTP, etc) is running in a Windows environment without access to the Registry. In that situation, you will probably want to use the `-hostkey` command-line option to configure the expected host key(s); see [section 3.11.3.22](#).

For situations where PuTTY's automated host key management simply picks the wrong host name to store a key under, you may want to consider setting a 'logical host name' instead; see [section 4.14.5](#).

To configure manual host keys via the GUI, enter some text describing the host key into the edit box in the 'Manually configure host keys for this connection' container, and press the 'Add' button. The text will appear in the 'Host keys or fingerprints to accept' list box. You can remove keys again with the 'Remove' button.

The text describing a host key can be in one of the following formats:

- An SHA-256-based host key fingerprint of the form displayed in PuTTY's Event Log and host key dialog boxes, i.e. 'SHA256:' followed by 43 case-sensitive characters.

- An MD5-based host key fingerprint, i.e. sixteen 2-digit hex numbers separated by colons, optionally preceded by the prefix ‘MD5:’. (The case of the characters does not matter.)
- A base64-encoded blob describing an SSH-2 public key in OpenSSH's one-line public key format. How you acquire a public key in this format is server-dependent; on an OpenSSH server it can typically be found in a location like /etc/ssh/ssh_host_rsa_key.pub.

If this box contains at least one host key or fingerprint when PuTTY makes an SSH connection, then PuTTY's automated host key management is completely bypassed: the connection will be permitted if and only if the host key presented by the server is one of the keys listed in this box, and the host key store in the Registry will be neither read *nor written*, unless you explicitly do so.

If the box is empty (as it usually is), then PuTTY's automated host key management will work as normal.

4.19.4 Configuring PuTTY to accept host certificates

In some environments, the SSH host keys for a lot of servers will all be signed in turn by a central ‘certification authority’ (‘CA’ for short). This simplifies host key configuration for users, because if they configure their SSH client to accept host keys certified by that CA, then they don’t need to individually confirm each host key the first time they connect to that server.

In order to do this, press the ‘Configure host CAs’ button in the ‘Host keys’ configuration panel. This will launch a secondary configuration dialog box where you can configure what CAs PuTTY will accept signatures from.

Note that this configuration is common to all saved sessions. Everything in the main PuTTY configuration is specific to one saved session, and you can prepare a separate session with all the configuration different. But there’s only one copy of the host CA configuration, and it applies to all sessions PuTTY runs, whether saved or not.

(Otherwise, it would be useless – configuring a CA by hand for each new host wouldn’t be any more convenient than pressing the ‘confirm’ button for each new host’s host key.)

To set up a new CA using this config box:

First, load the CA’s public key from a file, or paste it directly into the ‘Public key of certification authority’ edit box. If your organisation signs its host keys in this way, they will publish the public key of their CA so that SSH users can include it in their configuration.

Next, in the ‘Valid hosts this key is trusted to certify’ box, configure at least one hostname wildcard to say what servers PuTTY should trust this CA to speak for. For example, suppose you work for Example

Corporation (`example.com`), and the Example Corporation IT department has advertised a CA that signs all the Example internal machines' host keys. Then probably you want to trust that CA to sign host keys for machines in the domain `example.com`, but not for anything else. So you might enter '`*.example.com`' into the 'Valid hosts' box.

It's important to limit what the CA key is allowed to sign. Don't just enter '*' in that box! If you do that, you're saying that Example Corporation IT department is authorised to sign a host key for *anything at all* you might decide to connect to – even if you're connecting out of the company network to a machine somewhere else, such as your own personal server. So that configuration would enable the Example IT department to act as a 'man-in-the-middle' between your PuTTY process and your server, and listen in to your communications – exactly the thing SSH is supposed to avoid.

So, if the CA was provided to you by the sysadmins responsible for `example.com` (or whatever), make sure PuTTY will *only* trust it for machines in the `example.com` domain.

For the full syntax of the 'Valid hosts' expression, see [section 4.19.4.1](#).

Finally, choose an identifying name for this CA; enter that name in the 'Name for this CA' edit box at the top of the window, and press 'Save' to record the CA in your configuration. The name you chose will appear in the list of saved CAs to the left of the 'Save' button.

The identifying name can be anything you like. It's there so that if you store multiple certificates you can tell which is which later when you want to edit or delete them. It also appears in the PuTTY Event Log when a server presents a certificate signed by that CA.

To reload an existing CA configuration, select it in the list box and press 'Load'. Then you can make changes, and save it again.

To remove a CA from your configuration completely, select it in the list and press ‘Delete’.

- [4.19.4.1 Expressions you can enter in ‘Valid hosts’](#)
- [4.19.4.2 RSA signature types in certificates](#)

4.19.4.1 Expressions you can enter in ‘Valid hosts’

The simplest thing you can enter in the ‘Valid hosts this key is trusted to certify’ edit box is just a hostname wildcard such as

‘*.example.com’. This matches any host in any subdomain, so both ‘ssh.example.com’ and ‘login.dept.example.com’ would match, but ‘prod.example.net’ would not.

But you can also enter multiple host name wildcards, and port number ranges, and make complicated Boolean expressions out of them using the operators ‘&&’ for ‘and’, ‘||’ for ‘or’, ‘!’ for ‘not’, and parentheses.

For example, here are some other things you could enter.

- ‘*.foo.example.com || *.bar.example.com’. This means the CA is trusted to sign the host key for a connection if the host name matches ‘*.foo.example.com’ or it matches ‘*.bar.example.com’. In other words, the CA has authority over those two particular subdomains of example.com, but not for anything else, like www.example.com.
- ‘*.example.com && ! *.extrasecure.example.com’. This means the CA is trusted to sign the host key for a connection if the host name matches ‘*.example.com’ *but does not* match ‘*.extrasecure.example.com’. (Imagine if there was one top-secret set of servers in your company that the main IT department didn’t have security clearance to administer.)
- ‘*.example.com && port:22’. This means the CA is trusted to sign the host key for a connection if the host name matches ‘*.example.com’ *and* the port number is 22. SSH servers running on other ports would not be covered.
- ‘(*.foo.example.com || *.bar.example.com) && port:0-1023’. This matches two subdomains of example.com, as before, but *also* restricts the port number to the range 0-1023.

A certificate configuration expression consists of one or more individual requirements which can each be a hostname wildcard, a single port number, or a port number range, combined together with these Boolean operators.

Unlike other languages such as C, there is no implied priority between ‘`&&`’ and ‘`||`’. If you write ‘`A && B || c`’ (where A, B and c are some particular requirements), then PuTTY will report a syntax error, because you haven’t said which of the ‘`&&`’ and ‘`||`’ takes priority tightly. You will have to write either ‘`(A && B) || c`’, meaning ‘both of A and B, or alternatively just c’, or ‘`A && (B || c)`’ (‘A, and also at least one of B and c’), to make it clear.

4.19.4.2 RSA signature types in certificates

RSA keys can be used to generate signatures with a choice of secure hash function. Typically, any version of OpenSSH new enough to support certificates at all will also be new enough to avoid using SHA-1, so the default settings of accepting the more modern SHA-256 and SHA-512 should be suitable for nearly all cases. For completeness, however, you can configure which types of RSA signature PuTTY will accept in a certificate from a CA using an RSA key.

4.20 The Cipher panel

PuTTY supports a variety of different encryption algorithms, and allows you to choose which one you prefer to use. You can do this by dragging the algorithms up and down in the list box (or moving them using the Up and Down buttons) to specify a preference order. When you make an SSH connection, PuTTY will search down the list from the top until it finds an algorithm supported by the server, and then use that.

PuTTY currently supports the following algorithms:

- ChaCha20-Poly1305, a combined cipher and MAC (SSH-2 only)
- AES (Rijndael) - 256, 192, or 128-bit SDCTR or CBC, or 256 or 128-bit GCM (SSH-2 only)
- Arcfour (RC4) - 256 or 128-bit stream cipher (SSH-2 only)
- Blowfish - 256-bit SDCTR (SSH-2 only) or 128-bit CBC
- Triple-DES - 168-bit SDCTR (SSH-2 only) or CBC
- Single-DES - 56-bit CBC (see below for SSH-2)

If the algorithm PuTTY finds is below the ‘warn below here’ line, you will see a warning box when you make the connection:

The first cipher supported by the server
is single-DES, which is below the configured
warning threshold.

Do you want to continue with this connection?

This warns you that the first available encryption is not a very secure one. Typically you would put the ‘warn below here’ line between the encryptions you consider secure and the ones you consider substandard. By default, PuTTY supplies a preference order intended to reflect a reasonable preference in terms of security and speed.

In SSH-2, the encryption algorithm is negotiated independently for each direction of the connection, although PuTTY does not support separate configuration of the preference orders. As a result you may

get two warnings similar to the one above, possibly with different encryptions.

Single-DES is not recommended in the SSH-2 protocol standards, but one or two server implementations do support it. PuTTY can use single-DES to interoperate with these servers if you enable the ‘Enable legacy use of single-DES in SSH-2’ option; by default this is disabled and PuTTY will stick to recommended ciphers.

4.21 The Auth panel

The Auth panel allows you to configure authentication options for SSH sessions.

- [4.21.1 'Display pre-authentication banner'](#)
- [4.21.2 'Bypass authentication entirely'](#)
- [4.21.3 'Disconnect if authentication succeeds trivially'](#)
- [4.21.4 'Attempt authentication using Pageant'](#)
- [4.21.5 'Attempt TIS or CryptoCard authentication'](#)
- [4.21.6 'Attempt keyboard-interactive authentication'](#)
- [4.21.7 'Allow agent forwarding'](#)
- [4.21.8 'Allow attempted changes of username in SSH-2'](#)

4.21.1 ‘Display pre-authentication banner’

SSH-2 servers can provide a message for clients to display to the prospective user before the user logs in; this is sometimes known as a pre-authentication ‘banner’. Typically this is used to provide information about the server and legal notices.

By default, PuTTY displays this message before prompting for a password or similar credentials (although, unfortunately, not before prompting for a login name, due to the nature of the protocol design). By unchecking this option, display of the banner can be suppressed entirely.

4.21.2 ‘Bypass authentication entirely’

In SSH-2, it is in principle possible to establish a connection without using SSH's mechanisms to identify or prove who you are to the server. An SSH server could prefer to handle authentication in the data channel, for instance, or simply require no user authentication whatsoever.

By default, PuTTY assumes the server requires authentication (we've never heard of one that doesn't), and thus must start this process with a username. If you find you are getting username prompts that you cannot answer, you could try enabling this option. However, most SSH servers will reject this.

This is not the option you want if you have a username and just want PuTTY to remember it; for that see [section 4.15.1](#). It's also probably not what if you're trying to set up passwordless login to a mainstream SSH server; depending on the server, you probably wanted public-key authentication ([chapter 8](#)) or perhaps GSSAPI authentication ([section 4.23](#)). (These are still forms of authentication, even if you don't have to interact with them.)

This option only affects SSH-2 connections. SSH-1 connections always require an authentication step.

4.21.3 ‘Disconnect if authentication succeeds trivially’

This option causes PuTTY to abandon an SSH session and disconnect from the server, if the server accepted authentication without ever having asked for any kind of password or signature or token.

This might be used as a security measure. There are some forms of attack against an SSH client user which work by terminating the SSH authentication stage early, and then doing something in the main part of the SSH session which *looks* like part of the authentication, but isn't really.

For example, instead of demanding a signature from your public key, for which PuTTY would ask for your key's passphrase, a compromised or malicious server might allow you to log in with no signature or password at all, and then print a message that *imitates* PuTTY's request for your passphrase, in the hope that you would type it in. (In fact, the passphrase for your public key should not be sent to any server.) PuTTY's main defence against attacks of this type is the ‘trust sigil’ system: messages in the PuTTY window that are truly originated by PuTTY itself are shown next to a small copy of the PuTTY icon, which the server cannot fake when it tries to imitate the same message using terminal output.

However, if you think you might be at risk of this kind of thing anyway (if you don't watch closely for the trust sigils, or if you think you're at extra risk of one of your servers being malicious), then you could enable this option as an extra defence. Then, if the server tries any of these attacks involving letting you through the authentication stage, PuTTY will disconnect from the server before it can send a follow-up fake prompt or other type of attack.

On the other hand, some servers *legitimately* let you through the SSH authentication phase trivially, either because they are genuinely

public, or because the important authentication step happens during the terminal session. (An example might be an SSH server that connects you directly to the terminal login prompt of a legacy mainframe.) So enabling this option might cause some kinds of session to stop working. It's up to you.

4.21.4 ‘Attempt authentication using Pageant’

If this option is enabled, then PuTTY will look for Pageant (the SSH private-key storage agent) and attempt to authenticate with any suitable public keys Pageant currently holds.

This behaviour is almost always desirable, and is therefore enabled by default. In rare cases you might need to turn it off in order to force authentication by some non-public-key method such as passwords.

This option can also be controlled using the `-noagent` command-line option. See [section 3.11.3.9](#).

See [chapter 9](#) for more information about Pageant in general.

4.21.5 ‘Attempt TIS or CryptoCard authentication’

TIS and CryptoCard authentication are (despite their names) generic forms of simple challenge/response authentication available in SSH protocol version 1 only. You might use them if you were using S/Key one-time passwords, for example, or if you had a physical security token that generated responses to authentication challenges. They can even be used to prompt for simple passwords.

With this switch enabled, PuTTY will attempt these forms of authentication if the server is willing to try them. You will be presented with a challenge string (which may be different every time) and must supply the correct response in order to log in. If your server supports this, you should talk to your system administrator about precisely what form these challenges and responses take.

4.21.6 ‘Attempt keyboard-interactive authentication’

The SSH-2 equivalent of TIS authentication is called ‘keyboard-interactive’. It is a flexible authentication method using an arbitrary sequence of requests and responses; so it is not only useful for challenge/response mechanisms such as S/Key, but it can also be used for (for example) asking the user for a new password when the old one has expired.

PuTTY leaves this option enabled by default, but supplies a switch to turn it off in case you should have trouble with it.

4.21.7 ‘Allow agent forwarding’

This option allows the SSH server to open forwarded connections back to your local copy of Pageant. If you are not running Pageant, this option will do nothing.

See [chapter 9](#) for general information on Pageant, and [section 9.4](#) for information on agent forwarding. Note that there is a security risk involved with enabling this option; see [section 9.6](#) for details.

4.21.8 ‘Allow attempted changes of username in SSH-2’

In the SSH-1 protocol, it is impossible to change username after failing to authenticate. So if you mis-type your username at the PuTTY ‘login as:’ prompt, you will not be able to change it except by restarting PuTTY.

The SSH-2 protocol *does* allow changes of username, in principle, but does not make it mandatory for SSH-2 servers to accept them. In particular, OpenSSH does not accept a change of username; once you have sent one username, it will reject attempts to try to authenticate as another user. (Depending on the version of OpenSSH, it may quietly return failure for all login attempts, or it may send an error message.) For this reason, PuTTY will by default not prompt you for your username more than once, in case the server complains. If you know your server can cope with it, you can enable the ‘Allow attempted changes of username’ option to modify PuTTY’s behaviour.

4.22 The Credentials panel

This subpane of the Auth panel contains configuration options that specify actual *credentials* to present to the server: key files and certificates.

- [4.22.1 'Private key file for authentication'](#)
- [4.22.2 'Certificate to use with the private key'](#)
- [4.22.3 'Plugin to provide authentication responses'](#)

4.22.1 ‘Private key file for authentication’

This box is where you enter the name of your private key file if you are using public key authentication. See [chapter 8](#) for information about public key authentication in SSH.

This key must be in PuTTY's native format (*.PPK). If you have a private key in another format that you want to use with PuTTY, see [section 8.2.15](#).

You can use the authentication agent Pageant so that you do not need to explicitly configure a key here; see [chapter 9](#).

If a private key file is specified here with Pageant running, PuTTY will first try asking Pageant to authenticate with that key, and ignore any other keys Pageant may have. If that fails, PuTTY will ask for a passphrase as normal. You can also specify a *public* key file in this case (in RFC 4716 or OpenSSH format), as that's sufficient to identify the key to Pageant, but of course if Pageant isn't present PuTTY can't fall back to using this file itself.

4.22.2 ‘Certificate to use with the private key’

(This is optional. If you don't know you need it, you can leave this blank.)

In some environments, user authentication keys can be signed in turn by a ‘certifying authority’ (‘CA’ for short), and user accounts on an SSH server can be configured to automatically trust any key that's certified by the right signature.

This can be a convenient setup if you have a very large number of servers. When you change your key pair, you might otherwise have to edit the `authorized_keys` file on every server individually, to make them all accept the new key. But if instead you configure all those servers once to accept keys signed as yours by a CA, then when you change your public key, all you have to do is to get the new key certified by the same CA as before, and then all your servers will automatically accept it without needing individual reconfiguration.

One way to use a certificate is to incorporate it into your private key file. [Section 8.2.9](#) explains how to do that using PuTTYgen. But another approach is to tell PuTTY itself where to find the public certificate file, and then it will automatically present that certificate when authenticating with the corresponding private key.

To do this, enter the pathname of the certificate file into the ‘Certificate to use with the private key’ file selector.

When this setting is configured, PuTTY will honour it no matter whether the private key is found in a file, or loaded into Pageant.

4.22.3 ‘Plugin to provide authentication responses’

An SSH server can use the ‘keyboard-interactive’ protocol to present a series of arbitrary questions and answers. Sometimes this is used for ordinary passwords, but sometimes the server will use the same mechanism for something more complicated, such as a one-time password system.

Some of these systems can be automated. For this purpose, PuTTY allows you to provide a separate program to act as a ‘plugin’ which will take over the authentication and send answers to the questions on your behalf.

If you have been provided with a plugin of this type, you can configure it here, by entering a full command line in the ‘Plugin command to run’ box.

(If you want to *write* a plugin of this type, see [appendix H](#) for the full specification of how the plugin is expected to behave.)

4.23 The GSSAPI panel

The ‘GSSAPI’ subpanel of the ‘Auth’ panel controls the use of GSSAPI authentication. This is a mechanism which delegates the authentication exchange to a library elsewhere on the client machine, which in principle can authenticate in many different ways but in practice is usually used with the Kerberos single sign-on protocol to implement passwordless login.

GSSAPI authentication is only available in the SSH-2 protocol.

PuTTY supports two forms of GSSAPI-based authentication. In one of them, the SSH key exchange happens in the normal way, and GSSAPI is only involved in authenticating the user. The checkbox labelled ‘Attempt GSSAPI authentication’ controls this form.

In the other method, GSSAPI-based authentication is combined with the SSH key exchange phase. If this succeeds, then the SSH authentication step has nothing left to do. See [section 4.18.1.1](#) for more information about this method. The checkbox labelled ‘Attempt GSSAPI key exchange’ controls this form. (The same checkbox appears on the ‘Kex’ panel.)

If one or both of these controls is enabled, then GSSAPI authentication will be attempted in one form or the other, and (typically) if your client machine has valid Kerberos credentials loaded, then PuTTY should be able to authenticate automatically to servers that support Kerberos logins.

If both of those checkboxes are disabled, PuTTY will not try any form of GSSAPI at all, and the rest of this panel will be unused.

- [4.23.1 ‘Allow GSSAPI credential delegation’](#)
- [4.23.2 Preference order for GSSAPI libraries](#)

4.23.1 ‘Allow GSSAPI credential delegation’

GSSAPI credential delegation is a mechanism for passing on your Kerberos (or other) identity to the session on the SSH server. If you enable this option, then not only will PuTTY be able to log in automatically to a server that accepts your Kerberos credentials, but also you will be able to connect out from that server to other Kerberos-supporting services and use the same credentials just as automatically.

(This option is the Kerberos analogue of SSH agent forwarding; see [section 9.4](#) for some information on that.)

Note that, like SSH agent forwarding, there is a security implication in the use of this option: the administrator of the server you connect to, or anyone else who has cracked the administrator account on that server, could fake your identity when connecting to further Kerberos-supporting services. However, Kerberos sites are typically run by a central authority, so the administrator of one server is likely to already have access to the other services too; so this would typically be less of a risk than SSH agent forwarding.

If your connection is not using GSSAPI key exchange, it is possible for the delegation to expire during your session. See [section 4.18.1.1](#) for more information.

4.23.2 Preference order for GSSAPI libraries

GSSAPI is a mechanism which allows more than one authentication method to be accessed through the same interface. Therefore, more than one authentication library may exist on your system which can be accessed using GSSAPI.

PuTTY contains native support for a few well-known such libraries (including Windows' SSPI), and will look for all of them on your system and use whichever it finds. If more than one exists on your system and you need to use a specific one, you can adjust the order in which it will search using this preference list control.

One of the options in the preference list is to use a user-specified GSSAPI library. If the library you want to use is not mentioned by name in PuTTY's list of options, you can enter its full pathname in the 'User-supplied GSSAPI library path' field, and move the 'User-supplied GSSAPI library' option in the preference list to make sure it is selected before anything else.

On Windows, such libraries are files with a `.dll` extension, and must have been built in the same way as the PuTTY executable you're running; if you have a 32-bit DLL, you must run a 32-bit version of PuTTY, and the same with 64-bit (see [question A.6.10](#)). On Unix, shared libraries generally have a `.so` extension.

4.24 The TTY panel

The TTY panel lets you configure the remote pseudo-terminal.

- [4.24.1 ‘Don’t allocate a pseudo-terminal’](#)
- [4.24.2 Sending terminal modes](#)

4.24.1 ‘Don’t allocate a pseudo-terminal’

When connecting to a Unix system, most interactive shell sessions are run in a *pseudo-terminal*, which allows the Unix system to pretend it’s talking to a real physical terminal device but allows the SSH server to catch all the data coming from that fake device and send it back to the client.

Occasionally you might find you have a need to run a session *not* in a pseudo-terminal. In PuTTY, this is generally only useful for very specialist purposes; although in Plink (see [chapter 7](#)) it is the usual way of working.

4.24.2 Sending terminal modes

The SSH protocol allows the client to send ‘terminal modes’ for the remote pseudo-terminal. These usually control the server’s expectation of the local terminal’s behaviour.

If your server does not have sensible defaults for these modes, you may find that changing them here helps, although the server is at liberty to ignore your changes. If you don’t understand any of this, it’s safe to leave these settings alone.

(None of these settings will have any effect if no pseudo-terminal is requested or allocated.)

You can change what happens for a particular mode by selecting it in the list, choosing one of the options and specifying the exact value if necessary, and hitting ‘Set’. The effect of the options is as follows:

- If the ‘Auto’ option is selected, the PuTTY tools will decide whether to specify that mode to the server, and if so, will send a sensible value.

PuTTY proper will send modes that it has an opinion on (currently only the code for the Backspace key, ERASE, and whether the character set is UTF-8, IUTF8). Plink on Unix will propagate appropriate modes from the local terminal, if any.

- If ‘Nothing’ is selected, no value for the mode will be specified to the server under any circumstances.
- If a value is specified, it will be sent to the server under all circumstances. The precise syntax of the value box depends on the mode.

By default, all of the available modes are listed as ‘Auto’, which should do the right thing in most circumstances.

The precise effect of each setting, if any, is up to the server. Their names come from POSIX and other Unix systems, and they are most likely to have a useful effect on such systems. (These are the same settings that can usually be changed using the `stty` command once logged in to such servers.)

Some notable modes are described below; for fuller explanations, see your server documentation.

- `ERASE` is the character that when typed by the user will delete one space to the left. When set to ‘Auto’ (the default setting), this follows the setting of the local Backspace key in PuTTY (see [section 4.4.1](#)).

This and other special characters are specified using `^c` notation for Ctrl-C, and so on. Use `^<27>` or `^<0x1B>` to specify a character numerically, and `^~` to get a literal `^`. Other non-control characters are denoted by themselves. Leaving the box entirely blank indicates that *no* character should be assigned to the specified function, although this may not be supported by all servers.

- `QUIT` is a special character that usually forcefully ends the current process on the server (`SIGQUIT`). On many servers its default setting is Ctrl-backslash (`^\\`), which is easy to accidentally invoke on many keyboards. If this is getting in your way, you may want to change it to another character or turn it off entirely.
- Boolean modes such as `ECHO` and `ICANON` can be specified in PuTTY in a variety of ways, such as `true/false`, `yes/no`, and `0/1`. (Explicitly specifying a value of `no` is different from not sending the mode at all.)
- The boolean mode `IUTF8` signals to the server whether the terminal character set is UTF-8 or not, for purposes such as basic line editing; if this is set incorrectly, the backspace key may erase the wrong amount of text, for instance. However, simply setting this is not usually sufficient for the server to use UTF-8; POSIX servers will generally also require the locale to

be set (by some server-dependent means), although many newer installations default to UTF-8. Also, since this mode was added to the SSH protocol much later than the others, many servers (particularly older servers) do not honour this mode sent over SSH; indeed, a few poorly-written servers object to its mere presence, so you may find you need to set it to not be sent at all. When set to ‘Auto’, this follows the local configured character set (see [section 4.10.1](#)).

- Terminal speeds are configured elsewhere; see [section 4.15.4](#).

4.25 The X11 panel

The X11 panel allows you to configure forwarding of X11 over an SSH connection.

If your server lets you run X Window System graphical applications, X11 forwarding allows you to securely give those applications access to a local X display on your PC.

To enable X11 forwarding, check the ‘Enable X11 forwarding’ box. If your X display is somewhere unusual, you will need to enter its location in the ‘X display location’ box; if this is left blank, PuTTY will try to find a sensible default in the environment, or use the primary local display (`:0`) if that fails.

See [section 3.4](#) for more information about X11 forwarding.

- [4.25.1 Remote X11 authentication](#)
- [4.25.2 X authority file for local display](#)

4.25.1 Remote X11 authentication

If you are using X11 forwarding, the virtual X server created on the SSH server machine will be protected by authorisation data. This data is invented, and checked, by PuTTY.

The usual authorisation method used for this is called `MIT-MAGIC-COOKIE-1`. This is a simple password-style protocol: the X client sends some cookie data to the server, and the server checks that it matches the real cookie. The cookie data is sent over an unencrypted X11 connection; so if you allow a client on a third machine to access the virtual X server, then the cookie will be sent in the clear.

PuTTY offers the alternative protocol `XDM-AUTHORIZATION-1`. This is a cryptographically authenticated protocol: the data sent by the X client is different every time, and it depends on the IP address and port of the client's end of the connection and is also stamped with the current time. So an eavesdropper who captures an `XDM-AUTHORIZATION-1` string cannot immediately re-use it for their own X connection.

PuTTY's support for `XDM-AUTHORIZATION-1` is a somewhat experimental feature, and may encounter several problems:

- Some X clients probably do not even support `XDM-AUTHORIZATION-1`, so they will not know what to do with the data PuTTY has provided.
- This authentication mechanism will only work in SSH-2. In SSH-1, the SSH server does not tell the client the source address of a forwarded connection in a machine-readable format, so it's impossible to verify the `XDM-AUTHORIZATION-1` data.
- You may find this feature causes problems with some SSH servers, which will not clean up `XDM-AUTHORIZATION-1` data after a session, so that if you then connect to the same server using a client which only does `MIT-MAGIC-COOKIE-1` and are allocated the

same remote display number, you might find that out-of-date authentication data is still present on your server and your X connections fail.

PuTTY's default is `MIT-MAGIC-COOKIE-1`. If you change it, you should be sure you know what you're doing.

4.25.2 X authority file for local display

If you are using X11 forwarding, the local X server to which your forwarded connections are eventually directed may itself require authorisation.

Some Windows X servers do not require this: they do authorisation by simpler means, such as accepting any connection from the local machine but not from anywhere else. However, if your X server does require authorisation, then PuTTY needs to know what authorisation is required.

One way in which this data might be made available is for the X server to store it somewhere in a file which has the same format as the Unix `.xauthority` file. If this is how your Windows X server works, then you can tell PuTTY where to find this file by configuring this option. By default, PuTTY will not attempt to find any authorisation for your local display.

4.26 The Tunnels panel

The Tunnels panel allows you to configure tunnelling of arbitrary connection types through an SSH connection.

Port forwarding allows you to tunnel other types of network connection down an SSH session. See [section 3.5](#) for a general discussion of port forwarding and how it works.

The port forwarding section in the Tunnels panel shows a list of all the port forwardings that PuTTY will try to set up when it connects to the server. By default no port forwardings are set up, so this list is empty.

To add a port forwarding:

- Set one of the ‘Local’ or ‘Remote’ radio buttons, depending on whether you want to forward a local port to a remote destination (‘Local’) or forward a remote port to a local destination (‘Remote’). Alternatively, select ‘Dynamic’ if you want PuTTY to provide a local SOCKS 4/4A/5 proxy on a local port (note that this proxy only supports TCP connections; the SSH protocol does not support forwarding UDP).
- Enter a source port number into the ‘Source port’ box. For local forwardings, PuTTY will listen on this port of your PC. For remote forwardings, your SSH server will listen on this port of the remote machine. Note that most servers will not allow you to listen on port numbers less than 1024.
- If you have selected ‘Local’ or ‘Remote’ (this step is not needed with ‘Dynamic’), enter a hostname and port number separated by a colon, in the ‘Destination’ box. Connections received on the source port will be directed to this destination. For example, to connect to a POP-3 server, you might enter `popserver.example.com:110`. (If you need to enter a literal IPv6 address, enclose it in square brackets, for instance `[::1]:2200`.)

- Click the ‘Add’ button. Your forwarding details should appear in the list box.

To remove a port forwarding, simply select its details in the list box, and click the ‘Remove’ button.

In the ‘Source port’ box, you can also optionally enter an IP address to listen on, by specifying (for instance) 127.0.0.5:79. See [section 3.5](#) for more information on how this works and its restrictions.

In place of port numbers, you can enter service names, if they are known to the local system. For instance, in the ‘Destination’ box, you could enter popserver.example.com:pop3.

You can modify the currently active set of port forwardings in mid-session using ‘Change Settings’ (see [section 3.1.3.4](#)). If you delete a local or dynamic port forwarding in mid-session, PuTTY will stop listening for connections on that port, so it can be re-used by another program. If you delete a remote port forwarding, note that:

- The SSH-1 protocol contains no mechanism for asking the server to stop listening on a remote port.
- The SSH-2 protocol does contain such a mechanism, but not all SSH servers support it. (In particular, OpenSSH does not support it in any version earlier than 3.9.)

If you ask to delete a remote port forwarding and PuTTY cannot make the server actually stop listening on the port, it will instead just start refusing incoming connections on that port. Therefore, although the port cannot be reused by another program, you can at least be reasonably sure that server-side programs can no longer access the service at your end of the port forwarding.

If you delete a forwarding, any existing connections established using that forwarding remain open. Similarly, changes to global settings such as ‘Local ports accept connections from other hosts’ only take effect on new forwardings.

If the connection you are forwarding over SSH is itself a second SSH connection made by another copy of PuTTY, you might find the 'logical host name' configuration option useful to warn PuTTY of which host key it should be expecting. See [section 4.14.5](#) for details of this.

- [4.26.1 Controlling the visibility of forwarded ports](#)
- [4.26.2 Selecting Internet protocol version for forwarded ports](#)

4.26.1 Controlling the visibility of forwarded ports

The source port for a forwarded connection usually does not accept connections from any machine except the SSH client or server machine itself (for local and remote forwardings respectively). There are controls in the Tunnels panel to change this:

- The ‘Local ports accept connections from other hosts’ option allows you to set up local-to-remote port forwardings in such a way that machines other than your client PC can connect to the forwarded port. (This also applies to dynamic SOCKS forwarding.)
- The ‘Remote ports do the same’ option does the same thing for remote-to-local port forwardings (so that machines other than the SSH server machine can connect to the forwarded port.) Note that this feature is only available in the SSH-2 protocol, and not all SSH-2 servers support it (OpenSSH 3.0 does not, for example).

4.26.2 Selecting Internet protocol version for forwarded ports

This switch allows you to select a specific Internet protocol (IPv4 or IPv6) for the local end of a forwarded port. By default, it is set on ‘Auto’, which means that:

- for a local-to-remote port forwarding, PuTTY will listen for incoming connections in both IPv4 and (if available) IPv6
- for a remote-to-local port forwarding, PuTTY will choose a sensible protocol for the outgoing connection.

This overrides the general Internet protocol version preference on the Connection panel (see [section 4.14.4](#)).

Note that some operating systems may listen for incoming connections in IPv4 even if you specifically asked for IPv6, because their IPv4 and IPv6 protocol stacks are linked together. Apparently Linux does this, and Windows does not. So if you're running PuTTY on Windows and you tick ‘IPv6’ for a local or dynamic port forwarding, it will *only* be usable by connecting to it using IPv6; whereas if you do the same on Linux, you can also use it with IPv4. However, ticking ‘Auto’ should always give you a port which you can connect to using either protocol.

4.27 The Bugs and More Bugs panels

Not all SSH servers work properly. Various existing servers have bugs in them, which can make it impossible for a client to talk to them unless it knows about the bug and works around it.

Since most servers announce their software version number at the beginning of the SSH connection, PuTTY will attempt to detect which bugs it can expect to see in the server and automatically enable workarounds. However, sometimes it will make mistakes; if the server has been deliberately configured to conceal its version number, or if the server is a version which PuTTY's bug database does not know about, then PuTTY will not know what bugs to expect.

The Bugs and More Bugs panels (there are two because we have so many bug compatibility modes) allow you to manually configure the bugs PuTTY expects to see in the server. Each bug can be configured in three states:

- ‘Off’: PuTTY will assume the server does not have the bug.
- ‘On’: PuTTY will assume the server *does* have the bug.
- ‘Auto’: PuTTY will use the server's version number announcement to try to guess whether or not the server has the bug. (This option is not available for bugs that *cannot* be detected from the server version, e.g. because they must be acted on before the server version is known.)

(The PuTTY project has a defined policy about when we're prepared to add auto-detection for a bug workaround. See [section B.6](#).)

- [4.27.1 ‘Chokes on SSH-2 ignore messages’](#)
- [4.27.2 ‘Handles SSH-2 key re-exchange badly’](#)
- [4.27.3 ‘Chokes on PuTTY's SSH-2 ‘winadj’ requests’](#)

- [4.27.4 'Replies to requests on closed channels'](#)
- [4.27.5 'Ignores SSH-2 maximum packet size'](#)
- [4.27.6 'Discards data sent before its greeting'](#)
- [4.27.7 'Chokes on PuTTY's full KEXINIT'](#)
- [4.27.8 'Old RSA/SHA2 cert algorithm naming'](#)
- [4.27.9 'Requires padding on SSH-2 RSA signatures'](#)
- [4.27.10 'Only supports pre-RFC4419 SSH-2 DH GEX'](#)
- [4.27.11 'Miscomputes SSH-2 HMAC keys'](#)
- [4.27.12 'Misuses the session ID in SSH-2 PK auth'](#)
- [4.27.13 'Miscomputes SSH-2 encryption keys'](#)
- [4.27.14 'Chokes on SSH-1 ignore messages'](#)
- [4.27.15 'Refuses all SSH-1 password camouflage'](#)
- [4.27.16 'Chokes on SSH-1 RSA authentication'](#)

4.27.1 ‘Chokes on SSH-2 ignore messages’

An ignore message (SSH_MSG_IGNORE) is a message in the SSH protocol which can be sent from the client to the server, or from the server to the client, at any time. Either side is required to ignore the message whenever it receives it. PuTTY uses ignore messages in SSH-2 to confuse the encrypted data stream and make it harder to cryptanalyse. It also uses ignore messages for connection keepalives (see [section 4.14.1](#)).

If it believes the server to have this bug, PuTTY will stop using ignore messages. If this bug is enabled when talking to a correct server, the session will succeed, but keepalives will not work and the session might be less cryptographically secure than it could be.

4.27.2 ‘Handles SSH-2 key re-exchange badly’

Some SSH servers cannot cope with repeat key exchange at all, and will ignore attempts by the client to start one. Since PuTTY pauses the session while performing a repeat key exchange, the effect of this would be to cause the session to hang after an hour (unless you have your rekey timeout set differently; see [section 4.18.2](#) for more about rekeys). Other, very old, SSH servers handle repeat key exchange even more badly, and disconnect upon receiving a repeat key exchange request.

If this bug is detected, PuTTY will never initiate a repeat key exchange. If this bug is enabled when talking to a correct server, the session should still function, but may be less secure than you would expect.

This is an SSH-2-specific bug.

4.27.3 ‘Chokes on PuTTY’s SSH-2 ‘winadj’ requests’

PuTTY sometimes sends a special request to SSH servers in the middle of channel data, with the name

`winadj@putty.projects.tartarus.org` (see [section G.1](#)). The purpose of this request is to measure the round-trip time to the server, which PuTTY uses to tune its flow control. The server does not actually have to *understand* the message; it is expected to send back a `SSH_MSG_CHANNEL_FAILURE` message indicating that it didn’t understand it. (All PuTTY needs for its timing calculations is *some* kind of response.) It has been known for some SSH servers to get confused by this message in one way or another – because it has a long name, or because they can’t cope with unrecognised request names even to the extent of sending back the correct failure response, or because they handle it sensibly but fill up the server’s log file with pointless spam, or whatever. PuTTY therefore supports this bug-compatibility flag: if it believes the server has this bug, it will never send its ‘`winadj@putty.projects.tartarus.org`’ request, and will make do without its timing data.

4.27.4 ‘Replies to requests on closed channels’

The SSH protocol as published in RFC 4254 has an ambiguity which arises if one side of a connection tries to close a channel, while the other side simultaneously sends a request within the channel and asks for a reply. RFC 4254 leaves it unclear whether the closing side should reply to the channel request after having announced its intention to close the channel.

Discussion on the `ietf-ssh` mailing list in April 2014 formed a clear consensus that the right answer is no. However, because of the ambiguity in the specification, some SSH servers have implemented the other policy; for example, [OpenSSH used to](#) until it was fixed.

Because PuTTY sends channel requests with the ‘want reply’ flag throughout channels’ lifetime (see [section 4.27.3](#)), it’s possible that when connecting to such a server it might receive a reply to a request after it thinks the channel has entirely closed, and terminate with an error along the lines of ‘Received `SSH2_MSG_CHANNEL_FAILURE` for nonexistent channel 256’.

4.27.5 ‘Ignores SSH-2 maximum packet size’

When an SSH-2 channel is set up, each end announces the maximum size of data packet that it is willing to receive for that channel. Some servers ignore PuTTY's announcement and send packets larger than PuTTY is willing to accept, causing it to report 'Incoming packet was garbled on decryption'.

If this bug is detected, PuTTY never allows the channel's flow-control window to grow large enough to allow the server to send an oversized packet. If this bug is enabled when talking to a correct server, the session will work correctly, but download performance will be less than it could be.

4.27.6 ‘Discards data sent before its greeting’

Just occasionally, an SSH connection can be established over some channel that will accidentally discard outgoing data very early in the connection.

This is not typically seen as a bug in an actual SSH server, but it can sometimes occur in situations involving a complicated proxy process. An example is [Debian bug #991958](#), in which a connection going over the console of a User Mode Linux kernel can lose outgoing data before the kernel has fully booted.

You can work around this problem by manually enabling this bug flag, which will cause PuTTY to wait to send its initial SSH greeting until after it sees the greeting from the server.

Note that this bug flag can never be automatically detected, since auto-detection relies on the version string in the server's greeting, and PuTTY has to decide whether to expect this bug *before* it sees the server's greeting. So this is a manual workaround only.

4.27.7 ‘Chokes on PuTTY’s full KEXINIT’

At the start of an SSH connection, the client and server exchange long messages of type `SSH_MSG_KEXINIT`, containing lists of all the cryptographic algorithms they’re prepared to use. This is used to negotiate a set of algorithms that both ends can speak.

Occasionally, a badly written server might have a length limit on the list it’s prepared to receive, and refuse to make a connection simply because PuTTY is giving it too many choices.

A workaround is to enable this flag, which will make PuTTY wait to send `KEXINIT` until after it receives the one from the server, and then filter its own `KEXINIT` to leave out any algorithm the server doesn’t also announce support for. This will generally make PuTTY’s `KEXINIT` at most the size of the server’s, and will otherwise make no difference to the algorithm negotiation.

This flag is a minor violation of the SSH protocol, because both sides are supposed to send `KEXINIT` proactively. It still works provided *one* side sends its `KEXINIT` without waiting, but if both client and server waited for the other one to speak first, the connection would deadlock. We don’t know of any servers that do this, but if there is one, then this flag will make PuTTY unable to speak to them at all.

4.27.8 ‘Old RSA/SHA2 cert algorithm naming’

If PuTTY is trying to do SSH-2 user authentication using an RSA key, and the server is using one of the newer SHA-2 based versions of the SSH RSA protocol, and the user's key is also a certificate, then earlier versions of OpenSSH (up to 7.7) disagree with later versions about the right key algorithm string to send in the `SSH2_MSG_USERAUTH_REQUEST` packet. Modern versions send a string that indicates both the SHA-2 nature and the certificate nature of the key, such as ‘`rsa-sha2-512-cert-v01@openssh.com`’. Earlier versions would reject that, and insist on seeing ‘`ssh-rsa-cert-v01@openssh.com`’ followed by a SHA-2 based signature.

PuTTY should auto-detect the presence of this bug in earlier OpenSSH and adjust to send the right string.

4.27.9 ‘Requires padding on SSH-2 RSA signatures’

Versions below 3.3 of OpenSSH require SSH-2 RSA signatures to be padded with zero bytes to the same length as the RSA key modulus. The SSH-2 specification says that an unpadded signature **MUST** be accepted, so this is a bug. A typical symptom of this problem is that PuTTY mysteriously fails RSA authentication once in every few hundred attempts, and falls back to passwords.

If this bug is detected, PuTTY will pad its signatures in the way OpenSSH expects. If this bug is enabled when talking to a correct server, it is likely that no damage will be done, since correct servers usually still accept padded signatures because they're used to talking to OpenSSH.

This is an SSH-2-specific bug.

4.27.10 ‘Only supports pre-RFC4419 SSH-2 DH GEX’

The SSH key exchange method that uses Diffie-Hellman group exchange was redesigned after its original release, to use a slightly more sophisticated setup message. Almost all SSH implementations switched over to the new version. (PuTTY was one of the last.) A few old servers still only support the old one.

If this bug is detected, and the client and server negotiate Diffie-Hellman group exchange, then PuTTY will send the old message now known as `SSH2_MSG_KEX_DH_GEX_REQUEST_OLD` in place of the new `SSH2_MSG_KEX_DH_GEX_REQUEST`.

This is an SSH-2-specific bug.

4.27.11 ‘Miscomputes SSH-2 HMAC keys’

Versions 2.3.0 and below of the SSH server software from ssh.com compute the keys for their HMAC message authentication codes incorrectly. A typical symptom of this problem is that PuTTY dies unexpectedly at the beginning of the session, saying ‘Incorrect MAC received on packet’.

If this bug is detected, PuTTY will compute its HMAC keys in the same way as the buggy server, so that communication will still be possible. If this bug is enabled when talking to a correct server, communication will fail.

This is an SSH-2-specific bug.

4.27.12 ‘Misuses the session ID in SSH-2 PK auth’

Versions below 2.3 of OpenSSH require SSH-2 public-key authentication to be done slightly differently: the data to be signed by the client contains the session ID formatted in a different way. If public-key authentication mysteriously does not work but the Event Log (see [section 3.1.3.1](#)) thinks it has successfully sent a signature, it might be worth enabling the workaround for this bug to see if it helps.

If this bug is detected, PuTTY will sign data in the way OpenSSH expects. If this bug is enabled when talking to a correct server, SSH-2 public-key authentication will fail.

This is an SSH-2-specific bug.

4.27.13 ‘Miscomputes SSH-2 encryption keys’

Versions below 2.0.11 of the SSH server software from ssh.com compute the keys for the session encryption incorrectly. This problem can cause various error messages, such as ‘Incoming packet was garbled on decryption’, or possibly even ‘Out of memory’.

If this bug is detected, PuTTY will compute its encryption keys in the same way as the buggy server, so that communication will still be possible. If this bug is enabled when talking to a correct server, communication will fail.

This is an SSH-2-specific bug.

4.27.14 ‘Chokes on SSH-1 ignore messages’

An ignore message (SSH_MSG_IGNORE) is a message in the SSH protocol which can be sent from the client to the server, or from the server to the client, at any time. Either side is required to ignore the message whenever it receives it. PuTTY uses ignore messages to hide the password packet in SSH-1, so that a listener cannot tell the length of the user's password; it also uses ignore messages for connection keepalives (see [section 4.14.1](#)).

If this bug is detected, PuTTY will stop using ignore messages. This means that keepalives will stop working, and PuTTY will have to fall back to a secondary defence against SSH-1 password-length eavesdropping. See [section 4.27.15](#). If this bug is enabled when talking to a correct server, the session will succeed, but keepalives will not work and the session might be more vulnerable to eavesdroppers than it could be.

4.27.15 ‘Refuses all SSH-1 password camouflage’

When talking to an SSH-1 server which cannot deal with ignore messages (see [section 4.27.14](#)), PuTTY will attempt to disguise the length of the user’s password by sending additional padding *within* the password packet. This is technically a violation of the SSH-1 specification, and so PuTTY will only do it when it cannot use standards-compliant ignore messages as camouflage. In this sense, for a server to refuse to accept a padded password packet is not really a bug, but it does make life inconvenient if the server can also not handle ignore messages.

If this ‘bug’ is detected, PuTTY will assume that neither ignore messages nor padding are acceptable, and that it thus has no choice but to send the user’s password with no form of camouflage, so that an eavesdropping user will be easily able to find out the exact length of the password. If this bug is enabled when talking to a correct server, the session will succeed, but will be more vulnerable to eavesdroppers than it could be.

This is an SSH-1-specific bug. SSH-2 is secure against this type of attack.

4.27.16 ‘Chokes on SSH-1 RSA authentication’

Some SSH-1 servers cannot deal with RSA authentication messages at all. If Pageant is running and contains any SSH-1 keys, PuTTY will normally automatically try RSA authentication before falling back to passwords, so these servers will crash when they see the RSA attempt.

If this bug is detected, PuTTY will go straight to password authentication. If this bug is enabled when talking to a correct server, the session will succeed, but of course RSA authentication will be impossible.

This is an SSH-1-specific bug.

4.28 The ‘Bare ssh-connection’ protocol

In addition to SSH itself, PuTTY also supports a second protocol that is derived from SSH. It's listed in the PuTTY GUI under the name 'Bare ssh-connection'.

This protocol consists of just the innermost of SSH-2's three layers: it leaves out the cryptography layer providing network security, and it leaves out the authentication layer where you provide a username and prove you're allowed to log in as that user.

It is therefore **completely unsuited to any network connection**. Don't try to use it over a network!

The purpose of this protocol is for various specialist circumstances in which the ‘connection’ is not over a real network, but is a pipe or IPC channel between different processes running on the *same* computer. In these contexts, the operating system will already have guaranteed that each of the two communicating processes is owned by the expected user (so that no authentication is necessary), and that the communications channel cannot be tapped by a hostile user on the same machine (so that no cryptography is necessary either).

Examples of possible uses involve communicating with a strongly separated context such as the inside of a container, or a VM, or a different network namespace.

Explicit support for this protocol is new in PuTTY 0.75. As of 2021-04, the only known server for the bare ssh-connection protocol is the Unix program ‘psusan’ that is also part of the PuTTY tool suite.

(However, this protocol is also the same one used between instances of PuTTY to implement connection sharing: see [section 4.17.5](#). In fact, in the Unix version of PuTTY, when a sharing upstream records ‘Sharing this connection at [pathname]’ in the Event Log, it's possible to connect another instance of PuTTY directly to that Unix socket, by entering its pathname in the host name box and selecting ‘Bare ssh-connection’ as the protocol!)

Many of the options under the SSH panel also affect this protocol, although options to do with cryptography and authentication do not, for obvious reasons.

I repeat, **DON'T TRY TO USE THIS PROTOCOL FOR NETWORK CONNECTIONS!** That's not what it's for, and it's not at all safe to do it.

4.29 The Serial panel

The Serial panel allows you to configure options that only apply when PuTTY is connecting to a local serial line.

- [4.29.1 Selecting a serial line to connect to](#)
- [4.29.2 Selecting the speed of your serial line](#)
- [4.29.3 Selecting the number of data bits](#)
- [4.29.4 Selecting the number of stop bits](#)
- [4.29.5 Selecting the serial parity checking scheme](#)
- [4.29.6 Selecting the serial flow control scheme](#)

4.29.1 Selecting a serial line to connect to

The ‘Serial line to connect to’ box allows you to choose which serial line you want PuTTY to talk to, if your computer has more than one serial port.

On Windows, the first serial line is called com1, and if there is a second it is called com2, and so on.

This configuration setting is also visible on the Session panel, where it replaces the ‘Host Name’ box (see [section 4.1.1](#)) if the connection type is set to ‘Serial’.

4.29.2 Selecting the speed of your serial line

The ‘Speed’ box allows you to choose the speed (or ‘baud rate’) at which to talk to the serial line. Typical values might be 9600, 19200, 38400 or 57600. Which one you need will depend on the device at the other end of the serial cable; consult the manual for that device if you are in doubt.

This configuration setting is also visible on the Session panel, where it replaces the ‘Port’ box (see [section 4.1.1](#)) if the connection type is set to ‘Serial’.

4.29.3 Selecting the number of data bits

The ‘Data bits’ box allows you to choose how many data bits are transmitted in each byte sent or received through the serial line. Typical values are 7 or 8.

4.29.4 Selecting the number of stop bits

The ‘Stop bits’ box allows you to choose how many stop bits are used in the serial line protocol. Typical values are 1, 1.5 or 2.

4.29.5 Selecting the serial parity checking scheme

The ‘Parity’ box allows you to choose what type of parity checking is used on the serial line. The settings are:

- ‘None’: no parity bit is sent at all.
- ‘Odd’: an extra parity bit is sent alongside each byte, and arranged so that the total number of 1 bits is odd.
- ‘Even’: an extra parity bit is sent alongside each byte, and arranged so that the total number of 1 bits is even.
- ‘Mark’: an extra parity bit is sent alongside each byte, and always set to 1.
- ‘Space’: an extra parity bit is sent alongside each byte, and always set to 0.

4.29.6 Selecting the serial flow control scheme

The ‘Flow control’ box allows you to choose what type of flow control checking is used on the serial line. The settings are:

- ‘None’: no flow control is done. Data may be lost if either side attempts to send faster than the serial line permits.
- ‘XON/XOFF’: flow control is done by sending XON and XOFF characters within the data stream.
- ‘RTS/CTS’: flow control is done using the RTS and CTS wires on the serial line.
- ‘DSR/DTR’: flow control is done using the DSR and DTR wires on the serial line.

4.30 The Telnet panel

The Telnet panel allows you to configure options that only apply to Telnet sessions.

- [4.30.1 'Handling of OLD_ENVIRON ambiguity'](#)
- [4.30.2 Passive and active Telnet negotiation modes](#)
- [4.30.3 'Keyboard sends Telnet special commands'](#)
- [4.30.4 'Return key sends Telnet New Line instead of ^M'](#)

4.30.1 ‘Handling of OLD_ENVIRON ambiguity’

The original Telnet mechanism for passing environment variables was badly specified. At the time the standard (RFC 1408) was written, BSD telnet implementations were already supporting the feature, and the intention of the standard was to describe the behaviour the BSD implementations were already using.

Sadly there was a typing error in the standard when it was issued, and two vital function codes were specified the wrong way round. BSD implementations did not change, and the standard was not corrected. Therefore, it's possible you might find either BSD or RFC-compliant implementations out there. This switch allows you to choose which one PuTTY claims to be.

The problem was solved by issuing a second standard, defining a new Telnet mechanism called NEW_ENVIRON, which behaved exactly like the original OLD_ENVIRON but was not encumbered by existing implementations. Most Telnet servers now support this, and it's unambiguous. This feature should only be needed if you have trouble passing environment variables to quite an old server.

4.30.2 Passive and active Telnet negotiation modes

In a Telnet connection, there are two types of data passed between the client and the server: actual text, and *negotiations* about which Telnet extra features to use.

PuTTY can use two different strategies for negotiation:

- In *active* mode, PuTTY starts to send negotiations as soon as the connection is opened.
- In *passive* mode, PuTTY will wait to negotiate until it sees a negotiation from the server.

The obvious disadvantage of passive mode is that if the server is also operating in a passive mode, then negotiation will never begin at all. For this reason PuTTY defaults to active mode.

However, sometimes passive mode is required in order to successfully get through certain types of firewall and Telnet proxy server. If you have confusing trouble with a firewall, you could try enabling passive mode to see if it helps.

4.30.3 ‘Keyboard sends Telnet special commands’

If this box is checked, several key sequences will have their normal actions modified:

- the Backspace key on the keyboard will send the Telnet special backspace code;
- Control-C will send the Telnet special Interrupt Process code;
- Control-Z will send the Telnet special Suspend Process code.

You probably shouldn't enable this unless you know what you're doing.

4.30.4 ‘Return key sends Telnet New Line instead of ^M’

Unlike most other remote login protocols, the Telnet protocol has a special ‘new line’ code that is not the same as the usual line endings of Control-M or Control-J. By default, PuTTY sends the Telnet New Line code when you press Return, instead of sending Control-M as it does in most other protocols.

Most Unix-style Telnet servers don’t mind whether they receive Telnet New Line or Control-M; some servers do expect New Line, and some servers prefer to see ^M. If you are seeing surprising behaviour when you press Return in a Telnet session, you might try turning this option off to see if it helps.

4.31 The Rlogin panel

The Rlogin panel allows you to configure options that only apply to Rlogin sessions.

- [4.31.1 'Local username'](#)

4.31.1 ‘Local username’

Rlogin allows an automated (password-free) form of login by means of a file called `.rhosts` on the server. You put a line in your `.rhosts` file saying something like `jbloggs@pc1.example.com`, and then when you make an Rlogin connection the client transmits the username of the user running the Rlogin client. The server checks the username and hostname against `.rhosts`, and if they match it does not ask for a password.

This only works because Unix systems contain a safeguard to stop a user from pretending to be another user in an Rlogin connection. Rlogin connections have to come from port numbers below 1024, and Unix systems prohibit this to unprivileged processes; so when the server sees a connection from a low-numbered port, it assumes the client end of the connection is held by a privileged (and therefore trusted) process, so it believes the claim of who the user is.

Windows does not have this restriction: *any* user can initiate an outgoing connection from a low-numbered port. Hence, the Rlogin `.rhosts` mechanism is completely useless for securely distinguishing several different users on a Windows machine. If you have a `.rhosts` entry pointing at a Windows PC, you should assume that *anyone* using that PC can spoof your username in an Rlogin connection and access your account on the server.

The ‘Local username’ control allows you to specify what user name PuTTY should claim you have, in case it doesn’t match your Windows user name (or in case you didn’t bother to set up a Windows user name).

4.32 The SUPDUP panel

The SUPDUP panel allows you to configure options that only apply to SUPDUP sessions. See [section 3.10](#) for more about the SUPDUP protocol.

- [4.32.1 ‘Location string’](#)
- [4.32.2 ‘Extended ASCII Character set’](#)
- [4.32.3 ‘**MORE** processing’](#)
- [4.32.4 ‘Terminal scrolling’](#)

4.32.1 ‘Location string’

In SUPDUP, the client sends a piece of text of its choice to the server giving the user's location. This is typically displayed in lists of logged-in users.

By default, PuTTY just defaults this to "The Internet". If you want your location to show up as something more specific, you can configure it here.

4.32.2 ‘Extended ASCII Character set’

This declares what kind of character set extension your terminal supports. If the server supports it, it will send text using that character set. ‘None’ means the standard 95 printable ASCII characters. ‘ITS’ means ASCII extended with printable characters in the control character range. This character set is documented in the SUPDUP protocol definition. ‘WAITS’ is similar to ‘ITS’ but uses some alternative characters in the extended set: most prominently, it will display arrows instead of ^ and _, and } instead of ~. ‘ITS’ extended ASCII is used by ITS and Lisp machines, whilst ‘WAITS’ is only used by the WAITS operating system from the Stanford AI Laboratory.

4.32.3 **MORE processing'**

When **MORE** processing is enabled, the server causes output to pause at the bottom of the screen, until a space is typed.

4.32.4 ‘Terminal scrolling’

This controls whether the terminal will perform scrolling when the cursor goes below the last line, or if the cursor will return to the first line.

4.33 Storing configuration in a file

PuTTY does not currently support storing its configuration in a file instead of the Registry. However, you can work around this with a couple of batch files.

You will need a file called (say) PUTTY.BAT which imports the contents of a file into the Registry, then runs PuTTY, exports the contents of the Registry back into the file, and deletes the Registry entries. This can all be done using the Regedit command line options, so it's all automatic. Here is what you need in PUTTY.BAT:

```
@ECHO OFF
regedit /s putty.reg
regedit /s puttyrnd.reg
start /w putty.exe
regedit /ea new.reg
HKEY_CURRENT_USER\Software\SimonTatham\PuTTY
copy new.reg putty.reg
del new.reg
regedit /s puttydel.reg
```

This batch file needs two auxiliary files: PUTTYRND.REG which sets up an initial safe location for the PUTTY.RND random seed file, and PUTTYDEL.REG which destroys everything in the Registry once it's been successfully saved back to the file.

Here is PUTTYDEL.REG:

```
REGEDIT4
[-HKEY_CURRENT_USER\Software\SimonTatham\PuTTY]
```

Here is an example PUTTYRND.REG file:

```
REGEDIT4
[HKEY_CURRENT_USER\Software\SimonTatham\PuTTY]
"RandSeedFile"="a:\\putty.rnd"
```

You should replace a:\putty.rnd with the location where you want to store your random number data. If the aim is to carry around PuTTY and its settings on one USB stick, you probably want to store it on the USB stick.

Chapter 5: Using PSCP to transfer files securely

PSCP, the PuTTY Secure Copy client, is a tool for transferring files securely between computers using an SSH connection.

If you have an SSH-2 server, you might prefer PSFTP (see [chapter 6](#)) for interactive use. PSFTP does not in general work with SSH-1 servers, however.

- [5.1 Starting PSCP](#)
- [5.2 PSCP Usage](#)
 - [5.2.1 The basics](#)
 - [5.2.2 Options](#)
 - [5.2.3 Return value](#)
 - [5.2.4 Using public key authentication with PSCP](#)

5.1 Starting PSCP

PSCP is a command line application. This means that you cannot just double-click on its icon to run it and instead you have to bring up a console window. With Windows 95, 98, and ME, this is called an ‘MS-DOS Prompt’ and with Windows NT, 2000, and XP, it is called a ‘Command Prompt’. It should be available from the Programs section of your Start Menu.

To start PSCP it will need either to be on your PATH or in your current directory. To add the directory containing PSCP to your PATH environment variable, type into the console window:

```
set PATH=C:\path\to\putty\directory;%PATH%
```

This will only work for the lifetime of that particular console window. To set your PATH more permanently on Windows NT, 2000, and XP, use the Environment tab of the System Control Panel. On Windows 95, 98, and ME, you will need to edit your AUTOEXEC.BAT to include a set command like the one above.

<code>C:\>pscp

PuTTY Secure Copy client

Release 0.81

Usage: pscp [options] [user@]host:source target pscp
[options] source [source...] [user@]host:target pscp
[options] -ls [user@]host:filespec Options:

-V print version information and exit -pgpf print
PGP key fingerprints and exit -p preserve file
attributes -q quiet, don't show statistics -r copy
directories recursively -v show verbose messages -
load sessname Load settings from saved session -P
port connect to specified port -l user connect with
specified username -pwfile file login with password
read from specified file -1 -2 force use of particular
SSH protocol version -ssh -ssh-connection force use
of particular SSH protocol variant

-4 -6 force use of IPv4 or IPv6

-C enable compression

-i key private key file for user authentication -
noagent disable use of Pageant -agent enable use of
Pageant -no-trivial-auth

disconnect if SSH authentication succeeds trivially
-hostkey keyid

manually specify a host key (may be repeated) -
batch disable all interactive prompts -no-sanitise-
stderr don't strip control chars from standard error -
proxycmd command

use 'command' as local proxy -unsafe allow server-
side wildcards (DANGEROUS) -sftp force use of
SFTP protocol -scp force use of SCP protocol -sshlog
file

-sshrawlog file

log protocol details to a file -logoverwrite

`-logappend`

control what happens when a log file already exists
`</code>`

(PSCP's interface is much like the Unix `scp` command, if you're familiar with that.)

- [5.2.1 The basics](#)
 - [5.2.1.1 user](#)
 - [5.2.1.2 host](#)
 - [5.2.1.3 source](#)
 - [5.2.1.4 target](#)
- [5.2.2 Options](#)
 - [5.2.2.1 -ls list remote files](#)
 - [5.2.2.2 -p preserve file attributes](#)
 - [5.2.2.3 -q quiet, don't show statistics](#)
 - [5.2.2.4 -r copies directories recursively](#)
 - [5.2.2.5 -batch avoid interactive prompts](#)
 - [5.2.2.6 -sftp, -scp force use of particular file transfer protocol](#)
 - [5.2.2.7 -no-sanitise-stderr: control error message sanitisation](#)
- [5.2.3 Return value](#)
- [5.2.4 Using public key authentication with PSCP](#)

5.2.1 The basics

To receive (a) file(s) from a remote server:

```
pscp [options] [user@]host:source target
```

So to copy the file /etc/hosts from the server example.com as user fred to the file c:\temp\example-hosts.txt, you would type:

```
pscp fred@example.com:/etc/hosts c:\temp\example-hosts.txt
```

To send (a) file(s) to a remote server:

```
pscp [options] source [source...] [user@]host:target
```

So to copy the local file c:\documents\foo.txt to the server example.com as user fred to the file /tmp/foo you would type:

```
pscp c:\documents\foo.txt fred@example.com:/tmp/foo
```

You can use wildcards to transfer multiple files in either direction, like this:

```
pscp c:\documents\*.doc fred@example.com:docfiles  
pscp fred@example.com:source/*.c c:\source
```

However, in the second case (using a wildcard for multiple remote files) you may see a warning saying something like ‘warning: remote host tried to write to a file called ‘terminal.c’ when we requested a file called ‘*.c’. If this is a wildcard, consider upgrading to SSH-2 or using the ‘-unsafe’ option. Renaming of this file has been disallowed’.

This is due to a fundamental insecurity in the old-style SCP protocol: the client sends the wildcard string (*.c) to the server, and the server sends back a sequence of file names that match the wildcard pattern. However, there is nothing to stop the server sending back a *different* pattern and writing over one of your other files: if you request *.c, the server might send back the file name AUTOEXEC.BAT

and install a virus for you. Since the wildcard matching rules are decided by the server, the client cannot reliably verify that the filenames sent back match the pattern.

PSCP will attempt to use the newer SFTP protocol (part of SSH-2) where possible, which does not suffer from this security flaw. If you are talking to an SSH-2 server which supports SFTP, you will never see this warning. (You can force use of the SFTP protocol, if available, with `-sftp` - see [section 5.2.2.6](#).)

If you really need to use a server-side wildcard with an SSH-1 server, you can use the `-unsafe` command line option with PSCP:

```
pscp -unsafe fred@example.com:source/* .c c:\source
```

This will suppress the warning message and the file transfer will happen. However, you should be aware that by using this option you are giving the server the ability to write to *any* file in the target directory, so you should only use this option if you trust the server administrator not to be malicious (and not to let the server machine be cracked by malicious people). Alternatively, do any such download in a newly created empty directory. (Even in ‘unsafe’ mode, PSCP will still protect you against the server trying to get out of that directory using pathnames including ‘..’.)

- [5.2.1.1 user](#)
- [5.2.1.2 host](#)
- [5.2.1.3 source](#)
- [5.2.1.4 target](#)

5.2.1.1 user

The login name on the remote server. If this is omitted, and `host` is a PuTTY saved session, PSCP will use any username specified by that saved session. Otherwise, PSCP will attempt to use the local Windows username.

5.2.1.2 host

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.

5.2.1.3 source

One or more source files. Wildcards are allowed. The syntax of wildcards depends on the system to which they apply, so if you are copying *from* a Windows system *to* a UNIX system, you should use Windows wildcard syntax (e.g. `*.*`), but if you are copying *from* a UNIX system *to* a Windows system, you would use the wildcard syntax allowed by your UNIX shell (e.g. `*`).

If the source is a remote server and you do not specify a full pathname (in UNIX, a pathname beginning with a `/` (slash) character), what you specify as a source will be interpreted relative to your home directory on the remote server.

5.2.1.4 target

The filename or directory to put the file(s). When copying from a remote server to a local host, you may wish simply to place the file(s) in the current directory. To do this, you should specify a target of ..

For example: `pscp fred@example.com:/home/tom/.emacs .`

...would copy `/home/tom/.emacs` on the remote server to the current directory.

As with the source parameter, if the target is on a remote server and is not a full path name, it is interpreted relative to your home directory on the remote server.

5.2.2 Options

PSCP accepts all the general command line options supported by the PuTTY tools, except the ones which make no sense in a file transfer utility. See [section 3.11.3](#) for a description of these options. (The ones not supported by PSCP are clearly marked.)

PSCP also supports some of its own options. The following sections describe PSCP's specific command-line options.

- [5.2.2.1 -ls list remote files](#)
- [5.2.2.2 -p preserve file attributes](#)
- [5.2.2.3 -q quiet, don't show statistics](#)
- [5.2.2.4 -r copies directories recursively](#)
- [5.2.2.5 -batch avoid interactive prompts](#)
- [5.2.2.6 -sftp, -scp force use of particular file transfer protocol](#)
- [5.2.2.7 -no-sanitise-stderr; control error message sanitisation](#)

5.2.2.1 -ls list remote files

If the `-ls` option is given, no files are transferred; instead, remote files are listed. Only a hostname specification and optional remote file specification need be given. For example:

```
pscp -ls fred@example.com:dir1
```

The SCP protocol does not contain within itself a means of listing files. If SCP is in use, this option therefore assumes that the server responds appropriately to the command `ls -la`; this may not work with all servers.

If SFTP is in use, this option should work with all servers.

5.2.2.2 -p preserve file attributes

By default, files copied with PSCP are timestamped with the date and time they were copied. The **-p** option preserves the original timestamp on copied files.

5.2.2.3 -q quiet, don't show statistics

By default, PSCP displays a meter displaying the progress of the current transfer:

```
mibs.tar          | 168 KB | 84.0 kB/s | ETA: 00:00:13 |
13%
```

The fields in this display are (from left to right), filename, size (in kilobytes) of file transferred so far, estimate of how fast the file is being transferred (in kilobytes per second), estimated time that the transfer will be complete, and percentage of the file so far transferred. The -q option to PSCP suppresses the printing of these statistics.

5.2.2.4 -r copies directories recursively

By default, PSCP will only copy files. Any directories you specify to copy will be skipped, as will their contents. The **-r** option tells PSCP to descend into any directories you specify, and to copy them and their contents. This allows you to use PSCP to transfer whole directory structures between machines.

5.2.2.5 -batch avoid interactive prompts

If you use the `-batch` option, PSCP will never give an interactive prompt while establishing the connection. If the server's host key is invalid, for example (see [section 2.2](#)), then the connection will simply be abandoned instead of asking you what to do next.

This may help PSCP's behaviour when it is used in automated scripts: using `-batch`, if something goes wrong at connection time, the batch job will fail rather than hang.

5.2.2.6 -sftp, -scp force use of particular file transfer protocol

As mentioned in [section 5.2.1](#), there are two different file transfer protocols in use with SSH. Despite its name, PSCP (like many other ostensible scp clients) can use either of these protocols.

The older SCP protocol does not have a written specification and leaves a lot of detail to the server platform. Wildcards are expanded on the server. The simple design means that any wildcard specification supported by the server platform (such as brace expansion) can be used, but also leads to interoperability issues such as with filename quoting (for instance, where filenames contain spaces), and also the security issue described in [section 5.2.1](#).

The newer SFTP protocol, which is usually associated with SSH-2 servers, is specified in a more platform independent way, and leaves issues such as wildcard syntax up to the client. (PuTTY's SFTP wildcard syntax is described in [section 6.2.2](#).) This makes it more consistent across platforms, more suitable for scripting and automation, and avoids security issues with wildcard matching.

Normally PSCP will attempt to use the SFTP protocol, and only fall back to the SCP protocol if SFTP is not available on the server.

The -scp option forces PSCP to use the SCP protocol or quit.

The -sftp option forces PSCP to use the SFTP protocol or quit. When this option is specified, PSCP looks harder for an SFTP server, which may allow use of SFTP with SSH-1 depending on server setup.

5.2.2.7 -no-sanitise-stderr: control error message sanitisation

The `-no-sanitise-stderr` option will cause PSCP to pass through the server's standard-error stream literally, without stripping control characters from it first. This might be useful if the server were sending coloured error messages, but it also gives the server the ability to have unexpected effects on your terminal display. For more discussion, see [section 7.2.3.5](#).

5.2.3 Return value

PSCP returns an ERRORLEVEL of zero (success) only if the files were correctly transferred. You can test for this in a batch file, using code such as this:

```
pscp file*.* user@hostname:  
if errorlevel 1 echo There was an error
```

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 (see [section 5.2.1.2](#)). So you would do this:

- Run PuTTY, and create a PuTTY saved session (see [section 4.1.2](#)) which specifies your private key file (see [section 4.22.1](#)). You will probably also want to specify a username to log in as (see [section 4.15.1](#)).
- 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. See [section 3.11.3.18](#) for more information.

Thirdly, PSCP will attempt to authenticate using Pageant if Pageant is running (see [chapter 9](#)). 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](#).

Chapter 6: Using PSFTP to transfer files securely

PSFTP, the PuTTY SFTP client, is a tool for transferring files securely between computers using an SSH connection.

PSFTP differs from PSCP in the following ways:

- PSCP should work on virtually every SSH server. PSFTP uses the new SFTP protocol, which is a feature of SSH-2 only. (PSCP will also use this protocol if it can, but there is an SSH-1 equivalent it can fall back to if it cannot.)
- PSFTP allows you to run an interactive file transfer session, much like the Windows `ftp` program. You can list the contents of directories, browse around the file system, issue multiple `get` and `put` commands, and eventually log out. By contrast, PSCP is designed to do a single file transfer operation and immediately terminate.
- [6.1 Starting PSFTP](#)
 - [6.1.1 -b: specify a file containing batch commands](#)
 - [6.1.2 -bc: display batch commands as they are run](#)
 - [6.1.3 -be: continue batch processing on errors](#)
 - [6.1.4 -batch: avoid interactive prompts](#)
- [6.2 Running PSFTP](#)
 - [6.2.1 General quoting rules for PSFTP commands](#)
 - [6.2.2 Wildcards in PSFTP](#)
 - [6.2.3 The open command: start a session](#)
 - [6.2.4 The quit command: end your session](#)
 - [6.2.5 The close command: close your connection](#)
 - [6.2.6 The help command: get quick online help](#)
 - [6.2.7 The cd and pwd commands: changing the remote working directory](#)
 - [6.2.8 The lcd and lpwd commands: changing the local working directory](#)

- [6.2.9 The get command: fetch a file from the server](#)
- [6.2.10 The put command: send a file to the server](#)
- [6.2.11 The mget and mput commands: fetch or send multiple files](#)
- [6.2.12 The reget and reput commands: resuming file transfers](#)
- [6.2.13 The dir command: list remote files](#)
- [6.2.14 The chmod command: change permissions on remote files](#)
- [6.2.15 The del command: delete remote files](#)
- [6.2.16 The mkdir command: create remote directories](#)
- [6.2.17 The rmdir command: remove remote directories](#)
- [6.2.18 The mv command: move and rename remote files](#)
- [6.2.19 The ! command: run a local Windows command](#)
- [6.3 Using public key authentication with PSFTP](#)

6.1 Starting PSFTP

The usual way to start PSFTP is from a command prompt, much like PSCP. To do this, it will need either to be on your PATH or in your current directory. To add the directory containing PSFTP to your PATH environment variable, type into the console window:

```
set PATH=C:\path\to\putty\directory;%PATH%
```

Unlike PSCP, however, PSFTP has no complex command-line syntax; you just specify a host name and perhaps a user name:

```
psftp server.example.com
```

or perhaps

```
psftp fred@server.example.com
```

Alternatively, if you just type `psftp` on its own (or double-click the PSFTP icon in the Windows GUI), you will see the PSFTP prompt, and a message telling you PSFTP has not connected to any server:

```
C:\>psftp  
psftp: no hostname specified; use "open host.name" to connect  
psftp>
```

At this point you can type `open server.example.com` or `open fred@server.example.com` to start a session.

PSFTP accepts all the general command line options supported by the PuTTY tools, except the ones which make no sense in a file transfer utility. See [Section 3.11.3](#) for a description of these options. (The ones not supported by PSFTP are clearly marked.)

PSFTP also supports some of its own options. The following sections describe PSFTP's specific command-line options.

- [6.1.1 -b: specify a file containing batch commands](#)
- [6.1.2 -bc: display batch commands as they are run](#)

- [6.1.3 -be: continue batch processing on errors](#)
- [6.1.4 -batch: avoid interactive prompts](#)
 - [6.1.4.1 -no-sanitise-stderr: control error message sanitisation](#)

6.1.1 -b: specify a file containing batch commands

In normal operation, PSFTP is an interactive program which displays a command line and accepts commands from the keyboard.

If you need to do automated tasks with PSFTP, you would probably prefer to specify a set of commands in advance and have them executed automatically. The `-b` option allows you to do this. You use it with a file name containing batch commands. For example, you might create a file called `myscript.scr` containing lines like this:

```
cd /home/ftp/users/jeff  
del jam-old.tar.gz  
ren jam.tar.gz jam-old.tar.gz  
put jam.tar.gz  
chmod a+r jam.tar.gz
```

and then you could run the script by typing

```
psftp user@hostname -b myscript.scr
```

When you run a batch script in this way, PSFTP will abort the script if any command fails to complete successfully. To change this behaviour, you can add the `-be` option ([section 6.1.3](#)).

PSFTP will terminate after it finishes executing the batch script.

```
<code>C:\>psftp fred@hostname -b batchfile Sent  
username "fred"
```

Remote working directory is /home/fred Listing
directory /home/fred/lib drwxrwsr-x 4 fred fred 1024
Sep 6 10:42 .

drwxr-sr-x 25 fred fred 2048 Dec 14 09:36 ..

drwxrwsr-x 3 fred fred 1024 Apr 17 2000 jed
lrwxrwxrwx 1 fred fred 24 Apr 17 2000 timber
drwxrwsr-x 2 fred fred 1024 Mar 13 2000 trn
</code>

```
<code>C:\>psftp fred@hostname -bc -b batchfile  
Sent username "fred"
```

Remote working directory is /home/fred psftp> dir
lib

Listing directory /home/fred/lib drwxrwsr-x 4 fred
fred 1024 Sep 6 10:42 .

drwxr-sr-x 25 fred fred 2048 Dec 14 09:36 ..

```
drwxrwsr-x 3 fred fred 1024 Apr 17 2000 jed
lrwxrwxrwx 1 fred fred 24 Apr 17 2000 timber
drwxrwsr-x 2 fred fred 1024 Mar 13 2000 trn psftp>
quit
```

```
</code>
```

6.1.3 -be: continue batch processing on errors

When running a batch file, this additional option causes PSFTP to continue processing even if a command fails to complete successfully.

You might want this to happen if you wanted to delete a file and didn't care if it was already not present, for example.

6.1.4 -batch: avoid interactive prompts

If you use the `-batch` option, PSFTP will never give an interactive prompt while establishing the connection. If the server's host key is invalid, for example (see [section 2.2](#)), then the connection will simply be abandoned instead of asking you what to do next.

This may help PSFTP's behaviour when it is used in automated scripts: using `-batch`, if something goes wrong at connection time, the batch job will fail rather than hang.

- [6.1.4.1 -no-sanitise-stderr: control error message sanitisation](#)

6.1.4.1 -no-sanitise-stderr: control error message sanitisation

The `-no-sanitise-stderr` option will cause PSFTP to pass through the server's standard-error stream literally, without stripping control characters from it first. This might be useful if the server were sending coloured error messages, but it also gives the server the ability to have unexpected effects on your terminal display. For more discussion, see [section 7.2.3.5](#).

6.2 Running PSFTP

Once you have started your PSFTP session, you will see a `psftp>` prompt. You can now type commands to perform file-transfer functions. This section lists all the available commands.

Any line starting with a # will be treated as a comment and ignored.

- [6.2.1 General quoting rules for PSFTP commands](#)
- [6.2.2 Wildcards in PSFTP](#)
- [6.2.3 The open command: start a session](#)
- [6.2.4 The quit command: end your session](#)
- [6.2.5 The close command: close your connection](#)
- [6.2.6 The help command: get quick online help](#)
- [6.2.7 The cd and pwd commands: changing the remote working directory](#)
- [6.2.8 The lcd and lpwd commands: changing the local working directory](#)
- [6.2.9 The get command: fetch a file from the server](#)
- [6.2.10 The put command: send a file to the server](#)
- [6.2.11 The mget and mput commands: fetch or send multiple files](#)
- [6.2.12 The reget and reput commands: resuming file transfers](#)
- [6.2.13 The dir command: list remote files](#)
- [6.2.14 The chmod command: change permissions on remote files](#)
- [6.2.15 The del command: delete remote files](#)
- [6.2.16 The mkdir command: create remote directories](#)
- [6.2.17 The rmdir command: remove remote directories](#)
- [6.2.18 The mv command: move and rename remote files](#)
- [6.2.19 The ! command: run a local Windows command](#)

6.2.1 General quoting rules for PSFTP commands

Most PSFTP commands are considered by the PSFTP command interpreter as a sequence of words, separated by spaces. For example, the command `ren oldfilename newfilename` splits up into three words: `ren` (the command name), `oldfilename` (the name of the file to be renamed), and `newfilename` (the new name to give the file).

Sometimes you will need to specify file names that *contain* spaces. In order to do this, you can surround the file name with double quotes. This works equally well for local file names and remote file names: `psftp> get "spacey file name.txt" "save it under this name.txt"`

The double quotes themselves will not appear as part of the file names; they are removed by PSFTP and their only effect is to stop the spaces inside them from acting as word separators.

If you need to *use* a double quote (on some types of remote system, such as Unix, you are allowed to use double quotes in file names), you can do this by doubling it. This works both inside and outside double quotes. For example, this command `psftp> ren ""this"" "a file with ""quotes"" in it"`

will take a file whose current name is `"this"` (with a double quote character at the beginning and the end) and rename it to a file whose name is a file with `"quotes"` in it.

(The one exception to the PSFTP quoting rules is the `!` command, which passes its command line straight to Windows without splitting it up into words at all. See [section 6.2.19](#).)

6.2.2 Wildcards in PSFTP

Several commands in PSFTP support ‘wildcards’ to select multiple files.

For *local* file specifications (such as the first argument to `put`), wildcard rules for the local operating system are used. For instance, PSFTP running on Windows might require the use of `*.*` where PSFTP on Unix would need `*`.

For *remote* file specifications (such as the first argument to `get`), PSFTP uses a standard wildcard syntax (similar to POSIX wildcards):

- `*` matches any sequence of characters (including a zero-length sequence).
- `?` matches exactly one character.
- `[abc]` matches exactly one character which can be `a`, `b`, or `c`.

`[a-z]` matches any character in the range `a` to `z`.

`[^abc]` matches a single character that is *not* `a`, `b`, or `c`.

Special cases: `[-a]` matches a literal hyphen (`-`) or `a`; `[^-a]` matches all other characters. `[a^]` matches a literal caret (`^`) or `a`.

- `\` (backslash) before any of the above characters (or itself) removes that character's special meaning.

A leading period `(.)` on a filename is not treated specially, unlike in some Unix contexts; `get * will fetch all files, whether or not they start with a leading period.`

6.2.3 The open command: start a session

If you started PSFTP by double-clicking in the GUI, or just by typing `psftp` at the command line, you will need to open a connection to an SFTP server before you can issue any other commands (except `help` and `quit`).

To create a connection, type `open host.name`, or if you need to specify a user name as well you can type `open user@host.name`. You can optionally specify a port as well: `open user@host.name 22`.

Once you have issued this command, you will not be able to issue it again, even if the command fails (for example, if you mistype the host name or the connection times out). So if the connection is not opened successfully, PSFTP will terminate immediately.

6.2.4 The quit command: end your session

When you have finished your session, type the command `quit` to close the connection, terminate PSFTP and return to the command line (or just close the PSFTP console window if you started it from the GUI).

You can also use the `bye` and `exit` commands, which have exactly the same effect.

6.2.5 The close command: close your connection

If you just want to close the network connection but keep PSFTP running, you can use the `close` command. You can then use the `open` command to open a new connection.

6.2.6 The help command: get quick online help

If you type `help`, PSFTP will give a short list of the available commands.

If you type `help` with a command name - for example, `help get` - then PSFTP will give a short piece of help on that particular command.

6.2.7 The cd and pwd commands: changing the remote working directory

PSFTP maintains a notion of your ‘working directory’ on the server. This is the default directory that other commands will operate on. For example, if you type `get filename.dat` then PSFTP will look for `filename.dat` in your remote working directory on the server.

To change your remote working directory, use the `cd` command. If you don’t provide an argument, `cd` will return you to your home directory on the server (more precisely, the remote directory you were in at the start of the connection).

To display your current remote working directory, type `pwd`.

6.2.8 The lcd and lpwd commands: changing the local working directory

As well as having a working directory on the remote server, PSFTP also has a working directory on your local machine (just like any other Windows process). This is the default local directory that other commands will operate on. For example, if you type get filename.dat then PSFTP will save the resulting file as filename.dat in your local working directory.

To change your local working directory, use the lcd command. To display your current local working directory, type lpwd.

6.2.9 The get command: fetch a file from the server

To download a file from the server and store it on your local PC, you use the `get` command.

In its simplest form, you just use this with a file name:

```
get myfile.dat
```

If you want to store the file locally under a different name, specify the local file name after the remote one:

```
get myfile.dat newname.dat
```

This will fetch the file on the server called `myfile.dat`, but will save it to your local machine under the name `newname.dat`.

To fetch an entire directory recursively, you can use the `-r` option:

```
get -r mydir  
get -r mydir newname
```

(If you want to fetch a file whose name starts with a hyphen, you may have to use the `--` special argument, which stops `get` from interpreting anything as a switch after it. For example, ‘`get -- - silly-name-`’.)

6.2.10 The put command: send a file to the server

To upload a file to the server from your local PC, you use the `put` command.

In its simplest form, you just use this with a file name:

```
put myfile.dat
```

If you want to store the file remotely under a different name, specify the remote file name after the local one: `put myfile.dat newname.dat`

This will send the local file called `myfile.dat`, but will store it on the server under the name `newname.dat`.

To send an entire directory recursively, you can use the `-r` option:

```
put -r mydir put -r mydir newname
```

(If you want to send a file whose name starts with a hyphen, you may have to use the `--` special argument, which stops `put` from interpreting anything as a switch after it. For example, ‘`put -- - silly-name-`’.)

6.2.11 The `mget` and `mput` commands: fetch or send multiple files

`mget` works almost exactly like `get`, except that it allows you to specify more than one file to fetch at once. You can do this in two ways:

- by giving two or more explicit file names ('`mget file1.txt file2.txt`')
- by using a wildcard ('`mget *.txt`').

Every argument to `mget` is treated as the name of a file to fetch (unlike `get`, which will interpret at most one argument like that, and a second argument will be treated as an alternative name under which to store the retrieved file), or a wildcard expression matching more than one file.

The `-r` and `--` options from `get` are also available with `mget`.

`mput` is similar to `put`, with the same differences.

6.2.12 The reget and reput commands: resuming file transfers

If a file transfer fails half way through, and you end up with half the file stored on your disk, you can resume the file transfer using the `reget` and `reput` commands. These work exactly like the `get` and `put` commands, but they check for the presence of the half-written destination file and start transferring from where the last attempt left off.

The syntax of `reget` and `reput` is exactly the same as the syntax of `get` and `put`:

```
reget myfile.dat
reget myfile.dat newname.dat
reget -r mydir
```

These commands are intended mainly for resuming interrupted transfers. They assume that the remote file or directory structure has not changed in any way; if there have been changes, you may end up with corrupted files. In particular, the `-r` option will not pick up changes to files or directories already transferred in full.

6.2.13 The `dir` command: list remote files

To list the files in your remote working directory, just type `dir`.

You can also list the contents of a different directory by typing `dir` followed by the directory name: `dir /home/fred` `dir sources`

And you can list a subset of the contents of a directory by providing a wildcard: `dir /home/fred/*.txt` `dir sources/*.c`

The `ls` command works exactly the same way as `dir`.

```
<code>chmod go-rwx,u+w privatefile
```

```
chmod a+r public*
```

```
chmod 640 groupfile1 groupfile2
```

```
</code>
```

The `modes` parameter can be a set of octal digits in the Unix style. (If you don't know what this means, you probably don't want to be using it!) Alternatively, it can be a list of permission modifications, separated by commas. Each modification consists of:

- The people affected by the modification. This can be `u` (the owning user), `g` (members of the owning group), or `o` (everybody else - ‘others’), or some combination of those. It can also be `a` (‘all’) to affect everybody at once.
- A `+` or `-` sign, indicating whether permissions are to be added or removed.
- The actual permissions being added or removed. These can be `r` (permission to read the file), `w` (permission to write to the file), and `x` (permission

to execute the file, or in the case of a directory, permission to access files within the directory).

So the above examples would do:

- The first example: `go-rwx` removes read, write and execute permissions for members of the owning group and everybody else (so the only permissions left are the ones for the file owner). `u+w` adds write permission for the file owner.
- The second example: `a+r` adds read permission for everybody to all files and directories starting with ‘public’.

In addition to all this, there are a few extra special cases for Unix systems. On non-Unix systems these are unlikely to be useful:

- You can specify `u+s` and `u-s` to add or remove the Unix set-user-ID bit. This is typically only useful for special purposes; refer to your Unix documentation if you're not sure about it.
- You can specify `g+s` and `g-s` to add or remove the Unix set-group-ID bit. On a file, this works similarly to the set-user-ID bit (see your Unix documentation again); on a directory it ensures

that files created in the directory are accessible by members of the group that owns the directory.

- You can specify +t and -t to add or remove the Unix ‘sticky bit’. When applied to a directory, this means that the owner of a file in that directory can delete the file (whereas normally only the owner of the *directory* would be allowed to).

6.2.15 The del command: delete remote files

To delete a file on the server, type `del` and then the filename or filenames:

```
del oldfile.dat del file1.txt file2.txt del *.o
```

Files will be deleted without further prompting, even if multiple files are specified.

`del` will only delete files. You cannot use it to delete directories; use `rmdir` for that.

The `rm` command works exactly the same way as `del`.

6.2.16 The `mkdir` command: create remote directories

To create a directory on the server, type `mkdir` and then the directory name:

```
mkdir newstuff
```

You can specify multiple directories to create at once:

```
mkdir dir1 dir2 dir3
```

6.2.17 The `rmdir` command: remove remote directories

To remove a directory on the server, type `rmdir` and then the directory name or names: `rmdir oldstuff rmdir *.old ancient`

Directories will be deleted without further prompting, even if multiple directories are specified.

Most SFTP servers will probably refuse to remove a directory if the directory has anything in it, so you will need to delete the contents first.

6.2.18 The `mv` command: move and rename remote files

To rename a single file on the server, type `mv`, then the current file name, and then the new file name: `mv oldfile newname`

You can also move the file into a different directory and change the name:

```
mv oldfile dir/newname
```

To move one or more files into an existing subdirectory, specify the files (using wildcards if desired), and then the destination directory:

```
mv file dir mv file1 dir1/file2 dir2 mv *.c *.h ..
```

The `rename` and `ren` commands work exactly the same way as `mv`.

6.2.19 The ! command: run a local Windows command

You can run local Windows commands using the ! command. This is the only PSFTP command that is not subject to the command quoting rules given in [section 6.2.1](#). If any command line begins with the ! character, then the rest of the line will be passed straight to Windows without further translation.

For example, if you want to move an existing copy of a file out of the way before downloading an updated version, you might type: psftp>
!ren myfile.dat myfile.bak psftp> get myfile.dat

using the Windows ren command to rename files on your local PC.

6.3 Using public key authentication with PSFTP

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

Firstly, PSFTP can use PuTTY saved sessions in place of hostnames. So you might do this:

- Run PuTTY, and create a PuTTY saved session (see [section 4.1.2](#)) which specifies your private key file (see [section 4.22.1](#)). You will probably also want to specify a username to log in as (see [section 4.15.1](#)).
- In PSFTP, you can now use the name of the session instead of a hostname: type `psftp sessionname`, 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. See [section 3.11.3.18](#) for more information.

Thirdly, PSFTP will attempt to authenticate using Pageant if Pageant is running (see [chapter 9](#)). So you would do this:

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

For more general information on public-key authentication, see [chapter 8](#).

Chapter 7: Using the command-line connection tool Plink

Plink is a command-line connection tool similar to UNIX ssh. It is mostly used for automated operations, such as making CVS access a repository on a remote server.

Plink is probably not what you want if you want to run an interactive session in a console window.

- [7.1 Starting Plink](#)
- [7.2 Using Plink](#)
 - [7.2.1 Using Plink for interactive logins](#)
 - [7.2.2 Using Plink for automated connections](#)
 - [7.2.3 Plink command line options](#)
- [7.3 Using Plink in batch files and scripts](#)
- [7.4 Using Plink with CVS](#)
- [7.5 Using Plink with WinCVS](#)

7.1 Starting Plink

Plink is a command line application. This means that you cannot just double-click on its icon to run it and instead you have to bring up a console window. In Windows 95, 98, and ME, this is called an ‘MS-DOS Prompt’, and in Windows NT, 2000, and XP, it is called a ‘Command Prompt’. It should be available from the Programs section of your Start Menu.

In order to use Plink, the file `plink.exe` will need either to be on your PATH or in your current directory. To add the directory containing Plink to your PATH environment variable, type into the console window:

```
set PATH=C:\path\to\putty\directory;%PATH%
```

This will only work for the lifetime of that particular console window. To set your PATH more permanently on Windows NT, 2000, and XP, use the Environment tab of the System Control Panel. On Windows 95, 98, and ME, you will need to edit your `AUTOEXEC.BAT` to include a `set` command like the one above.

<code>C:\>plink

Plink: command-line connection utility

Release 0.81

Usage: plink [options] [user@]host [command]

("host" can also be a PuTTY saved session name)

Options:

-V print version information and exit -pgpfp print PGP key fingerprints and exit -v show verbose messages

-load sessname Load settings from saved session -ssh -telnet -rlogin -raw -serial force use of a particular protocol -ssh-connection

force use of the bare ssh-connection protocol -P port connect to specified port -l user connect with specified username -batch disable all interactive prompts -proxycmd command

use 'command' as local proxy -sercfg configuration-string (e.g. 19200,8,n,1,X) Specify the serial configuration (serial only) The following options only apply to SSH connections: -pwfile file login with password read from specified file -D [listen-IP:]listen-port

Dynamic SOCKS-based port forwarding -L [listen-IP:]listen-port:host:port Forward local port to remote address -R [listen-IP:]listen-port:host:port Forward remote port to local address -X -x enable / disable X11 forwarding -A -a enable / disable agent forwarding -t -T enable / disable pty allocation -1 -2 force use of particular SSH protocol version

-4 -6 force use of IPv4 or IPv6

-C enable compression

-i key private key file for user authentication -
noagent disable use of Pageant -agent enable use of
Pageant -no-trivial-auth

disconnect if SSH authentication succeeds trivially
-noshare disable use of connection sharing -share
enable use of connection sharing -hostkey keyid

manually specify a host key (may be repeated) -
sanitise-stderr, -sanitise-stdout, -no-sanitise-stderr, -
no-sanitise-stdout do/don't strip control chars from
standard output/error -no-antispoof omit anti-
spoofing prompt after authentication -m file read
remote command(s) from file -s remote command is
an SSH subsystem (SSH-2 only) -N don't start a
shell/command (SSH-2 only) -nc host:port

open tunnel in place of session (SSH-2 only) -
sshlog file

-sshrawlog file

log protocol details to a file -logoverwrite

-logappend

control what happens when a log file already exists

-shareexists

test whether a connection-sharing upstream exists

</code>

Once this works, you are ready to use Plink.

- [7.2.1 Using Plink for interactive logins](#)
- [7.2.2 Using Plink for automated connections](#)
- [7.2.3 Plink command line options](#)
 - [7.2.3.1 -batch: disable all interactive prompts](#)
 - [7.2.3.2 -s: remote command is SSH subsystem](#)
 - [7.2.3.3 -share: Test and try to share an existing connection.](#)
 - [7.2.3.4 -shareexists: test for connection-sharing upstream](#)
 - [7.2.3.5 -sanitise-stream: control output sanitisation](#)
 - [7.2.3.6 -no-antispoof: turn off authentication spoofing protection prompt](#)

7.2.1 Using Plink for interactive logins

To make a simple interactive connection to a remote server, just type plink and then the host name:

```
C:\>plink login.example.com
```

```
Debian GNU/Linux 2.2 flunky.example.com
flunky login:
```

You should then be able to log in as normal and run a session. The output sent by the server will be written straight to your command prompt window, which will most likely not interpret terminal control codes in the way the server expects it to. So if you run any full-screen applications, for example, you can expect to see strange characters appearing in your window. Interactive connections like this are not the main point of Plink.

In order to connect with a different protocol, you can give the command line options -ssh, -ssh-connection, -telnet, -rlogin, or -raw. To make an SSH connection, for example:

```
C:\>plink -ssh login.example.com
login as:
```

If you have already set up a PuTTY saved session, then instead of supplying a host name, you can give the saved session name. This allows you to use public-key authentication, specify a user name, and use most of the other features of PuTTY:

```
C:\>plink my-ssh-session
Sent username "fred"
Authenticating with public key "fred@winbox"
Last login: Thu Dec  6 19:25:33 2001 from :0.0
fred@flunky:~$
```

(You can also use the -load command-line option to load a saved session; see [section 3.11.3.1](#). If you use -load, the saved session exists, and it specifies a hostname, you cannot also specify a host or

user@host argument - it will be treated as part of the remote command.)

7.2.2 Using Plink for automated connections

More typically Plink is used with the SSH protocol, to enable you to talk directly to a program running on the server. To do this you have to ensure Plink is *using* the SSH protocol. You can do this in several ways:

- Use the `-ssh` option as described in [section 7.2.1](#).
- Set up a PuTTY saved session that describes the server you are connecting to, and that also specifies the protocol as SSH.
- Set the Windows environment variable `PLINK_PROTOCOL` to the word `ssh`.

Usually Plink is not invoked directly by a user, but run automatically by another process. Therefore you typically do not want Plink to prompt you for a user name or a password.

Next, you are likely to need to avoid the various interactive prompts Plink can produce. You might be prompted to verify the host key of the server you're connecting to, to enter a user name, or to enter a password.

To avoid being prompted for the server host key when using Plink for an automated connection, you can first make a *manual* connection (using either of PuTTY or Plink) to the same server, verify the host key (see [section 2.2](#) for more information), and select 'Accept' to add the host key to the Registry. After that, Plink commands connecting to that server should not give a host key prompt unless the host key changes. Alternatively, you can specify the appropriate host key(s) on Plink's command line every time you use it; see [section 3.11.3.22](#).

To avoid being prompted for a user name, you can:

- Use the `-l` option to specify a user name on the command line. For example, `plink login.example.com -l fred`.
- Set up a PuTTY saved session that describes the server you are connecting to, and that also specifies the username to log in

as (see [section 4.15.1](#)).

To avoid being prompted for a password, you should almost certainly set up public-key authentication. (See [chapter 8](#) for a general introduction to public-key authentication.) Again, you can do this in two ways:

- Set up a PuTTY saved session that describes the server you are connecting to, and that also specifies a private key file (see [section 4.22.1](#)). For this to work without prompting, your private key will need to have no passphrase.
- Store the private key in Pageant. See [chapter 9](#) for further information.

Once you have done all this, you should be able to run a remote command on the SSH server machine and have it execute automatically with no prompting:

```
C:\>plink login.example.com -l fred echo hello, world  
hello, world
```

```
C:\>
```

Or, if you have set up a saved session with all the connection details:

```
C:\>plink mysession echo hello, world  
hello, world
```

```
C:\>
```

Then you can set up other programs to run this Plink command and talk to it as if it were a process on the server machine.

7.2.3 Plink command line options

Plink accepts all the general command line options supported by the PuTTY tools. See [section 3.11.3](#) for a description of these options.

Plink also supports some of its own options. The following sections describe Plink's specific command-line options.

- [7.2.3.1 -batch: disable all interactive prompts](#)
- [7.2.3.2 -s: remote command is SSH subsystem](#)
- [7.2.3.3 -share: Test and try to share an existing connection.](#)
- [7.2.3.4 -shareexists: test for connection-sharing upstream](#)
- [7.2.3.5 -sanitise-stream: control output sanitisation](#)
- [7.2.3.6 -no-antispoof: turn off authentication spoofing protection prompt](#)

7.2.3.1 -batch: disable all interactive prompts

If you use the -batch option, Plink will never give an interactive prompt while establishing the connection. If the server's host key is invalid, for example (see [section 2.2](#)), then the connection will simply be abandoned instead of asking you what to do next.

This may help Plink's behaviour when it is used in automated scripts: using -batch, if something goes wrong at connection time, the batch job will fail rather than hang.

7.2.3.2 -s: remote command is SSH subsystem

If you specify the `-s` option, Plink passes the specified command as the name of an SSH ‘subsystem’ rather than an ordinary command line.

(This option is only meaningful with the SSH-2 protocol.)

7.2.3.3 -share: Test and try to share an existing connection.

This option tries to detect if an existing connection can be shared (See [section 4.17.5](#) for more information about SSH connection sharing.) and reuses that connection.

A Plink invocation of the form: `plink -share <session>`

will test whether there is currently a viable ‘upstream’ for the session in question, which can be specified using any syntax you’d normally use with Plink to make an actual connection (a host/port number, a bare saved session name, `-load`, etc). If no ‘upstream’ viable session is found and `-share` is specified, this connection will become the ‘upstream’ connection for subsequent connection sharing tries.

(This option is only meaningful with the SSH-2 protocol.)

7.2.3.4 -shareexists: test for connection-sharing upstream

This option does not make a new connection; instead it allows testing for the presence of an existing connection that can be shared. (See [section 4.17.5](#) for more information about SSH connection sharing.) A Plink invocation of the form: `plink -shareexists <session>`

will test whether there is currently a viable ‘upstream’ for the session in question, which can be specified using any syntax you’d normally use with Plink to make an actual connection (a host/port number, a bare saved session name, `-load`, etc). It returns a zero exit status if a usable ‘upstream’ exists, nonzero otherwise.

(This option is only meaningful with the SSH-2 protocol.)

7.2.3.5 -sanitise-stream: control output sanitisation

In some situations, Plink applies a sanitisation pass to the output received from the server, to strip out control characters such as backspace and the escape character.

The idea of this is to prevent remote processes from sending confusing escape sequences through the standard error channel when Plink is being used as a transport for something like git or CVS. If the server actually wants to send an error message, it will probably be plain text; if the server abuses that channel to try to write over unexpected parts of your terminal display, Plink will try to stop it.

By default, this only happens for output channels which are sent to a Windows console device, or a Unix terminal device. (Any output stream going somewhere else is likely to be needed by an 8-bit protocol and must not be tampered with at all.) It also stops happening if you tell Plink to allocate a remote pseudo-terminal (see [section 3.11.3.12](#) and [section 4.24.1](#)), on the basis that in that situation you often *want* escape sequences from the server to go to your terminal.

But in case Plink guesses wrong about whether you want this sanitisation, you can override it in either direction, using one of these options:

-sanitise-stderr

Sanitise server data written to Plink's standard error channel, regardless of terminals and consoles and remote ptys.

-no-sanitise-stderr

Do not sanitise server data written to Plink's standard error channel.

-sanitise-stdout

Sanitise server data written to Plink's standard output channel.

`-no-sanitise-stdout`

Do not sanitise server data written to Plink's standard output channel.

7.2.3.6 -no-antspoof: turn off authentication spoofing protection prompt

In SSH, some possible server authentication methods require user input (for example, password authentication, or entering a private key passphrase), and others do not (e.g. a private key held in Pageant).

If you use Plink to run an interactive login session, and if Plink authenticates without needing any user interaction, and if the server is malicious or compromised, it could try to trick you into giving it authentication data that should not go to the server (such as your private key passphrase), by sending what *looks* like one of Plink's local prompts, as if Plink had not already authenticated.

To protect against this, Plink's default policy is to finish the authentication phase with a final trivial prompt looking like this:
Access granted. Press Return to begin session.

so that if you saw anything that looked like an authentication prompt *after* that line, you would know it was not from Plink.

That extra interactive step is inconvenient. So Plink will turn it off in as many situations as it can:

- If Plink's standard input is not pointing at a console or terminal device – for example, if you're using Plink as a transport for some automated application like version control – then you *can't* type passphrases into the server anyway. In that situation, Plink won't try to protect you from the server trying to fool you into doing so.
- If Plink is in batch mode (see [section 7.2.2](#)), then it *never* does any interactive authentication. So anything looking like an interactive authentication prompt is automatically suspect, and so Plink omits the anti-spoofing prompt.

But if you still find the protective prompt inconvenient, and you trust the server not to try a trick like this, you can turn it off using the '-no-antispooft' option.

7.3 Using Plink in batch files and scripts

Once you have set up Plink to be able to log in to a remote server without any interactive prompting (see [section 7.2.2](#)), you can use it for lots of scripting and batch purposes. For example, to start a backup on a remote machine, you might use a command like:

```
plink root@myserver /etc/backups/do-backup.sh
```

Or perhaps you want to fetch all system log lines relating to a particular web area:

```
plink mysession grep /~fred/ /var/log/httpd/access.log > fredlog
```

Any non-interactive command you could usefully run on the server command line, you can run in a batch file using Plink in this way.

7.4 Using Plink with CVS

To use Plink with CVS, you need to set the environment variable `cvs_RSH` to point to Plink: set `cvs_RSH=\path\to\plink.exe`

You also need to arrange to be able to connect to a remote host without any interactive prompts, as described in [section 7.2.2](#).

You should then be able to run CVS as follows: `cvs -d :ext:user@sessionname:/path/to/repository co module`

If you specified a username in your saved session, you don't even need to specify the 'user' part of this, and you can just say: `cvs -d :ext:sessionname:/path/to/repository co module`

7.5 Using Plink with WinCVS

Plink can also be used with WinCVS. Firstly, arrange for Plink to be able to connect to a remote host non-interactively, as described in [section 7.2.2](#).

Then, in WinCVS, bring up the ‘Preferences’ dialogue box from the *Admin* menu, and switch to the ‘Ports’ tab. Tick the box there labelled ‘Check for an alternate rsh name’ and in the text entry field to the right enter the full path to plink.exe. Select ‘OK’ on the ‘Preferences’ dialogue box.

Next, select ‘Command Line’ from the WinCVS ‘Admin’ menu, and type a CVS command as in [section 7.4](#), for example: cvs -d :ext:user@hostname:/path/to/repository co module

or (if you're using a saved session):

```
cvs -d :ext:user@sessionname:/path/to/repository co module
```

Select the folder you want to check out to with the ‘Change Folder’ button, and click ‘OK’ to check out your module. Once you've got modules checked out, WinCVS will happily invoke plink from the GUI for CVS operations.

Chapter 8: Using public keys for SSH authentication

- [8.1 Public key authentication - an introduction](#)
- [8.2 Using PuTTYgen, the PuTTY key generator](#)
 - [8.2.1 Generating a new key](#)
 - [8.2.2 Selecting the type of key](#)
 - [8.2.3 Selecting the size \(strength\) of the key](#)
 - [8.2.4 Selecting the prime generation method](#)
 - [8.2.5 The 'Generate' button](#)
 - [8.2.6 The 'Key fingerprint' box](#)
 - [8.2.7 Setting a comment for your key](#)
 - [8.2.8 Setting a passphrase for your key](#)
 - [8.2.9 Adding a certificate to your key](#)
 - [8.2.10 Saving your private key to a disk file](#)
 - [8.2.11 Saving your public key to a disk file](#)
 - [8.2.12 'Public key for pasting into OpenSSH authorized_keys file'](#)
 - [8.2.13 Parameters for saving key files](#)
 - [8.2.14 Reloading a private key](#)
 - [8.2.15 Dealing with private keys in other formats](#)
 - [8.2.16 PuTTYgen command-line configuration](#)
- [8.3 Getting ready for public key 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](#)), 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](#) for further details.

There is more than one public-key algorithm available. The most common are RSA and ECDSA, 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](#).

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, PSFTP, and Plink, as well as the PuTTY authentication agent, Pageant (see [chapter 9](#)). PuTTYgen generates RSA, DSA, ECDSA, and EdDSA keys.

When you run PuTTYgen you will see a window where you have two main 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](#)
- [8.2.2 Selecting the type of key](#)
- [8.2.3 Selecting the size \(strength\) of the key](#)
- [8.2.4 Selecting the prime generation method](#)
- [8.2.5 The ‘Generate’ button](#)
- [8.2.6 The ‘Key fingerprint’ box](#)
- [8.2.7 Setting a comment for your key](#)
- [8.2.8 Setting a passphrase for your key](#)
- [8.2.9 Adding a certificate to your key](#)
- [8.2.10 Saving your private key to a disk file](#)
- [8.2.11 Saving your public key to a disk file](#)
- [8.2.12 ‘Public key for pasting into OpenSSH authorized_keys file’](#)
- [8.2.13 Parameters for saving key files](#)
 - [8.2.13.1 PPK file version](#)
 - [8.2.13.2 Options affecting passphrase hashing](#)
- [8.2.14 Reloading a private key](#)
- [8.2.15 Dealing with private keys in other formats](#)
- [8.2.16 PuTTYgen command-line configuration](#)

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](#) and [section 8.2.3](#).
- Then press the ‘Generate’ button, to actually generate the key. [Section 8.2.5](#) describes this step.
- Once you have generated the key, select a comment field ([section 8.2.7](#)) and a passphrase ([section 8.2.8](#)).
- Now you’re ready to save the private key to disk; press the ‘Save private key’ button. (See [section 8.2.10](#)).

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 OpenSSH authorized_keys file’ box (see [section 8.2.12](#)), or by using the ‘Save public key’ button ([section 8.2.11](#)). However, you don’t need to do this immediately; if you want, you can load the private key back into PuTTYgen later (see [section 8.2.14](#)) and the public key will be available for copying and pasting again.

[Section 8.3](#) 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.

The current version of the SSH protocol, SSH-2, supports several different key types, although specific servers may not support all of them. PuTTYgen can generate:

- An RSA key for use with the SSH-2 protocol.
- A DSA key for use with the SSH-2 protocol.
- An ECDSA (elliptic curve DSA) key for use with the SSH-2 protocol.
- An EdDSA key (Edwards-curve DSA, another elliptic curve algorithm) for use with the SSH-2 protocol.

PuTTYgen can also generate an RSA key suitable for use with the old SSH-1 protocol (which only supports RSA); for this, you need to select the ‘SSH-1 (RSA)’ option. Since the SSH-1 protocol is no longer considered secure, it’s rare to need this option.

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.

- For RSA and DSA, 2048 bits should currently be sufficient for most purposes. (Smaller keys of these types are no longer considered secure, and PuTTYgen will warn if you try to generate them.)
- For ECDSA, only 256, 384, and 521 bits are supported, corresponding to NIST-standardised elliptic curves. (Elliptic-curve keys do not need as many bits as RSA keys for equivalent security, so these numbers are smaller than the RSA recommendations.)
- For EdDSA, the only valid sizes are 255 bits (these keys are also known as ‘Ed25519’ and are commonly used) and 448 bits (‘Ed448’, which is much less common at the time of writing). (256 is also accepted for backward compatibility, but the effect is the same as 255.)

8.2.4 Selecting the prime generation method

(This is entirely optional. Unless you know better, it's entirely sensible to skip this and use the default settings.)

On the ‘Key’ menu, you can also optionally change the method for generating the prime numbers used in the generated key. This is used for RSA and DSA keys only. (The other key types don't require generating prime numbers at all.)

The prime-generation method does not affect compatibility: a key generated with any of these methods will still work with all the same SSH servers.

The available methods are:

- Use probable primes (fast)
- Use proven primes (slower)
- Use proven primes with even distribution (slowest)

The ‘probable primes’ method sounds unsafe, but it's the most commonly used prime-generation strategy. There is in theory a possibility that it might accidentally generate a number that isn't prime, but the software does enough checking to make that probability vanishingly small (less than 1 in 2^{80} , or 1 in 10^{24}). So, in practice, nobody worries about it very much.

The other methods cause PuTTYgen to use numbers that it is *sure* are prime, because it generates the output number together with a proof of its primality. This takes more effort, but it eliminates that theoretical risk in the probabilistic method.

There is one way in which PuTTYgen's ‘proven primes’ method is not strictly better than its ‘probable primes’ method. If you use PuTTYgen to generate an RSA key on a computer that is potentially susceptible to timing- or cache-based side-channel attacks, such as a shared computer, the ‘probable primes’ method is designed to

resist such attacks, whereas the ‘proven primes’ methods are not. (This is only a concern for RSA keys; for other key types, primes are either not secret or not involved.)

You might choose to switch from probable to proven primes if you have a local security standard that demands it, or if you don't trust the probabilistic argument for the safety of the usual method.

For RSA keys, there's also an option on the ‘Key’ menu to use ‘strong’ primes as the prime factors of the public key. A ‘strong’ prime is a prime number chosen to have a particular structure that makes certain factoring algorithms more difficult to apply, so some security standards recommend their use. However, the most modern factoring algorithms are unaffected, so this option is probably not worth turning on *unless* you have a local standard that recommends it.

8.2.5 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.

8.2.6 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; it is supposed to be more manageable for human beings than the public key itself.

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](#)) and the Unix `ssh-add` utility, will list key fingerprints rather than the whole public key.

By default, PuTTYgen will display SSH-2 key fingerprints in the ‘SHA256’ format. If you need to see the fingerprint in the older ‘MD5’ format (which looks like `aa:bb:cc:...`), you can choose ‘Show fingerprint as MD5’ from the ‘Key’ menu, but bear in mind that this is less cryptographically secure; it may be feasible for an attacker to create a key with the same fingerprint as yours.

8.2.7 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.

8.2.8 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](#)) 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) 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.

8.2.9 Adding a certificate to your key

In some environments, user authentication keys can be signed in turn by a ‘certifying authority’ ('CA' for short), and user accounts on an SSH server can be configured to automatically trust any key that's certified by the right signature.

This can be a convenient setup if you have a very large number of servers. When you change your key pair, you might otherwise have to edit the `authorized_keys` file on every server individually, to make them all accept the new key. But if instead you configure all those servers *once* to accept keys signed as yours by a CA, then when you change your public key, all you have to do is to get the new key certified by the same CA as before, and then all your servers will automatically accept it without needing individual reconfiguration.

To get your key signed by a CA, you'll probably send the CA the new *public* key (not the private half), and get back a modified version of the public key with the certificate included.

If you want to incorporate the certificate into your PPK file for convenience, you can use the ‘Add certificate to key’ menu option in PuTTYgen's ‘Key’ menu. This will give you a single file containing your private key and the certificate, which is everything you need to authenticate to a server prepared to accept that certificate.

To remove the certificate again and restore the uncertified PPK file, there's also a ‘Remove certificate from key’ option.

(However, you don't *have* to incorporate the certificate into your PPK file. You can equally well use it separately, via the ‘Certificate to use with the private key’ option in PuTTY itself. See [section 4.22.2](#). It's up to you which you find more convenient.)

When the currently loaded key in PuTTYgen contains a certificate, the large ‘Public key for pasting’ edit box (see [section 8.2.12](#)) is replaced by a button that brings up an information box telling you

about the certificate, such as who it certifies your key as belonging to, when it expires (if ever), and the fingerprint of the CA key that signed it in turn.

8.2.10 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 4.22.1](#)) or tell Pageant to load (see [section 9.2.2](#)).

(You can optionally change some details of the PPK format for your saved key files; see [section 8.2.13](#). But the defaults should be fine for most purposes.)

8.2.11 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 8.2.12](#).) 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](#) 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.

8.2.12 ‘Public key for pasting into OpenSSH authorized_keys file’

The OpenSSH server, among others, requires your public key to be given to it in a one-line format before it will accept authentication with your private key. (SSH-1 servers also used this method.)

The ‘Public key for pasting into OpenSSH 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](#) for general instructions on configuring public-key authentication once you have generated a key.

8.2.13 Parameters for saving key files

Selecting ‘Parameters for saving key files...’ from the ‘Key’ menu lets you adjust some aspects of PPK-format private key files stored on disk. None of these options affect compatibility with SSH servers.

In most cases, it's entirely sensible to leave all of these at their default settings.

- [8.2.13.1 PPK file version](#)
- [8.2.13.2 Options affecting passphrase hashing](#)

8.2.13.1 PPK file version

This defaults to version 3, which is fine for most uses.

You might need to select PPK version 2 if you need your private key file to be loadable in older versions of PuTTY (0.74 and older), or in other tools which do not yet support the version 3 format (which was introduced in 2021).

The version 2 format is less resistant to brute-force decryption, and doesn't support any of the following options to control that.

8.2.13.2 Options affecting passphrase hashing

All of the following options only affect keys saved with passphrases. They control how much work is required to decrypt the key (which happens every time you type its passphrase). This allows you to trade off the cost of legitimate use of the key against the resistance of the encrypted key to password-guessing attacks.

These options only affect PPK version 3.

Key derivation function

The variant of the Argon2 key derivation function to use. You might change this if you consider your exposure to side-channel attacks to be different to the norm.

Memory to use for passphrase hash

The amount of memory needed to decrypt the key, in Kbyte.

Time to use for passphrase hash

Controls how much time is required to attempt decrypting the key. You can either specify an approximate time in milliseconds (on this machine), or explicitly specify a number of hash passes (which is what the time is turned into during encryption).

Parallelism for passphrase hash

Number of parallelisable threads that can be used to decrypt the key. The default, 1, forces the process to run single-threaded, even on machines with multiple cores.

8.2.14 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 8.2.15](#) for information about importing foreign key formats.

8.2.15 Dealing with private keys in other formats

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 some OpenSSH key formats contained 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 8.2.10](#)) - 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.

For OpenSSH there are two options. Modern OpenSSH actually has two formats it uses for storing private keys: an older ('PEM-style') format, and a newer 'native' format with better resistance to passphrase guessing and support for comments. 'Export OpenSSH key' will automatically choose the oldest format supported for the key type, for maximum backward compatibility with older versions of OpenSSH; for newer key types like Ed25519, it will use the newer format as that is the only legal option. If you have some specific reason for wanting to use OpenSSH's newer format even for RSA, DSA, or ECDSA keys – for instance, you know your file will only be used by OpenSSH 6.5 or newer (released in 2014), and want the

extra security – you can choose ‘Export OpenSSH key (force new file format)’.

Most clients for the older SSH-1 protocol 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. Hence, the export options are not available if you have generated an SSH-1 key.

8.2.16 PuTTYgen command-line configuration

PuTTYgen supports a set of command-line options to configure many of the same settings you can select in the GUI. This allows you to start it up with your own preferences ready-selected, which might be useful if you generate a lot of keys. (For example, you could make a Windows shortcut that runs PuTTYgen with some command line options, or a batch file or Powershell script that you could distribute to a whole organisation containing your local standards.)

The options supported on the command line are:

-t *keytype*

Type of key to generate. You can select rsa, dsa, ecdsa, eddsa, ed25519, ed448, or rsa1. See [section 8.2.2](#).

-b *bits*

Size of the key to generate, in bits. See [section 8.2.3](#).

--primes *method*

Method for generating prime numbers. You can select probable, proven, and proven-even. See [section 8.2.4](#).

--strong-rsa

When generating an RSA key, make sure the prime factors of the key modulus are ‘strong primes’. See [section 8.2.4](#).

--ppk-param *key=value*, ...

Allows setting all the same details of the PPK save file format described in [section 8.2.13](#).

Aspects to change are specified as a series of *key=value* pairs separated by commas. The keys are:

version

The PPK format version: either 3 or 2.

`kdf`

The variant of Argon2 to use: argon2id, argon2i, and argon2d.

`memory`

The amount of memory needed to decrypt the key, in Kbyte.

`time`

Specifies how much time is required to attempt decrypting the key, in milliseconds.

`passes`

Alternative to `time`: specifies the number of hash passes required to attempt decrypting the key.

`parallelism`

Number of parallelisable threads that can be used to decrypt the key.

`-E fptype`

Algorithm to use when displaying key fingerprints. You can select sha256 or md5. See [section 8.2.6](#).

```
<code>chmod go-w $HOME $HOME/.ssh  
$HOME/.ssh/authorized_keys
```

</code>

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 4.22.1](#) for details.
- Specify the key file on the command line with the `-i` option. See [section 3.11.3.18](#) for details.
- Load the private key into Pageant (see [chapter 9](#)). 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](#)
- [9.2 The Pageant main window](#)
 - [9.2.1 The key list box](#)
 - [9.2.2 The 'Add Key' button](#)
 - [9.2.3 The 'Remove Key' button](#)
- [9.3 The Pageant command line](#)
 - [9.3.1 Making Pageant automatically load keys on startup](#)
 - [9.3.2 Making Pageant run another program](#)
 - [9.3.3 Integrating with Windows OpenSSH](#)
 - [9.3.4 Unix-domain sockets: integrating with WSL 1](#)
 - [9.3.5 Starting with the key list visible](#)
 - [9.3.6 Restricting the Windows process ACL](#)
- [9.4 Using agent forwarding](#)
- [9.5 Loading keys without decrypting them](#)
- [9.6 Security considerations](#)

9.1 Getting started with Pageant

Before you run Pageant, you need to have a private key in *.PPK format. See [chapter 8](#) 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. (You may need to use Windows' 'Show hidden icons' arrow to see the Pageant icon.) 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 4.21.4](#) and [section 3.11.3.9](#) 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.

If you want Pageant to stay running but forget all the keys it has acquired, select ‘Remove All Keys’ from the System tray menu.

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](#)
- [9.2.2 The ‘Add Key’ button](#)
- [9.2.3 The ‘Remove Key’ button](#)

```
<code>Ed25519  
SHA256:TddlQk20DVs4LRcAsIfDN9pInKpY06D+  
h4kSHwWAj4w RSA 2048  
SHA256:8DFtyHm3kQihgy52nzX96qMcEVOq7/yJ  
mmwQQhBWYFg </code>
```

For each key, the list box will tell you:

- The type of the key. Currently, this can be ‘RSA’ (an RSA key for use with the SSH-2 protocol), ‘DSA’ (a DSA key for use with the SSH-2 protocol), ‘NIST’ (an ECDSA key for use with the SSH-2 protocol), ‘Ed25519’ (an Ed25519 key for use with the SSH-2 protocol), ‘Ed448’ (an Ed448 key for use with the SSH-2 protocol), or ‘SSH-1’ (an RSA key for use with the old SSH-1 protocol). (If the key has an associated certificate, this is shown here with a ‘cert’ suffix.)
- The size (in bits) of the key, for key types that come in different sizes. (For ECDSA ‘NIST’ keys, this is indicated as ‘p256’ or ‘p384’ or ‘p521’.)
- 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.

For SSH-2 keys, by default this is shown in the ‘SHA256’ format. You can change to the older ‘MD5’ format (which looks like aa:bb:cc:....) with the ‘Fingerprint type’ drop-down, but bear in mind that this format is less secure and should be avoided for comparison purposes where possible.

If some of the keys loaded into Pageant have certificates attached, then Pageant will default to showing the fingerprint of the underlying key. This way, a certified and uncertified version of the same key will have the same fingerprint, so you can see that they match. You can instead use the ‘Fingerprint type’ drop-down to ask for a different fingerprint to be shown for certified keys, which includes the certificate as part of the fingerprinted data. That way you can tell two certificates apart.

- The comment attached to the key.

- The state of deferred decryption, if enabled for this key. See [section 9.5](#).

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](#) 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](#)); 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](#)
- [9.3.2 Making Pageant run another program](#)
- [9.3.3 Integrating with Windows OpenSSH](#)
- [9.3.4 Unix-domain sockets: integrating with WSL 1](#)
- [9.3.5 Starting with the key list visible](#)
- [9.3.6 Restricting the Windows process ACL](#)

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.

You can specify the '--encrypted' option to defer decryption of these keys; see [section 9.5](#).

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
```

9.3.3 Integrating with Windows OpenSSH

Windows's own port of OpenSSH uses the same mechanism as Pageant to talk to its SSH agent (Windows named pipes). This means that Windows OpenSSH can talk directly to Pageant, if it knows where to find Pageant's named pipe.

When Pageant starts up, it can optionally write out a file containing an OpenSSH configuration directive that tells the Windows ssh.exe where to find Pageant. If you include this file from your Windows SSH configuration, then ssh.exe should automatically use Pageant as its agent, so that you can keep your keys in one place and have both SSH clients able to use them.

The option is `--openssh-config`, and you follow it with a filename.

To refer to this file from your main OpenSSH configuration, you can use the ‘Include’ directive. For example, you might run Pageant like this (with your own username substituted, of course):

```
pageant --openssh-config C:\Users\Simon\.ssh\pageant.conf
```

and then add a directive like this to your main ‘.ssh\config’ file (assuming that lives in the same directory that you just put pageant.conf):

```
Include pageant.conf
```

Note: this technique only works with *Windows*'s port of OpenSSH, which lives at `C:\Windows\System32\OpenSSH\ssh.exe` if you have it installed. (If not, it can be installed as a Windows optional feature, e.g., via Settings > Apps & features > Optional features > Add a feature > OpenSSH Client.) There are other versions of OpenSSH for Windows, notably the one that comes with Windows git. Those will likely not work with the same configuration, because they tend to depend on Unix emulation layers like MinGW or MSys, so they won't speak Windows native pathname syntax or understand named

pipes. The above instructions will only work with Windows's own version of OpenSSH.

So, if you want to use Windows git with an SSH key held in Pageant, you'll have to set the environment variable `GIT_SSH`, to point at a different program. You could point it at `c:\Windows\System32\OpenSSH\ssh.exe` once you've done this setup – but it's just as easy to point it at Plink!

9.3.4 Unix-domain sockets: integrating with WSL 1

Pageant can listen on the WinSock implementation of ‘Unix-domain sockets’. These interoperate with the Unix-domain sockets found in the original Windows Subsystem for Linux (now known as WSL 1). So if you ask Pageant to listen on one of these, then your WSL 1 processes can talk directly to Pageant.

To configure this, run Pageant with the option `--unix`, followed with a pathname. Then, in WSL 1, set the environment variable `SSH_AUTH_SOCK` to point at the WSL translation of that pathname.

For example, you might run

```
pageant --unix C:\Users\Simon\.ssh\agent.sock
```

and in WSL 1, set the environment variable

```
SSH_AUTH_SOCK=/mnt/c/Users/Simon/.ssh/agent.sock
```

Alternatively, you can add a line to your `.ssh/config` file inside WSL that says

```
IdentityAgent /mnt/c/Users/Simon/.ssh/agent.sock
```

although doing it like that may mean that `ssh-add` commands won't find the agent, even though `ssh` itself will.

Security note: Unix-domain sockets are protected against access by other users by the file protections on their containing directory. So if your Windows machine is multiuser, make sure you create the socket inside a directory that other users can't access at all. (In fact, that's a good idea on general principles.)

Compatibility note: WSL 2 processes cannot talk to Pageant by this mechanism, because WSL 2's Unix-domain sockets are managed by

a separate Linux kernel, and not by the same kernel that WinSock talks to.

9.3.5 Starting with the key list visible

Start Pageant with the `--keylist` option to show the main window as soon as it starts up.

9.3.6 Restricting the Windows process ACL

Pageant supports the same `-restrict-acl` option as the other PuTTY utilities to lock down the Pageant process's access control; see [section 3.11.3.28](#) for why you might want to do this.

By default, if Pageant is started with `-restrict-acl`, it won't pass this to any PuTTY sessions started from its System Tray submenu. Use `-restrict-putty-acl` to change this. (Again, see [section 3.11.3.28](#) for details.)

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, whether agent forwarding in SSH-2 is available depends on your server. Pageant's protocol is compatible with the OpenSSH server, but 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 4.21.7](#)). Open the session as normal. (Alternatively, you can use the `-A` command line option; see [section 3.11.3.10](#) 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 Loading keys without decrypting them

You can add keys to Pageant *without* decrypting them. The key file will be held in Pageant's memory still encrypted, and when a client program first tries to use the key, Pageant will display a dialog box prompting for the passphrase so that the key can be decrypted.

This works the same way whether the key is used by an instance of PuTTY running locally, or a remote client connecting to Pageant through agent forwarding.

To add a key to Pageant in this encrypted form, press the 'Add Key (encrypted)' button in the Pageant main window, or alternatively right-click on the Pageant icon in the system tray and select 'Add Key (encrypted)' from there. Pageant will bring up a file dialog, in just the same way as it would for the plain 'Add Key' button. But it won't ask for a passphrase. Instead, the key will be listed in the main window with '(encrypted)' after it.

To start Pageant up in the first place with encrypted keys loaded into it, you can use the '--encrypted' option on the command line. For example:

```
C:\PuTTY\pageant.exe --encrypted d:\main.ppk
```

After a key has been decrypted for the first use, it remains decrypted, so that it can be used again. The main window will list the key with '(re-encryptable)' after it. You can revert it to the previous state, where a passphrase is required, using the 'Re-encrypt' button in the Pageant main window.

You can also 're-encrypt' all keys that were added encrypted by choosing 'Re-encrypt All Keys' from the System tray menu. (Note that this does *not* discard cleartext keys that were not previously added encrypted!)

CAUTION: When Pageant displays a prompt to decrypt an already-loaded key, it cannot give keyboard focus to the prompt dialog box. As far as I know this is a deliberate defensive measure by Windows, against malicious software. So make sure you click in the prompt window before typing your passphrase, or else the passphrase might be sent to somewhere you didn't want to trust with it!

9.6 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 any of 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.)

Chapter 10: Common error messages

This chapter lists a number of common error messages which PuTTY and its associated tools can produce, and explains what they mean in more detail.

We do not attempt to list *all* error messages here: there are many which should never occur, and some which should be self-explanatory. If you get an error message which is not listed in this chapter and which you don't understand, report it to us as a bug (see [appendix B](#)) and we will add documentation for it.

- [10.1 ‘The host key is not cached for this server’](#)
- [10.2 ‘WARNING - POTENTIAL SECURITY BREACH!’](#)
- [10.3 ‘This server presented a certified host key which was signed by a different certification authority ...’](#)
- [10.4 ‘SSH protocol version 2 required by our configuration but remote only provides \(old, insecure\) SSH-1’](#)
- [10.5 ‘The first cipher supported by the server is ... below the configured warning threshold’](#)
- [10.6 ‘Remote side sent disconnect message type 2 \(protocol error\): "Too many authentication failures for root"’](#)
- [10.7 ‘Out of memory’](#)
- [10.8 ‘Internal error’, ‘Internal fault’, ‘Assertion failed’](#)
- [10.9 ‘Unable to use key file’, ‘Couldn’t load private key’, ‘Couldn’t load this key’](#)
- [10.10 ‘Server refused our key’, ‘Server refused our public key’, ‘Key refused’](#)
- [10.11 ‘Access denied’, ‘Authentication refused’](#)
- [10.12 ‘No supported authentication methods available’](#)
- [10.13 ‘Incorrect MAC received on packet’ or ‘Incorrect CRC received on packet’](#)
- [10.14 ‘Incoming packet was garbled on decryption’](#)
- [10.15 ‘PuTTY X11 proxy: various errors’](#)

- [10.16 ‘Network error: Software caused connection abort’](#)
- [10.17 ‘Network error: Connection reset by peer’](#)
- [10.18 ‘Network error: Connection refused’](#)
- [10.19 ‘Network error: Connection timed out’](#)
- [10.20 ‘Network error: Cannot assign requested address’](#)

10.1 ‘The host key is not cached for this server’

This error message occurs when PuTTY connects to a new SSH server. Every server identifies itself by means of a host key; once PuTTY knows the host key for a server, it will be able to detect if a malicious attacker redirects your connection to another machine.

If you see this message, it means that PuTTY has not seen this host key before, and has no way of knowing whether it is correct or not. You should attempt to verify the host key by other means, such as asking the machine's administrator.

If you see this message and you know that your installation of PuTTY *has* connected to the same server before, it may have been recently upgraded to SSH protocol version 2. SSH protocols 1 and 2 use separate host keys, so when you first use SSH-2 with a server you have only used SSH-1 with before, you will see this message again. You should verify the correctness of the key as before.

See [section 2.2](#) for more information on host keys.

10.2 ‘WARNING - POTENTIAL SECURITY BREACH!’

This message, followed by ‘The server's host key does not match the one PuTTY has cached for this server’, means that PuTTY has connected to the SSH server before, knows what its host key *should* be, but has found a different one.

(If the message instead talks about a ‘certified host key’, see instead [section 10.3](#).)

This may mean that a malicious attacker has replaced your server with a different one, or has redirected your network connection to their own machine. On the other hand, it may simply mean that the administrator of your server has accidentally changed the key while upgrading the SSH software; this *shouldn't* happen but it is unfortunately possible.

You should contact your server's administrator and see whether they expect the host key to have changed. If so, verify the new host key in the same way as you would if it was new.

See [section 2.2](#) for more information on host keys.

10.3 ‘This server presented a certified host key which was signed by a different certification authority ...’

If you've configured PuTTY to trust at least one certification authority for signing host keys (see [section 4.19.4](#)), then it will ask the SSH server to send it any available certified host keys. If the server sends back a certified key signed by a *different* certification authority, PuTTY will present this variant of the host key prompt, preceded by ‘WARNING - POTENTIAL SECURITY BREACH!’

One reason why this can happen is a deliberate attack. Just like an ordinary man-in-the-middle attack which substitutes a wrong host key, a particularly ambitious attacker might substitute an entire wrong certification authority, and hope that you connect anyway.

But it's also possible in some situations that this error might arise legitimately. For example, if your organisation's IT department has just rolled out a new CA key which you haven't yet entered in PuTTY's configuration, or if your CA configuration involves two overlapping domains, or something similar.

So, unfortunately, you'll have to work out what to do about it yourself: make an exception for this specific case, or abandon this connection and install a new CA key before trying again (if you're really sure you trust the CA), or edit your configuration in some other way, or just stop trying to use this server.

If you're convinced that this particular server is legitimate even though the CA is not one you trust, PuTTY will let you cache the certified host key, treating it in the same way as an uncertified one. Then that particular certificate will be accepted for future connections to this specific server, even though other certificates signed by the same CA will still be rejected.

10.4 ‘SSH protocol version 2 required by our configuration but remote only provides (old, insecure) SSH-1’

By default, PuTTY only supports connecting to SSH servers that implement SSH protocol version 2. If you see this message, the server you're trying to connect to only supports the older SSH-1 protocol.

If the server genuinely only supports SSH-1, then you need to either change the ‘SSH protocol version’ setting (see [section 4.17.4](#)), or use the `-1` command-line option; in any case, you should not treat the resulting connection as secure.

You might start seeing this message with new versions of PuTTY (from 0.68 onwards) where you didn't before, because it used to be possible to configure PuTTY to automatically fall back from SSH-2 to SSH-1. This is no longer supported, to prevent the possibility of a downgrade attack.

10.5 ‘The first cipher supported by the server is ... below the configured warning threshold’

This occurs when the SSH server does not offer any ciphers which you have configured PuTTY to consider strong enough. By default, PuTTY puts up this warning only for Blowfish, single-DES, and Arcfour encryption.

See [section 4.20](#) for more information on this message.

(There are similar messages for other cryptographic primitives, such as host key algorithms.)

10.6 ‘Remote side sent disconnect message type 2 (protocol error): "Too many authentication failures for root"

This message is produced by an OpenSSH (or Sun SSH) server if it receives more failed authentication attempts than it is willing to tolerate.

This can easily happen if you are using Pageant and have a large number of keys loaded into it, since these servers count each offer of a public key as an authentication attempt. This can be worked around by specifying the key that's required for the authentication in the PuTTY configuration (see [section 4.22.1](#)); PuTTY will ignore any other keys Pageant may have, but will ask Pageant to do the authentication, so that you don't have to type your passphrase.

On the server, this can be worked around by disabling public-key authentication or (for Sun SSH only) by increasing `MaxAuthTries` in `sshd_config`.

10.7 ‘Out of memory’

This occurs when PuTTY tries to allocate more memory than the system can give it. This *may* happen for genuine reasons: if the computer really has run out of memory, or if you have configured an extremely large number of lines of scrollback in your terminal.

PuTTY is not able to recover from running out of memory; it will terminate immediately after giving this error.

However, this error can also occur when memory is not running out at all, because PuTTY receives data in the wrong format. In SSH-2 and also in SFTP, the server sends the length of each message before the message itself; so PuTTY will receive the length, try to allocate space for the message, and then receive the rest of the message. If the length PuTTY receives is garbage, it will try to allocate a ridiculous amount of memory, and will terminate with an ‘Out of memory’ error.

This can happen in SSH-2, if PuTTY and the server have not enabled encryption in the same way (see [question A.7.3](#) in the FAQ).

This can also happen in PSCP or PSFTP, if your login scripts on the server generate output: the client program will be expecting an SFTP message starting with a length, and if it receives some text from your login scripts instead it will try to interpret them as a message length. See [question A.7.4](#) for details of this.

10.8 ‘Internal error’, ‘Internal fault’, ‘Assertion failed’

Any error beginning with the word ‘Internal’ should *never* occur. If it does, there is a bug in PuTTY by definition; please see [appendix B](#) and report it to us.

Similarly, any error message starting with ‘Assertion failed’ is a bug in PuTTY. Please report it to us, and include the exact text from the error message box.

10.9 ‘Unable to use key file’, ‘Couldn’t load private key’, ‘Couldn’t load this key’

Various forms of this error are printed in the PuTTY window, or written to the PuTTY Event Log (see [section 3.1.3.1](#)) when trying public-key authentication, or given by Pageant when trying to load a private key.

If you see one of these messages, it often indicates that you've tried to load a key of an inappropriate type into PuTTY, Plink, PSCP, PSFTP, or Pageant.

You may have tried to load an SSH-2 key in a ‘foreign’ format (OpenSSH or ssh.com) directly into one of the PuTTY tools, in which case you need to import it into PuTTY's native format (*.PPK) using PuTTYgen – see [section 8.2.15](#).

Alternatively, you may have specified a key that's inappropriate for the connection you're making. The SSH-2 and the old SSH-1 protocols require different private key formats, and a SSH-1 key can't be used for a SSH-2 connection (or vice versa).

10.10 ‘Server refused our key’, ‘Server refused our public key’, ‘Key refused’

Various forms of this error are printed in the PuTTY window, or written to the PuTTY Event Log (see [section 3.1.3.1](#)) when trying public-key authentication.

If you see one of these messages, it means that PuTTY has sent a public key to the server and offered to authenticate with it, and the server has refused to accept authentication. This usually means that the server is not configured to accept this key to authenticate this user.

This is almost certainly not a problem with PuTTY. If you see this type of message, the first thing you should do is check your server configuration carefully. Common errors include having the wrong permissions or ownership set on the public key or the user's home directory on the server. Also, read the PuTTY Event Log; the server may have sent diagnostic messages explaining exactly what problem it had with your setup.

[Section 8.3](#) has some hints on server-side public key setup.

10.11 ‘Access denied’, ‘Authentication refused’

Various forms of this error are printed in the PuTTY window, or written to the PuTTY Event Log (see [section 3.1.3.1](#)) during authentication.

If you see one of these messages, it means that the server has refused all the forms of authentication PuTTY has tried and it has no further ideas.

It may be worth checking the Event Log for diagnostic messages from the server giving more detail.

This error can be caused by buggy SSH-1 servers that fail to cope with the various strategies we use for camouflaging passwords in transit. Upgrade your server, or use the workarounds described in [section 4.27.14](#) and possibly [section 4.27.15](#).

10.12 ‘No supported authentication methods available’

This error indicates that PuTTY has run out of ways to authenticate you to an SSH server. This may be because PuTTY has TIS or keyboard-interactive authentication disabled, in which case see [section 4.21.5](#) and [section 4.21.6](#).

10.13 ‘Incorrect MAC received on packet’ or ‘Incorrect CRC received on packet’

This error occurs when PuTTY decrypts an SSH packet and its checksum is not correct. This probably means something has gone wrong in the encryption or decryption process. It's difficult to tell from this error message whether the problem is in the client, in the server, or in between.

In particular, if the network is corrupting data at the TCP level, it may only be obvious with cryptographic protocols such as SSH, which explicitly check the integrity of the transferred data and complain loudly if the checks fail. Corruption of protocols without integrity protection (such as HTTP) will manifest in more subtle failures (such as misdisplayed text or images in a web browser) which may not be noticed.

Occasionally this has been caused by server bugs. An example is the bug described at [section 4.27.11](#), although you're very unlikely to encounter that one these days.

In this context MAC stands for Message Authentication Code. It's a cryptographic term, and it has nothing at all to do with Ethernet MAC (Media Access Control) addresses, or with the Apple computer.

10.14 ‘Incoming packet was garbled on decryption’

This error occurs when PuTTY decrypts an SSH packet and the decrypted data makes no sense. This probably means something has gone wrong in the encryption or decryption process. It's difficult to tell from this error message whether the problem is in the client, in the server, or in between.

If you get this error, one thing you could try would be to fiddle with the setting of ‘Miscomputes SSH-2 encryption keys’ (see [section 4.27.13](#)) or ‘Ignores SSH-2 maximum packet size’ (see [section 4.27.5](#)) on the Bugs panel.

10.15 ‘PuTTY X11 proxy: various errors’

This family of errors are reported when PuTTY is doing X forwarding. They are sent back to the X application running on the SSH server, which will usually report the error to the user.

When PuTTY enables X forwarding (see [section 3.4](#)) it creates a virtual X display running on the SSH server. This display requires authentication to connect to it (this is how PuTTY prevents other users on your server machine from connecting through the PuTTY proxy to your real X display). PuTTY also sends the server the details it needs to enable clients to connect, and the server should put this mechanism in place automatically, so your X applications should just work.

A common reason why people see one of these messages is because they used SSH to log in as one user (let's say 'fred'), and then used the Unix su command to become another user (typically 'root'). The original user, 'fred', has access to the X authentication data provided by the SSH server, and can run X applications which are forwarded over the SSH connection. However, the second user ('root') does not automatically have the authentication data passed on to it, so attempting to run an X application as that user often fails with this error.

If this happens, *it is not a problem with PuTTY*. You need to arrange for your X authentication data to be passed from the user you logged in as to the user you used su to become. How you do this depends on your particular system; in fact many modern versions of su do it automatically.

10.16 ‘Network error: Software caused connection abort’

This is a generic error produced by the Windows network code when it kills an established connection for some reason. For example, it might happen if you pull the network cable out of the back of an Ethernet-connected computer, or if Windows has any other similar reason to believe the entire network has become unreachable.

Windows also generates this error if it has given up on the machine at the other end of the connection ever responding to it. If the network between your client and server goes down and your client then tries to send some data, Windows will make several attempts to send the data and will then give up and kill the connection. In particular, this can occur even if you didn't type anything, if you are using SSH-2 and PuTTY attempts a key re-exchange. (See [section 4.18.2](#) for more about key re-exchange.)

(It can also occur if you are using keepalives in your connection. Other people have reported that keepalives *fix* this error for them. See [section 4.14.1](#) for a discussion of the pros and cons of keepalives.)

We are not aware of any reason why this error might occur that would represent a bug in PuTTY. The problem is between you, your Windows system, your network and the remote system.

10.17 ‘Network error: Connection reset by peer’

This error occurs when the machines at each end of a network connection lose track of the state of the connection between them. For example, you might see it if your SSH server crashes, and manages to reboot fully before you next attempt to send data to it.

However, the most common reason to see this message is if you are connecting through a firewall or a NAT router which has timed the connection out. See [question A.7.8](#) in the FAQ for more details. You may be able to improve the situation by using keepalives; see [section 4.14.1](#) for details on this.

Note that Windows can produce this error in some circumstances without seeing a connection reset from the server, for instance if the connection to the network is lost.

10.18 ‘Network error: Connection refused’

This error means that the network connection PuTTY tried to make to your server was rejected by the server. Usually this happens because the server does not provide the service which PuTTY is trying to access.

Check that you are connecting with the correct protocol (SSH, Telnet, etc), and check that the port number is correct. If that fails, consult the administrator of your server.

This error can also be caused by a firewall in between you and the server, which rejects the connection and sends back the same type of error packet as the real server would have sent.

10.19 ‘Network error: Connection timed out’

This error means that the network connection PuTTY tried to make to your server received no response at all from the server. Usually this happens because the server machine is completely isolated from the network, or because it is turned off.

Check that you have correctly entered the host name or IP address of your server machine. If that fails, consult the administrator of your server.

Unix also generates this error when it tries to send data down a connection and contact with the server has been completely lost during a connection. (There is a delay of minutes before Unix gives up on receiving a reply from the server.) This can occur if you type things into PuTTY while the network is down, but it can also occur if PuTTY decides of its own accord to send data: due to a repeat key exchange in SSH-2 (see [section 4.18.2](#)) or due to keepalives ([section 4.14.1](#)).

10.20 ‘Network error: Cannot assign requested address’

This means that the operating system rejected the parameters of the network connection PuTTY tried to make, usually without actually trying to connect to anything, because they were simply invalid.

A common way to provoke this error is to accidentally try to connect to port 0, which is not a valid port number.

Appendix A: PuTTY FAQ

This FAQ is published on the PuTTY web site, and also provided as an appendix in the manual.

- [A.1 Introduction](#)
 - [A.1.1 What is PuTTY?](#)
- [A.2 Features supported in PuTTY](#)
 - [A.2.1 Does PuTTY support SSH-2?](#)
 - [A.2.2 Does PuTTY support reading OpenSSH or ssh.com SSH-2 private key files?](#)
 - [A.2.3 Does PuTTY support SSH-1?](#)
 - [A.2.4 Does PuTTY support local echo?](#)
 - [A.2.5 Does PuTTY support storing settings, so I don't have to change them every time?](#)
 - [A.2.6 Does PuTTY support storing its settings in a disk file?](#)
 - [A.2.7 Does PuTTY support full-screen mode, like a DOS box?](#)
 - [A.2.8 Does PuTTY have the ability to remember my password so I don't have to type it every time?](#)
 - [A.2.9 Is there an option to turn off the annoying host key prompts?](#)
 - [A.2.10 Will you write an SSH server for the PuTTY suite, to go with the client?](#)
 - [A.2.11 Can PSCP or PSFTP transfer files in ASCII mode?](#)
- [A.3 Ports to other operating systems](#)
 - [A.3.1 What ports of PuTTY exist?](#)
 - [A.3.2 Is there a port to Unix?](#)
 - [A.3.3 What's the point of the Unix port? Unix has OpenSSH.](#)
 - [A.3.4 Will there be a port to Windows CE or PocketPC?](#)
 - [A.3.5 Is there a port to Windows 3.1?](#)
 - [A.3.6 Will there be a port to the Mac?](#)
 - [A.3.7 Will there be a port to EPOC?](#)

- [A.3.8 Will there be a port to the iPhone?](#)
- [A.4 Embedding PuTTY in other programs](#)
 - [A.4.1 Is the SSH or Telnet code available as a DLL?](#)
 - [A.4.2 Is the SSH or Telnet code available as a Visual Basic component?](#)
 - [A.4.3 How can I use PuTTY to make an SSH connection from within another program?](#)
- [A.5 Details of PuTTY's operation](#)
 - [A.5.1 What terminal type does PuTTY use?](#)
 - [A.5.2 Where does PuTTY store its data?](#)
 - [A.5.3 Why do small PuTTY icons appear next to the login prompts?](#)
 - [A.5.4 Why has Plink started saying 'Press Return to begin session'?](#)
- [A.6 HOWTO questions](#)
 - [A.6.1 What login name / password should I use?](#)
 - [A.6.2 What commands can I type into my PuTTY terminal window?](#)
 - [A.6.3 How can I make PuTTY start up maximised?](#)
 - [A.6.4 How can I create a Windows shortcut to start a particular saved session directly?](#)
 - [A.6.5 How can I start an SSH session straight from the command line?](#)
 - [A.6.6 How do I copy and paste between PuTTY and other Windows applications?](#)
 - [A.6.7 How do I use all PuTTY's features \(public keys, proxying, cipher selection, etc.\) in PSCP, PSFTP and Plink?](#)
 - [A.6.8 How do I use PSCP.EXE? When I double-click it gives me a command prompt window which then closes instantly.](#)
 - [A.6.9 How do I use PSCP to copy a file whose name has spaces in?](#)
 - [A.6.10 Should I run the 32-bit or the 64-bit version?](#)
- [A.7 Troubleshooting](#)
 - [A.7.1 Why do I see 'Fatal: Protocol error: Expected control record' in PSCP?](#)

- [A.7.2 I clicked on a colour in the Colours panel, and the colour didn't change in my terminal.](#)
- [A.7.3 After trying to establish an SSH-2 connection, PuTTY says 'Out of memory' and dies.](#)
- [A.7.4 When attempting a file transfer, either PSCP or PSFTP says 'Out of memory' and dies.](#)
- [A.7.5 PSFTP transfers files much slower than PSCP.](#)
- [A.7.6 When I run full-colour applications, I see areas of black space where colour ought to be, or vice versa.](#)
- [A.7.7 When I change some terminal settings, nothing happens.](#)
- [A.7.8 My PuTTY sessions unexpectedly close after they are idle for a while.](#)
- [A.7.9 PuTTY's network connections time out too quickly when network connectivity is temporarily lost.](#)
- [A.7.10 When I cat a binary file, I get 'PuTTYPuTTYPuTTY' on my command line.](#)
- [A.7.11 When I cat a binary file, my window title changes to a nonsense string.](#)
- [A.7.12 My keyboard stops working once PuTTY displays the password prompt.](#)
- [A.7.13 One or more function keys don't do what I expected in a server-side application.](#)
- [A.7.14 Why do I see 'Couldn't load private key from ...'? Why can PuTTYgen load my key but not PuTTY?](#)
- [A.7.15 When I'm connected to a Red Hat Linux 8.0 system, some characters don't display properly.](#)
- [A.7.16 Since I upgraded to PuTTY 0.54, the scrollback has stopped working when I run screen.](#)
- [A.7.17 Since I upgraded Windows XP to Service Pack 2, I can't use addresses like 127.0.0.2.](#)
- [A.7.18 PSFTP commands seem to be missing a directory separator \(slash\).](#)
- [A.7.19 Do you want to hear about 'Software caused connection abort'?](#)
- [A.7.20 My SSH-2 session locks up for a few seconds every so often.](#)

- [A.7.21 PuTTY fails to start up. Windows claims that ‘the application configuration is incorrect’.](#)
 - [A.7.22 When I put 32-bit PuTTY in c:\WINDOWS\SYSTEM32 on my 64-bit Windows system, ‘Duplicate Session’ doesn’t work.](#)
 - [A.7.23 After I upgraded PuTTY to 0.68, I can no longer connect to my embedded device or appliance.](#)
 - [A.7.24 Since 0.78, I can’t find where to configure my SSH private key.](#)
- [A.8 Security questions](#)
 - [A.8.1 Is it safe for me to download PuTTY and use it on a public PC?](#)
 - [A.8.2 What does PuTTY leave on a system? How can I clean up after it?](#)
 - [A.8.3 How come PuTTY now supports DSA, when the website used to say how insecure it was?](#)
 - [A.8.4 Couldn’t Pageant use virtualLock\(\) to stop private keys being written to disk?](#)
 - [A.8.5 Is the version of PuTTY in the Microsoft Store legit?](#)
- [A.9 Administrative questions](#)
 - [A.9.1 Is putty.org your website?](#)
 - [A.9.2 Would you like me to register you a nicer domain name?](#)
 - [A.9.3 Would you like free web hosting for the PuTTY web site?](#)
 - [A.9.4 Would you link to my web site from the PuTTY web site?](#)
 - [A.9.5 Why don’t you move PuTTY to SourceForge?](#)
 - [A.9.6 Why can’t I subscribe to the putty-bugs mailing list?](#)
 - [A.9.7 If putty-bugs isn’t a general-subscription mailing list, what is?](#)
 - [A.9.8 How can I donate to PuTTY development?](#)
 - [A.9.9 Can I have permission to put PuTTY on a cover disk / distribute it with other software / etc?](#)
 - [A.9.10 Can you sign an agreement indemnifying us against security problems in PuTTY?](#)

- [A.9.11 Can you sign this form granting us permission to use/distribute PuTTY?](#)
 - [A.9.12 Can you write us a formal notice of permission to use PuTTY?](#)
 - [A.9.13 Can you sign *anything* for us?](#)
 - [A.9.14 If you won't sign *anything*, can you give us some sort of assurance that you won't make PuTTY closed-source in future?](#)
 - [A.9.15 Can you provide us with export control information / FIPS certification for PuTTY?](#)
 - [A.9.16 As one of our existing software vendors, can you just fill in this questionnaire for us?](#)
 - [A.9.17 The sha1sums / sha256sums / etc files on your download page don't match the binaries.](#)
- [A.10 Miscellaneous questions](#)
 - [A.10.1 Is PuTTY a port of OpenSSH, or based on OpenSSH or OpenSSL?](#)
 - [A.10.2 Where can I buy silly_putty?](#)
 - [A.10.3 What does 'PuTTY' mean?](#)
 - [A.10.4 How do I pronounce 'PuTTY'?](#)

A.1 Introduction

- [A.1.1 What is PuTTY?](#)

A.1.1 What is PuTTY?

PuTTY is a client program for the SSH, Telnet, Rlogin, and SUPDUP network protocols.

These protocols are all used to run a remote session on a computer, over a network. PuTTY implements the client end of that session: the end at which the session is displayed, rather than the end at which it runs.

In really simple terms: you run PuTTY on a Windows machine, and tell it to connect to (for example) a Unix machine. PuTTY opens a window. Then, anything you type into that window is sent straight to the Unix machine, and everything the Unix machine sends back is displayed in the window. So you can work on the Unix machine as if you were sitting at its console, while actually sitting somewhere else.

A.2 Features supported in PuTTY

In general, if you want to know if PuTTY supports a particular feature, you should look for it on the [PuTTY web site](#). In particular:

- try the [changes page](#), and see if you can find the feature on there. If a feature is listed there, it's been implemented. If it's listed as a change made *since* the latest version, it should be available in the development snapshots, in which case testing will be very welcome.
- try the [Wishlist page](#), and see if you can find the feature there. If it's on there, and not in the 'Recently fixed' section, it probably *hasn't* been implemented.
- [A.2.1 Does PuTTY support SSH-2?](#)
- [A.2.2 Does PuTTY support reading OpenSSH or ssh.com SSH-2 private key files?](#)
- [A.2.3 Does PuTTY support SSH-1?](#)
- [A.2.4 Does PuTTY support local echo?](#)
- [A.2.5 Does PuTTY support storing settings, so I don't have to change them every time?](#)
- [A.2.6 Does PuTTY support storing its settings in a disk file?](#)
- [A.2.7 Does PuTTY support full-screen mode, like a DOS box?](#)
- [A.2.8 Does PuTTY have the ability to remember my password so I don't have to type it every time?](#)
- [A.2.9 Is there an option to turn off the annoying host key prompts?](#)
- [A.2.10 Will you write an SSH server for the PuTTY suite, to go with the client?](#)
- [A.2.11 Can PSCP or PSFTP transfer files in ASCII mode?](#)

A.2.1 Does PuTTY support SSH-2?

Yes. SSH-2 support has been available in PuTTY since version 0.50 in 2000.

Public key authentication (both RSA and DSA) in SSH-2 was new in version 0.52 in 2002.

A.2.2 Does PuTTY support reading OpenSSH or ssh.com SSH-2 private key files?

PuTTY doesn't support this natively (see [the wishlist entry](#) for reasons why not), but as of 0.53 PuTTYgen can convert both OpenSSH and ssh.com private key files into PuTTY's format.

A.2.3 Does PuTTY support SSH-1?

Yes. SSH-1 support has always been available in PuTTY.

However, the SSH-1 protocol has many weaknesses and is no longer considered secure; you should use SSH-2 instead if at all possible.

As of 0.68, PuTTY will no longer fall back to SSH-1 if the server doesn't appear to support SSH-2; you must explicitly ask for SSH-1.

A.2.4 Does PuTTY support local echo?

Yes. Version 0.52 has proper support for local echo.

In version 0.51 and before, local echo could not be separated from local line editing (where you type a line of text locally, and it is not sent to the server until you press Return, so you have the chance to edit it and correct mistakes *before* the server sees it). New in version 0.52, local echo and local line editing are separate options, and by default PuTTY will try to determine automatically whether to enable them or not, based on which protocol you have selected and also based on hints from the server. If you have a problem with PuTTY's default choice, you can force each option to be enabled or disabled as you choose. The controls are in the Terminal panel, in the section marked 'Line discipline options'.

A.2.5 Does PuTTY support storing settings, so I don't have to change them every time?

Yes, all of PuTTY's settings can be saved in named session profiles. You can also change the default settings that are used for new sessions. See [section 4.1.2](#) in the documentation for how to do this.

A.2.6 Does PuTTY support storing its settings in a disk file?

Not at present, although [section 4.33](#) in the documentation gives a method of achieving the same effect.

A.2.7 Does PuTTY support full-screen mode, like a DOS box?

Yes; this was added in version 0.52, in 2002.

A.2.8 Does PuTTY have the ability to remember my password so I don't have to type it every time?

No, it doesn't.

Remembering your password is a bad plan for obvious security reasons: anyone who gains access to your machine while you're away from your desk can find out the remembered password, and use it, abuse it or change it.

In addition, it's not even *possible* for PuTTY to automatically send your password in a Telnet session, because Telnet doesn't give the client software any indication of which part of the login process is the password prompt. PuTTY would have to guess, by looking for words like 'password' in the session data; and if your login program is written in something other than English, this won't work.

In SSH, remembering your password would be possible in theory, but there doesn't seem to be much point since SSH supports public key authentication, which is more flexible and more secure. See [chapter 8](#) in the documentation for a full discussion of public key authentication.

A.2.9 Is there an option to turn off the annoying host key prompts?

No, there isn't. And there won't be. Even if you write it yourself and send us the patch, we won't accept it.

Those annoying host key prompts are the *whole point* of SSH. Without them, all the cryptographic technology SSH uses to secure your session is doing nothing more than making an attacker's job slightly harder; instead of sitting between you and the server with a packet sniffer, the attacker must actually subvert a router and start modifying the packets going back and forth. But that's not all that much harder than just sniffing; and without host key checking, it will go completely undetected by client or server.

Host key checking is your guarantee that the encryption you put on your data at the client end is the *same* encryption taken off the data at the server end; it's your guarantee that it hasn't been removed and replaced somewhere on the way. Host key checking makes the attacker's job *astronomically* hard, compared to packet sniffing, and even compared to subverting a router. Instead of applying a little intelligence and keeping an eye on oss-security, the attacker must now perform a brute-force attack against at least one military-strength cipher. That insignificant host key prompt really does make *that* much difference.

If you're having a specific problem with host key checking - perhaps you want an automated batch job to make use of PSCP or Plink, and the interactive host key prompt is hanging the batch process - then the right way to fix it is to add the correct host key to the Registry in advance, or if the Registry is not available, to use the `-hostkey` command-line option. That way, you retain the *important* feature of host key checking: the right key will be accepted and the wrong ones will not. Adding an option to turn host key checking off completely is the wrong solution and we will not do it.

If you have host keys available in the common known_hosts format, we have a script called [`kh2reg.py`](#) to convert them to a Windows .REG file, which can be installed ahead of time by double-clicking or using REGEDIT.

A.2.10 Will you write an SSH server for the PuTTY suite, to go with the client?

Not one that you'd want to use.

While much of the protocol and networking code can be made common between a client and server, to make a *useful* general-purpose server requires all sorts of fiddly new code like interacting with OS authentication databases and the like.

A special-purpose SSH server (called Uppity) can now be built from the PuTTY source code, and indeed it is not usable as a general-purpose server; it exists mainly as a test harness.

If someone else wants to use this as a basis for writing a general-purpose SSH server, they'd be perfectly welcome to of course; but we don't have time, and we don't have motivation. The code is available if anyone else wants to try it.

A.2.11 Can PSCP or PSFTP transfer files in ASCII mode?

Unfortunately not.

This was a limitation of the file transfer protocols as originally specified: the SCP and SFTP protocols had no notion of transferring a file in anything other than binary mode. (This is still true of SCP.)

The current draft protocol spec of SFTP proposes a means of implementing ASCII transfer. At some point PSCP/PSFTP may implement this proposal.

A.3 Ports to other operating systems

The eventual goal is for PuTTY to be a multi-platform program, able to run on at least Windows, Mac OS and Unix.

PuTTY has been gaining a generalised porting layer, drawing a clear line between platform-dependent and platform-independent code. The general intention was for this porting layer to evolve naturally as part of the process of doing the first port; a Unix port has now been released and the plan seems to be working so far.

- [A.3.1 What ports of PuTTY exist?](#)
- [A.3.2 Is there a port to Unix?](#)
- [A.3.3 What's the point of the Unix port? Unix has OpenSSH.](#)
- [A.3.4 Will there be a port to Windows CE or PocketPC?](#)
- [A.3.5 Is there a port to Windows 3.1?](#)
- [A.3.6 Will there be a port to the Mac?](#)
- [A.3.7 Will there be a port to EPOC?](#)
- [A.3.8 Will there be a port to the iPhone?](#)

A.3.1 What ports of PuTTY exist?

Currently, release versions of PuTTY tools only run on Windows systems and Unix.

As of 0.68, the supplied PuTTY executables run on versions of Windows from XP onwards, up to and including Windows 11; and we know of no reason why PuTTY should not continue to work on future versions of Windows. We provide 32-bit and 64-bit Windows executables for the common x86 processor family; see [question A.6.10](#) for discussion of the compatibility issues around that. The 32-bit executables require a Pentium 4 or newer processor. We also provide executables for Windows on Arm processors.

(We used to also provide executables for Windows for the Alpha processor, but stopped after 0.58 due to lack of interest.) In the development code, a partial port to Mac OS exists (see [question A.3.6](#)).

Currently PuTTY does *not* run on Windows CE (see [question A.3.4](#)).

We do not have release-quality ports for any other systems at the present time. If anyone told you we had an Android port, or an iOS port, or any other port of PuTTY, they were mistaken. We don't.

There are some third-party ports to various platforms, mentioned on the [Links page of our website](#).

A.3.2 Is there a port to Unix?

There are Unix ports of most of the traditional PuTTY tools, and also one entirely new application.

If you look at the source release, you should find a `unix` subdirectory. You need `cmake` to build it; see the file `README` in the source distribution. This should build you:

- Unix ports of PuTTY, Plink, PSCP, and PSFTP, which work pretty much the same as their Windows counterparts;
- Command-line versions of PuTTYgen and Pageant, whose user interface is quite different to the Windows GUI versions;
- `pterm` - an `xterm`-type program which supports the same terminal emulation as PuTTY.

If you don't have Gtk, you should still be able to build the command-line tools.

A.3.3 What's the point of the Unix port? Unix has OpenSSH.

All sorts of little things. `pterm` is directly useful to anyone who prefers PuTTY's terminal emulation to `xterm`'s, which at least some people do. Unix Plink has apparently found a niche among people who find the complexity of OpenSSL makes OpenSSH hard to install (and who don't mind Plink not having as many features). Some users want to generate a large number of SSH keys on Unix and then copy them all into PuTTY, and the Unix PuTTYgen should allow them to automate that conversion process.

There were development advantages as well; porting PuTTY to Unix was a valuable path-finding effort for other future ports, and also allowed us to use the excellent Linux tool [Valgrind](#) to help with debugging, which has already improved PuTTY's stability on *all* platforms.

However, if you're a Unix user and you can see no reason to switch from OpenSSH to PuTTY/Plink, then you're probably right. We don't expect our Unix port to be the right thing for everybody.

A.3.4 Will there be a port to Windows CE or PocketPC?

We once did some work on such a port, but it only reached an early stage, and certainly not a useful one. It's no longer being actively worked on.

A.3.5 Is there a port to Windows 3.1?

PuTTY is a 32-bit application from the ground up, so it won't run on Windows 3.1 as a native 16-bit program; and it would be *very* hard to port it to do so, because of Windows 3.1's vile memory allocation mechanisms.

However, it is possible in theory to compile the existing PuTTY source in such a way that it will run under Win32s (an extension to Windows 3.1 to let you run 32-bit programs). In order to do this you'll need the right kind of C compiler - modern versions of Visual C at least have stopped being backwards compatible to Win32s. Also, the last time we tried this it didn't work very well.

A.3.6 Will there be a port to the Mac?

We hope so!

We attempted one around 2005, written as a native Cocoa application, but it turned out to be very slow to redraw its window for some reason we never got to the bottom of.

In 2015, after porting the GTK front end to work with GTK 3, we began another attempt based on making small changes to the GTK code and building it against the OS X Quartz version of GTK 3. This doesn't seem to have the window redrawing problem any more, so it's already got further than the last effort, but it is still substantially unfinished.

If any OS X and/or GTK programming experts are keen to have a finished version of this, we urge them to help out with some of the remaining problems! See the TODO list in `unix/main-gtk-application.c` in the source code.

A.3.7 Will there be a port to EPOC?

I hope so, but given that ports aren't really progressing very fast even on systems the developers *do* already know how to program for, it might be a long time before any of us get round to learning a new system and doing the port for that.

However, some of the work has been done by other people; see the [Links page of our website](#) for various third-party ports.

A.3.8 Will there be a port to the iPhone?

We have no plans to write such a port ourselves; none of us has an iPhone, and developing and publishing applications for it looks awkward and expensive.

However, there is a third-party SSH client for the iPhone and iPod Touch called [pTerm](#), which is apparently based on PuTTY. (This is nothing to do with our similarly-named `pterm`, which is a standalone terminal emulator for Unix systems; see [question A.3.2](#).)

A.4 Embedding PuTTY in other programs

- [A.4.1 Is the SSH or Telnet code available as a DLL?](#)
- [A.4.2 Is the SSH or Telnet code available as a Visual Basic component?](#)
- [A.4.3 How can I use PuTTY to make an SSH connection from within another program?](#)

A.4.1 Is the SSH or Telnet code available as a DLL?

No, it isn't. It would take a reasonable amount of rewriting for this to be possible, and since the PuTTY project itself doesn't believe in DLLs (they make installation more error-prone) none of us has taken the time to do it.

Most of the code cleanup work would be a good thing to happen in general, so if anyone feels like helping, we wouldn't say no.

See also [the wishlist entry](#).

A.4.2 Is the SSH or Telnet code available as a Visual Basic component?

No, it isn't. None of the PuTTY team uses Visual Basic, and none of us has any particular need to make SSH connections from a Visual Basic application. In addition, all the preliminary work to turn it into a DLL would be necessary first; and furthermore, we don't even know how to write VB components.

If someone offers to do some of this work for us, we might consider it, but unless that happens I can't see VB integration being anywhere other than the very bottom of our priority list.

A.4.3 How can I use PuTTY to make an SSH connection from within another program?

Probably your best bet is to use Plink, the command-line connection tool. If you can start Plink as a second Windows process, and arrange for your primary process to be able to send data to the Plink process, and receive data from it, through pipes, then you should be able to make SSH connections from your program.

This is what CVS for Windows does, for example.

A.5 Details of PuTTY's operation

- [A.5.1 What terminal type does PuTTY use?](#)
- [A.5.2 Where does PuTTY store its data?](#)
- [A.5.3 Why do small PuTTY icons appear next to the login prompts?](#)
- [A.5.4 Why has Plink started saying 'Press Return to begin session'?](#)

A.5.1 What terminal type does PuTTY use?

For most purposes, PuTTY can be considered to be an `xterm` terminal.

PuTTY also supports some terminal control sequences not supported by the real `xterm`: notably the Linux console sequences that reconfigure the colour palette, and the title bar control sequences used by DECTerm (which are different from the `xterm` ones; PuTTY supports both).

By default, PuTTY announces its terminal type to the server as `xterm`. If you have a problem with this, you can reconfigure it to say something else; `vt220` might help if you have trouble.

A.5.2 Where does PuTTY store its data?

On Windows, PuTTY stores most of its data (saved sessions, SSH host keys) in the Registry. The precise location is

HKEY_CURRENT_USER\Software\SimonTatham\PuTTY

and within that area, saved sessions are stored under Sessions while host keys are stored under SshHostKeys.

PuTTY also requires a random number seed file, to improve the unpredictability of randomly chosen data needed as part of the SSH cryptography. This is stored by default in a file called PUTTY.RND; this is stored by default in the 'Application Data' directory, or failing that, one of a number of fallback locations. If you want to change the location of the random number seed file, you can put your chosen pathname in the Registry, at

HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\RandSeedFile

You can ask PuTTY to delete all this data; see [question A.8.2](#).

On Unix, PuTTY stores all of this data in a directory `~/.putty` by default.

A.5.3 Why do small PuTTY icons appear next to the login prompts?

As of PuTTY 0.71, some lines of text in the terminal window are marked with a small copy of the PuTTY icon (as far as pixels allow).

This is to show trustworthiness. When the PuTTY icon appears next to a line of text, it indicates that that line of text was generated by PuTTY itself, and not generated by the server and sent to PuTTY.

Text that comes from the server does not have this icon, and we've arranged that the server should not be able to fake it. (There's no control sequence the server can send which will make PuTTY draw its own icon, and if the server tries to move the cursor back up to a line that *already* has an icon and overwrite the text, the icon will disappear.) This lets you tell the difference between (for example) a legitimate prompt in which PuTTY itself asks you for your private key passphrase, and a fake prompt in which the server tries to send the identical text to trick you into telling *it* your private key passphrase.

A.5.4 Why has Plink started saying ‘Press Return to begin session’?

As of PuTTY 0.71, if you use Plink for an interactive SSH session, then after the login phase has finished, it will present a final interactive prompt saying ‘Access granted. Press Return to begin session’.

This is another defence against servers trying to mimic the real authentication prompts after the session has started. When you pass through that prompt, you know that everything after it is generated by the server and not by Plink itself, so any request for your private key passphrase should be treated with suspicion.

In Plink, we can't use the defence described in [section A.5.3](#): Plink is running *in* the terminal, so anything it can write into the terminal, the server could write in the same way after the session starts. And we can't just print a separator line without a pause, because then the server could simply move the cursor back up to it and overwrite it (probably with a brief flicker, but you might easily miss that). The only robust defence anyone has come up with involves this pause.

If you trust your server not to be abusive, you can turn this off. It will also not appear in various other circumstances where Plink can be confident it isn't necessary. See [section 7.2.3.6](#) for details.

A.6 HOWTO questions

- [A.6.1 What login name / password should I use?](#)
- [A.6.2 What commands can I type into my PuTTY terminal window?](#)
- [A.6.3 How can I make PuTTY start up maximised?](#)
- [A.6.4 How can I create a Windows shortcut to start a particular saved session directly?](#)
- [A.6.5 How can I start an SSH session straight from the command line?](#)
- [A.6.6 How do I copy and paste between PuTTY and other Windows applications?](#)
- [A.6.7 How do I use all PuTTY's features \(public keys, proxying, cipher selection, etc.\) in PSCP, PSFTP and Plink?](#)
- [A.6.8 How do I use PSCP.EXE? When I double-click it gives me a command prompt window which then closes instantly.](#)
- [A.6.9 How do I use PSCP to copy a file whose name has spaces in?](#)
- [A.6.10 Should I run the 32-bit or the 64-bit version?](#)

A.6.1 What login name / password should I use?

This is not a question you should be asking *us*.

PuTTY is a communications tool, for making connections to other computers. We maintain the tool; we *don't* administer any computers that you're likely to be able to use, in the same way that the people who make web browsers aren't responsible for most of the content you can view in them. We cannot help with questions of this sort.

If you know the name of the computer you want to connect to, but don't know what login name or password to use, you should talk to whoever administers that computer. If you don't know who that is, see the next question for some possible ways to find out.

A.6.2 What commands can I type into my PuTTY terminal window?

Again, this is not a question you should be asking *us*. You need to read the manuals, or ask the administrator, of *the computer you have connected to*.

PuTTY does not process the commands you type into it. It's only a communications tool. It makes a connection to another computer; it passes the commands you type to that other computer; and it passes the other computer's responses back to you. Therefore, the precise range of commands you can use will not depend on PuTTY, but on what kind of computer you have connected to and what software is running on it. The PuTTY team cannot help you with that.

(Think of PuTTY as being a bit like a telephone. If you phone somebody up and you don't know what language to speak to make them understand you, it isn't *the telephone company's* job to find that out for you. We just provide the means for you to get in touch; making yourself understood is somebody else's problem.)

If you are unsure of where to start looking for the administrator of your server, a good place to start might be to remember how you found out the host name in the PuTTY configuration. If you were given that host name by e-mail, for example, you could try asking the person who sent you that e-mail. If your company's IT department provided you with ready-made PuTTY saved sessions, then that IT department can probably also tell you something about what commands you can type during those sessions. But the PuTTY maintainer team does not administer any server you are likely to be connecting to, and cannot help you with questions of this type.

A.6.3 How can I make PuTTY start up maximised?

Create a Windows shortcut to start PuTTY from, and set it as 'Run Maximized'.

A.6.4 How can I create a Windows shortcut to start a particular saved session directly?

To run a PuTTY session saved under the name ‘mysession’, create a Windows shortcut that invokes PuTTY with a command line like

```
\path\name\to\putty.exe -load "mysession"
```

(Note: prior to 0.53, the syntax was @session. This is now deprecated and may be removed at some point.)

A.6.5 How can I start an SSH session straight from the command line?

Use the command line `putty -ssh host.name`. Alternatively, create a saved session that specifies the SSH protocol, and start the saved session as shown in [question A.6.4](#).

A.6.6 How do I copy and paste between PuTTY and other Windows applications?

Copy and paste works similarly to the X Window System. You use the left mouse button to select text in the PuTTY window. The act of selection *automatically* copies the text to the clipboard: there is no need to press Ctrl-Ins or Ctrl-C or anything else. In fact, pressing Ctrl-C will send a Ctrl-C character to the other end of your connection (just like it does the rest of the time), which may have unpleasant effects. The *only* thing you need to do, to copy text to the clipboard, is to select it.

To paste the clipboard contents into a PuTTY window, by default you click the right mouse button. If you have a three-button mouse and are used to X applications, you can configure pasting to be done by the middle button instead, but this is not the default because most Windows users don't have a middle button at all.

You can also paste by pressing Shift-Ins.

A.6.7 How do I use all PuTTY's features (public keys, proxying, cipher selection, etc.) in PSCP, PSFTP and Plink?

Most major features (e.g., public keys, port forwarding) are available through command line options. See [section 3.11.3](#).

Not all features are accessible from the command line yet, although we'd like to fix this. In the meantime, you can use most of PuTTY's features if you create a PuTTY saved session, and then use the name of the saved session on the command line in place of a hostname. This works for PSCP, PSFTP and Plink (but don't expect port forwarding in the file transfer applications!).

A.6.8 How do I use PSCP.EXE? When I double-click it gives me a command prompt window which then closes instantly.

PSCP is a command-line application, not a GUI application. If you run it without arguments, it will simply print a help message and terminate.

To use PSCP properly, run it from a Command Prompt window. See [chapter 5](#) in the documentation for more details.

A.6.9 How do I use PSCP to copy a file whose name has spaces in?

If PSCP is using the newer SFTP protocol (which is usual with most modern servers), this is straightforward; all filenames with spaces in are specified using a single pair of quotes in the obvious way:

```
pscp "local file" user@host:  
pscp user@host:"remote file" .
```

However, if PSCP is using the older SCP protocol for some reason, things are more confusing. If you're specifying a file at the local end, you just use one set of quotes as you would normally do:

```
pscp "local filename with spaces" user@host:  
pscp user@host:myfile "local filename with spaces"
```

But if the filename you're specifying is on the *remote* side, you have to use backslashes and two sets of quotes:

```
pscp user@host:"\"remote filename with spaces\\"" local_filename  
pscp local_filename user@host:"\"remote filename with spaces\\""
```

Worse still, in a remote-to-local copy you have to specify the local file name explicitly, otherwise PSCP will complain that they don't match (unless you specified the -unsafe option). The following command will give an error message: c:\>pscp user@host:"\"oo er\\"" .
warning: remote host tried to write to a file called 'oo er'
when we requested a file called '\"oo er\"'.

Instead, you need to specify the local file name in full:

```
c:\>pscp user@host:"\"oo er\\"" "oo er"
```

A.6.10 Should I run the 32-bit or the 64-bit version?

If you're not sure, the 32-bit version is generally the safe option. It will run perfectly well on all processors and on all versions of Windows that PuTTY supports. PuTTY doesn't require to run as a 64-bit application to work well, and having a 32-bit PuTTY on a 64-bit system isn't likely to cause you any trouble.

The 64-bit version (first released in 0.68) will only run if you have a 64-bit processor *and* a 64-bit edition of Windows (both of these things are likely to be true of any recent Windows PC). It will run somewhat faster (in particular, the cryptography will be faster, especially during link setup), but it will consume slightly more memory.

If you need to use an external DLL for GSSAPI authentication, that DLL may only be available in a 32-bit or 64-bit form, and that will dictate the version of PuTTY you need to use. (You will probably know if you're doing this; see [section 4.23.2](#) in the documentation.)

A.7 Troubleshooting

- [A.7.1 Why do I see ‘Fatal: Protocol error: Expected control record’ in PSCP?](#)
- [A.7.2 I clicked on a colour in the Colours panel, and the colour didn’t change in my terminal.](#)
- [A.7.3 After trying to establish an SSH-2 connection, PuTTY says ‘Out of memory’ and dies.](#)
- [A.7.4 When attempting a file transfer, either PSCP or PSFTP says ‘Out of memory’ and dies.](#)
- [A.7.5 PSFTP transfers files much slower than PSCP.](#)
- [A.7.6 When I run full-colour applications, I see areas of black space where colour ought to be, or vice versa.](#)
- [A.7.7 When I change some terminal settings, nothing happens.](#)
- [A.7.8 My PuTTY sessions unexpectedly close after they are idle for a while.](#)
- [A.7.9 PuTTY’s network connections time out too quickly when network connectivity is temporarily lost.](#)
- [A.7.10 When I cat a binary file, I get ‘PuTTYPuTTYPuTTY’ on my command line.](#)
- [A.7.11 When I cat a binary file, my window title changes to a nonsense string.](#)
- [A.7.12 My keyboard stops working once PuTTY displays the password prompt.](#)
- [A.7.13 One or more function keys don’t do what I expected in a server-side application.](#)
- [A.7.14 Why do I see ‘Couldn’t load private key from ...’? Why can PuTTYgen load my key but not PuTTY?](#)
- [A.7.15 When I’m connected to a Red Hat Linux 8.0 system, some characters don’t display properly.](#)
- [A.7.16 Since I upgraded to PuTTY 0.54, the scrollback has stopped working when I run screen.](#)
- [A.7.17 Since I upgraded Windows XP to Service Pack 2, I can’t use addresses like 127.0.0.2.](#)
- [A.7.18 PSFTP commands seem to be missing a directory separator \(slash\).](#)

- [A.7.19 Do you want to hear about ‘Software caused connection abort’?](#)
- [A.7.20 My SSH-2 session locks up for a few seconds every so often.](#)
- [A.7.21 PuTTY fails to start up. Windows claims that ‘the application configuration is incorrect’.](#)
- [A.7.22 When I put 32-bit PuTTY in c:\WINDOWS\SYSTEM32 on my 64-bit Windows system, ‘Duplicate Session’ doesn’t work.](#)
- [A.7.23 After I upgraded PuTTY to 0.68, I can no longer connect to my embedded device or appliance.](#)
- [A.7.24 Since 0.78, I can’t find where to configure my SSH private key.](#)

A.7.1 Why do I see ‘Fatal: Protocol error: Expected control record’ in PSCP?

This happens because PSCP was expecting to see data from the server that was part of the PSCP protocol exchange, and instead it saw data that it couldn't make any sense of at all.

This almost always happens because the startup scripts in your account on the server machine are generating output. This is impossible for PSCP, or any other SCP client, to work around. You should never use startup files (`.bashrc`, `.cshrc` and so on) which generate output in non-interactive sessions.

This is not actually a PuTTY problem. If PSCP fails in this way, then all other SCP clients are likely to fail in exactly the same way. The problem is at the server end.

A.7.2 I clicked on a colour in the Colours panel, and the colour didn't change in my terminal.

That isn't how you're supposed to use the Colours panel.

During the course of a session, PuTTY potentially uses *all* the colours listed in the Colours panel. It's not a question of using only one of them and you choosing which one; PuTTY will use them *all*. The purpose of the Colours panel is to let you adjust the appearance of all the colours. So to change the colour of the cursor, for example, you would select 'Cursor Colour', press the 'Modify' button, and select a new colour from the dialog box that appeared. Similarly, if you want your session to appear in green, you should select 'Default Foreground' and press 'Modify'. Clicking on 'ANSI Green' won't turn your session green; it will only allow you to adjust the *shade* of green used when PuTTY is instructed by the server to display green text.

A.7.3 After trying to establish an SSH-2 connection, PuTTY says ‘Out of memory’ and dies.

If this happens just while the connection is starting up, this often indicates that for some reason the client and server have failed to establish a session encryption key. Somehow, they have performed calculations that should have given each of them the same key, but have ended up with different keys; so data encrypted by one and decrypted by the other looks like random garbage.

This causes an ‘out of memory’ error because the first encrypted data PuTTY expects to see is the length of an SSH message. Normally this will be something well under 100 bytes. If the decryption has failed, PuTTY will see a completely random length in the region of two *gigabytes*, and will try to allocate enough memory to store this non-existent message. This will immediately lead to it thinking it doesn't have enough memory, and panicking.

If this happens to you, it is quite likely to still be a PuTTY bug and you should report it (although it might be a bug in your SSH server instead); but it doesn't necessarily mean you've actually run out of memory.

A.7.4 When attempting a file transfer, either PSCP or PSFTP says ‘Out of memory’ and dies.

This is almost always caused by your login scripts on the server generating output. PSCP or PSFTP will receive that output when they were expecting to see the start of a file transfer protocol, and they will attempt to interpret the output as file-transfer protocol. This will usually lead to an ‘out of memory’ error for much the same reasons as given in [question A.7.3](#).

This is a setup problem in your account on your server, *not* a PSCP/PSFTP bug. Your login scripts should *never* generate output during non-interactive sessions; secure file transfer is not the only form of remote access that will break if they do.

On Unix, a simple fix is to ensure that all the parts of your login script that might generate output are in `.profile` (if you use a Bourne shell derivative) or `.login` (if you use a C shell). Putting them in more general files such as `.bashrc` or `.cshrc` is liable to lead to problems.

A.7.5 PSFTP transfers files much slower than PSCP.

The throughput of PSFTP 0.54 should be much better than 0.53b and prior; we've added code to the SFTP backend to queue several blocks of data rather than waiting for an acknowledgement for each. (The SCP backend did not suffer from this performance issue because SCP is a much simpler protocol.)

A.7.6 When I run full-colour applications, I see areas of black space where colour ought to be, or vice versa.

You almost certainly need to change the ‘Use background colour to erase screen’ setting in the Terminal panel. If there is too much black space (the commoner situation), you should enable it, while if there is too much colour, you should disable it. (See [section 4.3.5](#).) In old versions of PuTTY, this was disabled by default, and would not take effect until you reset the terminal (see [question A.7.7](#)). Since 0.54, it is enabled by default, and changes take effect immediately.

A.7.7 When I change some terminal settings, nothing happens.

Some of the terminal options (notably Auto Wrap and background-colour screen erase) actually represent the *default* setting, rather than the currently active setting. The server can send sequences that modify these options in mid-session, but when the terminal is reset (by server action, or by you choosing ‘Reset Terminal’ from the System menu) the defaults are restored.

In versions 0.53b and prior, if you change one of these options in the middle of a session, you will find that the change does not immediately take effect. It will only take effect once you reset the terminal.

In version 0.54, the behaviour has changed - changes to these settings take effect immediately.

A.7.8 My PuTTY sessions unexpectedly close after they are idle for a while.

Some types of firewall, and almost any router doing Network Address Translation (NAT, also known as IP masquerading), will forget about a connection through them if the connection does nothing for too long. This will cause the connection to be rudely cut off when contact is resumed.

You can try to combat this by telling PuTTY to send *keepalives*: packets of data which have no effect on the actual session, but which reassure the router or firewall that the network connection is still active and worth remembering about.

Keepalives don't solve everything, unfortunately; although they cause greater robustness against this sort of router, they can also cause a *loss* of robustness against network dropouts. See [section 4.14.1](#) in the documentation for more discussion of this.

A.7.9 PuTTY's network connections time out too quickly when network connectivity is temporarily lost.

This is a Windows problem, not a PuTTY problem. The timeout value can't be set on per application or per session basis. To increase the TCP timeout globally, you need to tinker with the Registry.

On Windows 95, 98 or ME, the registry key you need to create or change is

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\
MSTCP\MaxDataRetries

(it must be of type DWORD in Win95, or String in Win98/ME). (See MS Knowledge Base article [158474](#) for more information.)

On Windows NT, 2000, or XP, the registry key to create or change is

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\
Parameters\TcpMaxDataRetransmissions

and it must be of type DWORD. (See MS Knowledge Base articles [120642](#) and [314053](#) for more information.)

Set the key's value to something like 10. This will cause Windows to try harder to keep connections alive instead of abandoning them.

A.7.10 When I cat a binary file, I get ‘PuTTYPuTTYPuTTY’ on my command line.

Don't do that, then.

This is designed behaviour; when PuTTY receives the character Control-E from the remote server, it interprets it as a request to identify itself, and so it sends back the string ‘PuTTY’ as if that string had been entered at the keyboard. Control-E should only be sent by programs that are prepared to deal with the response. Writing a binary file to your terminal is likely to output many Control-E characters, and cause this behaviour. Don't do it. It's a bad plan.

To mitigate the effects, you could configure the answerback string to be empty (see [section 4.3.7](#)); but writing binary files to your terminal is likely to cause various other unpleasant behaviour, so this is only a small remedy.

A.7.11 When I cat a binary file, my window title changes to a nonsense string.

Don't do that, then.

It is designed behaviour that PuTTY should have the ability to adjust the window title on instructions from the server. Normally the control sequence that does this should only be sent deliberately, by programs that know what they are doing and intend to put meaningful text in the window title. Writing a binary file to your terminal runs the risk of sending the same control sequence by accident, and cause unexpected changes in the window title. Don't do it.

A.7.12 My keyboard stops working once PuTTY displays the password prompt.

No, it doesn't. PuTTY just doesn't display the password you type, so that someone looking at your screen can't see what it is.

Unlike the Windows login prompts, PuTTY doesn't display the password as a row of asterisks either. This is so that someone looking at your screen can't even tell how *long* your password is, which might be valuable information.

A.7.13 One or more function keys don't do what I expected in a server-side application.

If you've already tried all the relevant options in the PuTTY Keyboard panel, you may need to mail the PuTTY maintainers and ask.

It is *not* usually helpful just to tell us which application, which server operating system, and which key isn't working; in order to replicate the problem we would need to have a copy of every operating system, and every application, that anyone has ever complained about.

PuTTY responds to function key presses by sending a sequence of control characters to the server. If a function key isn't doing what you expect, it's likely that the character sequence your application is expecting to receive is not the same as the one PuTTY is sending. Therefore what we really need to know is *what* sequence the application is expecting.

The simplest way to investigate this is to find some other terminal environment, in which that function key *does* work; and then investigate what sequence the function key is sending in that situation. One reasonably easy way to do this on a Unix system is to type the command `cat`, and then press the function key. This is likely to produce output of the form `^[[11~`. You can also do this in PuTTY, to find out what sequence the function key is producing in that. Then you can mail the PuTTY maintainers and tell us 'I wanted the F1 key to send `^[[11~`, but instead it's sending `^[[OP`, can this be done?', or something similar.

You should still read the [Feedback page](#) on the PuTTY website (also provided as [appendix B](#) in the manual), and follow the guidelines contained in that.

A.7.14 Why do I see ‘Couldn’t load private key from ...’? Why can PuTTYgen load my key but not PuTTY?

It's likely that you've generated an SSH protocol 2 key with PuTTYgen, but you're trying to use it in an SSH-1 connection. SSH-1 and SSH-2 keys have different formats, and (at least in 0.52) PuTTY's reporting of a key in the wrong format isn't optimal.

To connect using SSH-2 to a server that supports both versions, you need to change the configuration from the default (see [question A.2.1](#)).

A.7.15 When I'm connected to a Red Hat Linux 8.0 system, some characters don't display properly.

A common complaint is that hyphens in man pages show up as acute.

With release 8.0, Red Hat appear to have made UTF-8 the default character set. There appears to be no way for terminal emulators such as PuTTY to know this (as far as we know, the appropriate escape sequence to switch into UTF-8 mode isn't sent).

A fix is to configure sessions to RH8 systems to use UTF-8 translation - see [section 4.10.1](#) in the documentation. (Note that if you use 'Change Settings', changes may not take place immediately - see [question A.7.7.](#)) If you really want to change the character set used by the server, the right place is /etc/sysconfig/i18n, but this shouldn't be necessary.

A.7.16 Since I upgraded to PuTTY 0.54, the scrollback has stopped working when I run screen.

PuTTY's terminal emulator has always had the policy that when the 'alternate screen' is in use, nothing is added to the scrollback. This is because the usual sorts of programs which use the alternate screen are things like text editors, which tend to scroll back and forth in the same document a lot; so (a) they would fill up the scrollback with a large amount of unhelpfully disordered text, and (b) they contain their own method for the user to scroll back to the bit they were interested in. We have generally found this policy to do the Right Thing in almost all situations.

Unfortunately, screen is one exception: it uses the alternate screen, but it's still usually helpful to have PuTTY's scrollback continue working. The simplest solution is to go to the Features control panel and tick 'Disable switching to alternate terminal screen'. (See [section 4.6.4](#) for more details.) Alternatively, you can tell screen itself not to use the alternate screen: the [screen FAQ](#) suggests adding the line 'termcapinfo xterm ti@:te@' to your .screenrc file.

The reason why this only started to be a problem in 0.54 is because screen typically uses an unusual control sequence to switch to the alternate screen, and previous versions of PuTTY did not support this sequence.

A.7.17 Since I upgraded Windows XP to Service Pack 2, I can't use addresses like 127.0.0.2.

Some people who ask PuTTY to listen on localhost addresses other than 127.0.0.1 to forward services such as SMB and Windows Terminal Services have found that doing so no longer works since they upgraded to WinXP SP2.

This is apparently an issue with SP2 that is acknowledged by Microsoft in MS Knowledge Base article [884020](#). The article links to a fix you can download.

(However, we've been told that SP2 also fixes the bug that means you need to use non-127.0.0.1 addresses to forward Terminal Services in the first place.)

A.7.18 PSFTP commands seem to be missing a directory separator (slash).

Some people have reported the following incorrect behaviour with PSFTP:

```
psftp> pwd  
Remote directory is /dir1/dir2  
psftp> get filename.ext  
/dir1/dir2filename.ext: no such file or directory
```

This is not a bug in PSFTP. There is a known bug in some versions of portable OpenSSH ([bug 697](#)) that causes these symptoms; it appears to have been introduced around 3.7.x. It manifests only on certain platforms (AIX is what has been reported to us).

There is a patch for OpenSSH attached to that bug; it's also fixed in recent versions of portable OpenSSH (from around 3.8).

A.7.19 Do you want to hear about ‘Software caused connection abort’?

In the documentation for PuTTY 0.53 and 0.53b, we mentioned that we'd like to hear about any occurrences of this error. Since the release of PuTTY 0.54, however, we've been convinced that this error doesn't indicate that PuTTY's doing anything wrong, and we don't need to hear about further occurrences. See [section 10.16](#) for our current documentation of this error.

A.7.20 My SSH-2 session locks up for a few seconds every so often.

Recent versions of PuTTY automatically initiate repeat key exchange once per hour, to improve session security. If your client or server machine is slow, you may experience this as a delay of anything up to thirty seconds or so.

These delays are inconvenient, but they are there for your protection. If they really cause you a problem, you can choose to turn off periodic rekeying using the ‘Kex’ configuration panel (see [Section 4.18](#)), but be aware that you will be sacrificing security for this. (Falling back to SSH-1 would also remove the delays, but would lose a *lot* more security still. We do not recommend it.)

A.7.21 PuTTY fails to start up. Windows claims that ‘the application configuration is incorrect’.

This is caused by a bug in certain versions of Windows XP which is triggered by PuTTY 0.58. This was fixed in 0.59. The '[xp-wont-run](#)' entry in PuTTY's wishlist has more details.

A.7.22 When I put 32-bit PuTTY in c:\WINDOWS\SYSTEM32 on my 64-bit Windows system, ‘Duplicate Session’ doesn’t work.

The short answer is not to put the PuTTY executables in that location.

On 64-bit systems, c:\WINDOWS\SYSTEM32 is intended to contain only 64-bit binaries; Windows' 32-bit binaries live in c:\WINDOWS\SYSWOW64. When a 32-bit PuTTY executable runs on a 64-bit system, it cannot by default see the 'real' c:\WINDOWS\SYSTEM32 at all, because the [File System Redirector](#) arranges that the running program sees the appropriate kind of binaries in SYSTEM32. Thus, operations in the PuTTY suite that involve it accessing its own executables, such as 'New Session' and 'Duplicate Session', will not work.

A.7.23 After I upgraded PuTTY to 0.68, I can no longer connect to my embedded device or appliance.

If your SSH server has started unexpectedly closing SSH connections after you enter your password, and it worked before 0.68, you may have a buggy server that objects to certain SSH protocol extensions.

The SSH protocol recently gained a new ‘terminal mode’, `IUTF8`, which PuTTY sends by default; see [section 4.24.2](#). This is the first new terminal mode since the SSH-2 protocol was defined. While servers are supposed to ignore modes they don't know about, some buggy servers will unceremoniously close the connection if they see anything they don't recognise. SSH servers in embedded devices, network appliances, and the like seem to disproportionately have this bug.

If you think you have such a server, from 0.69 onwards you can disable sending of the `IUTF8` mode: on the SSH / TTY panel, select `IUTF8` on the list, select ‘Nothing’, and press ‘Set’. (It's not possible to disable sending this mode in 0.68.)

A.7.24 Since 0.78, I can't find where to configure my SSH private key.

In PuTTY 0.78, the ‘Private key file for authentication’ control, where you specify a .PPK file for SSH public key authentication, moved to a new ‘Credentials’ panel in the configuration dialog. You can find this by opening the ‘SSH’ category in the tree view on the left, then opening the ‘Auth’ subcategory under that, then clicking on ‘Credentials’. On this page you’ll find the ‘Browse...’ button you need to select a .PPK file for authentication, as described in [section 4.22.1](#).

(This control had previously been on the ‘Auth’ panel since public key authentication was first released in 2002, so many online how-to guides still describe it there. The configuration controls were reorganised to make room for features added in 0.78, such as OpenSSH certificates.)

A.8 Security questions

- [A.8.1 Is it safe for me to download PuTTY and use it on a public PC?](#)
- [A.8.2 What does PuTTY leave on a system? How can I clean up after it?](#)
- [A.8.3 How come PuTTY now supports DSA, when the website used to say how insecure it was?](#)
- [A.8.4 Couldn't Pageant use virtualLock\(\) to stop private keys being written to disk?](#)
- [A.8.5 Is the version of PuTTY in the Microsoft Store legit?](#)

A.8.1 Is it safe for me to download PuTTY and use it on a public PC?

It depends on whether you trust that PC. If you don't trust the public PC, don't use PuTTY on it, and don't use any other software you plan to type passwords into either. It might be watching your keystrokes, or it might tamper with the PuTTY binary you download. There is *no* program safe enough that you can run it on an actively malicious PC and get away with typing passwords into it.

If you do trust the PC, then it's probably OK to use PuTTY on it (but if you don't trust the network, then the PuTTY download might be tampered with, so it would be better to carry PuTTY with you on a USB stick).

A.8.2 What does PuTTY leave on a system? How can I clean up after it?

PuTTY will leave some Registry entries, and a random seed file, on the PC (see [question A.5.2](#)). Windows 7 and up also remember some information about recently launched sessions for the ‘jump list’ feature.

If you are using PuTTY on a public PC, or somebody else's PC, you might want to clean this information up when you leave. You can do that automatically, by running the command `putty -cleanup`. See [section 3.11.2](#) in the documentation for more detail. (Note that this only removes settings for the currently logged-in user on multi-user systems.) If PuTTY was installed from the installer package, it will also appear in ‘Add/Remove Programs’. Current versions of the installer do not offer to remove the above-mentioned items, so if you want them removed you should run `putty -cleanup` before uninstalling.

A.8.3 How come PuTTY now supports DSA, when the website used to say how insecure it was?

DSA has a major weakness *if badly implemented*: it relies on a random number generator to far too great an extent. If the random number generator produces a number an attacker can predict, the DSA private key is exposed - meaning that the attacker can log in as you on all systems that accept that key.

The PuTTY policy changed because the developers were informed of ways to implement DSA which do not suffer nearly as badly from this weakness, and indeed which don't need to rely on random numbers at all. For this reason we now believe PuTTY's DSA implementation is probably OK.

The recently added elliptic-curve signature methods are also DSA-style algorithms, so they have this same weakness in principle. Our ECDSA implementation uses the same defence as DSA, while our Ed25519 implementation uses the similar system (but different in details) that the Ed25519 spec mandates.

A.8.4 Couldn't Pageant use `VirtualLock()` to stop private keys being written to disk?

Unfortunately not. The `VirtualLock()` function in the Windows API doesn't do a proper job: it may prevent small pieces of a process's memory from being paged to disk while the process is running, but it doesn't stop the process's memory as a whole from being swapped completely out to disk when the process is long-term inactive. And Pageant spends most of its time inactive.

A.8.5 Is the version of PuTTY in the Microsoft Store legit?

The free-of-charge ‘PuTTY’ application at [this link](#) is published and maintained by us. The copy there is the latest release, usually updated within a few days of us publishing it on our own website.

There have been other copies of PuTTY on the store, some looking quite similar, and some charging money. Those were uploaded by other people, and we can't guarantee anything about them.

The first version we published to the Microsoft Store was 0.76 (some time after its initial release on our website).

A.9 Administrative questions

- [A.9.1 Is putty.org your website?](#)
- [A.9.2 Would you like me to register you a nicer domain name?](#)
- [A.9.3 Would you like free web hosting for the PuTTY web site?](#)
- [A.9.4 Would you link to my web site from the PuTTY web site?](#)
- [A.9.5 Why don't you move PuTTY to SourceForge?](#)
- [A.9.6 Why can't I subscribe to the putty-bugs mailing list?](#)
- [A.9.7 If putty-bugs isn't a general-subscription mailing list, what is?](#)
- [A.9.8 How can I donate to PuTTY development?](#)
- [A.9.9 Can I have permission to put PuTTY on a cover disk / distribute it with other software / etc?](#)
- [A.9.10 Can you sign an agreement indemnifying us against security problems in PuTTY?](#)
- [A.9.11 Can you sign this form granting us permission to use/distribute PuTTY?](#)
- [A.9.12 Can you write us a formal notice of permission to use PuTTY?](#)
- [A.9.13 Can you sign anything for us?](#)
- [A.9.14 If you won't sign anything, can you give us some sort of assurance that you won't make PuTTY closed-source in future?](#)
- [A.9.15 Can you provide us with export control information / FIPS certification for PuTTY?](#)
- [A.9.16 As one of our existing software vendors, can you just fill in this questionnaire for us?](#)
- [A.9.17 The sha1sums / sha256sums / etc files on your download page don't match the binaries.](#)

A.9.1 Is putty.org your website?

No, it isn't. `putty.org` is run by an opportunist who uses it to advertise their own commercial SSH implementation to people looking for our free one. We don't own that site, we can't control it, and we don't advise anyone to use it in preference to our own site.

The real PuTTY web site, run by the PuTTY team, has always been at <https://www.chiark.greenend.org.uk/~sgtatham/putty/>.

A.9.2 Would you like me to register you a nicer domain name?

No, thank you. Even if you can find one (most of them seem to have been registered already, by people who didn't ask whether we actually wanted it before they applied), we're happy with the PuTTY web site being exactly where it is. It's not hard to find (just type 'putty' into google.com and we're the first link returned), and we don't believe the administrative hassle of moving the site would be worth the benefit.

In addition, if we *did* want a custom domain name, we would want to run it ourselves, so we knew for certain that it would continue to point where we wanted it, and wouldn't suddenly change or do strange things. Having it registered for us by a third party who we don't even know is not the best way to achieve this.

A.9.3 Would you like free web hosting for the PuTTY web site?

We already have some, thanks.

A.9.4 Would you link to my web site from the PuTTY web site?

Only if the content of your web page is of definite direct interest to PuTTY users. If your content is unrelated, or only tangentially related, to PuTTY, then the link would simply be advertising for you.

One very nice effect of the Google ranking mechanism is that by and large, the most popular web sites get the highest rankings. This means that when an ordinary person does a search, the top item in the search is very likely to be a high-quality site or the site they actually wanted, rather than the site which paid the most money for its ranking.

The PuTTY web site is held in high esteem by Google, for precisely this reason: lots of people have linked to it simply because they like PuTTY, without us ever having to ask anyone to link to us. We feel that it would be an abuse of this esteem to use it to boost the ranking of random advertisers' web sites. If you want your web site to have a high Google ranking, we'd prefer that you achieve this the way we did - by being good enough at what you do that people will link to you simply because they like you.

In particular, we aren't interested in trading links for money (see above), and we *certainly* aren't interested in trading links for other links (since we have no advertising on our web site, our Google ranking is not even directly worth anything to us). If we don't want to link to you for free, then we probably won't want to link to you at all.

If you have software based on PuTTY, or specifically designed to interoperate with PuTTY, or in some other way of genuine interest to PuTTY users, then we will probably be happy to add a link to you on our Links page. And if you're running a particularly valuable mirror of the PuTTY web site, we might be interested in linking to you from our Mirrors page.

A.9.5 Why don't you move PuTTY to SourceForge?

Partly, because we don't want to move the web site location (see [question A.9.2](#)).

Also, security reasons. PuTTY is a security product, and as such it is particularly important to guard the code and the web site against unauthorised modifications which might introduce subtle security flaws. Therefore, we prefer that the Git repository, web site and FTP site remain where they are, under the direct control of system administrators we know and trust personally, rather than being run by a large organisation full of people we've never met and which is known to have had breakins in the past.

No offence to SourceForge; I think they do a wonderful job. But they're not ideal for everyone, and in particular they're not ideal for us.

A.9.6 Why can't I subscribe to the putty-bugs mailing list?

Because you're not a member of the PuTTY core development team. The putty-bugs mailing list is not a general newsgroup-like discussion forum; it's a contact address for the core developers, and an *internal* mailing list for us to discuss things among ourselves. If we opened it up for everybody to subscribe to, it would turn into something more like a newsgroup and we would be completely overwhelmed by the volume of traffic. It's hard enough to keep up with the list as it is.

A.9.7 If putty-bugs isn't a general-subscription mailing list, what is?

There isn't one, that we know of.

If someone else wants to set up a mailing list or other forum for PuTTY users to help each other with common problems, that would be fine with us, though the PuTTY team would almost certainly not have the time to read it. It's probably better to use one of the established newsgroups for this purpose (see [section B.1.2](#)).

A.9.8 How can I donate to PuTTY development?

Please, *please* don't feel you have to. PuTTY is completely free software, and not shareware. We think it's very important that everybody who wants to use PuTTY should be able to, whether they have any money or not; so the last thing we would want is for a PuTTY user to feel guilty because they haven't paid us any money. If you want to keep your money, please do keep it. We wouldn't dream of asking for any.

Having said all that, if you still really *want* to give us money, we won't argue :-) The easiest way for us to accept donations is if you send money to <anakin@pobox.com> using PayPal (www.paypal.com). If you don't like PayPal, talk to us; we can probably arrange some alternative means.

Small donations (tens of dollars or tens of euros) will probably be spent on beer or curry, which helps motivate our volunteer team to continue doing this for the world. Larger donations will be spent on something that actually helps development, if we can find anything (perhaps new hardware, or a new version of Windows), but if we can't find anything then we'll just distribute the money among the developers. If you want to be sure your donation is going towards something worthwhile, ask us first. If you don't like these terms, feel perfectly free not to donate. We don't mind.

A.9.9 Can I have permission to put PuTTY on a cover disk / distribute it with other software / etc?

Yes. For most things, you need not bother asking us explicitly for permission; our licence already grants you permission.

See [section B.9](#) for more details.

A.9.10 Can you sign an agreement indemnifying us against security problems in PuTTY?

No!

A vendor of physical security products (e.g. locks) might plausibly be willing to accept financial liability for a product that failed to perform as advertised and resulted in damage (e.g. valuables being stolen). The reason they can afford to do this is because they sell a *lot* of units, and only a small proportion of them will fail; so they can meet their financial liability out of the income from all the rest of their sales, and still have enough left over to make a profit. Financial liability is intrinsically linked to selling your product for money.

There are two reasons why PuTTY is not analogous to a physical lock in this context. One is that software products don't exhibit random variation: *if* PuTTY has a security hole (which does happen, although we do our utmost to prevent it and to respond quickly when it does), every copy of PuTTY will have the same hole, so it's likely to affect all the users at the same time. So even if our users were all paying us to use PuTTY, we wouldn't be able to *simultaneously* pay every affected user compensation in excess of the amount they had paid us in the first place. It just wouldn't work.

The second, much more important, reason is that PuTTY users *don't* pay us. The PuTTY team does not have an income; it's a volunteer effort composed of people spending their spare time to try to write useful software. We aren't even a company or any kind of legally recognised organisation. We're just a bunch of people who happen to do some stuff in our spare time.

Therefore, to ask us to assume financial liability is to ask us to assume a risk of having to pay it out of our own *personal* pockets: out of the same budget from which we buy food and clothes and pay

our rent. That's more than we're willing to give. We're already giving a lot of our spare *time* to developing software for free; if we had to pay our own *money* to do it as well, we'd start to wonder why we were bothering.

Free software fundamentally does not work on the basis of financial guarantees. Your guarantee of the software functioning correctly is simply that you have the source code and can check it before you use it. If you want to be sure there aren't any security holes, do a security audit of the PuTTY code, or hire a security engineer if you don't have the necessary skills yourself: instead of trying to ensure you can get compensation in the event of a disaster, try to ensure there isn't a disaster in the first place.

If you *really* want financial security, see if you can find a security engineer who will take financial responsibility for the correctness of their review. (This might be less likely to suffer from the everything-failing-at-once problem mentioned above, because such an engineer would probably be reviewing a lot of *different* products which would tend to fail independently.) Failing that, see if you can persuade an insurance company to insure you against security incidents, and if the insurer demands it as a condition then get our code reviewed by a security engineer they're happy with.

A.9.11 Can you sign this form granting us permission to use/distribute PuTTY?

If your form contains any clause along the lines of ‘the undersigned represents and warrants’, we’re not going to sign it. This is particularly true if it asks us to warrant that PuTTY is secure; see [question A.9.10](#) for more discussion of this. But it doesn’t really matter what we’re supposed to be warranting: even if it’s something we already believe is true, such as that we don’t infringe any third-party copyright, we will not sign a document accepting any legal or financial liability. This is simply because the PuTTY development project has no income out of which to satisfy that liability, or pay legal costs, should it become necessary. We cannot afford to be sued. We are assuring you that *we have done our best*; if that isn’t good enough for you, tough.

The existing PuTTY licence document already gives you permission to use or distribute PuTTY in pretty much any way which does not involve pretending you wrote it or suing us if it goes wrong. We think that really ought to be enough for anybody.

See also [question A.9.13](#) for another reason why we don’t want to do this sort of thing.

A.9.12 Can you write us a formal notice of permission to use PuTTY?

We could, in principle, but it isn't clear what use it would be. If you think there's a serious chance of one of the PuTTY copyright holders suing you (which we don't!), you would presumably want a signed notice from *all* of them; and we couldn't provide that even if we wanted to, because many of the copyright holders are people who contributed some code in the past and with whom we subsequently lost contact. Therefore the best we would be able to do even *in theory* would be to have the core development team sign the document, which wouldn't guarantee you that some other copyright holder might not sue.

See also [question A.9.13](#) for another reason why we don't want to do this sort of thing.

A.9.13 Can you sign *anything* for us?

Not unless there's an incredibly good reason.

We are generally unwilling to set a precedent that involves us having to enter into individual agreements with PuTTY users. We estimate that we have literally *millions* of users, and we absolutely would not have time to go round signing specific agreements with every one of them. So if you want us to sign something specific for you, you might usefully stop to consider whether there's anything special that distinguishes you from 999,999 other users, and therefore any reason we should be willing to sign something for you without it setting such a precedent.

If your company policy requires you to have an individual agreement with the supplier of any software you use, then your company policy is simply not well suited to using popular free software, and we urge you to consider this as a flaw in your policy.

A.9.14 If you won't sign anything, can you give us some sort of assurance that you won't make PuTTY closed-source in future?

Yes and no.

If what you want is an assurance that some *current version* of PuTTY which you've already downloaded will remain free, then you already have that assurance: it's called the PuTTY Licence. It grants you permission to use, distribute and copy the software to which it applies; once we've granted that permission (which we have), we can't just revoke it.

On the other hand, if you want an assurance that *future* versions of PuTTY won't be closed-source, that's more difficult. We could in principle sign a document stating that we would never release a closed-source PuTTY, but that wouldn't assure you that we *would* keep releasing *open-source* PuTTYS: we would still have the option of ceasing to develop PuTTY at all, which would surely be even worse for you than making it closed-source! (And we almost certainly wouldn't *want* to sign a document guaranteeing that we would actually continue to do development work on PuTTY; we certainly wouldn't sign it for free. Documents like that are called contracts of employment, and are generally not signed except in return for a sizeable salary.)

If we *were* to stop developing PuTTY, or to decide to make all future releases closed-source, then you would still be free to copy the last open release in accordance with the current licence, and in particular you could start your own fork of the project from that release. If this happened, I confidently predict that *somebody* would do that, and that some kind of a free PuTTY would continue to be developed. There's already precedent for that sort of thing happening in free software. We can't guarantee that *somebody other than you* would do it, of course; you might have to do it yourself. But we can assure

you that there would be nothing *preventing* anyone from continuing free development if we stopped.

(Finally, we can also confidently predict that if we made PuTTY closed-source and someone made an open-source fork, most people would switch to the latter. Therefore, it would be pretty stupid of us to try it.)

A.9.15 Can you provide us with export control information / FIPS certification for PuTTY?

Some people have asked us for an Export Control Classification Number (ECCN) for PuTTY. We don't know whether we have one, and as a team of free software developers based in the UK we don't have the time, money, or effort to deal with US bureaucracy to investigate any further. We believe that PuTTY falls under 5D002 on the US Commerce Control List, but that shouldn't be taken as definitive. If you need to know more you should seek professional legal advice. The same applies to any other country's legal requirements and restrictions.

Similarly, some people have asked us for FIPS certification of the PuTTY tools. Unless someone else is prepared to do the necessary work and pay any costs, we can't provide this.

A.9.16 As one of our existing software vendors, can you just fill in this questionnaire for us?

We periodically receive requests like this, from organisations which have apparently sent out a form letter to everyone listed in their big spreadsheet of ‘software vendors’ requiring them all to answer some long list of questions about supported OS versions, paid support arrangements, compliance with assorted local regulations we haven’t heard of, contact phone numbers, and other such administrivia.

Many of the questions are obviously meaningless when applied to PuTTY (we don’t provide any paid support in the first place!), most of the rest could have been answered with only a very quick look at our website, and some we are actively unwilling to answer (we are private individuals, why would we want to give out our home phone numbers to large corporations?).

We don’t make a habit of responding in full to these questionnaires, because *we are not a software vendor*.

A software *vendor* is a company to which you are paying lots of money in return for some software. They know who you are, and they know you’re paying them money; so they have an incentive to fill in your forms and questionnaires, to research any local regulations you cite if they don’t already know about them, and generally to provide every scrap of information you might possibly need in the most convenient manner for you, because they want to keep being paid.

But we are a team of free software developers, and that means your relationship with us is nothing like that at all. If you once downloaded our software from our website, that’s great and we hope you found it useful, but it doesn’t mean we have the least idea who you are, or any incentive to do lots of unpaid work to support our ‘relationship’ with you.

It's not that we are unwilling to *provide information*. We put as much of it as we can on our website for your convenience, and if you actually need to know some fact about PuTTY which you haven't been able to find on the website (and which is not obviously inapplicable to free software in the first place) then please do ask us, and we'll try to answer as best we can. But we put up the website and this FAQ precisely so that we *don't* have to keep answering the same questions over and over again, so we aren't prepared to fill in completely generic form-letter questionnaires for people who haven't done their best to find the answers here first.

If you work for an organisation which you think might be at risk of making this mistake, we urge you to reorganise your list of software suppliers so that it clearly distinguishes paid vendors who know about you from free software developers who don't have any idea who you are. Then, only send out these mass mailings to the former.

A.9.17 The sha1sums / sha256sums / etc files on your download page don't match the binaries.

People report this every so often, and usually the reason turns out to be that they've matched up the wrong checksums file with the wrong binaries.

The PuTTY download page contains more than one version of the software. There's a *latest release* version; there are the *development snapshots*; and when we're in the run-up to making a release, there are also *pre-release* builds of the upcoming new version. Each one has its own collection of binaries, and its own collection of checksums files to go with them.

So if you've downloaded the release version of the actual program, you need the release version of the checksums too, otherwise you will see a mismatch. Similarly, the development snapshot binaries go with the development snapshot checksums, and so on. (We've colour-coded the download page in an effort to reduce this confusion a bit.)

Another thing to watch out for: as of 0.71, executables like `putty.exe` come in two flavours for each platform: the standalone versions on the website, each of which contains embedded help, and the versions installed by the installer, which use a separate help file also in the installer. We provide checksums for both; the latter are indicated with '(installer version)' after the filename.

If you have double-checked all that, and you still think there's a real mismatch, then please send us a report carefully quoting everything relevant:

- the exact URL you got your binary from
- the checksum of the binary after you downloaded
- the exact URL you got your checksums file from

- the checksum that file says the binary should have.

A.10 Miscellaneous questions

- [A.10.1 Is PuTTY a port of OpenSSH, or based on OpenSSH or OpenSSL?](#)
- [A.10.2 Where can I buy silly_putty?](#)
- [A.10.3 What does ‘PuTTY’ mean?](#)
- [A.10.4 How do I pronounce ‘PuTTY’?](#)

A.10.1 Is PuTTY a port of OpenSSH, or based on OpenSSH or OpenSSL?

No, it isn't. PuTTY is almost completely composed of code written from scratch for PuTTY. The only code we share with OpenSSH is the detector for SSH-1 CRC compensation attacks, written by CORE SDI S.A; we share no code at all with OpenSSL.

A.10.2 Where can I buy silly putty?

You're looking at the wrong web site; the only PuTTY we know about here is the name of a computer program.

If you want the kind of putty you can buy as an executive toy, the PuTTY team can personally recommend Thinking Putty, which you can buy from Crazy Aaron's Putty World, at www.puttyworld.com.

A.10.3 What does ‘PuTTY’ mean?

It's the name of a popular SSH and Telnet client. Any other meaning is in the eye of the beholder. It's been rumoured that ‘PuTTY’ is the antonym of ‘getty’, or that it's the stuff that makes your Windows useful, or that it's a kind of plutonium Teletype. We couldn't possibly comment on such allegations.

A.10.4 How do I pronounce ‘PuTTY’?

Exactly like the English word ‘putty’, which we pronounce /'pʌti/.

Appendix B: Feedback and bug reporting

This is a guide to providing feedback to the PuTTY development team. It is provided as both a web page on the PuTTY site, and an appendix in the PuTTY manual.

[Section B.1](#) gives some general guidelines for sending any kind of e-mail to the development team. Following sections give more specific guidelines for particular types of e-mail, such as bug reports and feature requests.

- [B.1 General guidelines](#)
 - [B.1.1 Sending large attachments](#)
 - [B.1.2 Other places to ask for help](#)
- [B.2 Reporting bugs](#)
- [B.3 Reporting security vulnerabilities](#)
- [B.4 Requesting extra features](#)
- [B.5 Requesting features that have already been requested](#)
- [B.6 Workarounds for SSH server bugs](#)
- [B.7 Support requests](#)
- [B.8 Web server administration](#)
- [B.9 Asking permission for things](#)
- [B.10 Mirroring the PuTTY web site](#)
- [B.11 Praise and compliments](#)
- [B.12 E-mail address](#)

B.1 General guidelines

The PuTTY development team gets a *lot* of mail. If you can possibly solve your own problem by reading the manual, reading the FAQ, reading the web site, asking a fellow user, perhaps posting to a newsgroup (see [section B.1.2](#)), or some other means, then it would make our lives much easier.

We get so much e-mail that we literally do not have time to answer it all. We regret this, but there's nothing we can do about it. So if you can *possibly* avoid sending mail to the PuTTY team, we recommend you do so. In particular, support requests ([section B.7](#)) are probably better sent to newsgroups, or passed to a local expert if possible.

The PuTTY contact email address is a private mailing list containing four or five core developers. Don't be put off by it being a mailing list: if you need to send confidential data as part of a bug report, you can trust the people on the list to respect that confidence. Also, the archives aren't publicly available, so you shouldn't be letting yourself in for any spam by sending us mail.

Please use a meaningful subject line on your message. We get a lot of mail, and it's hard to find the message we're looking for if they all have subject lines like 'PuTTY bug'.

- [B.1.1 Sending large attachments](#)
- [B.1.2 Other places to ask for help](#)

B.1.1 Sending large attachments

Since the PuTTY contact address is a mailing list, e-mails larger than 40Kb will be held for inspection by the list administrator, and will not be allowed through unless they really appear to be worth their large size.

If you are considering sending any kind of large data file to the PuTTY team, it's almost always a bad idea, or at the very least it would be better to ask us first whether we actually need the file. Alternatively, you could put the file on a web site and just send us the URL; that way, we don't have to download it unless we decide we actually need it, and only one of us needs to download it instead of it being automatically copied to all the developers.

(If the file contains confidential information, then you could encrypt it with our Secure Contact Key; see [section F.1](#) for details. Please *only* use this for information that *needs* to be confidential.) Some people like to send mail in MS Word format. Please *don't* send us bug reports, or any other mail, as a Word document. Word documents are roughly fifty times larger than writing the same report in plain text. In addition, most of the PuTTY team read their e-mail on Unix machines, so copying the file to a Windows box to run Word is very inconvenient. Not only that, but several of us don't even *have* a copy of Word!

Some people like to send us screen shots when demonstrating a problem. Please don't do this without checking with us first - we almost never actually need the information in the screen shot. Sending a screen shot of an error box is almost certainly unnecessary when you could just tell us in plain text what the error was. (On some versions of Windows, pressing Ctrl-C when the error box is displayed will copy the text of the message to the clipboard.) Sending a full-screen shot is *occasionally* useful, but it's probably still wise to check whether we need it before sending it.

If you *must* mail a screen shot, don't send it as a .BMP file. BMPs have no compression and they are *much* larger than other image formats such as PNG, TIFF and GIF. Convert the file to a properly compressed image format before sending it.

Please don't mail us executables, at all. Our mail server blocks all incoming e-mail containing executables, as a defence against the vast numbers of e-mail viruses we receive every day. If you mail us an executable, it will just bounce.

If you have made a tiny modification to the PuTTY code, please send us a *patch* to the source code if possible, rather than sending us a huge .ZIP file containing the complete sources plus your modification. If you've only changed 10 lines, we'd prefer to receive a mail that's 30 lines long than one containing multiple megabytes of data we already have.

B.1.2 Other places to ask for help

There are two Usenet newsgroups that are particularly relevant to the PuTTY tools:

- [comp.security.ssh](#), for questions specific to using the SSH protocol;
- [comp.terminals](#), for issues relating to terminal emulation (for instance, keyboard problems).

Please use the newsgroup most appropriate to your query, and remember that these are general newsgroups, not specifically about PuTTY.

If you don't have direct access to Usenet, you can access these newsgroups through Google Groups ([groups.google.com](#)).

B.2 Reporting bugs

If you think you have found a bug in PuTTY, your first steps should be:

- Check the [Wishlist page](#) on the PuTTY website, and see if we already know about the problem. If we do, it is almost certainly not necessary to mail us about it, unless you think you have extra information that might be helpful to us in fixing it. (Of course, if we actually *need* specific extra information about a particular bug, the Wishlist page will say so.)
- Check the [Change Log](#) on the PuTTY website, and see if we have already fixed the bug in the development snapshots.
- Check the [FAQ](#) on the PuTTY website (also provided as [appendix A](#) in the manual), and see if it answers your question. The FAQ lists the most common things which people think are bugs, but which aren't bugs.
- Download the latest development snapshot and see if the problem still happens with that. This really is worth doing. As a general rule we aren't very interested in bugs that appear in the release version but not in the development version, because that usually means they are bugs we have *already fixed*. On the other hand, if you can find a bug in the development version that doesn't appear in the release, that's likely to be a new bug we've introduced since the release and we're definitely interested in it.

If none of those options solved your problem, and you still need to report a bug to us, it is useful if you include some general information:

- Tell us what version of PuTTY you are running. To find this out, use the 'About PuTTY' option from the System menu. Please *do not* just tell us 'I'm running the latest version'; e-mail can be

delayed and it may not be obvious which version was the latest at the time you sent the message.

- PuTTY is a multi-platform application; tell us what version of what OS you are running PuTTY on. (If you're running on Unix, or Windows for Arm, tell us, or we'll assume you're running on Windows for Intel as this is overwhelmingly the case.)
- Tell us what protocol you are connecting with: SSH, Telnet, Rlogin, SUPDUP, or Raw mode, or a serial connection.
- Tell us what kind of server you are connecting to; what OS, and if possible what SSH server (if you're using SSH). You can get some of this information from the PuTTY Event Log (see [section 3.1.3.1](#) in the manual).
- Send us the contents of the PuTTY Event Log, unless you have a specific reason not to (for example, if it contains confidential information that you think we should be able to solve your problem without needing to know).
- Try to give us as much information as you can to help us see the problem for ourselves. If possible, give us a step-by-step sequence of *precise* instructions for reproducing the fault.
- Don't just tell us that PuTTY 'does the wrong thing'; tell us exactly and precisely what it did, and also tell us exactly and precisely what you think it should have done instead. Some people tell us PuTTY does the wrong thing, and it turns out that it was doing the right thing and their expectations were wrong. Help to avoid this problem by telling us exactly what you think it should have done, and exactly what it did do.
- If you think you can, you're welcome to try to fix the problem yourself. A patch to the code which fixes a bug is an excellent addition to a bug report. However, a patch is never a *substitute* for a good bug report; if your patch is wrong or inappropriate,

and you haven't supplied us with full information about the actual bug, then we won't be able to find a better solution.

- <https://www.chiark.greenend.org.uk/~sgtatham/bugs.html> is an article on how to report bugs effectively in general. If your bug report is *particularly* unclear, we may ask you to go away, read this article, and then report the bug again.

It is reasonable to report bugs in PuTTY's documentation, if you think the documentation is unclear or unhelpful. But we do need to be given exact details of *what* you think the documentation has failed to tell you, or *how* you think it could be made clearer. If your problem is simply that you don't *understand* the documentation, we suggest posting to a newsgroup (see [section B.1.2](#)) and seeing if someone will explain what you need to know. *Then*, if you think the documentation could usefully have told you that, send us a bug report and explain how you think we should change it.

B.3 Reporting security vulnerabilities

If you've found a security vulnerability in PuTTY, you might well want to notify us using an encrypted communications channel, to avoid disclosing information about the vulnerability before a fixed release is available.

For this purpose, we provide a GPG key suitable for encryption: the Secure Contact Key. See [section F.1](#) for details of this.

(Of course, vulnerabilities are also bugs, so please do include as much information as possible about them, the same way you would with any other bug report.)

B.4 Requesting extra features

If you want to request a new feature in PuTTY, the very first things you should do are:

- Check the [Wishlist page](#) on the PuTTY website, and see if your feature is already on the list. If it is, it probably won't achieve very much to repeat the request. (But see [section B.5](#) if you want to persuade us to give your particular feature higher priority.)
- Check the Wishlist and [Change Log](#) on the PuTTY website, and see if we have already added your feature in the development snapshots. If it isn't clear, download the latest development snapshot and see if the feature is present. If it is, then it will also be in the next release and there is no need to mail us at all.

If you can't find your feature in either the development snapshots or the Wishlist, then you probably do need to submit a feature request. Since the PuTTY authors are very busy, it helps if you try to do some of the work for us:

- Do as much of the design as you can. Think about ‘corner cases’; think about how your feature interacts with other existing features. Think about the user interface; if you can't come up with a simple and intuitive interface to your feature, you shouldn't be surprised if we can't either. Always imagine whether it's possible for there to be more than one, or less than one, of something you'd assumed there would be one of. (For example, if you were to want PuTTY to put an icon in the System tray rather than the Taskbar, you should think about what happens if there's more than one PuTTY active; how would the user tell which was which?)
- If you can program, it may be worth offering to write the feature yourself and send us a patch. However, it is likely to be helpful if you confer with us first; there may be design issues you haven't

thought of, or we may be about to make big changes to the code which your patch would clash with, or something. If you check with the maintainers first, there is a better chance of your code actually being usable. Also, read the design principles listed in [appendix E](#): if you do not conform to them, we will probably not be able to accept your patch.

B.5 Requesting features that have already been requested

If a feature is already listed on the Wishlist, then it usually means we would like to add it to PuTTY at some point. However, this may not be in the near future. If there's a feature on the Wishlist which you would like to see in the *near* future, there are several things you can do to try to increase its priority level:

- Mail us and vote for it. (Be sure to mention that you've seen it on the Wishlist, or we might think you haven't even *read* the Wishlist). This probably won't have very *much* effect; if a huge number of people vote for something then it may make a difference, but one or two extra votes for a particular feature are unlikely to change our priority list immediately. Offering a new and compelling justification might help. Also, don't expect a reply.
- Offer us money if we do the work sooner rather than later. This sometimes works, but not always. The PuTTY team all have full-time jobs and we're doing all of this work in our free time; we may sometimes be willing to give up some more of our free time in exchange for some money, but if you try to bribe us for a *big* feature it's entirely possible that we simply won't have the time to spare - whether you pay us or not. (Also, we don't accept bribes to add *bad* features to the Wishlist, because our desire to provide high-quality software to the users comes first.)
- Offer to help us write the code. This is probably the *only* way to get a feature implemented quickly, if it's a big one that we don't have time to do ourselves.

B.6 Workarounds for SSH server bugs

It's normal for SSH implementations to automatically enable workarounds for each other's bugs, using the software version strings that are exchanged at the start of the connection. Typically an SSH client will have a list of server version strings that it believes to have particular bugs, and auto-enable the appropriate set of workarounds when it sees one of those strings. (And servers will have a similar list of workarounds for *client* software they believe to be buggy.) If you've found a bug in an SSH server, and you'd like us to add an auto-detected workaround for it, our policy is that **the server implementor should fix it first**.

If the server implementor has fixed it in the latest version, and can give us a complete description of the version strings that go with the bug, then we're happy to use those version strings as a trigger to automatically enable our workaround (assuming one is possible). We *won't* accept requests to auto-enable workarounds for an open-ended set of version strings, such as 'any version of FooServer, including future ones not yet released'.

The aim of this policy is to encourage implementors to gradually converge on the actual standardised SSH protocol. If we enable people to continue violating the spec, by installing open-ended workarounds in PuTTY for bugs they're never going to fix, then we're contributing to an ecosystem in which everyone carries on having bugs and everyone else carries on having to work around them.

An exception: if an SSH server is no longer maintained *at all* (e.g. the company that produced it has gone out of business), and every version of it that was ever released has a bug, then that's one situation in which we may be prepared to add a workaround rule that matches all versions of that software. (The aim is to stop implementors from continuing to release software with the bug – and if they're not releasing it *at all* any more, then that's already done!) We do recognise that sometimes it will be difficult to get the server

maintainer to fix a bug, or even to answer support requests at all. Or it might take them a very long time to get round to doing anything about it. We're not completely unwilling to compromise: we're prepared to add *manually enabled* workarounds to PuTTY even for bugs that an implementation hasn't fixed yet. We just won't *automatically* enable the workaround unless the server maintainer has also done their part.

B.7 Support requests

If you're trying to make PuTTY do something for you and it isn't working, but you're not sure whether it's a bug or not, then *please* consider looking for help somewhere else. This is one of the most common types of mail the PuTTY team receives, and we simply don't have time to answer all the questions. Questions of this type include:

- If you want to do something with PuTTY but have no idea where to start, and reading the manual hasn't helped, try posting to a newsgroup (see [section B.1.2](#)) and see if someone can explain it to you.
- If you have tried to do something with PuTTY but it hasn't worked, and you aren't sure whether it's a bug in PuTTY or a bug in your SSH server or simply that you're not doing it right, then try posting to a newsgroup (see [section B.1.2](#)) and see if someone can solve your problem. Or try doing the same thing with a different SSH client and see if it works with that. Please do not report it as a PuTTY bug unless you are really sure it *is* a bug in PuTTY.
- If someone else installed PuTTY for you, or you're using PuTTY on someone else's computer, try asking them for help first. They're more likely to understand how they installed it and what they expected you to use it for than we are.
- If you have successfully made a connection to your server and now need to know what to type at the server's command prompt, or other details of how to use the server-end software, talk to your server's system administrator. This is not the PuTTY team's problem. PuTTY is only a communications tool, like a telephone; if you can't speak the same language as the person at the other end of the phone, it isn't the telephone company's job to teach it to you.

If you absolutely cannot get a support question answered any other way, you can try mailing it to us, but we can't guarantee to have time to answer it.

B.8 Web server administration

If the PuTTY web site is down (Connection Timed Out), please don't bother mailing us to tell us about it. Most of us read our e-mail on the same machines that host the web site, so if those machines are down then we will notice *before* we read our e-mail. So there's no point telling us our servers are down.

Of course, if the web site has some other error (Connection Refused, 404 Not Found, 403 Forbidden, or something else) then we might *not* have noticed and it might still be worth telling us about it.

If you want to report a problem with our web site, check that you're looking at our *real* web site and not a mirror. The real web site is at <https://www.chiark.greenend.org.uk/~sgtatham/putty/>; if that's not where you're reading this, then don't report the problem to us until you've checked that it's really a problem with the main site. If it's only a problem with the mirror, you should try to contact the administrator of that mirror site first, and only contact us if that doesn't solve the problem (in case we need to remove the mirror from our list).

B.9 Asking permission for things

PuTTY is distributed under the MIT Licence (see [appendix D](#) for details). This means you can do almost *anything* you like with our software, our source code, and our documentation. The only things you aren't allowed to do are to remove our copyright notices or the licence text itself, or to hold us legally responsible if something goes wrong.

So if you want permission to include PuTTY on a magazine cover disk, or as part of a collection of useful software on a CD or a web site, then *permission is already granted*. You don't have to mail us and ask. Just go ahead and do it. We don't mind.

(If you want to distribute PuTTY alongside your own application for use with that application, or if you want to distribute PuTTY within your own organisation, then we recommend, but do not insist, that you offer your own first-line technical support, to answer questions about the interaction of PuTTY with your environment. If your users mail us directly, we won't be able to tell them anything useful about your specific setup.)

If you want to use parts of the PuTTY source code in another program, then it might be worth mailing us to talk about technical details, but if all you want is to ask permission then you don't need to bother. You already have permission.

If you just want to link to our web site, just go ahead. (It's not clear that we *could* stop you doing this, even if we wanted to!)

B.10 Mirroring the PuTTY web site

If you want to set up a mirror of the PuTTY website, go ahead and set one up. Please don't bother asking us for permission before setting up a mirror. You already have permission.

If the mirror is in a country where we don't already have plenty of mirrors, we may be willing to add it to the list on our [mirrors page](#). Read the guidelines on that page, make sure your mirror works, and email us the information listed at the bottom of the page.

Note that we do not *promise* to list your mirror: we get a lot of mirror notifications and yours may not happen to find its way to the top of the list.

Also note that we link to all our mirror sites using the `rel="nofollow"` attribute. Running a PuTTY mirror is not intended to be a cheap way to gain search rankings.

If you have technical questions about the process of mirroring, then you might want to mail us before setting up the mirror (see also the [guidelines on the Mirrors page](#)); but if you just want to ask for permission, you don't need to. You already have permission.

B.11 Praise and compliments

One of the most rewarding things about maintaining free software is getting e-mails that just say ‘thanks’. We are always happy to receive e-mails of this type.

Regrettably we don't have time to answer them all in person. If you mail us a compliment and don't receive a reply, *please* don't think we've ignored you. We did receive it and we were happy about it; we just didn't have time to tell you so personally.

To everyone who's ever sent us praise and compliments, in the past and the future: *you're welcome!*

B.12 E-mail address

The actual address to mail is <putty@projects.tartarus.org>.

Appendix C: PPK file format

This appendix documents the file format used by PuTTY to store private keys.

In this appendix, binary data structures are described using data type representations such as ‘uint32’, ‘string’ and ‘mpint’ as used in the SSH protocol standards themselves. These are defined authoritatively by [RFC 4251 section 5](#), ‘Data Type Representations Used in the SSH Protocols’.

- [C.1 Overview](#)
- [C.2 Outer layer](#)
- [C.3 Private key encodings](#)
 - [C.3.1 RSA](#)
 - [C.3.2 DSA](#)
 - [C.3.3 NIST elliptic-curve keys](#)
 - [C.3.4 EdDSA elliptic-curve keys \(Ed25519 and Ed448\)](#)
- [C.4 Key derivation](#)
- [C.5 Older versions of the PPK format](#)
 - [C.5.1 Version 2](#)
 - [C.5.2 Version 1](#)

C.1 Overview

A PPK file stores a private key, and the corresponding public key. Both are contained in the same file.

The file format can be completely unencrypted, or it can encrypt the private key. The *public* key is stored in cleartext in both cases. (This enables PuTTY to send the public key to an SSH server to see whether it will accept it, and not bother prompting for the passphrase unless the server says yes.) When the key file is encrypted, the encryption key is derived from a passphrase. An encrypted PPK file is also tamper-proofed using a MAC (authentication code), also derived from the same passphrase. The MAC protects the encrypted private key data, but it also covers the cleartext parts of the file. So you can't edit the public half of the key without invalidating the MAC and causing the key file as a whole to become useless.

This MAC protects the key file against active cryptographic attacks in which the public half of a key pair is modified in a controlled way that allows an attacker to deduce information about the private half from the resulting corrupted signatures. Any attempt to do that to a PPK file should be reliably caught by the MAC failing to validate.

(Such an attack would only be useful if the key file was stored in a location where the attacker could modify it without also having full access to the process that you type passphrases into. But that's not impossible; for example, if your home directory was on a network file server, then the file server's administrator could access the key file but not processes on the client machine.) The MAC also covers the *comment* on the key. This stops an attacker from swapping keys with each other and editing the comments to disguise the fact. As a consequence, PuTTYgen cannot edit the comment on a key unless you decrypt the key with your passphrase first.

(The circumstances in which *that* attack would be useful are even more restricted. One example might be that the different keys trigger

specific actions on the server you're connecting to and one of those actions is more useful to the attacker than the other. But once you have a MAC at all, it's no extra effort to make it cover as much as possible, and usually sensible.)

C.2 Outer layer

The outer layer of a PPK file is text-based. The PuTTY tools will always use LF line termination when writing PPK files, but will tolerate CR+LF and CR-only on input.

The first few lines identify it as a PPK, and give some initial data about what's stored in it and how. They look like this:

```
PuTTY-User-Key-File-version: algorithm-name
Encryption: encryption-type
Comment: key-comment-string
```

version is a decimal number giving the version number of the file format itself. The current file format version is 3.

algorithm-name is the SSH protocol identifier for the public key algorithm that this key is used for (such as ‘ssh-dss’ or ‘ecdsa-sha2-nistp384’).

encryption-type indicates whether this key is stored encrypted, and if so, by what method. Currently the only supported encryption types are ‘aes256-cbc’ and ‘none’.

key-comment-string is a free text field giving the comment. This can contain any byte values other than 13 and 10 (CR and LF).

The next part of the file gives the public key. This is stored unencrypted but base64-encoded ([RFC 4648](#)), and is preceded by a header line saying how many lines of base64 data are shown, looking like this:

```
Public-Lines: number-of-lines
that many lines of base64 data
```

The base64-encoded data in this blob is formatted in exactly the same way as an SSH public key sent over the wire in the SSH protocol itself. That is also the same format as the base64 data

stored in OpenSSH's authorized_keys file, except that in a PPK file the base64 data is split across multiple lines. But if you remove the newlines from the middle of this section, the resulting base64 blob is in the right format to go in an authorized_keys line.

If the key is encrypted (i.e. **encryption-type** is not 'none'), then the next thing that appears is a sequence of lines specifying how the keys for encrypting the file are to be derived from the passphrase:

```
Key-Derivation: argon2-flavour
Argon2-Memory: decimal-integer
Argon2-Passes: decimal-integer
Argon2-Parallelism: decimal-integer
Argon2-Salt: hex-string
```

argon2-flavour is one of the identifiers 'Argon2d', 'Argon2i' or 'Argon2id', all describing variants of the Argon2 password-hashing function.

The three integer values are used as parameters for Argon2, which allows you to configure the amount of memory used (in Kbyte), the number of passes of the algorithm to run (to tune its running time), and the degree of parallelism required by the hash function. The salt is decoded into a sequence of binary bytes and used as an additional input to Argon2. (It is chosen randomly when the key file is written, so that a guessing attack can't be mounted in parallel against multiple key files.)

The next part of the file gives the private key. This is base64-encoded in the same way:

```
Private-Lines: number-of-lines
that many lines of base64 data
```

The binary data represented in this base64 blob may be encrypted, depending on the **encryption-type** field in the key file header shown above:

- If **encryption-type** is 'none', then this data is stored in plain text.

- If **encryption-type** is ‘aes256-cbc’, then this data is encrypted using AES, with a 256-bit key length, in the CBC cipher mode. The key and initialisation vector are derived from the passphrase: see [section C.4](#).

In order to encrypt the private key data with AES, it must be a multiple of 16 bytes (the AES cipher block length). This is achieved by appending random padding to the data before encrypting it. When decoding it after decryption, the random data can be ignored: the internal structure of the data is enough to tell you when you've reached the end of the meaningful part.

Unlike public keys, the binary encoding of private keys is not specified at all in the SSH standards. See [section C.3](#) for details of the private key format for each key type supported by PuTTY.

The final thing in the key file is the MAC:

Private-MAC: **hex-mac-data**

hex-mac-data is a hexadecimal-encoded value, 64 digits long (i.e. 32 bytes), generated using the HMAC-SHA-256 algorithm with the following binary data as input:

- string: the **algorithm-name** header field.
- string: the **encryption-type** header field.
- string: the **key-comment-string** header field.
- string: the binary public key data, as decoded from the base64 lines after the ‘Public-Lines’ header.
- string: the plaintext of the binary private key data, as decoded from the base64 lines after the ‘Private-Lines’ header. If that data was stored encrypted, then the decrypted version of it is used in this MAC preimage, *including* the random padding mentioned above.

The MAC key is derived from the passphrase: see [section C.4](#).

C.3 Private key encodings

This section describes the private key format for each key type supported by PuTTY.

Because the PPK format also contains the public key (and both public and private key are protected by the same MAC to ensure they can't be made inconsistent), there is no need for the private key section of the file to repeat data from the public section. So some of these formats are very short.

In all cases, a decoding application can begin reading from the start of the decrypted private key data, and know when it has read all that it needs. This allows random padding after the meaningful data to be safely ignored.

- [C.3.1 RSA](#)
- [C.3.2 DSA](#)
- [C.3.3 NIST elliptic-curve keys](#)
- [C.3.4 EdDSA elliptic-curve keys \(Ed25519 and Ed448\)](#)

C.3.1 RSA

RSA keys are stored using an **algorithm-name** of ‘ssh-rsa’. (Keys stored like this are also used by the updated RSA signature schemes that use hashes other than SHA-1.)

The public key data has already provided the key modulus and the public encoding exponent. The private data stores:

- `mpint`: the private decoding exponent of the key.
- `mpint`: one prime factor p of the key.
- `mpint`: the other prime factor q of the key. (RSA keys stored in this format are expected to have exactly two prime factors.)
- `mpint`: the multiplicative inverse of q modulo p .

C.3.2 DSA

DSA keys are stored using an **algorithm-name** of ‘ssh-dss’.

The public key data has already provided the key parameters (the large prime p , the small prime q and the group generator g), and the public key y . The private key stores:

- `mpint`: the private key x , which is the discrete logarithm of y in the group generated by $g \bmod p$.

C.3.3 NIST elliptic-curve keys

NIST elliptic-curve keys are stored using one of the following **algorithm-name** values, each corresponding to a different elliptic curve and key size:

- ‘ecdsa-sha2-nistp256’
- ‘ecdsa-sha2-nistp384’
- ‘ecdsa-sha2-nistp521’

The public key data has already provided the public elliptic curve point. The private key stores:

- `mpint`: the private exponent, which is the discrete log of the public point.

C.3.4 EdDSA elliptic-curve keys (Ed25519 and Ed448)

EdDSA elliptic-curve keys are stored using one of the following **algorithm-name** values, each corresponding to a different elliptic curve and key size:

- ‘ssh-ed25519’
- ‘ssh-ed448’

The public key data has already provided the public elliptic curve point. The private key stores:

- `moint`: the private exponent, which is the discrete log of the public point.

C.4 Key derivation

When a key file is encrypted, there are three pieces of key material that need to be computed from the passphrase:

- the key for the symmetric cipher used to encrypt the private key
- the initialisation vector for that cipher encryption
- the key for the MAC.

If **encryption-type** is ‘aes256-cbc’, then the symmetric cipher key is 32 bytes long, and the initialisation vector is 16 bytes (one cipher block). The length of the MAC key is also chosen to be 32 bytes.

If **encryption-type** is ‘none’, then all three of these pieces of data have zero length. (The MAC is still generated and checked in the key file format, but it has a zero-length key.)

If the amount of key material required is not zero, then the passphrase is fed to the Argon2 key derivation function, in whichever mode is described in the ‘Key-Derivation’ header in the key file, with parameters derived from the various ‘Argon2-Parameter:’ headers.

(If the key is unencrypted, then all those headers are omitted, and Argon2 is not run at all.)

Argon2 takes two extra string inputs in addition to the passphrase and the salt: a secret key, and some ‘associated data’. In PPK’s use of Argon2, these are both set to the empty string.

The ‘tag length’ parameter to Argon2 (i.e. the amount of data it is asked to output) is set to the sum of the lengths of all of the data items required, i.e. (cipher key length + IV length + MAC key length). The output data is interpreted as the concatenation of the cipher key, the IV and the MAC key, in that order.

So, for ‘aes256-cbc’, the tag length will be $32+16+32 = 80$ bytes; of the 80 bytes of output data, the first 32 bytes are used as the 256-bit

AES key, the next 16 as the CBC IV, and the final 32 bytes as the HMAC-SHA-256 key.

C.5 Older versions of the PPK format

- [C.5.1 Version 2](#)
- [C.5.2 Version 1](#)

C.5.1 Version 2

PPK version 2 was used by PuTTY 0.52 to 0.74 inclusive.

In PPK version 2, the MAC algorithm used was HMAC-SHA-1 (so the `Private-MAC` line contained only 40 hex digits).

The ‘Key-Derivation:’ header and all the ‘Argon2-Parameter:’ headers were absent. Instead of using Argon2, the key material for encrypting the private blob was derived from the passphrase in a totally different way, as follows.

The cipher key for ‘aes256-cbc’ was constructed by generating two SHA-1 hashes, concatenating them, and taking the first 32 bytes of the result. (So you’d get all 20 bytes of the first hash output, and the first 12 of the second). Each hash preimage was as follows:

- `uint32`: a sequence number. This is 0 in the first hash, and 1 in the second. (The idea was to extend this mechanism to further hashes by continuing to increment the sequence number, if future changes required even longer keys.)
- the passphrase, without any prefix length field.

In PPK v2, the CBC initialisation vector was all zeroes.

The MAC key was 20 bytes long, and was a single SHA-1 hash of the following data:

- the fixed string ‘`putty-private-key-file-mac-key`’, without any prefix length field.
- the passphrase, without any prefix length field. (If the key is stored unencrypted, the passphrase was taken to be the empty string for these purposes.)

C.5.2 Version 1

PPK version 1 was a badly designed format, only used during initial development, and not recommended for production use.

PPK version 1 was never used by a released version of PuTTY. It was only emitted by some early development snapshots between version 0.51 (which did not support SSH-2 public keys at all) and 0.52 (which already used version 2 of this file format). I hope there are no PPK v1 files in use anywhere. But just in case, the old badly designed format is documented here anyway.

In PPK version 1, the input to the MAC does not include any of the header fields or the public key. It is simply the private key data (still in plaintext and including random padding), all by itself (without a wrapping string).

PPK version 1 keys must therefore be rigorously validated after loading, to ensure that the public and private parts of the key were consistent with each other.

PPK version 1 only supported the RSA and DSA key types. For RSA, this validation can be done using only the provided data (since the private key blob contains enough information to reconstruct the public values anyway). But for DSA, that isn't quite enough.

Hence, PPK version 1 DSA keys extended the private data so that immediately after x was stored an extra value:

- string: a SHA-1 hash of the public key data, whose preimage consists of
 - string: the large prime p
 - string: the small prime q
 - string: the group generator g

The idea was that checking this hash would verify that the key parameters had not been tampered with, and then the loading

application could directly verify that $g^x = y$.

In an *unencrypted* version 1 key file, the MAC is replaced by a plain SHA-1 hash of the private key data. This is indicated by the ‘Private-MAC:’ header being replaced with ‘Private-Hash:’ instead.

Appendix D: PuTTY Licence

PuTTY is copyright 1997-2024 Simon Tatham.

Portions copyright Robert de Bath, Joris van Rantwijk, Delian Delchev, Andreas Schultz, Jeroen Massar, Wez Furlong, Nicolas Barry, Justin Bradford, Ben Harris, Malcolm Smith, Ahmad Khalifa, Markus Kuhn, Colin Watson, Christopher Staite, Lorenz Diener, Christian Brabandt, Jeff Smith, Pavel Kryukov, Maxim Kuznetsov, Svyatoslav Kuzmich, Nico Williams, Viktor Dukhovni, Josh Dersch, Lars Brinkhoff, and CORE SDI S.A.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ‘Software’), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ‘AS IS’, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Appendix E: PuTTY hacking guide

This appendix lists a selection of the design principles applying to the PuTTY source code. If you are planning to send code contributions, you should read this first.

- [E.1 Cross-OS portability](#)
- [E.2 Multiple backends treated equally](#)
- [E.3 Multiple sessions per process on some platforms](#)
- [E.4 C, not C++](#)
- [E.5 Security-conscious coding](#)
- [E.6 Independence of specific compiler](#)
- [E.7 Small code size](#)
- [E.8 Single-threaded code](#)
- [E.9 Keystrokes sent to the server wherever possible](#)
- [E.10 640×480 friendliness in configuration panels](#)
- [E.11 Coroutines in protocol code](#)
- [E.12 Explicit vtable structures to implement traits](#)
- [E.13 Do as we say, not as we do](#)

E.1 Cross-OS portability

Despite Windows being its main area of fame, PuTTY is no longer a Windows-only application suite. It has a working Unix port; a Mac port is in progress; more ports may or may not happen at a later date.

Therefore, embedding Windows-specific code in core modules such as `ssh.c` is not acceptable. We went to great lengths to *remove* all the Windows-specific stuff from our core modules, and to shift it out into Windows-specific modules. Adding large amounts of Windows-specific stuff in parts of the code that should be portable is almost guaranteed to make us reject a contribution.

The PuTTY source base is divided into platform-specific modules and platform-generic modules. The Unix-specific modules are all in the `unix` subdirectory; the Windows-specific modules are in the `windows` subdirectory.

All the modules in the main source directory and other subdirectories - notably *all* of the code for the various back ends - are platform-generic. We want to keep them that way.

This also means you should stick to the C semantics guaranteed by the C standard: try not to make assumptions about the precise size of basic types such as `int` and `long int`; don't use pointer casts to do endianness-dependent operations, and so on.

(Even *within* a platform front end you should still be careful of some of these portability issues. The Windows front end compiles on both 32- and 64-bit x86 and also Arm.)

Our current choice of C standards version is *mostly* C99. With a couple of exceptions, you can assume that C99 features are available (in particular `<stdint.h>`, `<stdbool.h>` and `inline`), but you shouldn't use things that are new in C11 (such as `<uchar.h>` or `_Generic`).

The exceptions to that rule are due to the need for Visual Studio compatibility:

- Don't use variable-length arrays. Visual Studio doesn't support them even now that it's adopted the rest of C99. We use `-Wvla` when building with `gcc` and `clang`, to make it easier to avoid accidentally breaking that rule.
- For historical reasons, we still build with one older VS version which lacks `<inttypes.h>`. So that file is included centrally in `defs.h`, and has a set of workaround definitions for the `PRIx64`-type macros we use. If you need to use another one of those macros, you need to add a workaround definition in `defs.h`, and don't casually re-include `<inttypes.h>` anywhere else in the source file.

Here are a few portability assumptions that we *do* currently allow (because we'd already have to thoroughly vet the existing code if they ever needed to change, and it doesn't seem worth doing that unless we really have to):

- You can assume `int` is *at least* 32 bits wide. (We've never tried to port PuTTY to a platform with 16-bit `int`, and it doesn't look likely to be necessary in future.)
- Similarly, you can assume `char` is exactly 8 bits. (Exceptions to that are even less likely to be relevant to us than `short int`.)
- You can assume that using `memset` to write zero bytes over a whole structure will have the effect of setting all its pointer fields to `NULL`. (The standard itself guarantees this for *integer* fields, but not for pointers.)
- You can assume that `time_t` has POSIX semantics, i.e. that it represents an integer number of non-leap seconds since 1970-01-01 00:00:00 UTC. (Times in this format are used in X authorisation, but we could work around that by carefully distinguishing local `time_t` from time values used in the wire protocol; but these semantics of `time_t` are also baked into the shared library API used by the GSSAPI authentication code, which would be much harder to change.)

- You can assume that the execution character encoding is a superset of the printable characters of ASCII. (In particular, it's fine to do arithmetic on a `char` value representing a Latin alphabetic character, without bothering to allow for EBCDIC or other non-consecutive encodings of the alphabet.)

On the other hand, here are some particular things *not* to assume:

- Don't assume anything about the *signedness* of `char`. In particular, you *must* cast `char` values to `unsigned char` before passing them to any `<cctype.h>` function (because those expect a non-negative character value, or `EOF`). If you need a particular signedness, explicitly specify `signed char` or `unsigned char`, or use C99 `int8_t` or `uint8_t`.
- From past experience with MacOS, we're still a bit nervous about '`\n`' and '`\r`' potentially having unusual meanings on a given platform. So it's fine to say `\n` in a string you're passing to `printf`, but in any context where those characters appear in a standardised wire protocol or a binary file format, they should be spelled '`\012`' and '`\015`' respectively.

E.2 Multiple backends treated equally

PuTTY is not an SSH client with some other stuff tacked on the side. PuTTY is a generic, multiple-backend, remote VT-terminal client which happens to support one backend which is larger, more popular and more useful than the rest. Any extra feature which can possibly be general across all backends should be so: localising features unnecessarily into the SSH back end is a design error. (For example, we had several code submissions for proxy support which worked by hacking `ssh.c`. Clearly this is completely wrong: the `network.h` abstraction is the place to put it, so that it will apply to all back ends equally, and indeed we eventually put it there after another contributor sent a better patch.) The rest of PuTTY should try to avoid knowing anything about specific back ends if at all possible. To support a feature which is only available in one network protocol, for example, the back end interface should be extended in a general manner such that *any* back end which is able to provide that feature can do so. If it so happens that only one back end actually does, that's just the way it is, but it shouldn't be relied upon by any code.

E.3 Multiple sessions per process on some platforms

Some ports of PuTTY - notably the in-progress Mac port - are constrained by the operating system to run as a single process potentially managing multiple sessions.

Therefore, the platform-independent parts of PuTTY never use global variables to store per-session data. The global variables that do exist are tolerated because they are not specific to a particular login session. The random number state in `sshrand.c`, the timer list in `timing.c` and the queue of top-level callbacks in `callback.c` serve all sessions equally. But most data is specific to a particular network session, and is therefore stored in dynamically allocated data structures, and pointers to these structures are passed around between functions.

Platform-specific code can reverse this decision if it likes. The Windows code, for historical reasons, stores most of its data as global variables. That's OK, because *on Windows* we know there is only one session per PuTTY process, so it's safe to do that. But changes to the platform-independent code should avoid introducing global variables, unless they are genuinely cross-session.

E.4 C, not C++

PuTTY is written entirely in C, not in C++.

We have made *some* effort to make it easy to compile our code using a C++ compiler: notably, our `snew`, `snewn` and `sresize` macros explicitly cast the return values of `malloc` and `realloc` to the target type. (This has type checking advantages even in C: it means you never accidentally allocate the wrong size piece of memory for the pointer type you're assigning it to. C++ friendliness is really a side benefit.) We want PuTTY to continue being pure C, at least in the platform-independent parts and the currently existing ports. Patches which switch the Makefiles to compile it as C++ and start using classes will not be accepted.

The one exception: a port to a new platform may use languages other than C if they are necessary to code on that platform. If your favourite PDA has a GUI with a C++ API, then there's no way you can do a port of PuTTY without using C++, so go ahead and use it. But keep the C++ restricted to that platform's subdirectory; if your changes force the Unix or Windows ports to be compiled as C++, they will be unacceptable to us.

E.5 Security-conscious coding

PuTTY is a network application and a security application. Assume your code will end up being fed deliberately malicious data by attackers, and try to code in a way that makes it unlikely to be a security risk.

In particular, try not to use fixed-size buffers for variable-size data such as strings received from the network (or even the user). We provide functions such as `dupcat` and `dupprintf`, which dynamically allocate buffers of the right size for the string they construct. Use these wherever possible.

E.6 Independence of specific compiler

Windows PuTTY can currently be compiled with any of three Windows compilers: MS Visual C, the Cygwin / mingw32 GNU tools, and clang (in MS compatibility mode).

This is a really useful property of PuTTY, because it means people who want to contribute to the coding don't depend on having a specific compiler; so they don't have to fork out money for MSVC if they don't already have it, but on the other hand if they *do* have it they also don't have to spend effort installing gcc alongside it. They can use whichever compiler they happen to have available, or install whichever is cheapest and easiest if they don't have one.

Therefore, we don't want PuTTY to start depending on which compiler you're using. Using GNU extensions to the C language, for example, would ruin this useful property (not that anyone's ever tried it!); and more realistically, depending on an MS-specific library function supplied by the MSVC C library (`_snprintf`, for example) is a mistake, because that function won't be available under the other compilers. Any function supplied in an official Windows DLL as part of the Windows API is fine, and anything defined in the C library standard is also fine, because those should be available irrespective of compilation environment. But things in between, available as non-standard library and language extensions in only one compiler, are disallowed.

(`_snprintf` in particular should be unnecessary, since we provide `dupprintf`; see [section E.5](#).)

Compiler independence should apply on all platforms, of course, not just on Windows.

E.7 Small code size

PuTTY is tiny, compared to many other Windows applications. And it's easy to install: it depends on no DLLs, no other applications, no service packs or system upgrades. It's just one executable. You install that executable wherever you want to, and run it.

We want to keep both these properties - the small size, and the ease of installation - if at all possible. So code contributions that depend critically on external DLLs, or that add a huge amount to the code size for a feature which is only useful to a small minority of users, are likely to be thrown out immediately.

We do vaguely intend to introduce a DLL plugin interface for PuTTY, whereby seriously large extra features can be implemented in plugin modules. The important thing, though, is that those DLLs will be *optional*; if PuTTY can't find them on startup, it should run perfectly happily and just won't provide those particular features. A full installation of PuTTY might one day contain ten or twenty little DLL plugins, which would cut down a little on the ease of installation - but if you really needed ease of installation you *could* still just install the one PuTTY binary, or just the DLLs you really needed, and it would still work fine.

Depending on *external* DLLs is something we'd like to avoid if at all possible (though for some purposes, such as complex SSH authentication mechanisms, it may be unavoidable). If it can't be avoided, the important thing is to follow the same principle of graceful degradation: if a DLL can't be found, then PuTTY should run happily and just not supply the feature that depended on it.

E.8 Single-threaded code

PuTTY and its supporting tools, or at least the vast majority of them, run in only one OS thread.

This means that if you're devising some piece of internal mechanism, there's no need to use locks to make sure it doesn't get called by two threads at once. The only way code can be called re-entrantly is by recursion.

That said, most of Windows PuTTY's network handling is triggered off Windows messages requested by `wsAAAsyncSelect()`, so if you call `MessageBox()` deep within some network event handling code you should be aware that you might be re-entered if a network event comes in and is passed on to our window procedure by the `MessageBox()` message loop.

Also, the front ends can use multiple threads if they like. For example, the Windows front-end code spawns subthreads to deal with bidirectional blocking I/O on non-network streams such as Windows pipes. However, it keeps tight control of its auxiliary threads, and uses them only for that one purpose, as a form of `select()`. Pretty much all the code outside `windows/handle-io.c` is *only* ever called from the one primary thread; the others just loop round blocking on file handles, and signal the main thread (via Windows event objects) when some real work needs doing. This is not considered a portability hazard because that code is already Windows-specific and needs rewriting on other platforms.

One important consequence of this: PuTTY has only one thread in which to do everything. That 'everything' may include managing more than one login session ([section E.3](#)), managing multiple data channels within an SSH session, responding to GUI events even when nothing is happening on the network, and responding to network requests from the server (such as repeat key exchange) even when the program is dealing with complex user interaction

such as the re-configuration dialog box. This means that *almost none* of the PuTTY code can safely block.

E.9 Keystrokes sent to the server wherever possible

In almost all cases, PuTTY sends keystrokes to the server. Even weird keystrokes that you think should be hot keys controlling PuTTY. Even Alt-F4 or Alt-Space, for example. If a keystroke has a well-defined escape sequence that it could usefully be sending to the server, then it should do so, or at the very least it should be configurably able to do so.

To unconditionally turn a key combination into a hot key to control PuTTY is almost always a design error. If a hot key is really truly required, then try to find a key combination for it which *isn't* already used in existing PuTTYS (either it sends nothing to the server, or it sends the same thing as some other combination). Even then, be prepared for the possibility that one day that key combination might end up being needed to send something to the server - so make sure that there's an alternative way to invoke whatever PuTTY feature it controls.

E.10 640×480 friendliness in configuration panels

There's a reason we have lots of tiny configuration panels instead of a few huge ones, and that reason is that not everyone has a 1600×1200 desktop. 640×480 is still a viable resolution for running Windows (and indeed it's still the default if you start up in safe mode), so it's still a resolution we care about.

Accordingly, the PuTTY configuration box, and the PuTTYgen control window, are deliberately kept just small enough to fit comfortably on a 640×480 display. If you're adding controls to either of these boxes and you find yourself wanting to increase the size of the whole box, *don't*. Split it into more panels instead.

E.11 Coroutines in protocol code

Large parts of the code in modules implementing wire protocols (mainly SSH) are structured using a set of macros that implement (something close to) Donald Knuth's 'coroutines' concept in C.

Essentially, the purpose of these macros are to arrange that a function can call `crReturn()` to return to its caller, and the next time it is called control will resume from just after that `crReturn` statement.

This means that any local (automatic) variables declared in such a function will be corrupted every time you call `crReturn`. If you need a variable to persist for longer than that, you *must* make it a field in some appropriate structure containing the persistent state of the coroutine – typically the main state structure for a protocol layer.

See

<https://www.chiark.greenend.org.uk/~sgtatham/coroutines.html> for a more in-depth discussion of what these macros are for and how they work.

Another caveat: most of these coroutines are not *guaranteed* to run to completion, because the SSH connection (or whatever) that they're part of might be interrupted at any time by an unexpected network event or user action. So whenever a coroutine-managed variable refers to a resource that needs releasing, you should also ensure that the cleanup function for its containing state structure can reliably release it even if the coroutine is aborted at an arbitrary point.

For example, if an SSH packet protocol layer has to have a field that sometimes points to a piece of allocated memory, then you should ensure that when you free that memory you reset the pointer field to `NULL`. Then, no matter when the protocol layer's cleanup function is called, it can reliably free the memory if there is any, and not crash if there isn't.

E.12 Explicit vtable structures to implement traits

A lot of PuTTY's code is written in a style that looks structurally rather like an object-oriented language, in spite of PuTTY being a pure C program.

For example, there's a single data type called `ssh_hash`, which is an abstraction of a secure hash function, and a bunch of functions called things like `ssh_hash_foo` that do things with those data types. But in fact, PuTTY supports many different hash functions, and each one has to provide its own implementation of those functions.

In C++ terms, this is rather like having a single abstract base class, and multiple concrete subclasses of it, each of which fills in all the pure virtual methods in a way that's compatible with the data fields of the subclass. The implementation is more or less the same, as well: in C, we do explicitly in the source code what the C++ compiler will be doing behind the scenes at compile time.

But perhaps a closer analogy in functional terms is the Rust concept of a 'trait', or the Java idea of an 'interface'. C++ supports a multi-level hierarchy of inheritance, whereas PuTTY's system – like traits or interfaces – has only two levels, one describing a generic object of a type (e.g. a hash function) and another describing a specific implementation of that type (e.g. SHA-256).

The PuTTY code base has a standard idiom for doing this in C, as follows.

Firstly, we define two `struct` types for our trait. One of them describes a particular *kind* of implementation of that trait, and it's full of (mostly) function pointers. The other describes a specific *instance* of an implementation of that trait, and it will contain a pointer to a `const` instance of the first type. For example:

```

typedef struct MyAbstraction MyAbstraction;
typedef struct MyAbstractionVtable MyAbstractionVtable;

struct MyAbstractionVtable {
    MyAbstraction *(*new)(const MyAbstractionVtable *vt);
    void (*free)(MyAbstraction *);
    void (*modify)(MyAbstraction *, unsigned some_parameter);
    unsigned (*query)(MyAbstraction *, unsigned
some_parameter);
};

struct MyAbstraction {
    const MyAbstractionVtable *vt;
};

```

Here, we imagine that `MyAbstraction` might be some kind of object that contains mutable state. The associated vtable structure shows what operations you can perform on a `MyAbstraction`: you can create one (dynamically allocated), free one you already have, or call the example methods ‘`modify`’ (to change the state of the object in some way) and ‘`query`’ (to return some value derived from the object’s current state).

(In most cases, the vtable structure has a name ending in ‘`vtable`’. But for historical reasons a lot of the crypto primitives that use this scheme – ciphers, hash functions, public key methods and so on – instead have names ending in ‘`alg`’, on the basis that the primitives they implement are often referred to as ‘encryption algorithms’, ‘hash algorithms’ and so forth.)

Now, to define a concrete instance of this trait, you’d define a `struct` that contains a `MyAbstraction` field, plus any other data it might need:

```

struct MyImplementation {
    unsigned internal_data[16];
    SomeOtherType *dynamic_subthing;

    MyAbstraction myabs;
};

```

Next, you’d implement all the necessary methods for that implementation of the trait, in this kind of style:

```

static MyAbstraction *myimpl_new(const MyAbstractionVtable *vt)
{
    MyImplementation *impl = snew(MyImplementation);
    memset(impl, 0, sizeof(*impl));
    impl->dynamic_subthing = allocate_some_other_type();
    impl->myabs.vt = vt;
    return &impl->myabs;
}

static void myimpl_free(MyAbstraction *myabs)
{
    MyImplementation *impl = container_of(myabs,
MyImplementation, myabs);
    free_other_type(impl->dynamic_subthing);
    sfree(impl);
}

static void myimpl_modify(MyAbstraction *myabs, unsigned param)
{
    MyImplementation *impl = container_of(myabs,
MyImplementation, myabs);
    impl->internal_data[param] += do_something_with(impl-
>dynamic_subthing);
}

static unsigned myimpl_query(MyAbstraction *myabs, unsigned
param)
{
    MyImplementation *impl = container_of(myabs,
MyImplementation, myabs);
    return impl->internal_data[param];
}

```

Having defined those methods, now we can define a `const` instance of the vtable structure containing pointers to them:

```

const MyAbstractionVtable MyImplementation_vt = {
    .new = myimpl_new,
    .free = myimpl_free,
    .modify = myimpl_modify,
    .query = myimpl_query,
};

```

In principle, this is all you need. Client code can construct a new instance of a particular implementation of `MyAbstraction` by digging out the `new` method from the vtable and calling it (with the vtable itself

as a parameter), which returns a `MyAbstraction *` pointer that identifies a newly created instance, in which the `vt` field will contain a pointer to the same vtable structure you passed in. And once you have an instance object, say `MyAbstraction *myabs`, you can dig out one of the other method pointers from the vtable it points to, and call that, passing the object itself as a parameter.

But in fact, we don't do that, because it looks pretty ugly at all the call sites. Instead, what we generally do in this code base is to write a set of static inline wrapper functions in the same header file that defined the `MyAbstraction` structure types, like this:

```
static inline MyAbstraction *myabs_new(const
MyAbstractionVtable *vt)
{ return vt->new(vt); }
static inline void myabs_free(MyAbstraction *myabs)
{ myabs->vt->free(myabs); }
static inline void myimpl_modify(MyAbstraction *myabs, unsigned
param)
{ myabs->vt->modify(myabs, param); }
static inline unsigned myimpl_query(MyAbstraction *myabs,
unsigned param)
{ return myabs->vt->query(myabs, param); }
```

And now call sites can use those reasonably clean-looking wrapper functions, and shouldn't ever have to directly refer to the `vt` field inside any `myabs` object they're holding. For example, you might write something like this:

```
MyAbstraction *myabs = myabs_new(&MyImplementation_vtable);
myabs_update(myabs, 10);
unsigned output = myabs_query(myabs, 2);
myabs_free(myabs);
```

and then all this code can use a different implementation of the same abstraction by just changing which vtable pointer it passed in in the first line.

Some things to note about this system:

- The implementation instance type (here ‘`MyImplementation`’) contains the abstraction type (‘`MyAbstraction`’) as one of its fields. But that field is not necessarily at the start of the structure. So you can't just *cast* pointers back and forth between the two types. Instead:
 - You ‘up-cast’ from implementation to abstraction by taking the address of the `MyAbstraction` field. You can see the example `new` method above doing this, returning `&impl->myabs`. All new methods do this on return.
 - Going in the other direction, each method that was passed a generic `MyAbstraction *myabs` parameter has to recover a pointer to the specific implementation type `MyImplementation *impl`. The idiom for doing that is to use the ‘`container_of`’ macro, also seen in the Linux kernel code. Generally, `container_of(p, Type, field)` says: ‘I'm confident that the pointer value ‘`p`’ is pointing to the field called ‘`field`’ within a larger `struct` of type `Type`. Please return me the pointer to the containing structure.’ So in this case, we take the ‘`myabs`’ pointer passed to the function, and ‘down-cast’ it into a pointer to the larger and more specific structure type `MyImplementation`, by adjusting the pointer value based on the offset within that structure of the field called ‘`myabs`’.

This system is flexible enough to permit ‘multiple inheritance’, or rather, multiple *implementation*: having one object type implement more than one trait. For example, the `ProxySocket` type implements both the `Socket` trait and the `Plug` trait that connects to it, because it has to act as an adapter between another instance of each of those types.

It's also perfectly possible to have the same object implement the *same* trait in two different ways. At the time of writing this I can't think of any case where we actually do this, but a theoretical example might be if you needed to support a trait like `Comparable` in two ways that sorted by different criteria. There would be no difficulty doing this in the PuTTY system: simply

have your implementation struct contain two (or more) fields of the same abstraction type. The fields will have different names, which makes it easy to explicitly specify which one you're returning a pointer to during up-casting, or which one you're down-casting from using `container_of`. And then both sets of implementation methods can recover a pointer to the same containing structure.

- Unlike in C++, all objects in PuTTY that use this system are dynamically allocated. The ‘constructor’ functions (whether they’re virtualised across the whole abstraction or specific to each implementation) always allocate memory and return a pointer to it. The ‘free’ method (our analogue of a destructor) always expects the input pointer to be dynamically allocated, and frees it. As a result, client code doesn’t need to know how large the implementing object type is, because it will never need to allocate it (on the stack or anywhere else).
- Unlike in C++, the abstraction’s ‘vtable’ structure does not only hold methods that you can call on an instance object. It can also hold several other kinds of thing:
 - Methods that you can call *without* an instance object, given only the vtable structure identifying a particular implementation of the trait. You might think of these as ‘static methods’, as in C++, except that they’re *virtual* – the same code can call the static method of a different ‘class’ given a different vtable pointer. So they’re more like ‘virtual static methods’, which is a concept C++ doesn’t have. An example is the `pubkey_bits` method in `ssh_keyalg`.
 - The most important case of a ‘virtual static method’ is the new method that allocates and returns a new object. You can think of it as a ‘virtual constructor’ – another concept C++ doesn’t have. (However, not all types need one of these: see below.)
 - The vtable can also contain constant data relevant to the class as a whole – ‘virtual constant data’. For example, a cryptographic hash function will contain an integer field giving the length of the output hash, and most crypto

primitives will contain a string field giving the identifier used in the SSH protocol that describes that primitive.

The effect of all of this is that you can make other pieces of code able to use any instance of one of these types, by passing it an actual vtable as a parameter. For example, the `hash_simple` function takes an `ssh_hashalg` vtable pointer specifying any hash algorithm you like, and internally, it creates an object of that type, uses it, and frees it. In C++, you'd probably do this using a template, which would mean you had multiple specialisations of `hash_simple` – and then it would be much more difficult to decide at *run time* which one you needed to use. Here, `hash_simple` is still just one function, and you can decide as late as you like which vtable to pass to it.

- The abstract *instance* structure can also contain publicly visible data fields (this time, usually treated as mutable) which are common to all implementations of the trait. For example, `BinaryPacketProtocol` has lots of these.
- Not all abstractions of this kind want virtual constructors. It depends on how different the implementations are.

With a crypto primitive like a hash algorithm, the constructor call looks the same for every implementing type, so it makes sense to have a standardised virtual constructor in the vtable and a `ssh_hash_new` wrapper function which can make an instance of whatever vtable you pass it. And then you make all the vtable objects themselves globally visible throughout the source code, so that any module can call (for example)
`ssh_hash_new(&ssh_sha256).`

But with other kinds of object, the constructor for each implementing type has to take a different set of parameters. For example, implementations of `Socket` are not generally interchangeable at construction time, because constructing different kinds of socket require totally different kinds of address parameter. In that situation, it makes more sense to keep the vtable structure itself private to the implementing source file, and

instead, publish an ordinary constructing function that allocates and returns an instance of that particular subtype, taking whatever parameters are appropriate to that subtype.

- If you do have virtual constructors, you can choose whether they take a vtable pointer as a parameter (as shown above), or an *existing* instance object. In the latter case, they can refer to the object itself as well as the vtable. For example, you could have a trait come with a virtual constructor called ‘clone’, meaning ‘Make a copy of this object, no matter which implementation it is.’
- Sometimes, a single vtable structure type can be shared between two completely different object types, and contain all the methods for both. For example, `ssh_compression_alg` contains methods to create, use and free `ssh_compressor` and `ssh_decompressor` objects, which are not interchangeable – but putting their methods in the same vtable means that it’s easy to create a matching pair of objects that are compatible with each other.
- Passing the vtable itself as an argument to the `new` method is not compulsory: if a given `new` implementation is only used by a single vtable, then that function can simply hard-code the vtable pointer that it writes into the object it constructs. But passing the vtable is more flexible, because it allows a single constructor function to be shared between multiple slightly different object types. For example, SHA-384 and SHA-512 share the same `new` method and the same implementation data type, because they’re very nearly the same hash algorithm – but a couple of the other methods in their vtables are different, because the ‘reset’ function has to set up the initial algorithm state differently, and the ‘digest’ method has to write out a different amount of data.

One practical advantage of having the `myabs_foo` family of inline wrapper functions in the header file is that if you change your mind later about whether the vtable needs to be passed to `new`,

you only have to update the `myabs_new` wrapper, and then the existing call sites won't need changing.

- Another piece of ‘stunt object orientation’ made possible by this scheme is that you can write two vtables that both use the same structure layout for the implementation object, and have an object *transform from one to the other* part way through its lifetime, by overwriting its own vtable pointer field. For example, the `sesschan` type that handles the server side of an SSH terminal session will sometimes transform in mid-lifetime into an SCP or SFTP file-transfer channel in this way, at the point where the client sends an ‘exec’ or ‘subsystem’ request that indicates that that’s what it wants to do with the channel.

This concept would be difficult to arrange in C++. In Rust, it wouldn’t even *make sense*, because in Rust, objects implementing a trait don’t even contain a vtable pointer at all – instead, the ‘trait object’ type (identifying a specific instance of some implementation of a given trait) consists of a pair of pointers, one to the object itself and one to the vtable. In that model, the only way you could make an existing object turn into a different trait would be to know where all the pointers to it were stored elsewhere in the program, and persuade all their owners to rewrite them.

- Another stunt you can do is to have a vtable that doesn’t have a corresponding implementation structure at all, because the only methods implemented in it are the constructors, and they always end up returning an implementation of some other vtable. For example, some of PuTTY’s crypto primitives have a hardware-accelerated version and a pure software version, and decide at run time which one to use (based on whether the CPU they’re running on supports the necessary acceleration instructions). So, for example, there are vtables for `ssh_sha256_sw` and `ssh_sha256_hw`, each of which has its own data layout and its own implementations of all the methods; and then there’s a top-level vtable `ssh_sha256`, which only provides the ‘new’ method,

and implements it by calling the ‘new’ method on one or other of the subtypes depending on what it finds out about the machine it’s running on. That top-level selector vtable is nearly always the one used by client code. (Except for the test suite, which has to instantiate both of the subtypes in order to make sure they both pass the tests.)

As a result, the top-level selector vtable `ssh_sha256` doesn’t need to implement any method that takes an `ssh_cipher *` parameter, because no `ssh_cipher` object is ever constructed whose `vt` field points to `&ssh_sha256`: they all point to one of the other two full implementation vtables.

E.13 Do as we say, not as we do

The current PuTTY code probably does not conform strictly to *all* of the principles listed above. There may be the occasional SSH-specific piece of code in what should be a backend-independent module, or the occasional dependence on a non-standard X library function under Unix.

This should not be taken as a licence to go ahead and violate the rules. Where we violate them ourselves, we're not happy about it, and we would welcome patches that fix any existing problems. Please try to help us make our code better, not worse!

Appendix F: PuTTY download keys and signatures

We create GPG signatures for all the PuTTY files distributed from our web site, so that users can be confident that the files have not been tampered with. Here we identify our public keys, and explain our signature policy so you can have an accurate idea of what each signature guarantees. This description is provided as both a web page on the PuTTY site, and an appendix in the PuTTY manual.

As of release 0.58, all of the PuTTY executables contain fingerprint material (usually accessed via the `-pgpfp` command-line option), such that if you have an executable you trust, you can use it to establish a trust path, for instance to a newer version downloaded from the Internet.

As of release 0.67, the Windows executables and installer also contain built-in signatures that are automatically verified by Windows' own mechanism ('Authenticode'). The keys used for that are different, and are not covered here.

(Note that none of the keys, signatures, etc mentioned here have anything to do with keys used with SSH - they are purely for verifying the origin of files distributed by the PuTTY team.)

- [F.1 Public keys](#)
- [F.2 Security details](#)
 - [F.2.1 The Development Snapshots key](#)
 - [F.2.2 The Releases key](#)
 - [F.2.3 The Secure Contact Key](#)
 - [F.2.4 The Master Keys](#)
- [F.3 Key rollover](#)

F.1 Public keys

We maintain multiple keys, stored with different levels of security due to being used in different ways. See [section F.2](#) below for details.

The keys we provide are:

Snapshot Key

Used to sign routine development builds of PuTTY: nightly snapshots, pre-releases, and sometimes also custom diagnostic builds we send to particular users.

Release Key

Used to sign manually released versions of PuTTY.

Secure Contact Key

An encryption-capable key suitable for people to send confidential messages to the PuTTY team, e.g. reports of vulnerabilities.

Master Key

Used to tie all the above keys into the GPG web of trust. The Master Key signs all the other keys, and other GPG users have signed it in turn.

The current issue of those keys are available for download from the PuTTY website, and are also available on PGP keyservers using the key IDs listed below.

[Master Key \(2023\)](#)

RSA, 4096-bit. Key ID: B15D9EFC216B06A1. Fingerprint: 28D4 7C46
55E7 65A6 D827 AC66 B15D 9EFC 216B 06A1

[Release Key \(2023\)](#)

RSA, 3072-bit. Key ID: 1993D21BCAD1AA77. Fingerprint: F412 BA3A
A30F DC0E 77B4 E387 1993 D21B CAD1 AA77

Snapshot Key_(2023)

RSA, 3072-bit. Key ID: 10625E553F53FAAD. Fingerprint: 74CC 6DD9
ABA7 31D4 C5A0 C2D0 1062 5E55 3F53 FAAD

Secure Contact Key_(2023)

RSA, 3072-bit. Key ID: 1559F6A8929F5EFC. Fingerprint: 01F5 A2B1
1388 D64B 707F 897F 1559 F6A8 929F 5EFC

F.2 Security details

The various keys have various different security levels. This section explains what those security levels are, and how far you can expect to trust each key.

- [F.2.1 The Development Snapshots key](#)
- [F.2.2 The Releases key](#)
- [F.2.3 The Secure Contact Key](#)
- [F.2.4 The Master Keys](#)

F.2.1 The Development Snapshots key

The Development Snapshots private key is stored *without a passphrase*. This is necessary, because the snapshots are generated every night without human intervention, so nobody would be able to type a passphrase.

The snapshots are built and signed on a team member's home computers, before being uploaded to the web server from which you download them.

Therefore, a signature from the Development Snapshots key *DOES* protect you against:

- People tampering with the PuTTY binaries between the PuTTY web site and you.
- The maintainers of our web server attempting to abuse their root privilege to tamper with the binaries.

But it *DOES NOT* protect you against:

- People tampering with the binaries before they are uploaded to our download servers.
- People tampering with the build machines so that the next set of binaries they build will be malicious in some way.
- People stealing the unencrypted private key from the build machine it lives on.

Of course, we take all reasonable precautions to guard the build machines. But when you see a signature, you should always be certain of precisely what it guarantees and precisely what it does not.

F.2.2 The Releases key

The Releases key is more secure: because it is only used at release time, to sign each release by hand, we can store it encrypted.

The Releases private key is kept encrypted on the developers' own local machines. So an attacker wanting to steal it would have to also steal the passphrase.

F.2.3 The Secure Contact Key

The Secure Contact Key is stored with a similar level of security to the Release Key: it is stored with a passphrase, and no automated script has access to it.

F.2.4 The Master Keys

The Master Key signs almost nothing. Its purpose is to bind the other keys together and certify that they are all owned by the same people and part of the same integrated setup. The only signatures produced by the Master Key, ever, should be the signatures on the other keys.

The Master Key is especially long, and its private key and passphrase are stored with special care.

We have collected some third-party signatures on the Master Key, in order to increase the chances that you can find a suitable trust path to them.

We have uploaded our various keys to public keyservers, so that even if you don't know any of the people who have signed our keys, you can still be reasonably confident that an attacker would find it hard to substitute fake keys on all the public keyservers at once.

F.3 Key rollover

Our current keys were generated in July 2023.

Each new Master Key is signed with the old one, to show that it really is owned by the same people and not substituted by an attacker.

Each new Master Key also signs the previous Release Keys, in case you're trying to verify the signatures on a release prior to the rollover and can find a chain of trust to those keys from any of the people who have signed our new Master Key.

Each release is signed with the Release Key that was current at the time of release. We don't go back and re-sign old releases with newly generated keys.

The details of all previous keys are given here.

Keys generated in the 2021 rollover

Master Key (2021)

RSA, 3072-bit. Key ID: DD4355EAAC1119DE. Fingerprint: A872 D42F
1660 890F 0E05 223E DD43 55EA AC11 19DE

Release Key (2021)

RSA, 3072-bit. Key ID: E4F83EA2AA4915EC. Fingerprint: 2CF6 134B
D3F7 7A65 88EB D668 E4F8 3EA2 AA49 15EC

Snapshot Key (2021)

RSA, 3072-bit. Key ID: B43979F89F446CFD. Fingerprint: 1FD3 BCAC
E532 FBE0 6A8C 09E2 B439 79F8 9F44 6CFD

Secure Contact Key (2021)

RSA, 3072-bit. Key ID: 012C59D4211BD62A. Fingerprint: E30F 1354
2A04 BE0E 56F0 5801 012C 59D4 211B D62A

Keys generated in the 2018 rollover

Master Key (2018)

RSA, 4096-bit. Key ID: 76BC7FE4EBFD2D9E. Fingerprint: 24E1 B1C5
75EA 3C9F F752 A922 76BC 7FE4 EBFD 2D9E

Release Key (2018)

RSA, 3072-bit. Key ID: 6289A25F4AE8DA82. Fingerprint: E273 94AC
A3F9 D904 9522 E054 6289 A25F 4AE8 DA82

Snapshot Key (2018)

RSA, 3072-bit. Key ID: 38BA7229B7588FD1. Fingerprint: C92B 52E9
9AB6 1DDA 33DB 2B7A 38BA 7229 B758 8FD1

Secure Contact Key (2018)

RSA, 3072-bit. Key ID: 657D487977F95C98. Fingerprint: A680 0082
2998 6E46 22CA 0E43 657D 4879 77F9 5C98

Key generated in 2016 (when we first introduced the Secure Contact Key)

Secure Contact Key (2016)

RSA, 2048-bit. Main key ID: 2048R/8A0AF00B (long version:
2048R/C4FCAAD08A0AF00B). Encryption subkey ID: 2048R/50C2CF5C
(long version: 2048R/9EB39CC150C2CF5C). Fingerprint: 8A26 250E
763F E359 75F3 118F C4FC AAD0 8A0A F00B

Keys generated in the 2015 rollover

Master Key (2015)

RSA, 4096-bit. Key ID: 4096R/04676F7C (long version:
4096R/AB585DC604676F7C). Fingerprint: 440D E3B5 B7A1 CA85 B3CC
1718 AB58 5DC6 0467 6F7C

Release Key (2015)

RSA, 2048-bit. Key ID: 2048R/B43434E4 (long version:
2048R/9DFE2648B43434E4). Fingerprint: 0054 DDAA 8ADA 15D2 768A
6DE7 9DFE 2648 B434 34E4

Snapshot Key (2015)

RSA, 2048-bit. Key ID: 2048R/D15F7E8A (long version:
2048R/EEF20295D15F7E8A). Fingerprint: 0A3B 0048 FE49 9B67 A234
FEB6 EEF2 0295 D15F 7E8A

Original keys generated in 2000 (two sets, RSA and DSA)

Master Key (original RSA)

RSA, 1024-bit. Key ID: 1024R/1E34AC41 (long version:
1024R/9D5877BF1E34AC41). Fingerprint: 8F 15 97 DA 25 30 AB 0D
88 D1 92 54 11 CF 0C 4C

Master Key (original DSA)

DSA, 1024-bit. Key ID: 1024D/6A93B34E (long version:
1024D/4F5E6DF56A93B34E). Fingerprint: 313C 3E76 4B74 C2C5 F2AE
83A8 4F5E 6DF5 6A93 B34E

Release Key (original RSA)

RSA, 1024-bit. Key ID: 1024R/B41CAE29 (long version:
1024R/EF39CCC0B41CAE29). Fingerprint: AE 65 D3 F7 85 D3 18 E0
3B 0C 9B 02 FF 3A 81 FE

Release Key (original DSA)

DSA, 1024-bit. Key ID: 1024D/08B0A90B (long version:
1024D/FECDF3F08B0A90B). Fingerprint: 00B1 1009 38E6 9800 6518
F0AB FECDF3F 08B0 A90B

Snapshot Key (original RSA)

RSA, 1024-bit. Key ID: 1024R/32B903A9 (long version:
1024R/FAAED21532B903A9). Fingerprint: 86 8B 1F 79 9C F4 7F BD
8B 1B D7 8E C6 4E 4C 03

Snapshot Key (original DSA)

DSA, 1024-bit. Key ID: 1024D/7D3E4A00 (long version:
1024D/165E56F77D3E4A00). Fingerprint: 63DD 8EF8 32F5 D777 9FF0
2947 165E 56F7 7D3E 4A00

Appendix G: SSH-2 names specified for PuTTY

There are various parts of the SSH-2 protocol where things are specified using a textual name. Names ending in @putty.projects.tartarus.org are reserved for allocation by the PuTTY team. Allocated names are documented here.

- [G.1 Connection protocol channel request names](#)
- [G.2 Key exchange method names](#)
- [G.3 Encryption algorithm names](#)
- [G.4 Agent extension request names](#)

G.1 Connection protocol channel request names

These names can be sent in a SSH_MSG_CHANNEL_REQUEST message.

simple@putty.projects.tartarus.org

This is sent by a client to announce that it will not have more than one channel open at a time in the current connection (that one being the one the request is sent on). The intention is that the server, knowing this, can set the window on that one channel to something very large, and leave flow control to TCP. There is no message-specific data.

winadj@putty.projects.tartarus.org

PuTTY sends this request along with some SSH_MSG_CHANNEL_WINDOW_ADJUST messages as part of its window-size tuning. It can be sent on any type of channel. There is no message-specific data. Servers MUST treat it as an unrecognised request and respond with SSH_MSG_CHANNEL_FAILURE.

(Some SSH servers get confused by this message, so there is a bug-compatibility mode for disabling it. See [section 4.27.3](#).)

G.2 Key exchange method names

rsa-sha1-draft-00@putty.projects.tartarus.org
rsa-sha256-draft-00@putty.projects.tartarus.org
rsa1024-sha1-draft-01@putty.projects.tartarus.org
rsa1024-sha256-draft-01@putty.projects.tartarus.org
rsa2048-sha256-draft-01@putty.projects.tartarus.org
rsa1024-sha1-draft-02@putty.projects.tartarus.org
rsa2048-sha512-draft-02@putty.projects.tartarus.org
rsa1024-sha1-draft-03@putty.projects.tartarus.org
rsa2048-sha256-draft-03@putty.projects.tartarus.org
rsa1024-sha1-draft-04@putty.projects.tartarus.org
rsa2048-sha256-draft-04@putty.projects.tartarus.org

These appeared in various drafts of what eventually became RFC 4432. They have been superseded by rsa1024-sha1 and rsa2048-sha256.

G.3 Encryption algorithm names

arcfour128-draft-00@putty.projects.tartarus.org

arcfour256-draft-00@putty.projects.tartarus.org

These were used in drafts of what eventually became
RFC 4345. They have been superseded by arcfour128 and
arcfour256.

G.4 Agent extension request names

The SSH agent protocol, which is only specified in an Internet-Draft at the time of writing ([draft-miller-ssh-agent](#)), defines an extension mechanism. These names can be sent in an `SSH_AGENTC_EXTENSION` message.

`add-ppk@putty.projects.tartarus.org`

The payload is a single SSH-2 string containing a keypair in the PPK format defined in [appendix C](#). Compared to the standard `SSH_AGENTC_ADD_IDENTITY`, this extension allows adding keys in encrypted form, with the agent requesting a decryption passphrase from the user on demand, and able to revert the key to encrypted form.

`reencrypt@putty.projects.tartarus.org`

The payload is a single SSH-2 string specifying a public key blob, as in `SSH_AGENTC_REMOVE_IDENTITY`. Requests that the agent forget any cleartext form of a specific key.

Returns `SSH_AGENT_SUCCESS` if the agent ended up holding the key only in encrypted form (even if it was already encrypted); returns `SSH_AGENT_EXTENSION_FAILURE` if not (if it wasn't held by the agent at all, or only in cleartext form).

`reencrypt-all@putty.projects.tartarus.org`

No payload. Requests that the agent forget the cleartext form of any keys for which it holds an encrypted form.

If the agent holds any keys with an encrypted form (or no keys at all), returns `SSH_AGENT_SUCCESS` to indicate that no such keys are now held in cleartext form, followed by a `uint32` specifying how many keys remain in cleartext form (because the agent didn't hold an encrypted form for them). If the agent holds

nothing but keys in cleartext form, returns
SSH_AGENT_EXTENSION_FAILURE.

list-extended@putty.projects.tartarus.org

No payload. Returns SSH_AGENT_SUCCESS followed by a list of identities similar to SSH_AGENT_IDENTITIES_ANSWER, except that each key has an extra SSH-2 string at the end. Currently that string contains a single uint32 flags word, with the following bits defined:

Bit 0

If set, key is held with an encrypted form (so that the reencrypt extension can do something useful with it).

Bit 1

If set, key's cleartext form is not currently held (so the user will have to supply a passphrase before the key can be used).

Appendix H: PuTTY authentication plugin protocol

This appendix contains the specification for the protocol spoken over local IPC between PuTTY and an authentication helper plugin.

If you already have an authentication plugin and want to configure PuTTY to use it, see [section 4.22.3](#) for how to do that. This appendix is for people writing new authentication plugins.

- [H.1 Requirements](#)
- [H.2 Transport and configuration](#)
- [H.3 Data formats and marshalling](#)
- [H.4 Protocol versioning](#)
- [H.5 Overview and sequence of events](#)
- [H.6 Message formats](#)
 - [H.6.1 PLUGIN INIT](#)
 - [H.6.2 PLUGIN INIT RESPONSE](#)
 - [H.6.3 PLUGIN INIT FAILURE](#)
 - [H.6.4 PLUGIN PROTOCOL](#)
 - [H.6.5 PLUGIN PROTOCOL REJECT](#)
 - [H.6.6 PLUGIN PROTOCOL ACCEPT](#)
 - [H.6.7 PLUGIN KI SERVER REQUEST](#)
 - [H.6.8 PLUGIN KI SERVER RESPONSE](#)
 - [H.6.9 PLUGIN KI USER REQUEST](#)
 - [H.6.10 PLUGIN KI USER RESPONSE](#)
 - [H.6.11 PLUGIN AUTH SUCCESS](#)
 - [H.6.12 PLUGIN AUTH FAILURE](#)
- [H.7 References](#)

H.1 Requirements

The following requirements informed the specification of this protocol.

Automate keyboard-interactive authentication. We're motivated in the first place by the observation that the general SSH userauth method 'keyboard-interactive' (defined in [\[RFC4256\]](#)) can be used for many kinds of challenge/response or one-time-password styles of authentication, and in more than one of those, the necessary responses might be obtained from an auxiliary network connection, such as an HTTPS transaction. So it's useful if a user doesn't have to manually copy-type or copy-paste from their web browser into their SSH client, but instead, the process can be automated.

Be able to pass prompts on to the user. On the other hand, some userauth methods can be only *partially* automated; some of the server's prompts might still require human input. Also, the plugin automating the authentication might need to ask its own questions that are not provided by the SSH server. (For example, 'please enter the master key that the real response will be generated by hashing'.) So after the plugin intercepts the server's questions, it needs to be able to ask its own questions of the user, which may or may not be the same questions sent by the server.

Allow automatic generation of the username. Sometimes, the authentication method comes with a mechanism for discovering the username to be used in the SSH login. So the plugin has to start up early enough that the client hasn't committed to a username yet.

Future expansion route to other SSH userauth flavours. The initial motivation for this protocol is specific to keyboard-interactive. But other SSH authentication methods exist, and they may also benefit from automation in future. We're making no attempt here to predict what those methods might be or how they might be

automated, but we do need to leave a space where they can be slotted in later if necessary.

Minimal information loss. Keyboard-interactive prompts and replies should be passed to and from the plugin in a form as close as possible to the way they look on the wire in SSH itself. Therefore, the protocol resembles SSH in its data formats and marshalling (instead of, for example, translating from SSH binary packet style to another well-known format such as JSON, which would introduce edge cases in character encoding).

Half-duplex. Simultaneously trying to read one I/O stream and write another adds a lot of complexity to software. It becomes necessary to have an organised event loop containing `select` or `WaitForMultipleObjects` or similar, which can invoke the handler for whichever event happens soonest. There's no need to add that complexity in an application like this, which isn't transferring large amounts of bulk data or multiplexing unrelated activities. So, to keep life simple for plugin authors, we set the ground rule that it must always be 100% clear which side is supposed to be sending a message next. That way, the plugin can be written as sequential code progressing through the protocol, making simple read and write calls to receive or send each message.

Communicate success/failure, to facilitate caching in the plugin. A plugin might want to cache recently used data for next time, but only in the case where authentication using that data was actually successful. So the client has to tell the plugin what the outcome was, if it's known. (But this is best-effort only. Obviously the plugin cannot *depend* on hearing the answer, because any IPC protocol at all carries the risk that the other end might crash or be killed by things outside its control.)

H.2 Transport and configuration

Plugins are executable programs on the client platform.

The SSH client must be manually configured to use a plugin for a particular connection. The configuration takes the form of a command line, including the location of the plugin executable, and optionally command-line arguments that are meaningful to the particular plugin.

The client invokes the plugin as a subprocess, passing it a pair of 8-bit-clean pipes as its standard input and output. On those pipes, the client and plugin will communicate via the protocol specified below.

H.3 Data formats and marshalling

This protocol borrows the low-level data formatting from SSH itself, in particular the following wire encodings from [\[RFC4251\]](#) section 5:

byte

An integer between 0 and 0xFF inclusive, transmitted as a single byte of binary data.

boolean

The values ‘true’ or ‘false’, transmitted as the bytes 1 and 0 respectively.

uint32

An integer between 0 and 0xFFFFFFFF inclusive, transmitted as 4 bytes of binary data, in big-endian (‘network’) byte order.

string

A sequence of bytes, preceded by a **uint32** giving the number of bytes in the sequence. The length field does not include itself.

For example, the empty string is represented by four zero bytes (the **uint32** encoding of 0); the string "AB" is represented by the six bytes 0,0,0,2,'A','B'.

Unlike SSH itself, the protocol spoken between the client and the plugin is unencrypted, because local inter-process pipes are assumed to be secured by the OS kernel. So the binary packet protocol is much simpler than SSH proper, and is similar to SFTP and the OpenSSH agent protocol.

The data sent in each direction of the conversation consists of a sequence of **messages** exchanged between the SSH client and the plugin. Each message is encoded as a **string**. The contents of the string begin with a **byte** giving the message type, which determines the format of the rest of the message.

H.4 Protocol versioning

This protocol itself is versioned. At connection setup, the client states the highest version number it knows how to speak, and then the plugin responds by choosing the version number that will actually be spoken (which may not be higher than the client's value).

Including a version number makes it possible to make breaking changes to the protocol later.

Even version numbers represent released versions of this spec. Odd numbers represent drafts or development versions in between releases. A client and plugin negotiating an odd version number are not guaranteed to interoperate; the developer testing the combination is responsible for ensuring the two are compatible.

This document describes version 2 of the protocol, the first released version. (The initial drafts had version 1.)

H.5 Overview and sequence of events

At the very beginning of the user authentication phase of SSH, the client launches the plugin subprocess, if one is configured. It immediately sends the `PLUGIN_INIT` message, telling the plugin some initial information about where the SSH connection is to.

The plugin responds with `PLUGIN_INIT_RESPONSE`, which may optionally tell the SSH client what username to use.

The client begins trying to authenticate with the SSH server in the usual way, using the username provided by the plugin (if any) or alternatively one obtained via its normal (non-plugin) policy.

The client follows its normal policy for selecting authentication methods to attempt. If it chooses a method that this protocol does not cover, then the client will perform that method in its own way without consulting the plugin.

However, if the client and server decide to attempt a method that this protocol *does* cover, then the client sends `PLUGIN_PROTOCOL` specifying the SSH protocol id for the authentication method being used. The plugin responds with `PLUGIN_PROTOCOL_ACCEPT` if it's willing to assist with this auth method, or `PLUGIN_PROTOCOL_REJECT` if it isn't.

If the plugin sends `PLUGIN_PROTOCOL_REJECT`, then the client will proceed as if the plugin were not present. Later, if another auth method is negotiated (either because this one failed, or because it succeeded but the server wants multiple auth methods), the client may send a further `PLUGIN_PROTOCOL` and try again.

If the plugin sends `PLUGIN_PROTOCOL_ACCEPT`, then a protocol segment begins that is specific to that auth method, terminating in either `PLUGIN_AUTH_SUCCESS` or `PLUGIN_AUTH_FAILURE`. After that, again, the client may send a further `PLUGIN_PROTOCOL`.

Currently the only supported method is ‘keyboard-interactive’, defined in [\[RFC4256\]](#). Once the client has announced this to the server, the followup protocol is as follows:

Each time the server sends an `SSH_MSG_USERAUTH_INFO_REQUEST` message requesting authentication responses from the user, the SSH client translates the message into `PLUGIN_KI_SERVER_REQUEST` and passes it on to the plugin.

At this point, the plugin may optionally send back `PLUGIN_KI_USER_REQUEST` containing prompts to be presented to the actual user. The client will reply with a matching `PLUGIN_KI_USER_RESPONSE` after asking the user to reply to the question(s) in the request message. The plugin can repeat this cycle multiple times.

Once the plugin has all the information it needs to respond to the server’s authentication prompts, it sends `PLUGIN_KI_SERVER_RESPONSE` back to the client, which translates it into `SSH_MSG_USERAUTH_INFO_RESPONSE` to send on to the server.

After that, as described in [\[RFC4256\]](#), the server is free to accept authentication, reject it, or send another `SSH_MSG_USERAUTH_INFO_REQUEST`. Each `SSH_MSG_USERAUTH_INFO_REQUEST` is dealt with in the same way as above.

If the server terminates keyboard-interactive authentication with `SSH_MSG_USERAUTH_SUCCESS` or `SSH_MSG_USERAUTH_FAILURE`, the client informs the plugin by sending either `PLUGIN_AUTH_SUCCESS` or `PLUGIN_AUTH_FAILURE`. `PLUGIN_AUTH_SUCCESS` is sent when *that particular authentication method* was successful, regardless of whether the SSH server chooses to request further authentication afterwards: in particular, `SSH_MSG_USERAUTH_FAILURE` with the ‘partial success’ flag (see [\[RFC4252\]](#) section 5.1) translates into `PLUGIN_AUTH_SUCCESS`.

The plugin's standard input will close when the client no longer requires the plugin's services, for any reason. This could be because authentication is complete (with overall success or overall failure), or because the user has manually aborted the session in mid-authentication, or because the client crashed.

H.6 Message formats

This section describes the format of every message in the protocol.

As described in [section H.3](#), every message starts with the same two fields:

- **uint32**: overall length of the message
- **byte**: message type.

The length field does not include itself, but does include the type code.

The following subsections each give the format of the remainder of the message, after the type code.

The type codes themselves are defined here:

```
#define PLUGIN_INIT 1
#define PLUGIN_INIT_RESPONSE 2
#define PLUGIN_PROTOCOL 3
#define PLUGIN_PROTOCOL_ACCEPT 4
#define PLUGIN_PROTOCOL_REJECT 5
#define PLUGIN_AUTH_SUCCESS 6
#define PLUGIN_AUTH_FAILURE 7
#define PLUGIN_INIT_FAILURE 8

#define PLUGIN_KI_SERVER_REQUEST 20
#define PLUGIN_KI_SERVER_RESPONSE 21
#define PLUGIN_KI_USER_REQUEST 22
#define PLUGIN_KI_USER_RESPONSE 23
```

If this protocol is extended to be able to assist with further auth methods, their message type codes will also begin from 20, overlapping the codes for keyboard-interactive.

- [H.6.1 PLUGIN INIT](#)
- [H.6.2 PLUGIN INIT RESPONSE](#)
- [H.6.3 PLUGIN INIT FAILURE](#)
- [H.6.4 PLUGIN PROTOCOL](#)

- [H.6.5 PLUGIN PROTOCOL REJECT](#)
- [H.6.6 PLUGIN PROTOCOL ACCEPT](#)
- [H.6.7 PLUGIN KI SERVER REQUEST](#)
- [H.6.8 PLUGIN KI SERVER RESPONSE](#)
- [H.6.9 PLUGIN KI USER REQUEST](#)
- [H.6.10 PLUGIN KI USER RESPONSE](#)
- [H.6.11 PLUGIN AUTH SUCCESS](#)
- [H.6.12 PLUGIN AUTH FAILURE](#)

H.6.1 PLUGIN_INIT

Direction: client to plugin **When:** the first message sent at connection startup **What happens next:** the plugin will send PLUGIN_INIT_RESPONSE OR PLUGIN_INIT_FAILURE

Message contents after the type code:

- **uint32**: the highest version number of this protocol that the client knows how to speak.
- **string**: the hostname of the server. This will be the *logical* hostname, in cases where it differs from the physical destination of the network connection. Whatever name would be used by the SSH client to cache the server's host key, that's the same name passed in this message.
- **uint32**: the port number on the server. (Together with the host name, this forms a primary key identifying a particular server. Port numbers may be vital because a single host can run two unrelated SSH servers with completely different authentication requirements, e.g. system sshd on port 22 and Gerrit on port 29418.)
- **string**: the username that the client will use to log in, if the plugin chooses not to override it. An empty string means that the client has no opinion about this (and might, for example, prompt the user).

H.6.2 PLUGIN_INIT_RESPONSE

Direction: plugin to client **When:** response to PLUGIN_INIT

What happens next: the client will send PLUGIN_PROTOCOL, or perhaps terminate the session (if no auth method is ever negotiated that the plugin can help with) **Message contents after the type code:**

- **uint32**: the version number of this protocol that the connection will use. Must be no greater than the max version number sent by the client in PLUGIN_INIT.
- **string**: the username that the plugin suggests the client use. An empty string means that the plugin has no opinion and the client should stick with the username it already had (or prompt the user, if it had none).

H.6.3 PLUGIN_INIT_FAILURE

Direction: plugin to client

When: response to PLUGIN_INIT

What happens next: the session is over

Message contents after the type code:

- **string:** an error message to present to the user indicating why the plugin was unable to start up.

H.6.4 PLUGIN_PROTOCOL

Direction: client to plugin **When:** sent after PLUGIN_INIT_RESPONSE, or after a previous auth phase terminates with PLUGIN_AUTH_SUCCESS or PLUGIN_AUTH_FAILURE

What happens next: the plugin will send PLUGIN_PROTOCOL_ACCEPT or PLUGIN_PROTOCOL_REJECT

Message contents after the type code:

- **string:** the SSH protocol id of the auth method the client intends to attempt. Currently the only method specified for use in this protocol is ‘keyboard-interactive’.

H.6.5 PLUGIN_PROTOCOL_REJECT

Direction: plugin to client

When: sent after PLUGIN_PROTOCOL

What happens next: the client will either send another PLUGIN_PROTOCOL or terminate the session **Message contents after the type code:**

- **string:** an error message to present to the user, explaining why the plugin cannot help with this authentication protocol.

An example might be ‘unable to open <config file>: <OS error message>’, if the plugin depends on some configuration that the user has not set up.

If the plugin does not support this particular authentication protocol at all, this string should be left blank, so that no message will be presented to the user at all.

H.6.6 PLUGIN_PROTOCOL_ACCEPT

Direction: plugin to client

When: sent after PLUGIN_PROTOCOL

What happens next: depends on the auth protocol agreed on. For keyboard-interactive, the client will send PLUGIN_KI_SERVER_REQUEST or PLUGIN_AUTH_SUCCESS or PLUGIN_AUTH_FAILURE. No other method is specified.

Message contents after the type code: none.

H.6.7 PLUGIN_KI_SERVER_REQUEST

Direction: client to plugin

When: sent after PLUGIN_PROTOCOL, or after a previous PLUGIN_KI_SERVER_RESPONSE, when the SSH server has sent SSH_MSG_USERAUTH_INFO_REQUEST

What happens next: the plugin will send either PLUGIN_KI_USER_REQUEST or PLUGIN_KI_SERVER_RESPONSE

Message contents after the type code: the exact contents of the SSH_MSG_USERAUTH_INFO_REQUEST just sent by the server. See [\[RFC4256\]](#) section 3.2 for details. The summary:

- **string**: name of this prompt collection (e.g. to use as a dialog-box title)
- **string**: instructions to be displayed before this prompt collection
- **string**: language tag (deprecated)
- **uint32**: number of prompts in this collection
- That many copies of:
 - **string**: prompt (in UTF-8)
 - **boolean**: whether the response to this prompt is safe to echo to the screen

H.6.8 PLUGIN_KI_SERVER_RESPONSE

Direction: plugin to client **When:** response to
PLUGIN_KI_SERVER_REQUEST, perhaps after one or more intervening
pairs of PLUGIN_KI_USER_REQUEST and PLUGIN_KI_USER_RESPONSE

What happens next: the client will send a further
PLUGIN_KI_SERVER_REQUEST, OR PLUGIN_AUTH_SUCCESS OR
PLUGIN_AUTH_FAILURE

Message contents after the type code: the exact contents of the
SSH_MSG_USERAUTH_INFO_RESPONSE that the client should send back to
the server. See [\[RFC4256\]](#) section 3.4 for details. The summary:

- **uint32**: number of responses (must match the ‘number of
prompts’ field from the corresponding server request)
- That many copies of:
 - **string**: response to the *n*th prompt (in UTF-8)

H.6.9 PLUGIN_KI_USER_REQUEST

Direction: plugin to client **When:** response to
PLUGIN_KI_SERVER_REQUEST, if the plugin cannot answer the server's
auth prompts without presenting prompts of its own to the user **What**
happens next: the client will send PLUGIN_KI_USER_RESPONSE

Message contents after the type code: exactly the same as in
PLUGIN_KI_SERVER_REQUEST (see [section H.6.7](#)).

H.6.10 PLUGIN_KI_USER_RESPONSE

Direction: client to plugin

When: response to PLUGIN_KI_USER_REQUEST

What happens next: the plugin will send
PLUGIN_KI_SERVER_RESPONSE, or another PLUGIN_KI_USER_REQUEST

Message contents after the type code: exactly the same as in
PLUGIN_KI_SERVER_RESPONSE (see [section H.6.8](#)).

H.6.11 PLUGIN_AUTH_SUCCESS

Direction: client to plugin

When: sent after PLUGIN_KI_SERVER_RESPONSE, or (in unusual cases) after PLUGIN_PROTOCOL_ACCEPT

What happens next: the client will either send another PLUGIN_PROTOCOL or terminate the session

Message contents after the type code: none

H.6.12 PLUGIN_AUTH_FAILURE

Direction: client to plugin

When: sent after PLUGIN_KI_SERVER_RESPONSE, or (in unusual cases) after PLUGIN_PROTOCOL_ACCEPT

What happens next: the client will either send another PLUGIN_PROTOCOL or terminate the session

Message contents after the type code: none

H.7 References

[RFC4251] [RFC 4251](#), ‘The Secure Shell (SSH) Protocol Architecture’.

[RFC4252] [RFC 4252](#), ‘The Secure Shell (SSH) Authentication Protocol’.

[RFC4256] [RFC 4256](#), ‘Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)’ (better known by its wire id ‘keyboard-interactive’).