

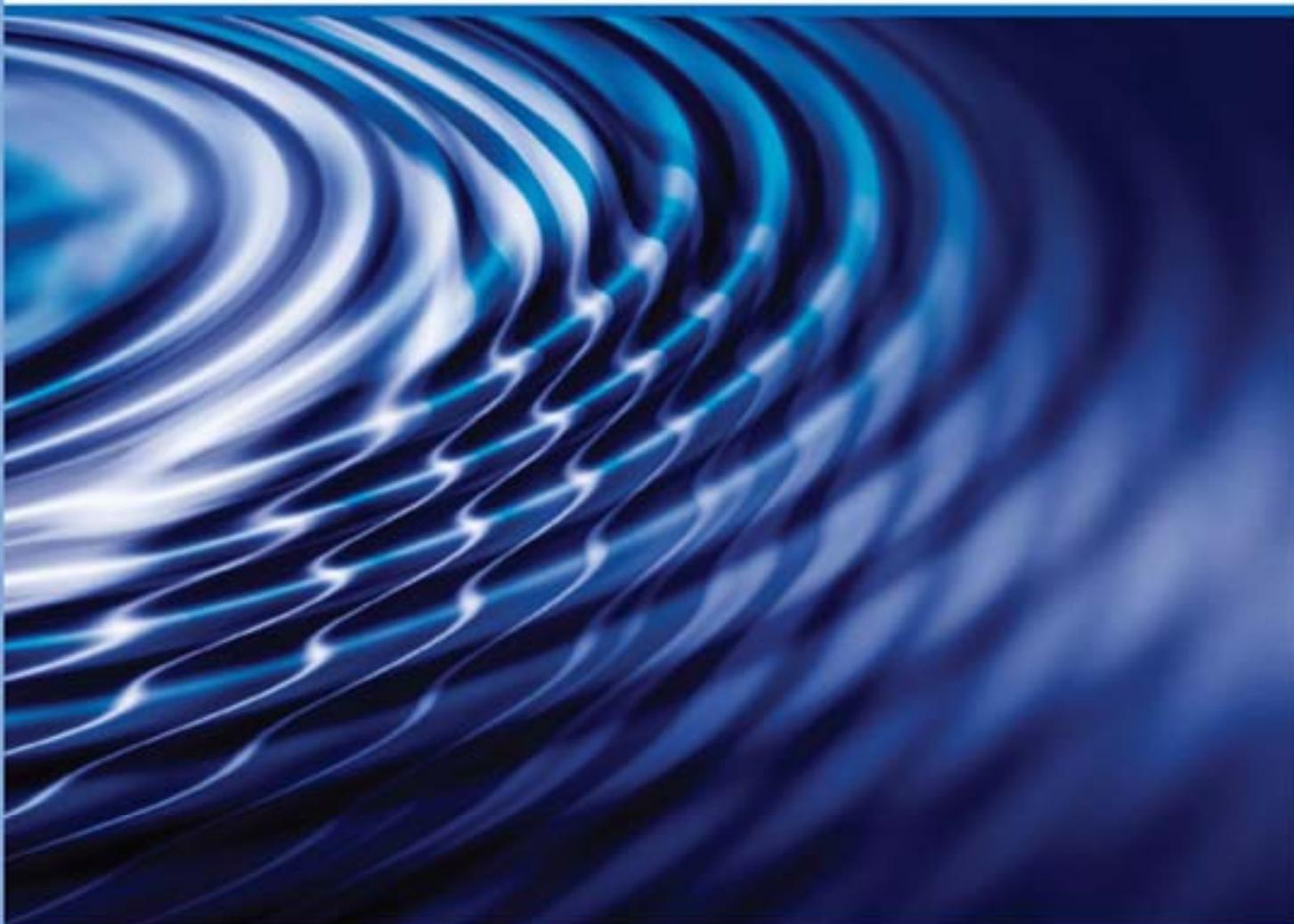
UNIX CLIENT AND SERVER
USER'S GUIDE



Attachmate®

Reflection®

for Secure IT



Reflection for Secure IT

UNIX Client and Server

Version 7.2



© 2010 Attachmate Corporation. All rights reserved.

No part of the documentation materials accompanying this Attachmate software product may be reproduced, transmitted, transcribed, or translated into any language, in any form by any means, without the written permission of Attachmate Corporation. The content of this document is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Attachmate Corporation. Attachmate Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this document.

Attachmate, the Attachmate logo, and Reflection are registered trademarks of Attachmate Corporation, in the USA. All other trademarks, trade names, or company names referenced herein are used for identification only and are the property of their respective owners.

Attachmate Corporation
1500 Dexter Avenue North
Seattle, WA 98109
USA
+1.206.217.7100
<http://www.attachmate.com>

Contents

Installation	7
Replace an Existing Secure Shell Program	9
Install on Linux	10
Install to a Non-Default Location on Linux	11
Install on Sun Solaris	12
Install to a Non-Default Location on Sun Solaris	13
Install on HP-UX	14
Install on IBM AIX	15
Migrate Settings from Existing Configuration Files	16
Install Reflection PKI Services Manager	17
Getting Started	19
Start and Stop the Server	19
Make an SSH Connection	21
Transfer Files Using sftp	22
Transfer Files Using scp	23
Understanding Secure Shell	24
Configuration Files	27
Client Configuration Files	27
Configuration File Format	28
Host Stanzas	28
Command Line Options	29
Server Configuration Files	29
Server Subconfiguration Files	30
Subconfiguration File Samples	31
Data Protection	33
Encryption	33
Data Integrity	34
Configuring Ciphers and MACs	34
FIPS Mode	35

Server Authentication	37
Public Key Authentication Overview	37
Create a New Host Key	39
Add a Key to the Client Known Hosts List	39
Display the Fingerprint of the Host Public Key	41
Server Certificate Authentication Overview	41
Obtain Authentication Certificates	42
Configure Server Certificate Authentication	44
Kerberos (GSSAPI) Authentication	47
Kerberos System Requirements	47
Configure Kerberos Server and Client Authentication	48
User Authentication	51
Password and Keyboard Interactive Authentication	52
Configure Password Authentication	52
Configure Keyboard Interactive Authentication	53
Public Key Authentication	54
Configure Public Key User Authentication	54
Use the Key Agent	56
Certificate Authentication for Users	57
Configure Certificate Authentication for Users	59
Pluggable Authentication Modules (PAM)	62
Configure PAM Authentication	63
RADIUS Authentication	65
Configure RADIUS Authentication	66
RSA SecurID Authentication	67
Configure SecurID Authentication	67
Configure Account Management on HP-UX Trusted Systems	69
Secure File Transfer	71
Secure File Transfer (sftp)	71
Use sftp Interactively	72
Run sftp Batch Files	73

Configuring the sftp Transfer Method (ASCII or Binary)	74
Secure File Copy (scp)	75
Smart Copy and Checkpoint Resume	76
Configure Upload and Download Access	77
Set File Permissions on Downloaded Files	78
Set File Permissions on Uploaded Files	79
Port Forwarding	83
Local Port Forwarding	84
Remote Port Forwarding	87
Configure Port Forwarding	89
FTP Forwarding	90
X Protocol Forwarding	91
Port Forwarding Settings	92
Controlling Access and Authorization	95
Access Control Settings	95
Using Allow and Deny Keywords	96
Configuring User Access	97
Configuring Group Access	98
Configuring Client Host Access	98
Debug Logging and Auditing	101
Client Debugging	101
Server Debugging	102
Auditing (Message Logging)	103
Solaris Audit Support	105
Troubleshooting	107
Troubleshooting Public Key Authentication	107
Troubleshooting Slow File Transfer Speed	109

Appendix	111
Files Used by the Client	112
Files Used by the Server	114
Client Configuration Keywords	118
Server Configuration Keywords	132
File and Directory Permissions	151
ssh Command Line Options	154
ssh Escape Sequences	161
ssh Exit Values	162
ssh-keygen Command Line Options	163
scp Command Line Options	167
sftp Command Line Options	172
Supported sftp Commands	176
ssh-add Command Line Options	181
ssh-agent Command Line Options	183
sshd Command Line Options	185
ssh-certview Command Reference	187
ssh-certtool Command Reference	189
winpki and pkid Command Reference	193
pkid_config Configuration File Reference	197
pki_mapfile Map File Reference	203
Sample Mapping Rules	210
Sample Map File with RuleType Stanzas	212
PKI Settings Migration	213
PKI Services Manager Return Codes	216
Glossary of Terms	219
Index	223

CHAPTER 1

Installation

In this Chapter

Replace an Existing Secure Shell Program	9
Install on Linux	10
Install to a Non-Default Location on Linux	11
Install on Sun Solaris	12
Install to a Non-Default Location on Sun Solaris	13
Install on HP-UX	14
Install on IBM AIX	15
Migrate Settings from Existing Configuration Files	16
Install Reflection PKI Services Manager	17

Reflection for Secure IT UNIX Client and Server provides secure connections between computers. Use Reflection for Secure IT for secure file transfer, secure remote administration of computers, and to tunnel application traffic securely across a network.

For information about supported platforms and additional system requirements, see *Technical note 1944* (<http://support.attachmate.com/techdocs/1944.html>).

Client features

Both the Reflection for Secure IT client and server install the following Secure Shell client features.

- ssh (Secure Shell client)
- ssh2_config (client configuration file)
- sftp (secure file transfer)
- scp (secure file copy)
- ssh-keygen (key generation utility)
- ssh-agent (key agent)
- ssh-add (add identities to the agent)
- ssh-askpass (X11 passphrase utility)
- ssh-certtool (certificate management utility)
- ssh-certview (certificate viewing utility)

By default, client executables are installed to `/usr/bin`. (On Linux `ssh-askpass` is installed to `/usr/libexec`.) The global client configuration file is installed to `/etc/ssh2/`.

Server features

The Reflection for Secure IT server includes all of the client features listed above plus the following Secure Shell server features.

- sshd (Secure Shell daemon)
- sshd2_config (server configuration file)
- A host public/private key pair (see note below)
- sftp-server (file transfer subsystem used by the server)

By default, the `sshd` server is installed to `/usr/sbin`. The `sftp-server` is installed to `/usr/bin`. (On Linux `sftp-server` is installed to `/usr/libexec`.) The server configuration file is installed to `/etc/ssh2`.

Note: The server installation package checks to see if an existing host key pair is already present. If no host key is found, the package creates a new host key pair and the server uses this pair for host authentication. If a host key already exists in `/etc/ssh2`, Reflection for Secure IT uses this key. If an OpenSSH host key is found in `/etc/ssh`, Reflection for Secure IT migrates the key to the correct format and location and uses the migrated key.

Replace an Existing Secure Shell Program

If you're installing on a system that is already running a Secure Shell client or server, you must uninstall the prior version before you install Reflection for Secure IT. This requirement applies to earlier versions of Reflection for Secure IT, as well as F-Secure SSH, OpenSSH, and other Secure Shell implementations.

To install on a system that is currently running Secure Shell

- 1 Log in as root.
- 2 (Server only) Stop the **sshd** service.
- 3 Uninstall your existing Secure Shell product.
- 4 (AIX only) Check for the existence of a hidden `.toc` file in the directory from which you ran **installp** to uninstall your previous version. If this file is present, remove or rename it.
- 5 Install the Reflection for Secure IT client or server.
- 6 If you use public key authentication, ensure that your files and directories are configured with correct permissions. This release of Reflection for Secure IT requires a greater degree of security than was required in previous releases. If files and directories are not sufficiently protected, public key authentication will fail. For details, see *File and Directory Permissions* (page [151](#)).

Note: The **StrictModes** setting affects the level of protection required for files and directories used for public key authentication. To ensure enforcement of a satisfactory level of security, this setting is now enabled by default. Some file and directory permissions are enforced even when this setting is disabled.

- 7 (Optional) If you had configured a non-default client or server configuration file, you will find a backup copy of your file in the configuration file directory. (For details see the note below.) Use these backup files to merge your non-default settings to the new configuration file.

Notes:

- The server installation package checks to see if an existing host key pair is already present. If no host key is found, the package creates a new host key pair and the server uses this pair for host authentication. If a host key already exists in `/etc/ssh2`, Reflection for Secure IT uses this key. If an OpenSSH host key is found in `/etc/ssh`, Reflection for Secure IT migrates the key to the correct format and location and uses the migrated key.
 - The details of how backup configuration files are created vary with the associated operating system.
 - On all platforms except AIX, if you have made any changes to the default client and/or server configuration file, the installer backs up the file when you uninstall. (The file extension added to this backup depends on the native installer.)
 - On AIX, no backup file is created when you uninstall; instead, a backup file is created if a non-default configuration file is present when you install Reflection for Secure IT.
 - Key pairs created with previous Reflection for Secure IT versions are compatible with the current version. No conversion is necessary.
 - The **StrictModes** default value is now "yes" for both the client and server.
 - If `/etc/pam.d/ssh` exists, it is backed up and a new file is put in place.
 - Subconfiguration files, if present, are not touched.
-

Install on Linux

To install Reflection for Secure IT on Linux

- 1 Log in as root.
- 2 Copy the installation package file to your computer and navigate to the directory that contains this file.
- 3 Use **rpm** to install the package:

```
rpm -ivh package_name.rpm
```

For example:

```
rpm -ivh rsit-client-7.2.0.999-i386-rhel.rpm
```

To uninstall

- 1 Log in as root.
- 2 Enter one of the following commands.

For	Use
server	<code>rpm -e --nodeps rsit-server</code>
client	<code>rpm -e --nodeps rsit-client</code>

Install to a Non-Default Location on Linux

You can use the **rpm --relocate** option to specify new target locations for installed files. Two modifications are supported.

- Specify a new target location for configuration files and keys that are installed by default to `/etc/ssh2`.
- Specify a new target location for binaries and man pages that are installed by default to `/usr`.

The following installed items are not relocated: startup and shutdown scripts, the cryptographic module, and the PKI client library.

To install to a non-standard location

- 1 Create the target directories.
- 2 Use the **rpm --relocate** option to specify your target directories. The general syntax is:

```
rpm --install --relocate /usr=PrefixDir --relocate /etc/ssh2=SysConfDir
package_file.rpm
```

For example

```
rpm --install --relocate /usr=/opt/rsit --relocate /etc/ssh2=/opt/rsit/etc
rsit-server-7.2.0.999-i386-rhel.rpm
```

Notes:

- Use **--relocate** modifications to the installation only as described above. Using other modifications will likely result in an unusable installation.
 - To provide access to binaries and man pages after installing to a non-default location, modify the system `PATH` and `MANPATH` variables.
-

Install on Sun Solaris

To install Reflection for Secure IT on Solaris

- 1 Log in as root.
- 2 Copy the installation package file to your computer and navigate to the directory that contains this file.
- 3 Use **uncompress** to unpack the package.

```
uncompress package_name.pkg.Z
```

For example:

```
uncompress rsit-client-7.2.0.999-sparc-solaris10.pkg.Z
```

- 4 Use **pkgadd** to install the package.

```
pkgadd -d package_name.pkg
```

For example:

```
pkgadd -d rsit-client-7.2.0.999-sparc-solaris10.pkg
```

Note: On systems running Solaris 10, you can use zones to partition a single Solaris instance into isolated application environments. For information about installing Reflection for Secure IT in a zones environment, refer to *Technical Note 2254* (<http://support.attachmate.com/techdocs/2254.html>).

To uninstall

- 1 Log in as root.
- 2 Use the **pkgrm** command to remove the package:

For	Use
server	pkgrm RSITsshs
client	pkgrm RSITsshc

Install to a Non-Default Location on Sun Solaris

Note: Installing to a non-default location is supported on Solaris 10; it is not available on Solaris 8 or 9.

To install Reflection for Secure IT to a non-default location, you can create a response file (a text file that provides information to the installer package) and use the PREFIX variable to identify the target directory for the installation. The PREFIX variable has the following effects:

- Configuration files and keys that are installed by default to `/etc/ssh2` are relocated to `$PREFIX/etc/ssh2`.
- Binaries and man pages that are installed by default to `/usr` are relocated to `$PREFIX`.

The following installed items are not relocated: startup and shutdown scripts, the cryptographic module, and the PKI client library.

To install to a non-default location

- 1 Create the target directory.
- 2 Create a response file (`rsp` in this example) that redirects the installation to your target directory (`/opt/rsit` in this example).


```
echo "PREFIX=/opt/rsit" > rsp
```
- 3 Use the **pkgadd -r** option to provide the relocation information during the installation. For example:

```
pkgadd -r rsp -d rsit-server-7.2.0.999-x64-solaris10.pkg
```

Note: To provide access to binaries and man pages after installing to a non-default location, modify the system PATH and MANPATH variables.

Install on HP-UX

To install Reflection for Secure IT on HP-UX

- 1 Log in as root.
- 2 Copy the installation package file to your computer and navigate to the directory that contains this file.
- 3 Use **uncompress** to unpack the package.

```
uncompress package_name.depot.Z
```

For example:

```
uncompress rsit-client-7.2.0.999-ia64-hpux-11.23.depot.Z
```

- 4 Use **swinstall** to install the unpacked package.

```
swinstall -s full_path_and_package_name.depot RSIT
```

For example:

```
swinstall -s /rsit/rsit-server-7.2.0.999-ia64-hpux-11.23.depot RSIT
```

To uninstall

- 1 Log in as root.
- 2 (Server only) Use the server script to stop the **sshd** service.
- 3 Use **swremove** to uninstall the package.

```
/sbin/init.d/sshd2 stop
```

```
swremove RSIT
```

Note: Installing to non-standard locations is not supported on HP-UX.

Install on IBM AIX

To install Reflection for Secure IT on IBM AIX

- 1 Log in as root.
- 2 Copy the installation package file to your computer and navigate to the directory that contains this file.
- 3 Use the **uncompress** command to unpack the package.

```
uncompress package_name.bff.Z
```

For example:

```
uncompress rsit-server-7.2.0.999-powerpc-aix5.bff.Z
```

- 4 Use the **installp** command to install the package.

```
installp -d. RSIT.ssh
```

To uninstall

- 1 Log in as root.
- 2 (Server only) Use the server script to stop the **sshd** service.

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

- 3 Use the **installp** command to uninstall the package.

```
installp -u RSIT.ssh
```

- 4 Remove the hidden `.toc` file in the directory from which you ran **installp** in step 3.

Note: Installing to non-standard locations is not supported on IBM AIX.

Migrate Settings from Existing Configuration Files

A migration script is installed with Reflection for Secure IT, which you can use to migrate settings configured using any of the following products:

F-Secure UNIX clients and servers
 Reflection for Secure IT 6.x UNIX clients and servers
 Reflection for Secure IT 7.x UNIX clients and servers.

The migration script is installed to:

```
/etc/ssh2/migrate.sh
```

The script examines your configuration files to determine if setting changes are required. If changes are needed, you are prompted to confirm that you want to apply these changes. After you confirm the migration, new configuration files are created with the required updates along with backups of your original files. All operations are detailed in the script's output and log files. The log files document which settings have been migrated and which cannot be migrated. Log files are created in the same directory as the converted file and have names based on the converted filename (for example, `sshd2_config_migration.log`).

To migrate global configuration files

Note: When you run the migration script with no arguments, it migrates files located in the `/etc/ssh2` directory. If `/etc/ssh2/sshd2_config` and `/etc/ssh2/ssh2_config` contain non-default settings, you are asked if you want to migrate these files. If these settings contain default values (which is the expected state after you uninstall the prior version and then install the current version), the script looks for the most recent backup files (for example `*.rpmsave`, `*.save` or `*.backup`) and asks if you want to migrate settings in the backup files.

- 1 Log in as root.
- 2 Uninstall the prior version.
- 3 Install the new version.
- 4 Run the migration script with no arguments:


```
/etc/ssh2/migrate.sh
```
- 5 Respond to the prompts.
- 6 Review the migrated settings and the migration log and, where required, merge settings from the migrated backup files into `sshd2_config` and `ssh2_config`.

To migrate a user configuration file

- 1 Log in as root.
- 2 Run the migration script and specify the file you want to migrate. For example:

```
/etc/ssh2/migrate.sh client ~/.ssh2/ssh2_config
```

To migrate PKI settings

You can use the following procedure to migrate certificate settings if Reflection PKI Services Manager is installed on a computer that has Reflection for Secure IT 6.x or F-Secure configuration files.

- 1 Log in as root.
- 2 Use **pkid** with the **-m** option to migrate settings from your prior version configuration files. For example:

To migrate PKI settings in `sshd2_config` and `ssh2_config` files located in `/etc/ssh2/` and migrate these settings to `pki_config` and `pki_map` files in the PKI Services Manager configuration folder:

```
/usr/local/sbin/pkid -m /etc/ssh2/
```

To migrate PKI settings in `sshd2_config.backup` and create new PKI Services Manager configuration files in the specified output directory:

```
/usr/local/sbin/pkid -b /output/path/ -m /etc/ssh2/sshd2_config.backup
```

- 3 Review the migration log, which is created in the `logs` directory located in the PKI Services Manager data directory. (By default, this log records at a level of "info". The level can be elevated using **-d**.)

Note: If the `pki_config` file in the destination folder already has a trust anchor configured, no migration occurs. This helps ensure that the migration won't overwrite modifications you have already configured.

Install Reflection PKI Services Manager

Reflection PKI Services Manager is a service that provides X.509 certificate validation services. If you need support for user or server certificate authentication, you'll need to download and install this application. It is available at no additional charge.

To install Reflection PKI Services Manager

- 1 Log in as root.
- 2 Copy the installation package file to your computer and navigate to the directory that contains this file.

- 3 Use `gzip` to unzip the package:

```
gzip -d package_name.tar.gz
```

For example:

```
gzip -d pkid_1.0.0.999-i386-solaris.gz
```

- 4 Use `tar` to expand the file:

```
tar -xf package_name.tar
```

This creates a directory based on the package name. For example:

```
pkid_1.0.0.999--i386-solaris/
```

- 5 Change to this directory. For example:

```
cd pkid_1.0.0.999-i386-solaris
```

- 6 Run the install script:

```
./install.sh
```

- 7 You are prompted to specify installation locations. To accept the default locations (recommended), press `Enter` in response to these prompts.

Notes:

- On UNIX the install script automatically starts the service.
 - Before Reflection PKI Services Manager can validate certificates you need to edit the default configuration and map files.
-

To uninstall

- 1 Log in as root.
- 2 Run the uninstall script. This script is installed to the `bin` directory in the PKI Services Manager data folder. The default path is:

```
/opt/attachmate/pkid/bin/uninstall.sh
```

Note: The uninstall script renames your existing configuration directory (`/opt/attachmate/pkid/config/` by default) using a name based on the current date, and time. For example, `config.20100101143755`. Your `local-store` directory and any certificates you have added to this directory remain unchanged.

CHAPTER 2

Getting Started

In this Chapter

Start and Stop the Server	19
Make an SSH Connection	21
Transfer Files Using sftp	22
Transfer Files Using scp	23
Understanding Secure Shell	24

Start and Stop the Server

The **sshd** service starts automatically after installation.

A script is installed, which you can use to start, stop, and restart the **sshd** service. The name and location of the script varies, depending on your operating system. When you use the script to start the server, the following **sshd** command is invoked.

```
sshd -oPidFile=sshd_PidFile_keyword_value
```

Note: Do not use **inetd** to launch **sshd**. This is not a supported configuration. Attempting this configuration in FIPS mode results in extremely long connection times for each user connection; this is because **sshd** needs to run required self tests for each connection.

To run the sshd service directly

- 1 Log in as root.
- 2 Include full path information:

```
/usr/sbin/sshd options
```

To run the server script on Linux

Note: The following commands work on all Linux platforms, although in some cases the actual script file is installed to a different location.

- 1 Log in as root.
- 2 Use the following commands to start, stop, and restart the **sshd** service:

```
/etc/init.d/sshd start
/etc/init.d/sshd stop
/etc/init.d/sshd restart
```

To run the server script or service on Sun Solaris

- 1 Log in as root.
- 2 Use the following to start, stop, and restart the **sshd** service:

- On Sun Solaris 8 and 9 use the following commands to start, stop, and restart the **sshd** service:

```
/etc/init.d/sshd2 start
/etc/init.d/sshd2 stop
/etc/init.d/sshd2 restart
```

- On Sun Solaris 10 use the following service options to start, stop, restart, and check the state of the service:

```
svcadm enable network/ssh
svcadm disable network/ssh
svcadm restart network/ssh
svcs -l network/ssh
```

To run the server script on HP-UX

- 1 Log in as root.
- 2 Use the following commands to start, stop, and restart the **sshd** service:

```
/sbin/init.d/sshd2 start
/sbin/init.d/sshd2 stop
/sbin/init.d/sshd2 restart
```

To run the server script on IBM AIX

- 1 Log in as root.
- 2 Use the following commands to start, stop, and restart the **sshd** service:

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

Make an SSH Connection

In most cases, you can connect to your host and log in using your password without making any changes to the default settings. Use **ssh** to connect to the remote server. The syntax is:

```
ssh [options] [user@]hostname[#port] [remote_command [arguments] ...]
```

When no user is specified, the client connects using your current login name. When no port is specified, the client uses the default port (which is 22 unless this has been changed in the client configuration file).

When no command is specified, **ssh** creates a new session on the remote host. When a command is specified, the command is executed on the host and then **ssh** exits. When no user is specified, the current user name is used.

To open a terminal session to a remote server using defaults

- 1 Use **ssh** to connect to the server. For example:

```
ssh joe@myhost
```

- 2 The first time you connect to a host, you see a prompt asking you to confirm the authenticity of the host. For example:

```
Host key not found in hostkeys database.
Key fingerprint:
xesem-cyvic-puhef-penyl-dugid-kxpif-tizyh-behen-gymum-fozyb-cuxex
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)?
```

You can confirm the validity of the host key by contacting the system administrator for that host. (For the procedure administrators can use to get this information, see *Display the Fingerprint of the Host Public Key* (page [41](#)).)

- 3 Enter `yes` in response to the prompt to accept the connection to this host. This adds the host key to your known host key list (in `~/.ssh2/hostkeys`). Hosts whose key you hold are trusted hosts, and you will not see the unknown host prompt in subsequent connections.
- 4 Enter your password to complete the connection.

Note: To simplify initial connections and eliminate the risk created by allowing users to accept unknown keys, administrators can manually add the host key to a user-specific or global known hosts list. For details, see *Add a Key to the Client Known Hosts List*. (page [39](#))

Transfer Files Using `sftp`

Use `sftp` to transfer files securely between the local computer and a remote host. You can also perform other file management commands, such as creating directories and changing file permissions. You can use `sftp` interactively or in combination with batch files to automate actions. For detailed information about command line options, see *sftp Command Line Options* (page [172](#)). For an `sftp` command reference, see *Supported sftp Commands* (page [176](#)).

To open an interactive `sftp` session

- 1 Connect to a remote host. For example:

```
sftp joe@myhost.com
```

Note: You can omit the user name if your name on the Secure Shell server is the same as your current user name.

After a successful connection is established, the following prompt appears:

```
sftp>
```

2 Do any of the following:

To	Use
View a list of supported commands	help ; for example: sftp> help
Learn more about supported commands	help command ; for example: sftp> help put
Transfer and manage files	<i>Supported commands</i> (page 176); for example, to transfer the file <code>demo</code> from the local working directory to the remote working directory: sftp> put demo
End the session	quit ; for example: sftp> quit

Note: The first time you connect to a host, you may see a prompt asking you to confirm the authenticity of the host. For more information, see *Make a Client Connection* (page [21](#)).

Transfer Files Using scp

Use **scp** to copy files securely between the local computer and a remote host, or to transfer files securely between two remote hosts. The basic syntax is:

```
scp [[user@]host[#port]:]source [[user@]host[#port]:]destination
```

Both source and destination file names can include host and user specification to indicate that files are to be copied to or from that host.

To copy a local file to the default remote directory

- Use the following example to get started:

```
scp file_src joe@myhost.com:
```

To copy remote files to the local working directory

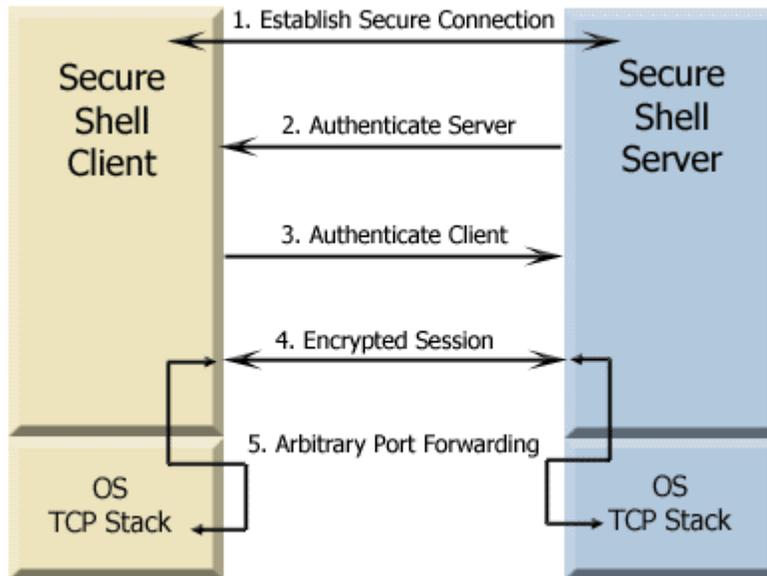
- Use the following example to get started:

```
scp joe@myhost:/demo*.htm .
```

For additional examples, see *Secure File Copy* (page [75](#)). For detailed information about command line options, see *scp Command Line Options* (page [167](#)).

Understanding Secure Shell

This diagram outlines the basic steps involved in creating a Secure Shell tunnel and using it to transmit data securely.



1. Establish the secure connection.

The client and server negotiate to establish a shared key and cipher to use for session encryption, and a hash to use for data integrity checking. For additional information, see *Data Protection* (page [33](#)).

2. Authenticate the server.

Server authentication enables the client to confirm the identity of the server. The server has only one chance to authenticate to the client during the authentication process. If this authentication fails, the connection fails. For additional information, see *Server Authentication* (page [37](#)).

3. Authenticate the client.

Client authentication enables the server to confirm the identity of the client user. By default, the client is allowed multiple authentication attempts. The server and client negotiate to agree on one or more authentication methods. For additional information, see *Client Authentication* (page [51](#)).

4. **Send data through the encrypted session.**

Once the encrypted session is established, all data exchanged between the Secure Shell server and client is encrypted. Users now have secure remote access to the server and can execute commands and transfer files securely through the secure channel. For additional information, see *Secure File Transfer* (page [71](#)).

5. **Use port forwarding to secure communications between other clients and servers.**

Port forwarding, also known as tunneling, provides a way to redirect communications through the Secure Shell channel of an active session. When port forwarding is configured, all data sent to a specified port is redirected through the secure channel. For additional information, see *Port Forwarding* (page [83](#)).

CHAPTER 3

Configuration Files

In this Chapter

Client Configuration Files	27
Configuration File Format	28
Host Stanzas	28
Command Line Options	29
Server Configuration Files	29
Server Subconfiguration Files	30
Subconfiguration File Samples	31

Client Configuration Files

Reflection for Secure IT configuration files control connections made using **ssh**. The settings in the client configuration files also control **scp** and **sftp** connections. The default, global, configuration file is:

```
/etc/ssh2/ssh2_config
```

This file is installed when you install Reflection for Secure IT. The installed file contains commented out lines showing default values for the client settings. A duplicate copy of this file is installed to

```
/etc/ssh2/ssh2_config.example.
```

In addition, you can create configuration files for individual users in:

```
~/.ssh2/ssh2_config
```

The **ssh** client processes settings cumulatively in the following order. If a setting is configured in more than one place, the last value processed overrides any previous value of the same setting.

1. System-wide configuration file: `/etc/ssh2/ssh2_config`
2. User-specific configuration file: `~/.ssh2/ssh2_config`
3. Optional user configuration file specified using the **-F** switch on the **ssh** command line.
4. Command line options used with **ssh**, **scp**, and **sftp**.

For detailed information about client configuration file keywords, see *Client Configuration Keywords* (page [118](#)).

Configuration File Format

The configuration file consists of keywords followed by values. You can use optional host stanzas to configure settings specific to individual hosts or groups of hosts. If a setting is configured in more than one place in the file, the value configured further down the list overrides the previous value.

Any line starting with a number sign (#) is a comment. Any empty line is ignored.

Regular Expressions

Regular expressions are evaluated using POSIX-Extended syntax. For details about regular expression rules, see:

<http://www.opengroup.org/onlinepubs/7990989775/xbd/re.html>

Keyword Syntax

Every keyword requires a value. The value can be separated from the keyword by spaces, or optional spaces and exactly one "=". Enclose the value in quotation marks (single or double) if it includes spaces. For example:

```
key value
```

```
key=value
```

```
key="value with spaces"
```

```
key=value1, value2
```

Keywords are not case sensitive.

Host Stanzas

Host stanzas are supported in client configuration files. Use host stanzas to apply different settings to different hosts. To create a host stanza, use a regular expression that identifies an individual host or a group of hosts. Place this at the beginning of a new line, followed by a colon (:). *This line cannot contain white space.* When you initiate a connection, the client matches host stanza expressions against the host name you specify for that connection. If the host stanza expression matches your specified host, values within that stanza are applied to the connection. The client continues to search for matching host stanzas and applies any relevant settings until the end of the file is reached. Because the last value of a keyword overrides any previous value for the same keyword, you need to place global settings above host-specific settings. Settings outside of any stanza apply to all connections, but can be superseded by subsequent settings placed within a stanza.

You can configure global settings by creating a stanza labeled with ". *:" Settings in this stanza apply to any host you specify on the command line.

Note: Global settings configured in this stanza do not apply to a connection in which no host is specified. To make a successful connection without specifying a host, you must use a configuration file in which the **Host** keyword appears outside of a host stanza.

The following example sets the default user name to 'joe', and changes the user name to 'guy' for connections to samplehost.

```
. *:
    user=joe

samplehost:
    user=guy
```

Command Line Options

You can configure client and server connections using command-line options in addition to, or instead of, using configuration files. Command line options override configuration file settings. Any option that can be configured in a configuration file, can also be set on the command line using the **-o** option. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
```

To configure multiple options, use multiple **-o** switches.

```
-o key1=value -o key2=value
```

The following command lines show two equivalent ways to specify an identification file.

```
ssh -i testfile myname@myserver
ssh -o IdentificationFile=testfile myname@myserver
```

Server Configuration Files

Reflection for Secure IT server configuration files contain configuration settings for the **sshd** server. The default global configuration file is `/etc/ssh2/sshd2_config`. You can specify an alternate file using the **-f** option on the **sshd** command line. You can also create and use optional subconfiguration files for specific client hosts or users.

A sample configuration file is installed to `/etc/ssh2/sshd2_config`. This file includes commented lines that show all available settings and their default values. A duplicate copy of this file is installed to `/etc/ssh2/sshd2_config.example`.

The basic format of the server configuration file is the same as the client configuration file. For details, see *Configuration File Format* (page [28](#)).

Changes you make to the main server configuration file affect new connections immediately; you do not need to restart the server. Existing connections remain active using their original settings; subsequent connections use the new settings.

Note: Changes to **Port**, **ListenAddress** and **FipsMode** require a restart.

The server processes settings cumulatively in the following order. If a setting is configured in more than one place, the last value processed overrides any previous value of the same setting.

1. The global configuration file, or an alternate file specified on the **sshd** command line using **-f**.
2. Any host-specific subconfiguration file(s) that you have created and identified using the **HostSpecificConfig** keyword.
3. Any user-specific subconfiguration file(s) that you have created and identified using the **UserSpecificConfig** keyword.
4. Command line options used with **sshd**.

Server Subconfiguration Files

You can create and use optional subconfiguration files to configure settings that you want to apply to a subset of users or client hosts. Subconfiguration files are read by the process forked for each new connection. These files are read at runtime; any changes you make affect all subsequent connections.

User-specific Subconfiguration Files

Use the **UserSpecificConfig** keyword to configure user-specific subconfiguration files. The syntax for this keyword is:

```
UserSpecificConfig user_expression subconfig_file
```

If the *user expression* (page 97) matches the user attempting a connection, the server uses the specified subconfiguration file. An example file is installed to:

```
/etc/ssh2/subconfig/user.example
```

The `user.example` file includes a list of keywords that are supported in user-specific subconfiguration files.

Security Note: If you configure a user-specific list for **RequiredAuthentications** that is different from the global allowed or required list, a malicious user attempting to authenticate can compare the client/server authentication negotiations of various accounts and use differences in the list of allowed authentications to determine that an account is valid on this system and different from other accounts on the system.

Host-specific Subconfiguration Files

Use the **HostSpecificConfig** keyword to configure settings to apply to a subset of client hosts. The syntax for this keyword is:

```
HostSpecificConfig host_expression subconfig_file
```

If the *host expression* (page 98) matches the client host, the server uses the specified subconfiguration file. An example file is installed to:

```
/etc/ssh2/subconfig/host.example
```

The `host.example` file includes a list of keywords that are supported in host-specific subconfiguration files.

Subconfiguration File Samples

The following sample files provide an example of how subconfiguration files might be used to apply connection settings to particular hosts and users. In the sample server configuration file, a host subconfiguration file is specified using the **HostSpecificConfig** keyword. In this example, settings in the host subconfiguration file apply to all users connecting from the `acme.com` domain. The host subconfiguration file uses the **UserSpecificConfig** keyword to specify a user subconfiguration file, whose settings apply only to connections from the user named `joe`, connecting from the `acme.com` domain.

Server Configuration File

Sample content for `/etc/ssh2/sshd2_config`.

```
Port=2222

RequireReverseMapping=yes

ResolveClientHostname=yes

#Specify a host-specific file for the users from acme.com
HostSpecificConfig=.*acme\.com /root/hostsubconfig

#Limit forwarding to user joe and constrain his forwarding rights
ForwardACL=allow remote joe .* peak.acme.com
```

Host Subconfiguration File

Sample content for `/root/hostsubconfig`.

```
AllowedAuthentications=publickey,password

Ciphers=aes128-cbc

#Allow sftp access only

SessionRestricted=Subsystem

#Specify a user-specific file for user joe
UserSpecificConfig=joe /root/joesubconfig
```

User Subconfiguration File

Sample content for `/root/joesubconfig`.

```
RequiredAuthentications=publickey
```

```
#Allow both shell and sftp access
```

```
SessionRestricted=shell,subsystem
```

CHAPTER 4

Data Protection

In this Chapter

Encryption	33
Data Integrity	34
Configuring Ciphers and MACs	34
FIPS Mode	35

Encryption

Encryption protects the confidentiality of data in transit. This protection is accomplished by encrypting the data before it is sent using a secret key and cipher. The received data must be decrypted using the same key and cipher. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server.

Reflection for Secure IT supports the following data encryption standards:

- Arcfour, Arcfour128, and Arcfour256 (stream mode)
- TripleDES (168-bit) CBC mode
- Cast (128-bit) CBC mode
- Blowfish (128-bit) CBC mode
- AES, also known as Rijndael (128-, 192-, or 256-bit) CBC mode and CTR mode

Data Integrity

Data integrity ensures that data is not altered in transit.

Secure Shell connections use MACs (message authentication codes) to ensure data integrity. The client and server independently compute a hash for each packet of transferred data. If the message has changed in transit, the hash values are different and the packet is rejected. The MAC used for a given session is the MAC highest in the client's order of preference that is also supported by the server.

Reflection for Secure IT supports the following MAC standards:

- hmac-sha1
- hmac-md5
- hmac-sha1-96
- hmac-md5-96
- hmac-ripemd-160
- hmac-sha256
- hmac-sha512

Configuring Ciphers and MACs

The client and server support the same keywords for configuring ciphers and MACs. Configure client keywords in `ssh2_config`. Configure server keywords in `sshd2_config`.

Keyword	Values
Ciphers	<p>Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'. You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.</p> <p>The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers).</p> <p>The default is 'AnyStdCipher'.</p>

MACs Allowed values are 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', 'hmac-md5-96', 'hmac-ripemd160', 'hmac-sha256', and 'hmac-sha512'. Use 'AnyMac' to support all of these. Use 'AnyStdMac' to support 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', and 'hmac-md5-96'. Additional options are 'none', 'any' (equivalent to AnyMac plus 'none'), and 'AnyStd' (equivalent to 'AnyStdMac' plus 'none'). Multiple MACs can also be specified as a comma-separated list. When 'none' is the agreed on MAC, no message authentication code is used. Because this provides no data integrity protection, options that include 'none' are not recommended.

Ciphers can also be defined on the **ssh**, **scp**, and **sftp** command line using **-c**. For example:

```
ssh -c blowfish-cbc joe@remote.com
```

MACs can also be defined on the **ssh** and **sftp** command line using **-m**. For example:

```
sftp -m hmac-md5 joe@remote.com
```

FIPS Mode

The United States Government's Federal Information Processing Standard (FIPS) 140-2 specifies security requirements for cryptographic modules. Cryptographic products are validated against a specific set of requirements and tested in 11 categories by independent, U.S. Government-certified testing laboratories. This validation is then submitted to the National Institute of Standards and Technology (NIST), which reviews the validation and issues a certificate. In addition, cryptographic algorithms may also be validated and certified based on other FIPS specifications. The list of certified products and the vendor's stated security policy (the definition of what the module has been certified to do) can be found at: <http://csrc.nist.gov/cryptval/vallists.htm>.

To configure Reflection for Secure IT to run in FIPS mode, use the **FipsMode** keyword. This keyword is supported for both the client and server.

Note: If you change the **FipsMode** setting on the server, you need to restart the server for the change to take full effect. A SIGHUP signal puts new sessions into FIPS-mode, but does not affect existing connections.)

Enabling FIPS Mode has the following effects:

- All connections must be made using algorithms that meet FIPS 140-2 standards. Algorithms that don't meet these standards are not available, except where these algorithms are allowed by NIST for legacy compatibility.
- Minimum public key sizes for both user and host keys are reset from the default of 512 bits up to 1024 bits.
- Because Reflection for Secure IT cannot verify the FIPS status of SecurID, GSSAPI, and RADIUS binaries, these authentication methods need to be manually disabled by the system administrator if they are not FIPS validated. To ensure that you have disabled *all* PAM authentication methods that are not FIPS validated, disable PAM (`UsePAM=no`) in the server configuration file (`/etc/ssh2/sshd2_config`).

CHAPTER 5

Server Authentication

In this Chapter

Public Key Authentication Overview	37
Create a New Host Key	39
Add a Key to the Client Known Hosts List	39
Display the Fingerprint of the Host Public Key	41
Server Certificate Authentication Overview	41
Obtain Authentication Certificates	42
Configure Server Certificate Authentication	44
Kerberos (GSSAPI) Authentication	47
Kerberos System Requirements	47
Configure Kerberos Server and Client Authentication	48

Public Key Authentication Overview

Reflection for Secure IT uses public key host authentication by default. The server automatically generates a new host key (or migrates an existing host key) during installation. The default key is an RSA 2048-bit key.

Public key cryptography uses a mathematical algorithm with a public/private key pair to encrypt and decrypt data. One of the keys is a public key, which can be freely distributed to communicating parties, and the other is a private key, which should be kept secure by the owner of the key. Data encrypted with the private key can be decrypted only with the public key; and data encrypted with the public key can be decrypted only with the private key.

When keys are used for authentication, the party being authenticated creates a digital signature using the private key of a public/private key pair. The recipient must use the corresponding public key to verify the authenticity of the digital signature. This means that the recipient must have a copy of the other party's public key and trust in the authenticity of that key.

How it Works

When public key authentication is used for host authentication, the following sequence of events takes place.

1. The Secure Shell client initiates a connection.
2. The server sends its public key to the client.
3. The client looks for this key in its trusted host key store.

If the client	This occurs
Finds the host key, and the client copy matches the key sent by the server	Authentication proceeds to the next step.
Does <i>not</i> find the host key	<p>The client displays a message that the host is unknown and provides a fingerprint of the host key. If the client is configured to allow the user to accept unknown keys (the default), the user can accept the key, and authentication proceeds to the next step.</p> <p>If strict host key checking is enforced, the client ends the connection.</p>
Finds a host key, and the client copy <i>doesn't</i> match the key sent by the server	<p>The client displays a warning that the key doesn't match the existing key and displays the fingerprint of the key sent by the server. If the client is configured to allow the user to accept unknown keys (the default), the user can accept the new key.</p> <p>If strict host key checking is enforced, the client ends the connection.</p>

4. To confirm that the server actually holds the private key that corresponds to the received public key, the client sends a challenge (an arbitrary message) to the server and computes a *hash* (page [220](#)) based on this message text.
5. The server creates a digital signature based on the challenge message. To do this, the server independently computes the message hash, and then encrypts the computed hash using its private key. The server attaches this digital signature to the original challenge and returns this signed message to the client.
6. The client decrypts the signature using the public key and compares the hash with its own computed hash. If the values match, host authentication is successful.

Create a New Host Key

In most cases, you do not need to make any changes to the default server host key. The server installation package checks to see if an existing host key pair is already present. If no host key is found, the package creates a new host key pair and the server uses this pair for host authentication. If a host key already exists in `/etc/ssh2`, Reflection for Secure IT uses this key. If an OpenSSH host key is found in `/etc/ssh`, Reflection for Secure IT migrates the key to the correct format and location and uses the migrated key.

To create and use a new host key

- 1 Log in as root.
- 2 Terminate any instances of **sshd** using the server script. (For additional information, see *Start and Stop the Server* (page 19).)
- 3 Use **ssh-keygen** to generate a new host key. For example:

```
ssh-keygen -P /etc/ssh2/hostkey2
```

Note: The **-P** option creates a key with no passphrase protection, which is required for host keys.

- 4 (Optional) If you use a new host key name and/or location, edit the server configuration file (`/etc/ssh2/sshd2_config`). Use the **HostKeyFile** keyword to specify the new name and location:

```
HostKeyFile=/etc/ssh2/hostkey2
```

This step is not required if you continue to use the default host key name (`/etc/ssh2/hostkey`).

- 5 Restart the service.

Add a Key to the Client Known Hosts List

By default, the first time a client attempts to connect to the server, the user sees a message indicating that this is an unknown host. This message includes a fingerprint that identifies the host key. To be sure that this is actually the correct host key, the user should contact the host system administrator who can confirm that this is the correct fingerprint. Without this verification, the client is at risk of a "man-in-the-middle" attack. To simplify initial connections and eliminate the risk created by allowing users to accept unknown keys, you can manually add the host key to the client known hosts list.

To add the server key to the client known hosts list

Note: You will need a correctly named copy of the server's public host key. Client copies of known host keys use the following file name format:

```
key_port_host,IP.pub
```

Where *port* is the port used for the ssh connection, *host* is the host name, and *IP* is the host IP address. (Earlier versions used *key_port_host.pub*, and this format is still supported.)

An easy way to obtain a correctly named key is to make an initial connection to the server and allow the client to accept and name the host key. You can then distribute this copy of the host key. This is the technique used in the following procedure.

- 1 From your server, use **ssh-keygen** to display the fingerprint of the server's public host key:

```
ssh-keygen -F /etc/ssh2/hostkey.pub
```

- 2 From a client that has not yet connected to this host, initiate a connection to your server:

```
ssh myname@myserver
```

You'll see a message saying that the host key is not in the host key database.

- 3 Confirm that the host key fingerprint in this message matches the actual host key fingerprint, and enter 'yes' to accept the host key.

You will see a message identifying the name and location of the host key you just accepted. For example:

```
Host key saved to /home/joe/.ssh2/hostkeys/key_22_myserver,10.10.1.123.pub
```

- 4 Copy this key to the known host list of your client computers:

- To add this host key for all users of the client computer, copy the host public key file to `/etc/ssh2/hostkeys`.

-or-

- To add this host key for an individual user, copy the host public key file to `~/.ssh2/hostkeys`.

- 5 (Optional) Enable **StrictHostKeyChecking** so that users cannot accept unknown host keys. You can add the following line to a system-wide configuration file (`/etc/ssh2/ssh2_config`), or a user-specific configuration file (`~/.ssh2/ssh2_config`).

```
StrictHostKeyChecking=yes
```

Display the Fingerprint of the Host Public Key

The first time a client user connects to the server, he or she sees a prompt that includes the fingerprint of the server's public host key. The server administrator can confirm the validity of the key by displaying the host key fingerprint on the server.

To display the fingerprint that identifies the server's public host key

- 1 Log in to the server.
- 2 Use **ssh-keygen** to display the host key fingerprint:

```
ssh-keygen -F /etc/ssh2/hostkey.pub
```

Server Certificate Authentication Overview

Certificate authentication is a form of public key authentication that solves some of the problems presented by public key authentication. With public key host authentication, the system administrator must either add the host public key for every server to each client's list of known hosts, or count on client users to confirm the host identity correctly when they connect to an unknown host. Certificate authentication avoids this problem by using a trusted third party, called the certification authority (CA), to verify the validity of information coming from the host. With certificates, you can configure authentication using a single trust anchor instead of multiple unique server public keys.

Reflection PKI Services Manager supports central management of PKI settings. You can install and configure a single instance of PKI Services Manager to provide certificate validation services for all supported Attachmate products.

Requirements

Requirement	Function
Reflection PKI Services Manager must be installed and correctly configured.	PKI Services Manager validates the certificate and uses a map file to determine which servers can authenticate with a valid certificate. You need to configure at least one trust anchor and one mapping rule for certificate validation to succeed. You may also need to configure access to intermediate certificates and to certificate revocation information.
A certificate signed by a CA and the associated private key must be installed on the server.	The server sends this certificate to the client to authenticate the server.

Requirement	Function
The Reflection for Secure IT UNIX client must have a copy of the PKI Services Manager public key and be configured to connect to PKI Services Manager.	The client communicates with PKI Services Manager to confirm the validity of the server certificate.

How it Works

1. The Reflection for Secure IT server presents a certificate to the client for server authentication.
2. The Reflection for Secure IT client connects to Reflection PKI Services Manager. (Set the server name and port for this connection using the Reflection for Secure IT client **PkidAddress** keyword.)
3. Reflection for Secure IT verifies the identity of PKI Services Manager using an installed public key. (Set the key name and location using the Reflection for Secure IT client **PkidPublicKey** keyword.)
4. Reflection for Secure IT sends the certificate and the server name to PKI Services Manager.
5. PKI Services Manager determines if the certificate is valid and whether the server is allowed to authenticate with this certificate based on the rules the PKI Services Manager administrator has configured in the PKI Services Manager map file (`/opt/attachmate/pkid/config/pki_mapfile` by default). This information is returned to Reflection for Secure IT.
6. If the certificate is valid and the server presenting it is an allowed identity for this certificate, server authentication is successful.

Obtain Authentication Certificates

Before you can configure authentication using certificates, you need a private key and an associated certificate signed by a trusted CA. For server authentication, these need to be installed and configured on the server. For user authentication, these need to be installed and configured on the client.

There are several ways to obtain the key and associated certificate. The approach you take depends on whether you want to obtain a certificate for an existing key, generate a new key and obtain a certificate for it, or obtain both the private key and the certificate from the CA.

To obtain a certificate for an existing private key

- 1 Use **ssh-certtool** to create a certificate request for your private key. For example:

```
ssh-certtool -p privatekey pkcs10 "CN=acme,OU=demo,C=US"
```

This creates a request file in PKCS#10 format. The default filename is `output.pkcs10`.

- 2 Submit the certificate request to the CA.

The CA returns a digitally signed certificate.

- 3 If the returned certificate is packaged as a PKCS#12 (*.pfx or *.p12) or PKCS#7 file, you can use **ssh-keygen** to extract the certificate from the returned package.

Use **-k** to extract the contents of a PKCS#12 file:

```
ssh-keygen -k package.pfx
```

Use **-7** to extract the contents of a PKCS#7 file:

```
ssh-keygen -7 pkcs7file
```

To generate a new private key and obtain a certificate

- 1 Use **ssh-certtool** to create a private key and a certificate request for this private key. For example to generate an RSA key:

```
ssh-certtool -n rsa pkcs10 "CN=acme,OU=demo,C=US"
```

This creates a request file in PKCS#10 format. The default filename is `output.pkcs10`.

- 2 Submit the certificate request to the CA.

The CA returns a digitally signed certificate.

- 3 If the returned certificate is packaged as a PKCS#12 (*.pfx or *.p12) or PKCS#7 file, you can use **ssh-keygen** to extract the certificate from the returned package.

Use **-k** to extract the contents of a PKCS#12 file:

```
ssh-keygen -k package.pfx
```

Use **-7** to extract the contents of a PKCS#7 file:

```
ssh-keygen -7 pkcs7file
```

To obtain both the private key and certificate from a CA

- 1 Submit your request to the CA.

The CA returns a PKCS#12 (*.pfx or *.p12) that contains both the private key and a digitally signed certificate.

- 2 Use **ssh-keygen** with the **-k** option to extract the key and the certificate from the returned package. For example:

```
ssh-keygen -k package.pfx
```

Configure Server Certificate Authentication

Before you begin, review the requirements described in the *Server Certificate Authentication Overview* (page [41](#)) topic.

To configure server authentication using certificates, you need to install and configure Reflection PKI Services Manager and configure your server and client. Use the following procedures to get started. Many additional variations are possible. For more information, see the Reflection PKI Services Manager User Guide, which is available from <http://support.attachmate.com/manuals/pki.html>.

You can install and configure a single instance of PKI Services Manager to support certificate authentication requests from multiple Reflection for Secure IT clients and/or servers. However, because Reflection for Secure IT settings allow only one entry for the PKI Services Manager address and port, this configuration creates a potential single point of failure. If PKI Services Manager is unreachable or the server is not running, all authentication attempts using certificates will fail. To provide load balancing and failover, you can define a round-robin DNS entry for the PKI Services Manager host name or place the PKI Services Manager host behind a load balancing server.

Note: Paths shown here are based on the default installation options.

To install and configure PKI Services Manager

- 1 Log in as root on the Reflection PKI Services Manager server.
- 2 *Install Reflection PKI Services Manager* (page [17](#)).
- 3 Put a copy of the certificate you want to designate as a trust anchor into your local store. The default PKI Services Manager store is in the following location:

```
/opt/attachmate/pkid/local-store
```

- 4 Open the PKI Services Manager configuration file in a text editor. The default name and location is:

```
/opt/attachmate/pkid/config/pki_config
```

- 5 Use the **TrustAnchor** keyword to identify your trust anchor. For example:

```
TrustAnchor = trustedca.crt
```

-OR-

```
TrustAnchor = CN=SecureCA,O=Acme,C=US
```

Note: To configure multiple trust anchors, add additional **TrustAnchor** lines.

- 6 Configure certificate revocation checking. For example:

To	Sample Configuration
Use CRLs stored on an LDAP server.	RevocationCheckOrder = crlserver CRLServers=ldap://crlserver
Use an OCSP responder.	RevocationCheckOrder = ocsp OCSPResponders = http://ocspresponder

Note: By default PKI Services Manager looks for CRLs in the local store. If you use this configuration, you need to copy the CRLs to your local store.

- 7 If intermediate certificates are required by the chain of trust in your certificates, configure access to these certificates. For example:

To	Sample Configuration
Use intermediate certificates you have added to your local store.	CertSearchOrder=local
Use certificates stored on an LDAP server.	CertSearchOrder=certserver CertServers=ldap://ldapserverserver

- 8 Save your changes to the configuration file.
- 9 Open the *PKI Services Manager map file* (page [203](#)) in a text editor. The default name and location is:

```
/opt/attachmate/pkid/config/pki_mapfile
```

- 10 Create a host **RuleType** stanza and add one or more rules that define which hosts can authenticate with a valid certificate. For example:

```
RuleType = host
  {myhost.com} Subject Contains "myhost"
```

For more sample rules, see *Sample PKI Services Manager Mapping Rules* (page [210](#)).

Note: After a certificate is determined to be valid, rules are processed in order (based on rule type then sequence). If the certificate meets the requirements defined in the conditional expression (or if the rule has no condition), the allowed identities specified in that rule are allowed to authenticate. No additional rules are applied after the first match. This means that if you create a rule with no conditions, all allowed identities must be included in that rule.

- 11 Test for valid PKI Services Manager configuration:

```
/usr/local/sbin/pkid -k
```

```
No errors. Configuration is valid:
```

- 12 Restart Reflection PKI Services Manager.

```
/usr/local/sbin/pkid restart
```

To configure the Reflection for Secure IT server

- 1 Install the *server certificate and associated private key* (page [42](#)). For example:

```
/etc/ssh2/server.key
```

```
/etc/ssh2/server.crt
```

- 2 Set permissions on the server key for user-only read-only access:

```
chmod 400 server.key
```

- 3 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.

- 4 Configure the following keywords:

```
HostCertificateFile=/etc/ssh2/server.crt
```

```
HostKeyFile=/etc/ssh2/server.key
```

- 5 *Restart the server* (page [19](#)).

To configure the Reflection for Secure IT client

- 1 If PKI Services Manager is not installed on the same host as the Reflection for Secure IT client, copy the PKI Services Manager public key to the Reflection for Secure IT client. The key location on PKI Services Manager is:

```
/opt/attachmate/pkid/config/pki_key.pub
```

Copy this to any location on the Reflection for Secure IT client. For example:

```
/etc/ssh2/pki_key.pub
```

- 2 Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor.
- 3 Edit **PkidPublicKey** to specify the location in which you placed the PKI Services Manager public key. For example:

```
PkidPublicKey=/etc/ssh2/pki_key.pub
```

- 4 Edit **PkidAddress** to specify the PKI Services Manager host and port. For example:

```
PkidAddress=pkiserver.acme.com:18081
```

Note: If you specify a host and omit the port, the default PKI Services Manager port (18081) is used.

- 5 Confirm that **HostKeyAlgorithms** is configured to prefer X.509 certificates over host keys. This is the default.

```
HostKeyAlgorithms=x509v3-sign-rsa,x509v3-sign-dss,ssh-rsa,ssh-dss
```

Kerberos (GSSAPI) Authentication

Kerberos is a security protocol that provides an alternate mechanism for both client and server authentication. Kerberos authentication relies on a trusted third party called the KDC (Key Distribution Center). The Secure Shell protocol supports Kerberos authentication via GSSAPI (Generic Security Services Application Programming Interface).

Advantages of using Kerberos authentication include:

- Using a trusted third party eliminates the key management tasks you encounter when you use public key authentication.
- When Kerberos is used for server authentication, no host key is required. This means that client users won't need to respond to an unknown host prompt.

Server Authentication using GSSAPI

By default, Secure Shell connections are established using this sequence of events:

1. Key exchange — the client and server negotiate a shared secret key, cipher, and hash for the session.
2. Server authentication — by default, the server presents a host key for this purpose.
3. Client authentication.

When GSSAPI is used for server authentication, the Kerberos KDC authenticates the server during the initial key exchange. No subsequent server authentication is needed, and the server never sends a host key to the client.

Client Authentication using GSSAPI

After a user has authenticated to the KDC, that user holds Kerberos credentials that can be used by other kerberized applications. When you configure Reflection for Secure IT to support GSSAPI, the server uses Kerberos credentials to authenticate client users. This means that users who have authenticated to the KDC need no additional authentication to connect to the server.

Kerberos System Requirements

Reflection for Secure IT supports the following Kerberos implementations:

- MIT Kerberos V5, release 1.5.4 or later

- Sun Solaris native Kerberos libraries

Reflection for Secure IT uses the following Kerberos libraries. You may need to configure the keywords shown here to specify the fully-qualified path to these libraries on your system. (The default values of these settings depend on your operating system.)

Library	Keyword	Used by
libgssapi_krb5.so	LibGssKrb5	Client (ssh2_config) and server (sshd2_config)
libkrb5.so	LibKrb5	Server only (sshd2_config)

Configure Kerberos Server and Client Authentication

Kerberos can be used for mutual authentication (both client and server), or for client authentication only.

- When the authentication method is `gssapi-keyex`, both server and client authentication occur during the key exchange portion of the connection negotiations. If this authentication fails, the connection fails; no subsequent authentication methods are attempted.
- When the authentication method is `gssapi-with-mic`, Kerberos is not used for server authentication. Client authentication using Kerberos is attempted after successful server authentication. If Kerberos authentication fails, other allowed authentication methods are tried.

Here's a quick summary of the important steps. The details are explained in the procedures that follow.

1. Configure connections to the KDC.
 - Add the host principal and install a keytab file on the Secure Shell server host.
 - Add client user principals.
2. Configure the **AllowedAuthentications** in the server configuration file (as needed).
3. Configure **AllowedAuthentications** and **GSSAPIDelegateCredentials** in the client configuration file (as needed).
4. Authenticate the client user to the KDC using **kinit** before you make a Secure Shell connection.

To configure connections to the KDC

- 1 Log in to your Secure Shell server.

- 2 Confirm that the server is configured to authenticate to your Kerberos realm. If not, install a correctly configured `krb5.conf` file.
- 3 Authenticate to your Kerberos realm using a principal with administrative rights:

```
kinit root/admin
```

- 4 Launch the Kerberos administration utility:

```
/usr/krb5/sbin/kadmin
```

- 5 Add a host principal for this server. For example, to add the host `myhost.sample.com`:

```
addprinc -randkey host/myhost.sample.com
```

- 6 Extract a keytab file for this server:

```
ktadd host/myhost.sample.com
```

- 7 Add a principal for each client user. For example, to add Joe:

```
addprinc joe
```

To configure Secure Shell settings on the server

- 1 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 2 Edit the **AllowedAuthentications** keyword:

To	Use
Authenticate both the server and the client using Kerberos	<code>AllowedAuthentications=gssapi-keyex</code>
Authenticate only the client using Kerberos	<code>AllowedAuthentications=gssapi-with-mic</code>

To configure the client

- 1 Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor.
- 2 Edit the **AllowedAuthentications** keyword:

To	Use
Authenticate both the server and the client using Kerberos	<code>AllowedAuthentications=gssapi-keyex</code>
Authenticate only the client using Kerberos	<code>AllowedAuthentications=gssapi-with-mic</code>

- 3 (Optional) Edit the **GSSAPIDelegateCredentials** keyword if you want to enable ticket forwarding:

```
GSSAPIDelegateCredentials=Yes
```

To obtain Kerberos credentials

Before you can connect to the Secure Shell server, you need to obtain your Kerberos credentials.

- 1 Use **kinit** to authenticate.

```
kinit -f
```

Note: The **-f** option is not required. This option requests a forwardable ticket. If ticket forwarding has been enabled (using **GSSAPIDelegateCredentials**) this ticket is forwarded to the server. This means that you can access other kerberized applications without having to obtain additional Kerberos credentials.

- 2 Enter your password for the Kerberos KDC.

CHAPTER 6

User Authentication

In this Chapter

Password and Keyboard Interactive Authentication	52
Public Key Authentication	54
Certificate Authentication for Users	57
Pluggable Authentication Modules (PAM)	62
RADIUS Authentication	65
RSA SecurID Authentication	67
Configure Account Management on HP-UX Trusted Systems	69

Several methods of client authentication are available, and both the client and server can be configured to determine which method — or methods — are used. The server can be configured to allow, require, or deny client authentication methods. During Secure Shell connection negotiations, the server presents a list of allowed and required methods from which the client and server negotiate one or more authentication methods.

Authentication attempts follow the order of preference set by the client. The connection uses the first authentication technique highest in the client order of preference that is also allowed by the server. If the server is configured to require more than one method, multiple authentication methods are needed to establish a connection.

Password and Keyboard Interactive Authentication

Reflection for Secure IT server supports both password and keyboard interactive authentication by default.

Authentication method	Description
Password	<p>Prompts the client user for the login password for that user on the Secure Shell server host.</p> <p>The password is sent to the host through the encrypted channel.</p>
Keyboard interactive	<p>Supports any procedure in which authentication data is entered using the keyboard, including simple password authentication, thereby enabling the Secure Shell client to support a range of authentication mechanisms, such as RSA SecurID tokens or RADIUS servers.</p> <p>A client administrator could, for example, configure keyboard interactive authentication to handle situations in which multiple prompts are required, such as for password updates.</p> <p>Keyboard data is sent to the host through the encrypted channel.</p>

Configure Password Authentication

Password authentication is supported by default; no configuration is required on either the server or the client to use this authentication method. Use these procedures if you want to modify the default server or client configuration.

Note: Password authentication can be also done using the keyboard-interactive method, which is the preferred method.

To configure password authentication on the client

- 1 Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor.
- 2 Edit the **AllowedAuthentications** keyword.

To configure password authentication on the server

- 1 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 2 Edit **AllowedAuthentications** or **RequiredAuthentications**.

- (Optional) Use **PasswordGuesses** to change the maximum number of attempts a user is allowed for password authentication. (The default is 3.) For example:

```
PasswordGuesses=5
```

Configure Keyboard Interactive Authentication

Keyboard-interactive authentication is supported by default; no configuration is required on either the server or the client to use this authentication method.

Follow these procedures if you want to modify the default server or client configuration.

To configure keyboard interactive authentication on the client

- Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor.
- Edit the **AllowedAuthentications** keyword. For example, to require keyboard interactive authentication:

```
AllowedAuthentications=keyboard-interactive
```

To configure keyboard interactive authentication on the server

- Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- Edit **AllowedAuthentications** or **RequiredAuthentications**. For example:

To	Do This
Support keyboard-interactive authentication, but not traditional password authentication	Remove password from the allowed list. For example: <pre>AllowedAuthentications=gssapi-keyex,gssapi-with-mic,publickey,keyboard-interactive</pre>
Require keyboard interactive authentication	Enter the following command: <pre>RequiredAuthentications=keyboard-interactive</pre>

- (Optional) Use **AuthKbdInt.Retries** to change the maximum number of attempts a user is allowed for keyboard-interactive authentication (the default is 3). For example:

```
AuthKbdInt.Retries=5
```

- (Optional) Configure account management using **AccountManagement**. For details, see *Pluggable Authentication Modules (PAM)* (page [62](#)).

Public Key Authentication

Public key authentication relies upon public/private key pairs. To configure public key authentication, each client user needs to create a key pair and upload the public key to the server. If the key is protected by a passphrase, the client user is prompted to enter that passphrase to complete the connection using public key authentication.

Configure Public Key User Authentication

Public key authentication requires both client and server configuration. Here's a quick summary of the important steps. The details are explained in the procedures that follow.

1. Create a key pair on the client.
2. Add a line to the client identification file (`~/.ssh2/identification`) that identifies the private key.
3. Copy the public key to the user's directory on the server (`~/.ssh2`).
4. Add a line to the user's authorization file (`~/.ssh2/authorization`) on the server that identifies the public key.

Note: To help ensure secure authentication, and prevent tampering, information leakage and spoofing, files and directories used by the client and server must be configured with correct permissions and ownership. If these conditions aren't met, Secure Shell connections and public key authentication may fail. For details, see *File and Directory Permissions* (page [151](#)).

To configure public key authentication on the client

- 1 (Optional) Modify the client's **AllowedAuthentications** setting.

Because public key authentication is allowed by default, this step is required only if you want to change this default. To modify the supported authentications, open the client configuration file (`/etc/ssh2/ssh2_config`). For example, to require public key authentication use:

```
AllowedAuthentications=publickey
```

- 2 Generate a public/private key pair using the **ssh-keygen** utility.

For example, the following command creates a default (2048-bit RSA) key pair (`mykey` and `mykey.pub`) in the current working directory. You are prompted to enter a passphrase during the key creation process. If you provide a passphrase, you will need to use it whenever you authenticate using this key.

```
ssh-keygen mykey
```

The next example uses **-P** to create a key that is not passphrase-protected. This option is less secure, but may be desirable for use with scripts and batch files. The **-t** specifies key type (DSA in this example). Because no key name is specified, the key is created using a default name and location, (`$HOME/.ssh2/id_dsa_1024_myhost_a` for this example, where `myhost` is the system's host name as returned by the **hostname** command).

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

- 3 Create (or edit) the client identification file. The default name and location for this file is `~/.ssh2/identification`. Configure this file for user-only write access (600 is recommended).
- 4 In the identification file, add a line for the private key you just created. The format for key entries is `IdKey`, followed by the private key name. For example:

```
IdKey /home/joe/mykey
IdKey id_dsa_1024_myhost_a
```

Note: If no path information is provided, the client looks for listed keys in `~/.ssh2/`.

To configure public key authentication on the server

- 1 (Optional) Modify the server's **AllowedAuthentications** or **RequiredAuthentications** settings.

Because public key authentication is allowed by default, this step is required only if you want to change the default settings. To modify the supported authentications, open the server configuration file (`/etc/ssh2/sshd2_config`). For example, to require public key authentication, use:

```
RequiredAuthentications=publickey
```

- 2 Copy the client public key to the user-specific configuration directory on the server. The default location is `~/.ssh2`.
- 3 Create (or edit) the key authorization file for this user on this server. This file contains a list of the keys the server accepts for user authentication. The default name and location is `~/.ssh2/authorization`. Configure this file for user-only write access (600 is recommended).

- 4 In the authorization file, add a line for the public key you just copied. The format for key entries is `key` followed by the public key name. For example:

```
Key /path/to/mykey.pub
Key id_dsa_1024_myhost_a.pub
```

Any listed key can be used by the server for user authentication. Keys are assumed to be in the user-specific configuration directory (by default, `~/.ssh2/`) unless you specify an absolute path. If the key presented by the client doesn't match any of the keys listed in the authorization file, public key authentication fails.

Use the Key Agent

You can use the key agent, **ssh-agent**, to manage the private keys that you use for authentication. The agent enables you to store private keys and use these keys to authenticate **ssh**, **scp**, and **sftp** sessions. Because passphrases are required only when you add keys to the agent, using the agent can simplify scripting that relies on **ssh**. By default, the connection to the agent can be forwarded, which means you can use the stored identities securely anywhere in the network.

Note: Because agent forwarding creates an added security risk, you may want to disallow it. Use **ForwardAgent** on the client and **AllowAgentForwarding** on the server.

To launch the agent in your current shell

- Use the following command:

```
eval `ssh-agent`
```

When you launch using **eval**, you need to terminate the process manually. You can use the PID, or use **-k**, as shown here:

```
ssh-agent
```

To launch the agent in a subshell

- Use the command argument to specify your shell; for example:

```
ssh-agent $SHELL
```

When you launch the agent in a subshell, it terminates automatically when you log out of the shell.

To add keys to the agent

- Use **ssh-add**; for example, to start the agent in your current shell and load it with the keys in your identification file, use the following command sequence:

```
eval `ssh-agent`  
  
ssh-add
```

You are prompted for passphrases when keys are added to the agent. After you have loaded the keys, you can connect to the servers that require any of the loaded keys without having to enter a passphrase.

Notes:

- When you run **ssh-agent** alone on the command line, a display appears showing how to configure required environment variables. However, these required variables aren't yet configured. To configure the environment variables you can copy the displayed text, paste it to the command line, and execute the command. Until you do this, you won't be able to use **ssh-add**. This additional step is not required when you use `eval` or `$SHELL` as shown in the preceding examples.
 - If you use X11, call **ssh-add** with '`< /dev/null`' to activate the **ssh-askpass** prompting window. This window is used for passphrase prompts.
 - If you are using private keys associated with X.509 certificates, use the **ssh-add -x** option to add these keys to the key agent:
ssh-add -x
-

Certificate Authentication for Users

Using certificates for client authentication solves some of the problems presented by public key authentication. With public key authentication, each client must upload a copy of the public key to every server. Certificate authentication avoids this problem by using a trusted third party, the certification authority (CA), to verify the validity of information coming from the client. With certificates, you can configure authentication using a single trust anchor instead of multiple unique client public keys.

Note: Reflection PKI Services Manager supports central management of PKI settings. You can install and configure a single instance of PKI Services Manager to provide certificate validation services for all supported Attachmate products.

Requirements

Requirement	Function
Reflection PKI Services Manager must be installed and correctly configured.	PKI Services Manager validates the certificate and uses a map file to determine which users can authenticate with a valid certificate. You need to configure at least one trust anchor and one mapping rule for certificate validation to succeed. You may also need to configure access to intermediate certificates and to certificate revocation information.
A certificate signed by a CA and the associated private key must be installed on the client.	The client sends this certificate to the server to authenticate the user.
The Reflection for Secure IT server must have a copy of the PKI Services Manager public key and be configured to connect to PKI Services Manager.	The server communicates with PKI Services Manager to confirm the validity of the user certificate.

How it Works

1. The Reflection for Secure IT client presents a certificate to the server for user authentication.
2. The Reflection for Secure IT server connects to Reflection PKI Services Manager. (Set the server name and port for this connection using the Reflection for Secure IT server **PkidAddress** keyword.)
3. Reflection for Secure IT verifies the identity of PKI Services Manager using an installed public key. (Set the key name and location using the Reflection for Secure IT server **PkidPublicKey** keyword.)
4. Reflection for Secure IT sends the certificate and user name to PKI Services Manager.
5. PKI Services Manager determines if the certificate is valid and determines if the user is allowed to authenticate with this certificate based on the rules the PKI Services Manager administrator has configured in the PKI Services Manager map file (`/opt/attachmate/pkid/config/pki_mapfile` by default). This information is returned to Reflection for Secure IT.
6. If the certificate is valid and the user presenting it is an allowed identity for this certificate, the Reflection for Secure IT server validates the user's digital signature to prove the client possesses the private key associated with the public key contained in the user's certificate. If the digital signature is verified, the user authentication is successful.

Configure Certificate Authentication for Users

Before you begin, review the requirements described in the *Certificate Authentication for Users* (page [57](#)) topic.

To configure user authentication using certificates, you need to install and configure Reflection PKI Services Manager and configure your server and client. Use the following procedures to get started. Many additional variations are possible. For more information, see the Reflection PKI Services Manager User Guide, which is available from <http://support.attachmate.com/manuals/pki.html>.

You can install and configure a single instance of PKI Services Manager to support certificate authentication requests from multiple Reflection for Secure IT clients and/or servers. However, because Reflection for Secure IT settings allow only one entry for the PKI Services Manager address and port, this configuration creates a potential single point of failure. If PKI Services Manager is unreachable or the server is not running, all authentication attempts using certificates will fail. To provide load balancing and failover, you can define a round-robin DNS entry for the PKI Services Manager host name or place the PKI Services Manager host behind a load balancing server.

Note: Paths shown here are based on the default installation options.

To install and configure PKI Services Manager

- 1 Log in as root on the Reflection PKI Services Manager server.
- 2 *Install Reflection PKI Services Manager* (page [17](#)).
- 3 Put a copy of the certificate you want to designate as a trust anchor into your local store. The default PKI Services Manager store is in the following location:

```
/opt/attachmate/pkid/local-store
```

- 4 Open the PKI Services Manager configuration file in a text editor. The default name and location is:

```
/opt/attachmate/pkid/config/pki_config
```

- 5 Use the **TrustAnchor** keyword to identify your trust anchor. For example:

```
TrustAnchor = trustedca.crt
```

-or-

```
TrustAnchor = CN=SecureCA,O=Acme,C=US
```

Note: To configure multiple trust anchors, add additional **TrustAnchor** lines.

- 6 Configure certificate revocation checking. For example:

To	Sample Configuration
Use CRLs stored on an LDAP server.	RevocationCheckOrder = crlserver CRLServers=ldap://crlserver
Use an OCSP responder.	RevocationCheckOrder = ocsp OCSPResponders = http://ocspresponder

Note: By default PKI Services Manager looks for CRLs in the local store. If you use this configuration, you need to copy the CRLs to your local store.

- 7 If intermediate certificates are required by the chain of trust in your certificates, configure access to these certificates. For example:

To	Sample Configuration
Use intermediate certificates you have added to your local store.	CertSearchOrder=local
Use certificates stored on an LDAP server.	CertSearchOrder=certserver CertServers=ldap://ldapserverserver

- 8 Save your changes to the configuration file.
- 9 Open the *PKI Services Manager map file* (page [203](#)) in a text editor. The default name and location is:

```
/opt/attachmate/pkid/config/pki_mapfile
```

- 10 Create a user **RuleType** stanza and add one or more rules that define which users can authenticate with a valid certificate. For example:

```
RuleType = user
{ %UPN.user% } UPN.host Equals "acme.com"
{ fred root } Subject.CN Contains "Fred"
```

For more sample rules, see *Sample PKI Services Manager Mapping Rules* (page [210](#)).

Note: After a certificate is determined to be valid, rules are processed in order (based on rule type then sequence). If the certificate meets the requirements defined in the conditional expression (or if the rule has no condition), the allowed identities specified in that rule are allowed to authenticate. No additional rules are applied after the first match.

- 11 Test for valid PKI Services Manager configuration:

```
/usr/local/sbin/pkid -k
```

```
No errors. Configuration is valid:
```

12 Restart Reflection PKI Services Manager.

```
/usr/local/sbin/pkid restart
```

To configure the Reflection for Secure IT server

- 1 If PKI Services Manager is not installed on the same host as the Reflection for Secure IT server, copy the PKI Services Manager public key to the Reflection for Secure IT server.

The key location on PKI Services Manager is:

```
/opt/attachmate/pkid/config/pki_key.pub
```

Copy this to any location on the Reflection for Secure IT host. For example:

```
/etc/ssh2/pki_key.pub
```

Note: This key file should be owned by root and not be writable by any user but root.

- 2 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 3 Edit **PkidPublicKey** to specify the location in which you placed the PKI Services Manager public key. For example:

```
PkidPublicKey=/etc/ssh2/pki_key.pub
```

- 4 Edit **PkidAddress** to specify the PKI Services Manager host and port. For example:

```
PkidAddress=pkiserver.acme.com:18081
```

Note: If you specify a host and omit the port, the default PKI Services Manager port (18081) is used.

- 5 Configure **AllowedAuthentications** or **RequiredAuthentications** to allow or require public key authentication. The defaults shown below allow public key authentication, but don't require it:

```
AllowedAuthentications=gssapi-with-mic,publickey,keyboard-interactive,password
```

```
RequiredAuthentications=
```

To configure the Reflection for Secure IT client

1 Obtain a user certificate and associated private key (page [42](#)).

2 Install the certificate and private key. For example:

```
~/ .ssh2/userkey
~/ .ssh2/userkey.crt
```

Note: The certificate must be in the same directory as the private key and use the same base name with a `.crt` file extension.

3 Set permissions on the user key for user-only read-only access:

```
chmod 400 userkey
```

4 Create (or edit) the client identification file. (The default is `~/ .ssh2/identification`.) Configure this file for user-only write access:

```
chmod 600 identification
```

5 Add a line to the client identification file that identifies the private key. Use the **CertKey** keyword. (Path information is optional if the key is in the `~/ .ssh2/` directory.) For example:

```
CertKey userkey
```

6 Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor.

7 Check your configuration of the following client settings. **AllowedAuthentications** must include `publickey`. **IdentificationFile** must specify the file you configured in step 3. The defaults are shown here:

```
AllowedAuthentications=gssapi-with-mic,publickey,keyboard-
interactive,password

IdentificationFile=~/ .ssh2/identification
```

Pluggable Authentication Modules (PAM)

You can configure the Reflection for Secure IT server to use Pluggable Authentication Modules (PAM) in combination with keyboard interactive authentication. PAM employs runtime pluggable modules that provide authentication-related services. These modules are divided into four categories: authentication, account management, session management, and password management.

When PAM is configured, Reflection for Secure IT transfers control of authentication to the PAM library. The PAM library loads the modules specified in the PAM configuration file, and the PAM library prompts Reflection for Secure IT to confirm successful authentication.

The following server keywords configure PAM authentication on the server.

Server keyword	Configuration information
AuthKbdInt.Required	To use PAM for authentication and password management: <code>AuthKbdInt.Required=pam</code>
AccountManagement	To use PAM for account management: <code>AccountManagement=pam</code>
UsePamSessions	To use PAM for session management: <code>UsePamSessions=yes</code>
PamServiceName	To specify the name of the PAM service. The default is: <code>PamServiceName=ssh</code>
PamServiceNameForInternalProcesses	To specify a PAM service to be used for internal processes. For example: <code>PamServiceNameForInternalProcesses ssh-shell</code>
PamServiceNameForSubsystems	To specify a PAM service to be used for subsystems. For example: <code>PAMServiceNameforSubsystems sftp ssh-sftp</code>

Configure PAM Authentication

When PAM is configured, Reflection for Secure IT transfers control of authentication to the PAM library.

To configure PAM authentication on the server

- 1 Edit your PAM configuration settings to support the required modules: *auth*, *account*, *password*, and *session*. If required modules are not defined, the connection will be refused.

On Linux systems, the following file is installed with the server:

```
/etc/pam.d/ssh
```

This file contains the default configuration information. For example, on SLES systems the `ssh` file includes the following:

```

#%PAM-1.0

auth    include      common-auth
auth    required     pam_nologin.so
account include      common-account
password include     common-password
session include      common-session

```

On other systems, create (or configure) `/etc/pam.conf`. For example, on HP-UX:

```

ssh auth    required /usr/lib/security/libpam_unix.1
ssh account required /usr/lib/security/libpam_unix.1
ssh password required /usr/lib/security/libpam_unix.1
ssh session required /usr/lib/security/libpam_unix.1

```

- 2 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 3 Confirm that **AllowedAuthentications** (or **RequiredAuthentications**) includes `keyboard-interactive` as an allowed authentication method (the default).
- 4 Configure **PamServiceName** to identify the name of your PAM service.
 - Use the default (`ssh`) if your PAM modules are defined in `/etc/pam.d/ssh`.
 - or-
 - If your PAM modules are defined in `pam.conf`, the value of **PamServiceName** must match your service name (`ssh` in the example shown above). If `ssh` is not defined in `pam.conf`, you may be able to use the default service name `other`.

- 5 Configure the server to use PAM.

<u>To use PAM for</u>	<u>In the server configuration file, add</u>
Authentication and password management	<code>AuthKbdInt.Required=pam</code>
Account management	<code>AccountManagement=pam</code>
Session management	<code>UsePamSessions=yes</code>

- 6 (Optional) To include the words "PAM authentication" in the prompt that client users see during authentication, include the following:

```
AuthKbdInt.Verbose=yes
```

To configure PAM authentication on the client

- Confirm that **AllowedAuthentications** includes keyboard-interactive as an allowed authentication method (the default).

RADIUS Authentication

RADIUS is an authentication, authorization, and accounting service that authenticates users by integrating with password databases, such as the a UNIX password file, Active Directory, LDAP, and simple text files containing user/password pairs. Reflection for Secure IT supports RADIUS for authentication purposes only.

Requirements

One or more RADIUS authentication servers must be configured. To configure Reflection for Secure IT, you need the name of the RADIUS server, the port used for RADIUS communication (usually 1812 or 1645), and the shared secret used by the RADIUS server. You'll use this information to create a RADIUS configuration file.

How it Works

The Reflection for Secure IT server acts as a RADIUS client in order to authenticate a user. Requests are sent to any RADIUS servers you have configured in the RADIUS file.

1. The Reflection for Secure IT server receives a keyboard-interactive authentication request from a client.
2. If RADIUS authentication is enabled, the Reflection for Secure IT server attempts to authenticate the user by sending an ACCESS-REQUEST message with the User-Name and Password attribute/value pair to the first RADIUS server you have configured.
3. The Reflection for Secure IT server waits for an ACCESS-ACCEPT or ACCESS-REJECT message from the RADIUS authentication server.
4. If the Reflection for Secure IT server receives an ACCESS-ACCEPT message, the client connection is allowed and the Reflection for Secure IT server provides user access based on the current server configuration. If the server receives an ACCESS-REJECT message, or it fails to receive a response, the server attempts to authenticate to any additional RADIUS servers you have configured. If no ACCESS-ACCEPT message is received from any RADIUS server, RADIUS authentication fails and the Reflection for Secure IT server attempts any other allowed authentications.

Note: Authentication fails if a user is able to authenticate to the RADIUS authentication server, but no account exists for that user on the Reflection for Secure IT server.

Configure RADIUS Authentication

When RADIUS is configured, Reflection for Secure IT transfers control of authentication to the RADIUS authentication server.

To configure the Reflection for Secure IT server

1. Create the following file and set owner-only read and write access (permissions = 600).

```
/etc/ssh2/radius_config
```

2. Open this file in a text editor. Add a line for each RADIUS server that identifies the server, the port used for RADIUS on that server, and the shared secret required for RADIUS clients to authenticate to that server. For example:

```
server1:1812:secret1
```

```
server2:1812:secret2
```

Note: RADIUS servers are contacted in order from top to bottom until a response to the authentication request is received.

- 3 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor. Edit the following keywords:

```
AllowedAuthentications=keyboard-interactive
AuthKbdInt.Required=radius
RadiusFile=/etc/ssh2/radius_config
```

To configure the client

- Enable keyboard-interactive authentication. (This is the default for all Reflection for Secure IT clients.)

RSA SecurID Authentication

RSA SecurID is a two-factor authentication solution from RSA Security, Inc that is based on hardware or software tokens. We recommend that you review the Authentication Manager documentation before using SecurID.

Reflection for Secure IT supports RSA SecurID authentication using PAM.

Required Item	Function
RSA Authentication Manager	Verifies authentication requests and centrally manages authentication policies.
RSA Authentication Agent	Intercepts authentication requests and directs them to the Authentication Manager for authentication.
<hr/> <p>Note: The RSA Authentication Agent for PAM must be running on the same computer as the Reflection for Secure IT server.</p> <hr/>	
Hardware Token	A hardware device, such as a key fob or PIN card, that generates a one-time authentication code.
RSA Authentication Agent for PAM	Transfers control of authentication to RSA.

Configure SecurID Authentication

Reflection for Secure IT supports the RSA Authentication Agent for PAM, which allows RSA SecurID tokens to be used when connecting to the server. The RSA Authentication Agent for PAM must be running on the same host as the Reflection for Secure IT server.

To configure the client

- Enable keyboard-interactive authentication. (This is the default for all Reflection for Secure IT clients.)

To configure the server

- 1 Install the RSA Authentication Agent on the computer running the Reflection for Secure IT server.
- 2 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 3 Enable keyboard-interactive authentication and configure the server to use PAM for authentication and password management:

```
AllowedAuthentications=keyboard-interactive
```

```
AuthKbdInt.Required=pam
```

To start the server

Note: You need to set the environment variables `VAR_ACE` and `LD_LIBRARY_PATH` before you start the Secure Shell server. Set `VAR_ACE` to the directory of the RSA Agent for PAM installation that contains the `sdconf.rec` file. Set `LD_LIBRARY_PATH` to the directory where the RSA/Server or RSA/Agent is installed.

- To set the environment variables and start the server:

```
$ VAR_ACE=/opt/ace/data LD_LIBRARY_PATH=/opt/ace/prog /usr/sbin/sshd2
```

Note: To make the environment variable changes persist through a restart, you can modify the *server startup script* (page [19](#)), or modify the root user's default profile.

Configure Account Management on HP-UX Trusted Systems

To ensure that Reflection for Secure IT correctly respects the login restrictions imposed by HP-UX Trusted Systems, you must configure the server to use PAM account management.

Caution: Configuring PAM account management is required for use with HP-UX Trusted Systems to ensure that only allowed users can connect. If the server is not configured to use PAM account management, in some circumstances the server will allow users access to the system who should be denied access.

To configure PAM account management on the server

- 1 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor.
- 2 Configure the server to use PAM for account management:

```
AccountManagement=pam
```

Note: To support PAM account management, PAM must be properly configured on the host system.

CHAPTER 7

Secure File Transfer

In this Chapter

Secure File Transfer (sftp)	71
Use sftp Interactively	72
Run sftp Batch Files	73
Configuring the sftp Transfer Method (ASCII or Binary)	74
Secure File Copy (scp)	75
Smart Copy and Checkpoint Resume	76
Configure Upload and Download Access	77
Set File Permissions on Downloaded Files	78
Set File Permissions on Uploaded Files	79

Reflection for Secure IT supports programs that you can use for secure file transfer: **sftp** and **scp**. Both commands can help you manage file transfers between computers efficiently and securely.

Use **sftp** to transfer files securely between the local computer and a remote host. You can also perform other file management commands, such as creating files and changing permissions. You can use **sftp** interactively or in combination with batch files to automate actions.

Use **scp** to copy files securely between the local computer and a remote host, or to transfer files securely between two remote hosts.

Secure File Transfer (sftp)

Secure file transfer (**sftp**) provides a secure alternative to **ftp**. You can run **sftp** interactively, or use it in combination with a batch file for automated, secure file transfer.

Because **sftp** uses authentication and encryption provided by **ssh**, a Secure Shell server must be running on the remote computer. Settings for **sftp** connections are controlled by the **ssh** client configuration file. For details about these settings, see *Client Configuration Keywords* (page [118](#)). You can also use the **-o** option to configure settings on the **sftp** command line.

Note: Command line options override configuration file settings.

For detailed information about command line options, see *sftp Command Line Options* (page [172](#)). For an **sftp** command reference, see *Supported sftp Commands* (page [176](#)).

Use sftp Interactively

You can use an interactive sftp session to execute one or more file management commands securely on a remote computer.

To open an interactive sftp session

- 1 Connect to a remote host. For example:

```
sftp joe@myhost.com
```

Note: You can omit the user name if your name on the Secure Shell server is the same as your current user name.

After a successful connection is established, the following prompt appears:

```
sftp>
```

- 2 Do any of the following:

To	Use
View a list of supported commands	help ; for example: sftp> help
Learn more about supported commands	help command ; for example: sftp> help put
Transfer and manage files	<i>Supported commands</i> (page 176); for example, to transfer the file <code>demo</code> from the local working directory to the remote working directory: sftp> put demo
End the session	quit ; for example: sftp> quit

Note: If you launch **sftp** without specifying a host name, you can use the interactive **open** and **close** commands to connect to one or more hosts during your interactive session.

Run sftp Batch Files

Using sftp batch files provides a secure way to automate file management.

To create and run an sftp batch file

- 1 Configure the client and server to support a non-interactive client authentication method, such as GSSAPI, or public key without passphrase protection.

Note: Authentication methods that require interaction are not supported when you use the **sftp** batch file option (**-B**).

- 2 On the client, create an **sftp** batch file. The batch file can use any of the interactive commands supported by **sftp** (page 176). For example, you might create a file called `demo` with commands such as:

```
get path/file1
get webfiles/*.htm
```

- 3 Use **sftp** to connect to the remote host and run the batch file. For example:

```
sftp -B demo myname@myhost.com
```

The client runs the commands in the batch file and then exits.

Notes:

- After a successful login, **sftp** executes each command in the batch file until a **bye**, **exit** or **quit** command is found, and then terminates the connection.
 - If a command in the batch file fails, **sftp** continues executing the remaining commands, and returns the error code of the first failed command. However, commands prefixed with "-" (dash) always return 0, even if the command fails. To configure a batch file that returns a separate error for each transfer command, use **scp**.
-

Configuring the sftp Transfer Method (ASCII or Binary)

SFTP supports two transfer methods: ASCII and binary. In ASCII mode individual letters, numbers, and characters are transferred using their ASCII character code, and the receiving computer saves these in the correct text format for that system. During ASCII transfers between UNIX and Windows computers, newline characters are converted as appropriate for each system. (You can also manually configure newline character conversion if necessary.) In binary transfers, data is transferred to the server byte-by-byte with no data conversion.

Reflection for Secure IT also provides a smart transfer option called *auto*. In auto mode, the transfer method is determined by file extension. Files with specified file extensions use ASCII transfer; all other files use binary transfer. The default list of ASCII file types is "txt, htm*, pl, php*". To modify this list for a given **sftp** session, use the **setext** command. To change the default file extension list, use the client keyword **FileCopyAsciiExtensions**.

sftp command	Effect
<code>ascii -s</code>	Displays the current transfer mode.
<code>binary</code>	Sets the current transfer mode to binary (the default).
<code>auto</code>	Sets the current transfer mode to auto.
<code>getext</code>	Displays the current list of file extensions that use ASCII file transfer when auto mode is enabled.
<code>setext</code>	Specifies the current list of file extensions that use ASCII file transfer when auto mode is enabled. To specify multiple extensions, use a comma or space-separated list; this command is not cumulative. Wildcard (zsh-glob) characters are supported. Don't precede file extensions with a period. To specify extensions containing spaces, use quotation marks around the extension or use a backslash as an escape character.
<code>ascii</code>	Sets the current transfer mode to ASCII. When no remote newline is explicitly stated, the client attempts to retrieve the newline convention from the server. If the server does not support this functionality, the client sets the remote newline to CRLF.
<code>ascii dos</code>	Sets the remote newline to CRLF.
<code>ascii unix</code>	Sets the remote newline to LF.

Client keyword	Effect
FileCopyAsciiExtensions	Specifies the default list of file types for ASCII file transfer when auto mode transfer is enabled. The default is 'txt, htm*', pl, php*!.

Secure File Copy (scp)

Use **scp** to copy files securely between the local computer and a remote host, or to transfer files securely between two remote hosts.

Both source and destination file names can include host and user specifications to indicate that files are to be copied to or from that host. Copies between two remote hosts are permitted. Wildcards are supported. When recursion is off (the default), name substitution occurs on file names only, not directories. When recursion is enabled (using **-r**), name substitution includes files and directories. By default, existing files are overwritten. To control overwrite behavior, use **--overwrite**. (If the files are identical no transfer occurs regardless of this setting value.)

Because **scp** uses authentication and encryption provided by **ssh**, a Secure Shell server must be running on the remote computer. Settings for **scp** connections are controlled by the **ssh** client configuration file. For details about these settings, see *Client Configuration Keywords* (page 118).

You can also use the **-o** option to configure settings on the **scp** command line. Command line options override configuration file settings.

Examples

The following samples show how you can use **scp** to transfer files securely — between a local computer and remote host, or between two remote hosts.

To	Example
Transfer a remote file (<code>file1</code>) to a specified local file (<code>file2</code>) and location	<code>scp joe@myhost:/source/file1 /destination/file2</code>
Copy all <code>*.htm</code> files from the current working directory on the local computer to joe's default directory on <code>myhost.com</code>	<code>scp *.htm joe@myhost.com:.</code>
Copy the specified file from remote <code>host1</code> to remote <code>host2</code>	<code>scp joe@host1:/dir/src_file joe@host2:/dir/dest_file</code>
<hr/> Note: Two authentications are required. <hr/>	

Smart Copy and Checkpoint Resume

Reflection for Secure IT UNIX clients and servers support features that help minimize the amount of time spent repeating unnecessary transfer of data.

Identical Files (Smart File Copy)

If a client user initiates a transfer and an identically named file already exists on the server, the server computes a hash of the server copy of the file and sends this value to the client. The client computes a hash of the client copy of the file and compares that to the value from the server. If the two hashes are identical, this indicates that the files are identical, and no data transfer occurs. The timestamp of the destination file is updated unless you transfer using the **scp -p** option.

Smart file copy is enabled by default. To disable from the client, set **SmartFileCopy** to no. To disable it from the server, set **SmartFileTransfer** to no. When smart file copy is disabled, existing files are always overwritten.

Automatic Resume of Interrupted File Transfer (Checkpoint Resume)

Reflection for Secure IT client and servers can resume an interrupted file transfer at the point at which the transfer was interrupted. For example, if a connection is dropped during a file upload, the client user can restart the transfer. The Reflection for Secure IT client determines the size of the file on the server, and requests a hash of that file from the server. The client computes the hash of the local file up to the length that the server already has. If the hashes are the same, the transfer resumes at that point in the file.

Note: Computing a hash to compare files does not produce useful data for ASCII transfers between systems with different line endings, so the hash comparison is skipped in this case and the complete file is always transferred.

Checkpoint resume is enabled by default. To disable from the client, set **CheckpointResume** to no. To disable it from the server, set **SmartFileTransfer** to no. When checkpoint resume is disabled, file transfer always starts over after an interruption.

Note: If you transfer files in a high latency network, the time required to send the hash values across the network can cause delays that exceed the benefit of using these features. In this case, you may be able to improve performance by disabling the smart copy and checkpoint resume features.

Configure Upload and Download Access

By default, users have full access to all directories permitted by their login account. You can use **AllowSftpCommands** to limit what kinds of actions users can perform using **sftp** and **scp**. This keyword supports a comma-separated list of one or more of the following: **all**, **none**, **browse**, **download**, **upload**, **delete**, **rename**. The **upload** option enables users to modify files, create files, create directories, or modify file attributes on the server. The **download** option enables users to read file contents.

AllowSftpCommands controls access from commands that use the SFTP subsystem. This includes both **scp** and **sftp** commands from Reflection for Secure IT clients and **sftp** commands from OpenSSH clients. It does not affect **scp** commands from OpenSSH clients; the OpenSSH **scp** command does not use the SFTP subsystem; it executes an **rcp** command through the secure channel.

Caution: Client users may have a number of ways to access server files and directories. Factors to consider when configuring your server include session access, tunneling access, and file and directory permissions configured on the system.

To configure upload and download permissions

Note: This change affects both **scp** and **sftp** transfers.

- 1 Open the server configuration file (`/etc/ssh2/sshd2_config`) in a text editor. (You can also configure this keyword in subconfiguration files.)

- 2 Edit the **AllowSftpCommands** keyword. For example,

To allow users to view and download files, but disallow any changes to the server files:

```
AllowSftpCommands = browse, download
```

To allow users to browse and upload files, but not view the contents of files on the server:

```
AllowSftpCommands = browse, upload
```

- 3 To prevent file access via terminal sessions or remote command execution (including OpenSSH **scp**), you can use the **SessionRestricted** keyword:

```
SessionRestricted = subsystem
```

Set File Permissions on Downloaded Files

When you download a file to the client using either **sftp** or **scp**, the file permissions of the downloaded file can depend on both the client configuration and the source file permissions.

If the file already exists on the client:

- The client file permissions remain the same after a transfer; the transfer updates the contents of the file contents, but does not modify existing file permissions.

If the file does not exist on the client, the following factors affect the permissions set on the transferred file.

- The downloaded file is given the same permissions as the source file provided there are no settings in effect on the client that prevent the creation of files with these permissions.
- If there are local settings in effect that limit the permissions of newly created files, these are applied to the downloaded file. These settings can be globally configured, or can be modified for the current session using the **umask** command.

To set permissions on downloaded files using umask:

- 1 Use **umask** to specify the limits you want for newly created files. For example, you can use either of the following equivalent commands to limit new files to user-only read and write access.

```
$ umask 066
```

-OR-

```
$ umask u=rwx,g=x,o=x
```

- 2 Connect to the server and download using either **sftp** or **scp**.

With the sample **umask** shown above, downloaded files are created on the client without group or world access.

The following session shows the use of **umask** to set permissions on files downloaded using **sftp**. The first file (`file1`) allows user, group, and world read/write access (666) on the server. The second file (`file2`) allows user read/write access, and group and world read-only access (644) on the server. After the download, both files allow user-only read/write access (600) on the client.

```
$ umask 066

$ sftp joe@myserver.com

Authentication successful.

sftp> ls -l file1

-rw-rw-rw-  0 joe  users    108 Sep 30 02:52 file1

sftp> get file1

/home/joe/file1          108  0.0KB/s  00:00 100%

sftp> lls -l file1

-rw-----  0 joe  users    8 Sep 30 11:47 file1

sftp> ls -l file2

-rw-r--r--  0 joe  users   225 Sep 30 02:56 file2

sftp> get file2

/home/joe/file2          225  0.0KB/s  00:00 100%

sftp> lls -l file2

-rw-----  0 joe  users   225 Sep 30 11:47 file2

sftp> exit

$
```

Set File Permissions on Uploaded Files

When you upload a file to the server using either **sftp** or **scp**, the file permissions of the uploaded file can depend on both the server configuration and the source file permissions.

If the file already exists on the server:

- The server file permissions remain the same after a transfer; the transfer updates the contents of the file contents, but does not modify existing file permissions.

If the file does not exist on the server, the following factors affect the permissions set on the transferred file. Items lower on this list override items higher on the list.

1. The uploaded file is given the same permissions as the source file provided there are no settings in effect on the server that prevent the creation of files with these permissions.

2. If the client requests a UMASK value using the **SetRemoteEnv** keyword, those permission limits are applied.
3. System-wide settings for new file creation are applied. (For example, these may be configured in standard system files such as `/etc/default/login` and `/etc/environment`, or using PAM.)
4. If a UMASK value is configured in a global Reflection for Secure IT environment file (`/etc/ssh2/environment`), those permission limits are applied.
5. If a UMASK value is configured in a user-specific Reflection for Secure IT environment file (`~/.ssh2/environment`), those permission limits are applied.

Note: UMASK is included by default in the list of environment variables allowed by **SettableEnvironmentVars**. If UMASK is not included in this list, you cannot modify UMASK values using an `environment` file on the server or using the client **SetRemoteEnv** keyword.

To set permissions on uploaded files on the server using the environment file

- 1 Create (or edit) the environment file.

To configure	Use this path and file name
User-specific settings	<code>~/.ssh2/environment</code>
Global settings	<code>/etc/ssh2/environment</code>

- 2 Add a line specifying the UMASK value that you want to apply to uploaded files. For example:

```
UMASK=066
```

To set permissions on uploaded files from the client using SetRemoteEnv

- Open the client configuration file (`/etc/ssh2/ssh2_config`) in a text editor. Add a line using **SetRemoteEnv** to specify the UMASK value you want to apply to uploaded files. For example:

```
SetRemoteEnv=UMASK=066
```

-OR-

- Use **SetRemoteEnv** on the command line to specify a UMASK value. For example:

```
sftp -oSetRemoteEnv=UMASK=066 joe@myserver.com
```

The following session shows the use of **SetRemoteEnv** to set permissions on a file uploaded using **scp**. The source file (`demo`) allows user, group, and world read/write access (644) on the client (`abchost`). After the upload, the file allows user-only read/write access (600) on the server (`xyzhost`).

```
joe@abchost:~> ls -l demo
-rw-r--r-- 1 joe users 30 2008-10-02 12:07 demo
joe@abchost:~> scp -oSetRemoteEnv=UMASK=066 demo joe@10.10.3.232:
Authentication successful.
demo                               30      0.0KB/s   00:00   100%
joe@abchost:~> ssh joe@10.10.3.232
Authentication successful.
Last login: Thu Oct  2 16:56:22 2008 from 150.215.83.121
[joe@xyzhost ~]$ ls -l demo
-rw----- 1 joe joe 30 Oct  2 16:57 demo
[joe@xyzhost ~]$
```


CHAPTER 8

Port Forwarding

In this Chapter

Local Port Forwarding	84
Remote Port Forwarding	87
Configure Port Forwarding	89
FTP Forwarding	90
X Protocol Forwarding	91
Port Forwarding Settings	92

Port forwarding, also known as tunneling, provides a way to redirect communications through the Secure Shell channel of an active session. When port forwarding is configured, all data sent to a specified port is redirected through the secure channel. You can configure any of the following.

- Local port forwarding moves data securely between an application client running on the Secure Shell client host and a remote application server.
- Remote port forwarding moves data securely between an application client running on the Secure Shell server host and a local application server.
- FTP forwarding allows you to forward all FTP communications through the Secure Shell tunnel.
- X11 forwarding moves X protocol data securely between an X server running on the Secure Shell client host and an X client running on the Secure Shell server host. This is a special category of dynamic remote port forwarding, and is configured using different settings.

Terminology

Port forwarding involves two sets of client and server applications — the Secure Shell client and server, and the client/server pair whose data is being forwarded. In this guide, the following terms are used as defined below in reference to port forwarding:

Term	Definition
Secure Shell server	The Reflection for Secure IT server daemon.
Secure Shell server host	The computer on which the Secure Shell server runs.
Secure Shell client	The Reflection for Secure IT client application.
Secure Shell client host	The computer on which the Secure Shell client runs.
Application client	The client program of the client/server pair whose data you want to forward. For example, this might be a mail client or Web browser.
Application client host	The computer on which the application client runs. This is often either the Secure Shell server host or the Secure Shell client host, but it can also be a third host.
Application server	The server program that communicates with your application client, such as a mail server or Web server.
Application server host	The computer on which the server application runs. This can be either the Secure Shell server host or the Secure Shell client host, or it can also be a third host.

Local Port Forwarding

Use local port forwarding to forward data securely from an application client running on the same computer as the Secure Shell client. When you configure local port forwarding, you designate an arbitrary local port to use for forwarding data, and a destination host and port to receive the data. Local port forwarding works as follows.

1. When the Secure Shell connection is established, the Secure Shell client opens a listening *socket* (page [222](#)) on the local computer (the one running the Secure Shell client) using the designated local port. In most cases, this socket is available only to applications running on the Secure Shell client host.

The gateway ports setting controls whether locally forwarded ports are available to remote applications. By default this setting is not enabled, and the client uses the loopback address ("localhost" or 127.0.0.1) when it opens a socket for local port forwarding. This prevents applications running on other computers from connecting to the forwarded port. When you enable gateway ports, a remote application client can open a socket using the Secure Shell client's Ethernet address (such as an IP address, a URL, or a DNS name). For example, a Secure Shell client running on acme.com might be configured to forward port 8088. When gateway ports are not enabled, the forwarded socket is localhost:8088. When gateway ports are enabled, the forwarded socket is acme.com:8088.

Caution: Enabling gateway ports reduces the security of your client host, network, and connection because it allows remote applications to use the forwarded port on your system without authenticating.

2. An application client is configured to connect to the forwarded port (rather than directly to the application server host and port). When that client establishes a connection, all data is sent to the listening port, and then redirected to the Secure Shell client.
3. The Secure Shell client encrypts the data and sends it securely through the Secure Shell channel to the Secure Shell server.
4. The Secure Shell server receives the data, decrypts it, and redirects it to the destination host and port used by the application server.

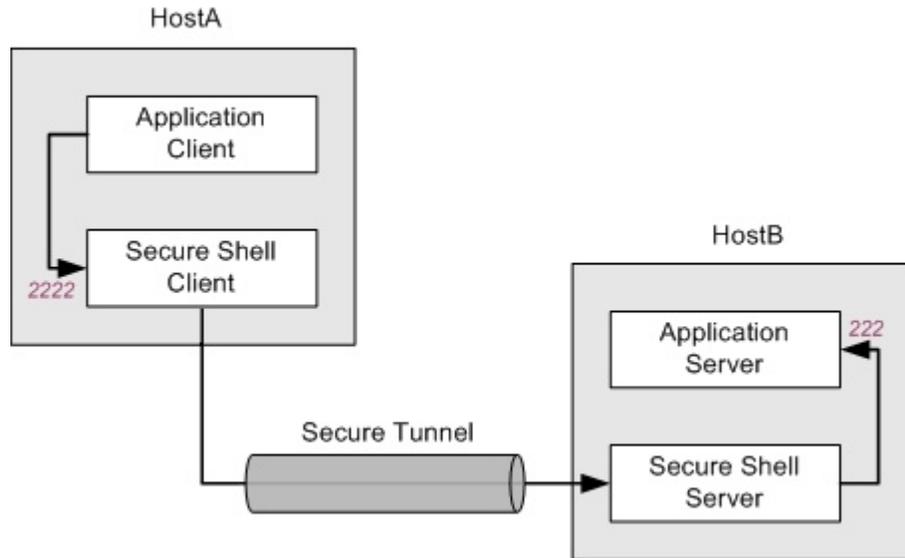
Note: If the final destination host and port are not on the Secure Shell server host, data is sent in the clear between the Secure Shell host and the application server host.

5. The return data from the application server is directed to the Secure Shell server, which encrypts it and sends it securely to the Secure Shell client through the SSH tunnel. The Secure Shell client decrypts the data and redirects it to the original application client.

The general command-line syntax for local port forwarding is:

```
ssh -L listening_port:app_host:hostport user@sshserver
```

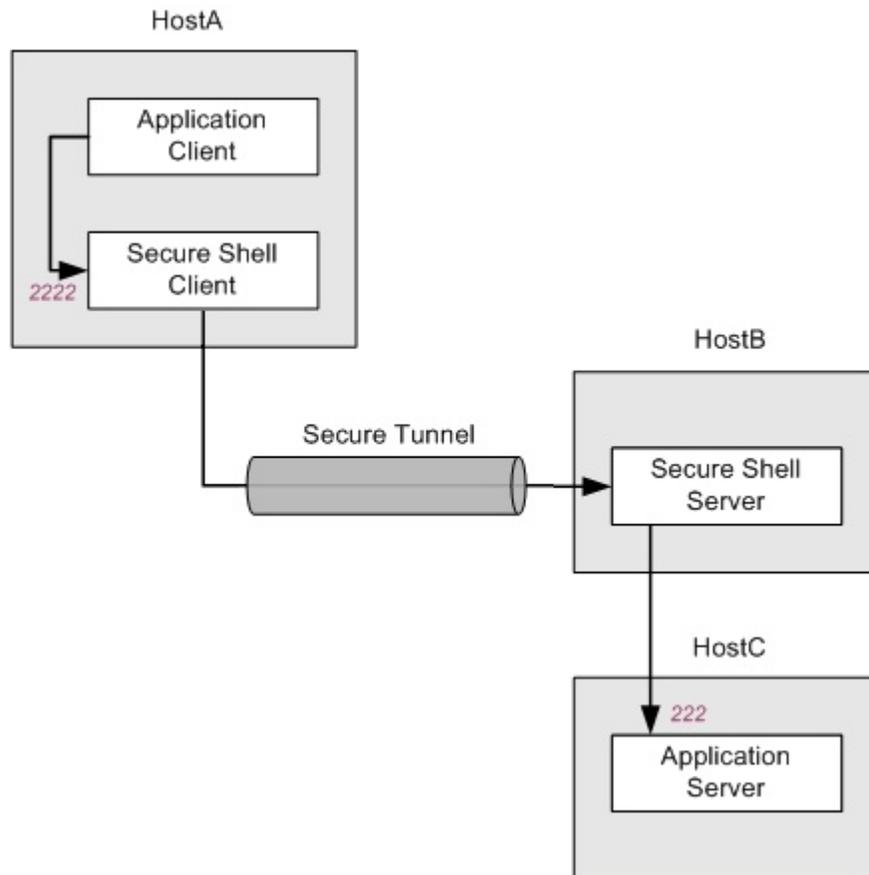
The diagrams that follow illustrate two ways to use this.



In the configuration shown above, the application client and the Secure Shell client both run on HostA. The Secure Shell server and application server both run on HostB. All data sent to port 2222 on HostA is forwarded to port 222 on HostB. In this arrangement, all data in transit is securely encrypted. The following command (in which `localhost` identifies the loopback address on HostB) configures this:

```
ssh -L 2222:localhost:222 user@HostB
```

The following diagram illustrates local port forwarding to a third host. In this configuration, the application server runs on a different host than the Secure Shell server. All data sent to port 2222 on HostA is forwarded to port 222 on HostC.



The following command configures this:

```
ssh -L 2222:HostC:222 user@HostB
```

Note: Data sent between HostB and HostC is not encrypted.

Remote Port Forwarding

Use remote port forwarding to forward data securely from an application client running on the Secure Shell server host. When you configure remote port forwarding, you designate an arbitrary remote port to use for forwarding data and a destination host and port to receive the data. Remote port forwarding works as follows.

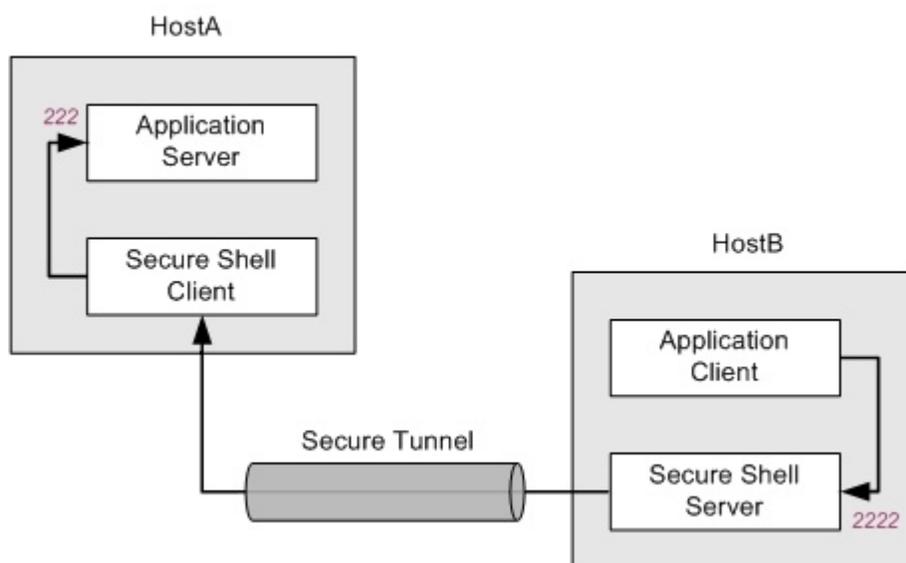
1. When the Secure Shell connection is established, the Secure Shell server opens a listening *socket* (page [222](#)) on the Secure Shell server host using the specified listening port.

2. A client application running on the Secure Shell server host is configured to connect to the listening port (rather than directly to the application server host and port). When that client establishes a connection, all data is sent to the listening port, and then redirected to the Secure Shell server.
3. The Secure Shell server encrypts the data and sends it securely through the SSH tunnel to the Secure Shell client.
4. The Secure Shell client receives data, decrypts it, and redirects it to the destination host and port (on the Secure Shell client host) that is used by the server application.
5. The return data from the server application is directed to the Secure Shell client, which encrypts it and sends it securely to the Secure Shell server through the SSH tunnel. The Secure Shell server decrypts the data and redirects it to the original client application.

The general command-line syntax for remote port forwarding is:

```
ssh -R listening_port:app_host:hostport user@sshserver
```

The diagram that follows illustrates one possible remote port forwarding configuration.



The application server and the Secure Shell client run on HostA. The Secure Shell server and application client both run on HostB. All data sent to port 2222 on HostB is forwarded to port 222 on HostA. In this arrangement, all data in transit is securely encrypted. The following command configures this.

```
ssh -R 2222:localhost:222 user@HostB
```

Configure Port Forwarding

You can establish a port forwarding tunnel using either the **ssh** command line or in the client configuration file (`/etc/ssh2/ssh2_config`).

To configure and use local port forwarding

- 1 Pick a local port to use for forwarding. (This procedure uses 2110 as an example.)

Note: This can be any available port, but don't use port values less than 1024. These ports are, by convention, reserved for services, and may not be available.

- 2 Configure your application client (for example your e-mail client) to connect to the forwarded port on the local host rather than to the remote application server socket. For this example:

Forwarded local port	Remote application server socket
localhost:2110	mailserver.com:110

- 3 Connect the Secure Shell client.

Use local port forwarding to send data from the forwarded local port to the remote application server. The general command line syntax is:

```
ssh -L listening_port:app_host:hostport user@sshserver
```

For this example, the mailserver runs on the same host as the Secure Shell server. The application host in this case is "localhost" on mailserver.com. The command-line configuration is:

```
ssh -L 2110:localhost:110 joe@mailserver.com
```

- 4 Use the application client as you normally would.

The data is forwarded securely from the listening port on the client host (localhost:2110) through the secure channel to the remote application server's listening socket on mailserver.com (localhost:110).

Forwarding to a Third Host

In the preceding example, the application server and Secure Shell server run on the same host. The forwarded data is encrypted for the entire transit. It's also possible to use port forwarding when the application server runs on a different host. For example:

```
ssh -L 2110:mailserver.com:110 user@sshserver.com
```

In this case, data is forwarded through the secure tunnel to sshserver.com. Data is then forwarded in the clear to port 2110 on mailserver.com.

FTP Forwarding

You can configure Reflection for Secure IT to forward FTP communications through the Secure Shell tunnel. FTP forwarding supports both active and passive mode transfers.

Advantages of using FTP forwarding include:

- You can continue to use FTP applications. All communications (including the FTP command channel and all data channels) are securely encrypted between the Secure Shell client and the Secure Shell server.
- If the Secure Shell server and the FTP server run on the same computer, only the Secure Shell port (22) has to be opened in the firewall. Without tunneling, FTP communications require opening the FTP port (21) and a wide range of non-privileged ports for passive mode transfers.
- The FTP client computer doesn't require any open ports in the firewall for active mode transfers.

Depending on your hardware resources, forwarding FTP connections using Secure Shell channels may lead to some variations in the transfer speeds when compared to plain FTP connections. If the network is faster than the CPU, tunneling FTP may result in slower transfer because of the encryption process. If the network is slower than the CPU, enabling Secure Shell compression may increase transfer rates.

Local FTP Forwarding

To forward FTP communications from a port used by a local FTP client to a remote FTP server, add the prefix "ftp/" before the local listening port.

In the following example, FTP communications sent from an FTP client (on the same computer as the Secure Shell client) are forwarded to an FTP server running on myhost.com. With this configuration, you would configure the FTP client to connect to localhost:2121.

```
ssh -L ftp/2121:myhost.com:21 user@myhost.com
```

-OR-

```
LocalForward=ftp/2121:myhost.com:21
```

Note: The FTP client must be on the same server as the Reflection for Secure IT client. You can configure local FTP forwarding to an FTP server on a different host than the Reflection for Secure IT server, but in that case data is unencrypted in transit from the Reflection for Secure IT server to the FTP server.

Remote FTP Forwarding

To forward FTP communications from a port used by a remote FTP client to a local FTP server, add the prefix "ftp/" before the remote listening port.

In the following example, FTP communications sent from an FTP client (on the same computer as the Secure Shell server) are forwarded to an FTP server (on the same computer as the Secure Shell client). With this configuration, you would configure the FTP client to connect to port 3333.

```
ssh -R ftp/3333:localhost:21 user@myhost.com
```

-OR-

```
RemoteForward=ftp/3333:localhost:21
```

Note: The FTP server must be on the same host as the Reflection for Secure IT client and the FTP client must be on the same host as the Reflection for Secure IT server.

X Protocol Forwarding

The X Window System provides support for graphical display on UNIX systems. X protocol forwarding provides a way to secure the communication between X clients and remote X servers. X forwarding is enabled by default. X forwarding works as follows:

1. If X forwarding is enabled, the Secure Shell client requests X forwarding when it connects to the server.
2. If X forwarding is supported by the server, the server sets itself up as a proxy X server on the server host, and sets the DISPLAY environment variable in the client shell to point to the proxy X display.
3. When you run an X client program on the server host, it connects to the proxy display.
4. The Secure Shell client acts as a proxy X client and connects to the X server on the client host.
5. All X protocol information is sent through the Secure Shell channel.

Working with X11 Settings

The client setting **ForwardX11** enables or disables X11 forwarding. (The default is yes.) The client setting **TrustX11Applications** specifies whether the X server treats forwarded X11 client applications as trusted. (The default is no.)

Under some conditions, the configuration of these settings may affect the launch speed of X client applications. This happens when more than two systems are involved. For example:

System1 runs an X server and the Secure Shell client.

System2 runs an X client application, the Secure Shell client, and the Secure Shell server.

System3 runs an X client application and the Secure Shell server.

When a user makes an **ssh** connection from System1 to System2 with `X11Forwarding=yes` (the default) and `TrustX11Applications=no` (the default), there is no delay in starting X applications.

If the user makes a subsequent **ssh** connection from the new shell to System3 with `X11Forwarding=yes` (the default) and `TrustX11Applications=no` (the default), there will be a long delay (as much as 6 seconds) after the user authenticates during which X applications started from System3 will not be displayed to the X server running on System1. This delay is added by the `xauth` application as it tries to communicate with the X server and register a new cookie. In order to avoid this delay and run the X applications from System3, set `TrustX11Applications=yes` for the second connection.

Note: Setting `TrustX11Applications=yes` for the second connection does not create any additional security risk to the X server running on System1. This is because the `xauth` application registers into the existing cookie created on System2 by the initial X11 forwarding (done from System1), for which `TrustX11Applications=no`.

Port Forwarding Settings

Use the following keywords or command line options to configure port forwarding.

Command Line Options

You can use the following options on the **ssh** command line.

Option	Description
-L <i>listening_port:host:hostport</i>	Open the specified port on the Secure Shell client host (<i>listening_port</i>) and forward data to the destination <i>host</i> and <i>hostport</i> .
-R <i>listening_port:host:hostport</i>	Open the specified port on the Secure Shell server host (<i>listening_port</i>) and forward data to the destination <i>host</i> and <i>hostport</i> .
-X	Enables X11 connection forwarding and treats X11 clients as untrusted. Untrusted remote X11 clients are prevented from tampering

Option	Description
	with data belonging to trusted X11 clients.
-x	Disables X11 connection forwarding.
-Y	Enables X11 connection forwarding and treats X11 clients as trusted.

Client Configuration Keywords

You can configure the following settings in the client configuration file. (The global file is `/etc/ssh2/ssh2_config`; the user-specific file is `~/.ssh2/ssh2_config`.)

Keyword	Description
ClearAllForwardings	Clears any local, remote, or dynamically forwarded ports that have already been processed from either a configuration file or the command line. <code>sep</code> and <code>sftp</code> clear all forwarded ports automatically, regardless of the value of this setting. The default is <code>no</code> .
ForwardX11	Equivalent to -X .
GatewayPorts	Controls whether forwarded ports on the Secure Shell client host are available to remote applications. The default is <code>no</code> , which prevents applications running on other computers from connecting to forwarded ports.
LocalForward <i>listening_port:host:hostport</i>	Equivalent to -L .
RemoteForward <i>listening_port:host:hostport</i>	Equivalent to -R .
TrustX11Applications	Specifies whether the X server treats forwarded X11 client applications as trusted. The default is <code>no</code> .
XauthPath	Specifies the full path of the <code>xauth</code> program. The default is <code>/usr/bin/xauth</code> .

Server Configuration Keywords

You can configure the following settings in the server configuration file (`/etc/ssh2/sshd2_config`).

Option	Description
AllowTCPForwarding	Enables or disables all port forwarding. The default is <code>yes</code> .
AllowX11Forwarding	Specifies whether X11 forwarding is allowed. The default is <code>yes</code> .
AllowTCPForwardingForGroups DenyTCPForwardingForGroups	Allows or denies port forwarding for specified groups. Regular expressions are supported.
AllowTCPForwardingForUsers DenyTCPForwardingForUsers	Allows or denies port forwarding only for specified users. Regular expressions are supported.
ForwardACL	Provides detailed control over port forwarding. For details, see <i>Server Configuration Keywords</i> (page 132).
GatewayPorts	Specifies whether remote hosts are allowed to connect to ports forwarded for the client. The default is <code>no</code> .
X11UseLocalHost	Specifies whether the server should bind X11 forwarding to the loopback address. The default is <code>yes</code> .

CHAPTER 9

Controlling Access and Authorization

In this Chapter

Access Control Settings	95
Using Allow and Deny Keywords	96
Configuring User Access	97
Configuring Group Access	98
Configuring Client Host Access	98

Access Control Settings

The table below provides an overview of server settings you can use to control client access to the server.

By default, all client users with an account on the server host can connect to the server, open a terminal session, and access all local files and directories allowed for their user account from any client computer. You can edit the server configuration file (`/etc/ssh2/ssh2_config`) to customize access for client users, groups, and computers.

To	Use
Set the maximum number of connections	MaxConnections
Allow access to specified session types only	SessionRestricted
Control access from client users	AllowUsers DenyUsers UserSpecificConfig
Control access from client groups	AllowGroups DenyGroups UserSpecificConfig
Control access from client hosts	AllowHosts DenyHosts HostSpecificConfig
Control access using TCP Wrappers	LibWrap

To	Use
Restrict sftp and scp users or groups to a confined directory tree	ChrootSftpUsers ChrootSftpGroups
Control upload and download access rights for sftp and scp users.	AllowSftpCommands
Restrict port forwarding	AllowTcpForwardingForGroups DenyTcpForwardingForGroups AllowTcpForwardingForUsers DenyTcpForwardingForUsers ForwardACL GatewayPorts AllowX11Forwarding X11UseLocalHost
Configure PAM authentication	AccountManagement AuthKbdInt.Required PamServiceName UsePamSessions

Using Allow and Deny Keywords

The following keywords are available for controlling access to users, groups, and/or client host computers:

AllowUsers, DenyUsers, AllowGroups, DenyGroups, AllowHosts, DenyHosts, AllowTcpForwardingForUsers, DenyTcpForwardingForUsers, AllowTcpForwardingForGroups, DenyTcpForwardingForGroups, ForwardACL

You can specify users, groups, or hosts for any of these keywords by using a single instance of the keyword with a comma-separated list of values, or by including multiple instances of the keyword, in which case the final assigned value is cumulative over all instances.

The server uses the following logic to determine whether to allow a connection.

1. Check to see if any "Deny" keywords are configured for a given access category (hosts, users, group, or TCP forwarding); and deny access if the client matches any denied expression.
2. Check to see if any "Allow" keywords are configured for the same category.
 - If no "Allow" keywords are configured, access is granted.
 - If *any* "Allow" keywords are configured, the server allows access *only* if the client matches an allowed expression.

Examples

The following samples show how you can allow access, deny access, or use a combination of allow and deny.

To	Example
Allow access only to users whose name starts with "abc".	<code>AllowUsers= abc.*</code>
Deny access to client hosts with an IP address that begins with 123.156.78, and allow access to users on any other client.	<code>DenyHosts=123\.156\.78\..*</code>
Allow access to all hosts in the acme.com domain except badpc, and deny access to clients from any other domain.	<code>AllowHosts=.*\.acme\.com</code> <code>DenyHosts=badpc\.acme\.com</code>
Deny access to all hosts in the acme.com domain, including mypc, and allow access to clients from any other domain.	<code>DenyHosts=.*\.acme\.com</code> <code>AllowHosts=mypc\.acme\.com</code> <code>AllowHosts=.*</code>

Note: Without the final line, no clients would be allowed access. This is because once any client is added to the allow list, clients are allowed access only if they match an allowed expression.

Note: You can also configure user-specific and host-specific settings using *subconfiguration files* (page [30](#)).

Configuring User Access

Edit the server configuration file (`/etc/ssh2/sshd2_config`) to control access to the server. The following keywords configure user access: **AllowUsers**, **DenyUsers**, **AllowTcpForwardingForUsers**, **DenyTcpForwardingForUsers**, **ForwardACL**, **ChrootSftpUsers**, **UserSpecificConfig**. You can specify user names alone, or use the following syntax to include group and/or host information:

```
user[%group][@host]
```

Where *user* is a regular expression for a user (numerical UIDs are not supported), *group* is a regular expression for a group, (numerical GIDs are not supported), and *host* is a regular expression for host (which can be a domain name, IP address, or subnet mask). For example, the following denies access to all members of the interns group at myhost.com:

```
DenyUsers=.*%interns@myhost.com
```

Configuring Group Access

Edit the server configuration file (`/etc/ssh2/sshd2_config`) to control access to the server. The following keywords configure group access: **AllowGroups**, **DenyGroups**, **AllowTcpForwardingForGroups**, **DenyTcpForwardingForGroups**, **ChrootSftpGroups**. These keywords support any valid regular expression. Numerical GIDs are not supported. For example:

```
DenyGroups=interns
```

Configuring Client Host Access

Edit the server configuration file (`/etc/ssh2/sshd2_config`) to control access to the server. The following keywords configure settings for client host computers: **AllowHosts**, **DenyHosts**, **HostSpecificConfig**. You can specify hosts using either IP addresses or domain names. The server first tries to match using the IP address of the client. If that fails, it tries to match using a domain name.

Note: The **ResolveClientHostname** setting controls whether the server attempts to resolve the client IP address to a domain name, and the default is 'yes'. The resolved domain name for a client is the fully qualified domain name. This means that when you add a host to the allow or deny list using a domain name, you must either use a fully qualified domain name, or a regular expression, to ensure that host domain names are handled correctly. For example, if you deny access to the client "mypc", the client `mypc.myhost.com` will be able to connect. You must explicitly deny access to `"mypc\myhost\com"` or use an expression such as `"mypc\.*"` to ensure that this client is denied access.

You can also configure the server to force a match based on IP address. To force matching to a specific IP address, start the host expression using a backslash followed by `i (\i)`. For example:

```
DenyHosts = \i123.45.78.9
```

To match a range of IP addresses using a CIDR (Classless Inter-Domain Routing) subnet, start the host expression using a backslash followed by `m (\m)`. For example:

```
DenyHosts = \m123.123.0.0/16
```

Note: If you use either `\i` or `\m` regular expressions are not supported within the IP address.

Notes

- To configure localhost in any allow or deny list, include IP addresses for all external interfaces and also the local loopback address (127.0.0.1 and 0:0:0:0:0:0:0:1).
 - To configure addresses in any allow or deny list, both IPv4 and IPv6 addresses must be specified. This is particularly important if you are configuring a deny list to ensure that access is blocked.
-

CHAPTER 10

Debug Logging and Auditing

In this Chapter

Client Debugging	101
Server Debugging	102
Auditing (Message Logging)	103
Solaris Audit Support	105

Reflection for Secure IT logs can provide you with detailed information to help in troubleshooting. The information included in the client log is different from that in the server log, and it is often useful to have both.

Client Debugging

You can configure debugging on the `ssh`, `sftp`, and `scp` command line. You can also configure debugging that applies to all of these session types in the client configuration file (`/etc/ssh2/ssh2_config`).

Command Line Options

Use the following command-line options to configure client-side debugging.

Option	Used by	Description
<code>-d debug_level</code>	<code>ssh</code> , <code>ssh-agent</code>	Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)
<code>-D debug_level</code>	<code>scp</code> , <code>sftp</code>	Equivalent to <code>ssh -d</code> .
<code>-v</code>	<code>ssh</code> , <code>scp</code> , <code>sftp</code>	Sets the debug level to verbose mode, which is equivalent to setting the debug level to 2.
<code>-q</code>	<code>ssh</code>	Enables quiet mode, which causes all warning and diagnostic messages, including banners, to be suppressed.

Configuration File Keywords

You can configure the following settings in the client configuration file. (The global file is `/etc/ssh2/ssh2_config`; the user-specific file is `~/.ssh2/ssh2_config`.)

Keyword	Description
LogLevel	Sets the verbosity level for messages sent to the facility specified by SyslogFacility .
QuietMode	Causes all warning and diagnostic messages, including banners, to be suppressed.
VerboseMode	Sets the debug level to 'verbose' mode. Equivalent to <code>-v</code> and LogLevel = verbose.
SyslogFacility	Specifies the facility code used for logging ssh , sftp , and scp messages. The default value is <code>USER</code> . Set the value to 'none' to disable client auditing.

Server Debugging

Server event messages can arise from different sources and be controlled by different configuration options. The following table summarizes **sshd** command-line options and server configuration keywords that affect logging; and describes where to find the output.

To	Use	Output Location	Notes
Debug a single client connection	<code>-d debug_level</code>	<code>stderr</code>	Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.) With this option, sshd terminates after the first client connection closes. This option is independent of the setting for LogLevel .
Enable persistent debugging	<code>-D debug_level</code>	<code>/etc/ssh2</code>	A debug file is created using the following file name format: <code>debugYYMMDD_HHMMSS_uniqueID</code> . This option is independent of the setting for LogLevel .
Suppress debug messages	<code>-q</code> QuietMode	N/A — affects syslog output only	This option overrides LogLevel .
View server startup messages	LogLevel	<code>stderr</code>	Output to <code>stderr</code> includes errors and warnings found while parsing <code>ssh2_config</code> .

Auditing (Message Logging)

The Reflection for Secure IT server provides the following auditing services, which are always enabled.

- Login history
- Currently logged in users
- Failed logins

Output locations are platform-dependent. For details refer to the following table.

Platform	Login history	Current login	Failed login
HPUX (11.11, 11.23) PARISC	/var/adm/wtmp	/etc/utmp	/var/adm/btmp
HPUX (11.23, 11.31) Itanium	/var/adm/wtmps	/etc/utmpx	/var/adm/btmps
AIX 5.2, 5.3, 6.1	/var/adm/wtmp /etc/security/ lastlog	/etc/utmp	/etc/security/ failedlogin /etc/security/ lastlog
Solaris 8, 9, 10	/var/adm/wtmpx	/var/adm/utmpx	/var/adm/ loginlog
RHEL 3, 4, 5	/var/log/lastlog /var/log/wtmp	/var/run/utmp	/var/log/btmp
SLES 9, 10	/var/log/wtmp	/var/run/utmp	/var/log/btmp

Notes:

- Some platforms write to more than one file.
- On some Linux systems, btmp is not present. The server writes to this database if it is present.

Keywords for Configuring Auditing

The output for **sshd** and **sftp-server** messages is affected by both Reflection for Secure IT configuration and **syslogd** configuration. For example, the following entry in `/etc/syslog.conf` configures a facility called `local6` and sends output from that facility to `/var/adm/rsit_log`.

```
local6.info      /var/adm/rsit_log
```

Note: The syntax shown above requires a tab between the two entries.

To configure Reflection for Secure IT to send **sshd** messages to the `local6` facility, include the following line in the server configuration file (`/etc/ssh2/sshd2_config`).

```
SysLogFacility local6
```

The table below summarizes keywords used for configuring auditing.

To	Use	Notes
Specify a facility code for sshd messages	SyslogFacility	The default is 'AUTH'. The value of SyslogFacility must correspond to a facility specified in <code>syslog.conf</code> .
Specify a facility code for sftp-server messages	SftpSysLogfacility	When no value is configured (the default) sftp-server uses the current facility configured for sshd . Use SftpSysLogFacility to specify an alternate facility for sftp server logging. Sending sftp messages to a different facility is often useful for auditing. The value of SftpSysLogFacility must correspond to a facility specified in <code>syslog.conf</code> .
Specify which categories of sftp server messages are sent to the facility specified by SftpSysLogFacility .	SftpLogCategory	The default is 'loginlogout,directorylistings,download,s,modifications,uploads', which configures logging of all categories. You can specify any of those options, plus 'all', or 'none'.

To	Use	Notes
Specify the level of logging to SysLogFacility and SftpSysLogFacility .	LogLevel	After the configuration file is read, messages are processed according to rules defined in <code>syslog.conf</code> . This level applies to both sshd and sftp logging.

Solaris Audit Support

The Solaris operating environment supports auditing of system events, such as file access, process operations and network activity. With auditing enabled, the system provides an audit trail of selected events in the form of a log file, which can be monitored to detect unauthorized use of the system. Auditing in the Solaris operating environment is provided by the Basic Security Module (BSM). Refer to the Solaris documentation for information about configuring BSM.

To generate audit records for Secure Shell connections

- 1 Verify that the Secure Shell (SSH) audit events are mapped to the correct audit class. The following line in the `/etc/security/audit_event` file defines that all Secure Shell events will belong to the login/logout class of events:

```
6172:AUE_ssh:login - ssh:lo
```

Note: If you are running Solaris 8, this entry does not exist, but must be added manually.

- 2 Edit the `/etc/security/audit_control` file, which lets you define a system-wide audit setting for all users. Add the login/logout event class to the `flags:` section:

```
flags:lo
```

- 3 (Optional) If some users need special audit settings, or you want to remove auditing for only some users, you can edit the `/etc/security/audit_user` file. The entries are of the following form:

```
user:always_audit-flags:never_audit_flags
```

For example, the following entry in the `/etc/security/audit_user` file disables auditing for a user 'joe':

```
joe::lo
```

To view the audit log

- 1 Locate the audit log in:

```
/var/audit
```

- 2 Use the **praudit** command to read the binary file format:

```
praudit audit_file
```

The audit entry for the Secure Shell login/logout events tells which user attempted to log in or out, from which host, and whether the connection succeeded or not.

Example 1

An entry for a user ‘joe’, logging on from host sphinx.company.com:

```
header,94,2,login - ssh,,Tue May 13 10:49:44 2010, + 863 msec
subject,joe,joe,other,joe,other,7763,7763,0 2805 sphinx.company.com
text,sshd login joe on /dev/pts/4
return,success,0
```

In this case, the user successfully logged on to the system, and was given a Secure Shell terminal session on `/dev/pts/4`.

Example 2

For Secure Shell logins not requiring a terminal session, such as remote commands or file transfers with **scp** or **sftp**, the terminal or tty number is replaced by the command the server executes on behalf of the user. For example:

```
header,116,2,login - ssh,,Tue May 13 10:49:58 2010, + 361 msec
subject,joe,joe,other,joe,other,7774,7774,0 2806 sphinx.company.com
text,sshd login joe on (no tty)
text,remote command: sftp
return,success,0
```

Example 3

An example of a failed login attempt:

```
header,81,2,AUE_ssh,,Tue May 13 11:22:51 2003, + 462 msec
subject,joe,joe,other,joe,other,8006,8006,0 0 sphinx.company.com
text,invalid password
return,failure: Interrupted system call,-1
```

CHAPTER 11

Troubleshooting

In this Chapter

Troubleshooting Public Key Authentication	107
Troubleshooting Slow File Transfer Speed	109

Troubleshooting Public Key Authentication

The Problem: Public key authentication is configured, but client users are unable to connect using public key authentication.

Check the client configuration

- 1 Confirm that there is a private/public key pair on the client and note the name and location of the private key.
- 2 Open the client configuration file. (If the user has a user-specific file, check both the global and user file.)
 - Confirm that **AllowedAuthentications** includes 'publickey.'
 - Check the **IdentificationFile** setting. Note the name and location of the file. (The default is `~/.ssh2/identification`).
- 3 Open the identification file
 - Confirm that this file includes a line that identifies the client's private key. For example:
`IdKey /home/joe/mykey`
 - Confirm that the key name exactly matches the private key of the key pair. (For example, if your private key has a file extension, this extension needs to be included.)
 - If no path is specified, confirm that the keys are located in the Secure Shell user directory (`~/.ssh2/`)

- 4 Check file and directory permissions. (The second and third bullet items are required if **StrictModes** is enabled on the client, which is the default.)
 - Is the private key readable only by the owner (600)?
 - Is the identification file configured to allow write-access only to the user (600 or 644)?
 - Are the user directory and all parent directories configured to allow write access only to the user (755 or less)?

Check the server configuration

- 1 Confirm that there's a copy of the user's public key in the user-specific configuration directory on the server. The default location is `~/.ssh2`.
- 2 Open the server configuration file.
 - Confirm that **AllowedAuthentications** includes 'publickey.'
 - Check the **AuthorizationFile** setting. Note the name and location of the file. (The default is `~/.ssh2/authorization`.)
- 3 Open the authorization file.
 - Confirm that this file includes a line that identifies the server's copy of the client's public key. For example:
`key /home/joe/mykey.pub`
 - Confirm that the key name exactly matches the public key, including the file extension.
 - If no path is specified, confirm that the key is located in the Secure Shell user directory. (The default is `~/.ssh2/`. This is configurable on the server with the **UserConfigDirectory** keyword.)
- 4 Check file and directory permissions. (The second bullet item is required if **StrictModes** is enabled on the server, which is the default.)
 - Is the authorization file configured to allow write-access only to the user (600 or 644)?
 - Are the user directory and all parent directories configured to allow write access only to the user (755 or less)?

Troubleshooting Slow File Transfer Speed

File transfer speed can be affected by number of factors, including the CPU power of your client and server systems, available *bandwidth* (page 219) for transfers, and *latency* (page 220) in your network. In most cases, you'll see the best performance using the Reflection for Secure IT default settings. In some cases, the following settings may affect transfer speeds.

Check compression settings

You can configure compression on both the client and server using the **Compression** keyword. You can specify compression values 0-9. The default value for the server is 6. The default value for the client is 0. Increasing the value increases the amount of compression. Using higher values results in the use of less network bandwidth, but at the cost of more CPU cycles.

- Lowering the **Compression** value or setting it to zero may improve performance if your files are already compressed, if your network bandwidth is large, or if your computer has limited CPU power.
- Increasing the **Compression** value may improve performance if your files are uncompressed, network bandwidth is small, or your CPU is not a limiting factor.

Check smart copy and checkpoint resume settings

Smart copy and checkpoint resume (page 76) help minimize the amount of time spent repeating unnecessary transfer of data. These features use a series of hashes sent between the client and server to determine if part or all of a file is identical. Identical content is not transferred. This functionality is enabled by default. If you transfer files in a high latency network, the time required to send the hash values across the network can cause delays that exceed the benefit of using these features.

- If you transfer large files across a high latency network, you may be able to improve performance by disabling the smart copy and checkpoint resume feature. To disable these features from the client, set **SmartFileCopy** and **CheckpointResume** to no. To disable these features from the server, set **SmartFileTransfer** to no.

Check High Performance Network (HPN) settings

Reflection for Secure IT supports HPN features that maximize file transfer performance. This functionality is enabled by default.

- To ensure best performance, confirm that **HPNDisabled** is set to 'no' (the default) on both the client and server.

Appendix

In this section

Files Used by the Client	112
Files Used by the Server	114
Client Configuration Keywords	118
Server Configuration Keywords	132
File and Directory Permissions	151
ssh Command Line Options	154
ssh Escape Sequences	161
ssh Exit Values	162
ssh-keygen Command Line Options	163
scp Command Line Options	167
sftp Command Line Options	172
Supported sftp Commands	176
ssh-add Command Line Options	181
ssh-agent Command Line Options	183
sshd Command Line Options	185
ssh-certview Command Reference	187
ssh-certtool Command Reference	189
winpki and pkid Command Reference	193
pkid_config Configuration File Reference	197
pki_mapfile Map File Reference	203
Sample Mapping Rules	210
Sample Map File with RuleType Stanzas	212
PKI Settings Migration	213
PKI Services Manager Return Codes	216

APPENDIX A

Files Used by the Client

`$HOME/.ssh2/ssh2_config`

User-specific configuration file. The format is the same as the system-wide configuration file. Recommended permissions = 644.

`/etc/ssh2/ssh2_config`

System-wide configuration file. This file is installed when you install Reflection for Secure IT. The installed file shows default values as commented out lines. Edit this file to change system-wide settings. For information about keywords and supported values, see `ssh2_config(5)`. Recommended permissions = 644.

`$HOME/.ssh2/hostkeys/key_*.pub`

This directory contains the public keys of hosts trusted by the current user. By default, keys are added automatically to this location when the user answers 'yes' in response to an unknown host prompt. (This behavior can be changed using the **StrictHostKeyChecking** keyword in the configuration file.) Starting with version 7.0, host keys use the following file name format:

`key_port_host,IP.pub`

Where *port* is the port used for the ssh connection, *host* is the host name, and *IP* is the host IP address.

Earlier versions used `key_port_host.pub`, and this format is still supported.

`/etc/ssh2/hostkeys/key_*.pub`

System-wide known hosts. Hosts with keys in this list are trusted for all users of the computer. No keys are installed to this location automatically. To add a system-wide trusted host, create this directory and put a copy of the host public key in it. Use the file name format described above for `$HOME/.ssh2/hostkeys/key_*.pub`.

`$HOME/.ssh2/identification`

An identification file is required if you use public keys or certificates for user authentication. (This is the default file name and location. You can redefine the name and/or location of the identification file on the **ssh** command line using **-i** or in the configuration file using the **IdentificationFile** keyword.) The identification file contains a list of one or more private keys held by a client user. Any listed key can be used by the client for user authentication. If more than one key is listed, the client tries the first key in the list, then continues trying the other keys in order. If no path information is provided, the client looks for listed keys in `$HOME/.ssh2/`. This file should have user-only write access (permissions = 600 or 644).

For standard keys use the following syntax to add keys to the list:

```
IdKey <keyname>
```

For example:

```
IdKey id_dsa_2048_a
```

For keys associated with an X.509 certificate use the following syntax.

```
CertKey <keyname>
```

The associated certificate must be in the same directory as the specified key and use the same base name with a `.crt` file extension.

Note: For public key authentication, you also need to configure the server. For certificate authentication, you need to install and configure Reflection PKI Services Manager and also configure the server.

Note: When **ChrootSftpUsers** or **ChrootSftpGroups** is enabled, connected users see additional subdirectories (`etc` on all platforms and `dev` on AIX) added to their home directory. These directories cannot be moved or deleted. The `etc` directory contains two required files. The `rsit.conf` file identifies the installation location of files required by Reflection for Secure IT. The `localtime` file is needed so that processes such as logging can get the current time. The system `localtime` file is in a location that cannot be accessed by a chrooted user. Users running on AIX also require `/dev/null`, which is needed for correct logging to `syslog`.

APPENDIX B

Files Used by the Server

The server uses system-wide files (in `/etc/ssh2`) for all connections. Files in user-specific directories (`~/.ssh2` by default) apply to connections from individual client users.

System-wide server files

`/etc/ssh2/sshd2_config`

The global server configuration file. This file must not be writable by group or other. For file format and supported settings see `sshd2_config(5)`. Recommended permissions = 644.

`/etc/ssh2/hostkey`

The default private key of the public/private key pair used to identify the server to clients. This file should be readable and writable only by root. This file must be limited to user-only read and write access. If permissions are not sufficiently restricted, public key authentication will fail. Recommended permissions = 600.

`/etc/ssh2/hostkey.pub`

The default public key of the public/private key pair used to authenticate the server to clients. Recommended permissions = 644.

`/etc/ssh2/subconfig`

Directory for optional user-specific and host-specific subconfiguration files. Recommended permissions = 700.

`/etc/ssh2/subconfig/<subconfig_file>`

User-specific and host-specific subconfiguration files. For details see **SUBCONFIGURATION FILES** in `sshd2_config(5)`.

`/etc/ssh2/environment`

If this file is present, it sets environment variable settings to use for all Secure Shell client connections to this server. (The keyword **SettableEnvironmentVars** controls which environment variables can be set.) Recommended permissions = 644. Note: Environment variable settings specified in this file override any values configured in standard system files such as `/etc/default/login` and `/etc/environment`. If the same environment variable is configured in this global file and also in a user-specific environment file (`~/.ssh2/environment`), the user-specific value overrides the global value. The pound sign (`#`) marks comment lines. The syntax is:

```
environment_variable=value
```

`/etc/nologin`

Limits login to root. If this file exists, only root is allowed to login. The text of `nologin` is displayed to anyone else who attempts to log in.

`<pidfile>/sshd2_22.pid`

Contains the PID of the process listening for incoming connections. The PID directory is determined by your operating system. The port number (22 by default) encoded in this name is determined by the value of the **Port** keyword. You can specify a different name or location using the **PidFile** keyword.

`/etc/motd`

The message-of-the-day file. The text of this file is displayed when a user logs in. The **PrintMotd** keyword can be used to turn off this display.

`/etc/ssh2/radius_config`

A user-created file listing one or more RADIUS authentication servers. The file name suggested above is not required. After you create this file, use the **RadiusFile** keyword to specify your file name. For each RADIUS server, you need to enter the name, port, and shared secret. Recommended permissions = 600. The syntax is:

```
server1:port1:shared_secret1
```

```
server2:port2:shared_secret2
```

User-specific server files

`~/.ssh2`

The default directory for user-specific files on the server. (You can specify a different location with the **UserConfigDirectory** keyword.) Recommended permissions = 700.

`~/.ssh2/authorization`

The default client authorization file. (You can specify a different file with the **AuthorizationFile** keyword.) This file is required for Secure Shell public key authentication of client users. Each user must have an authorization file in that user's directory. This file must be limited to user-only write access. If permissions are not sufficiently restricted, public key authentication will fail. Recommended permissions = 600.

The file contains a list of key files that the server will use during public key authentication. If the key presented by the client doesn't match any of the keys listed in the authorization file, public key authentication fails. Keywords are not case sensitive and the pound sign (#) marks comment lines. The supported keywords are:

key

Specifies keys the server will accept for this user. The format for key entries is "key" followed by the name of a file that contains a public key. Keys are assumed to be in the user-specific configuration directory (`~/.ssh2` by default) unless you specify an absolute path. For example, the following lines authorize the user to authenticate using either of the specified keys.

```
key mykey.pub
key id_rsa_2048_a.pub
```

options

Use this optional keyword to specify options that apply to the preceding key. All options for a given key must be configured on a single line. White space is allowed. Options must be configured on the line immediately following the line containing the key. The format is:

```
Options option_keyword="arg", [option_keyword="arg"],...
```

Three **Options** keywords are supported: **command**, **allow-from**, and **deny-from**

command *command*

The specified command is executed on the remote host, then the connection is closed. For example, with this configuration, the script "myscript" runs whenever mykey.pub is used for authentication.

```
key mykey.pub
options command="sh myscript"
```

allow-from *IP-address*

The key is allowed only for connections from the specified IP address. For example, the following configuration allows the specified key to be used only for connections from IP addresses starting with "150." and "10.10."

```
Key /home/joe/.ssh/mykey.pub
options allow-from="150\.*,10\.*"
```

deny-from *IP-address*

The key is not allowed for connections from the specified IP address.

Notes: To configure addresses in any allow or deny list, both IPv4 and IPv6 addresses must be specified. This is particularly important if you are configuring a deny list to ensure that access is blocked. To configure localhost in any allow or deny list, include IP addresses for all external interfaces and also the local loopback address (127.0.0.1 and 0:0:0:0:0:0:0:1).

~/.hushlogin

If this file is present, it suppresses display of the user's last login, the message of the day, and the mail check.

~/.ssh2/environment

If this file is present, it sets environment variables to set for this user at login. (The keyword **SettableEnvironmentVars** controls which environment variables can be set.) Recommended permissions = 644. Note: Environment variable settings specified in this file override any values configured in standard system files such as /etc/default/login and /etc/environment, and also override settings configured in the global file (/etc/ssh2/environment). The pound sign (#) marks comment lines. The syntax is:

environment_variable=value

APPENDIX C

Client Configuration Keywords

You can configure the following settings in the client configuration file. (The global file is `/etc/ssh2/ssh2_config`; the user-specific file is `~/.ssh2/ssh2_config`.) You can also configure these settings using the `-o` option on the `ssh` command line.

AddressFamily

Specifies which address formats are supported by the client. The allowed values are 'any' (allow the system to decide which address family to use), 'inet' (accept only IPv4), and 'inet6' (prefer IPv6 but accept IPv4). The default is 'inet'. You can also configure address family preference using the `-4` and `-6` command line options.

AllowedAuthentications

Specifies which authentication methods the client attempts, and the order in which they are tried. The supported methods are: 'gssapi-keyex', 'gssapi-with-mic', 'publickey', 'keyboard-interactive', and 'password'. Use a comma-separated list to specify supported methods. The client attempts authentication methods in order from first to last. The authentication technique used for the connection is the one highest in the client order of preference that is also allowed by the server. If the server is configured to require more than one method, multiple authentication methods may be needed to establish a connection. To support automated scripts, the least interactive methods should be placed first in the list. The default is 'gssapi-with-mic, publickey, keyboard-interactive, password'.

AuthenticationSuccessMsg

Specifies whether to display the following message when authentication has been completed successfully: "Authentication successful." The allowed values are 'yes' and 'no'. The default is 'yes'.

BatchMode

Specifies whether to disable all queries for user input, including password and passphrase prompts, which is useful for scripts and batch jobs. If **StrictHostKeyChecking** is set to 'ask' and **BatchMode** is set to 'yes', the client assumes a "no" response to all queries about unknown host keys. The allowed values are 'yes' and 'no'. The default is 'no'.

CheckHostIP

Specifies whether host IP address checking is performed using the host name and IP address encoded in the public key file name. When a user accepts a new host key, the key is added to the known hosts store using the format `key_port_host,IP.pub`. When **CheckHostIP** is enabled, host authentication fails if the actual IP of the specified host doesn't match the encoded IP address for that host. Enabling this setting helps detect DNS spoofing if the host key changes. The allowed values are 'yes' and 'no'. The default is 'no'.

Note: Host keys added to the host key store using versions earlier than v. 7.0 do not include the host IP address. Disable **CheckHostIP** if you use keys with the older format.

CheckpointResume

When this setting is 'yes' (the default), interrupted file transfer resumes at the point of interruption. When this setting is 'no' transfers always start over. Note: Checkpoint resume can also be disabled on the server using the **SmartFileTransfer** keyword.

Ciphers

Specifies one or more (comma-separated) encryption algorithms supported by the client. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'.

You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.

The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers).

You can also specify encryption algorithms on the **ssh** command line using the **-c** option. The default is 'AnyStdCipher'.

ClearAllForwardings

Clears any local, remote, or dynamically forwarded ports that have already been processed from either a configuration file or the command line. The allowed values are 'yes' and 'no'. The default is 'no'. Note: **scp** and **sftp** clear all forwarded ports automatically regardless of the value of this setting.

Compat.RSA.HashScheme

Specifies whether the MD5 hash algorithm is supported for verifying the digital signature for RSA keys used in public key or X.509 certificate authentication. The allowed values are 'yes' and 'no'. When this keyword is set to 'no' (the default), only signatures with SHA-1 hashes are accepted. When it is set to 'yes' signatures with either SHA-1 or MD5 hashes are accepted.

Compression

Specifies the level of compression. You can specify compression values 0-9. Increasing the value increases the amount of compression. Using higher values results in the use of less network bandwidth, but at the cost of more CPU cycles. Level 6 is equivalent to 'yes'. Level 0 is equivalent to 'no'. The default is 'no' (0).

Note: Compression can be disabled on the **ssh** command line using the **-C** option, but can only be enabled using this keyword.

ConnectionReuse

Specifies whether new **ssh**, **scp**, and **sftp** sessions can reuse an established connection. This feature allows you to start new sessions without having to reauthenticate. The allowed values are 'yes' and 'no'. The default is 'no'. When set to 'yes', a new session reuses an existing tunnel if the target host, port, and user are all identical to those used for the established connection. When set to 'no', the client establishes a new connection for each session, which means that each new connection repeats the authentication process and also applies any modified connection-specific settings (such as forwards and ciphers).

Note: Connection reuse may fail if the server administrator has configured restricted directory access using **ChrootSftpGroups** or **ChrootSftpUsers**.

ConnectionTimeout

Specifies the maximum time (in seconds) that the client waits when trying to connect to the server. The default is set to 0 (zero), which means that the client sets no limit and the actual limit is determined by the operating system.

DefaultDomain

Specifies a default domain name. You can add this setting to your configuration file if you want to be able to enter a short host name on the command line, but send a fully qualified domain name to make the connection. If you have configured a value for **DefaultDomain** and you enter a host name that doesn't contain any "." (dot) characters, the **DefaultDomain** value is concatenated to the host name using a "." character. (Note: You can include an optional dot at the beginning of the **DefaultDomain** string; the first "." in this string is ignored.) Any alphanumeric character is accepted as a value. For example, if **DefaultDomain** is set to either "acme.com" or ".acme.com", the command "ssh joe@myhost" is sent as "ssh joe@myhost.acme.com".

DontReadStdin

Redirects stdin from /dev/null, which prevents reading from stdin. You can also configure this on the **ssh** command line using the **-n** option. The allowed values are 'yes' and 'no'. The default is 'no'.

EscapeChar

Sets the escape character for the terminal session. The default character is a tilde (~). Setting the escape character to 'none' means that no escape character is available and the tilde acts like any other character. For details, see ESCAPE SEQUENCES in the **ssh** man page. You can also set the escape character on the **ssh** command line using the **-e** option.

ExitOnForwardFailure

Specifies whether **ssh** terminates the connection if all requested dynamic, local, and remote port forwardings cannot be configured. The allowed values are 'yes' and 'no'. The default is 'no'.

FileCopyAsciiExtensions

Specifies which file types use ASCII transfer during **sftp** sessions when auto mode transfer is enabled. All other files use binary transfer. Specify a comma or space-separated list. Wildcard (zsh-glob) characters are supported. Don't precede file extensions with a period. To specify extensions containing spaces, use quotation marks around the extension or use a backslash as an escape character. The default is 'txt, htm*, pl, php*'. (You can use the **setext** during an **sftp** session to specify a different file list for that session. Use **getext** to display the current list.)

Note: This setting is only relevant when auto transfer is enabled. The transfer method is set to binary by default. To enable auto transfer, use the **sftp** command "auto". To display the current transfer mode, use "ascii -s".

FipsMode

Specifies whether all connections will be made using security protocols and algorithms that meet FIPS 140-2 standards. The allowed values are 'yes' and 'no'. The default is 'no'.

ForcePTTYAllocation

Forces a tty allocation even if a command is specified. The allowed values are 'yes' and 'no'. The default is 'no'. You can also configure this on the **ssh** command line using the **-t** option.

ForwardAgent

Specifies whether a connection to the authentication agent (if established) is forwarded to the remote machine. The allowed values are 'yes' and 'no'. The default is 'yes'.

ForwardX11

Enables X11 connection forwarding and treats X11 clients as untrusted. Untrusted remote X11 clients are prevented from tampering with data belonging to trusted X11 clients. The allowed values are 'yes' and 'no'. The default is 'yes'. You can also configure this on the **ssh** command line using the **-X** option.

GatewayPorts

The gateway ports setting controls whether locally forwarded ports are available to remote applications. By default this setting is not enabled, and the client uses the loopback address ("localhost" or 127.0.0.1) when it opens a socket for local port forwarding. This prevents applications running on other computers from connecting to the forwarded port. When you enable gateway ports, a remote application client can open a socket using the Secure Shell client's Ethernet address (such as an IP address, a URL, or a DNS name). For example, a Secure Shell client running on acme.com might be configured to forward port 8088. When gateway ports are not enabled, the forwarded socket is localhost:8088. When gateway ports are enabled, the forwarded socket is acme.com:8088. The allowed values are 'yes' and 'no'. The default is 'no'. You can also configure this on the **ssh** command line using the **-g** option.

Caution: This option should be used with extreme caution (and never with Internet-facing network adapters), because the client performs no authentication of remote host connections. If the application to which this connection is forwarded does not perform its own authentication, then all remote hosts connections are allowed unrestricted access to that application.

GoBackground

Use this keyword when you have configured port forwarding and you want the Secure Shell session to run in the background. The allowed values are 'yes', 'no', and 'oneshot'. The default is 'no'. If at least one port forwarding rule is configured, both 'yes' and 'oneshot' send the session to the background after authentication is complete. When you specify 'yes', the Secure Shell session remains in the background and continues to accept forward requests indefinitely until you manually kill the process. (This is equivalent to using **-f** on the **ssh** command line.) When you specify 'oneshot', the background session waits for only one forwarded connection to occur and exits as soon as the forwarded connection is closed. (This is equivalent to using **-fo** on the **ssh** command line.)

GSSAPIDelegateCredentials

Specifies whether to forward (delegate) GSSAPI credentials to the server. The allowed values are 'yes' and 'no'. The default is 'yes'.

Host

Specifies the actual host name or IP address to use for a connection. The default is an empty string. This keyword can be used in combination with a host stanza expression to create an alternate name for connecting to a host. When this keyword appears outside any stanza, it can be used to specify a default host for the connection.

HostCA

This keyword is deprecated. It is a synonym for **TrustAnchor**.

HostCANoCRLs

This keyword is deprecated. It is a synonym for **TrustAnchor**. Note: Certificate revocation checking cannot be configured using the Reflection for Secure IT configuration file. Use Reflection PKI Services Manager to configure revocation checking.

HostCertNameCheck

Specifies whether server authentication using a certificate requires host name checking. When **HostCertNameCheck** is 'yes', authentication succeeds only if the host name or IP address specified for the connection is included in the allowed identity set for the certificate. (Use the PKI Services Manager map file to configure allowed identities.) When **HostCertNameCheck** is 'no', the client ignores the allowed identity set and accepts any valid certificate. When **HostCertNameCheck** is 'ask' (the default), the user receives a prompt when the server name is not an allowed identity, and is asked whether or not to continue.

HostKeyAlgorithms

Specifies, in order of preference, the host key algorithms proposed by the client. This setting is useful when the server is configured for both certificate and standard host key authentication. Secure Shell protocol allows only one attempt to authenticate the host. If the host presents a certificate, and the client is not configured for host authentication using certificates, the connection fails. (This is different from user authentication in which multiple authentication attempts are supported.) The default is 'x509v3-sign-rsa,x509v3-sign-dss,ssh-rsa,ssh-dss'.

HostKeyAlias

Specifies an alias to use instead of the real host name when a host key is saved to the client's directory of known host keys. Host keys are stored using this naming format: `key_port_host,IP.pub`. The value you specify replaces the `host` portion of the stored host key name. This option is useful for tunneling Secure Shell connections, or when multiple servers are running on a single host.

HPNDisabled

Specifies whether Reflection for Secure IT uses HPN dynamic TCP window features to enhance performance. When **HPNDisabled** = 'no' (the default), Reflection for Secure IT adjusts the TCP window and TCP receive buffers to optimize performance. When **HPNDisabled** is 'yes', the receive buffer is set to 64 KB.

IdentificationFile

Specifies an alternate identification file to use for public key authentication. The file location is assumed to be in the current working directory unless you specify a fully-qualified or relative path. The default identity file is `~/.ssh2/identification`. For details, see the **FILES** section below. You can also configure this on the **ssh** command line using the **-i** option.

IdentityFile

This keyword is deprecated. It is the equivalent of **IdentificationFile**.

KeepAlive

Specifies whether the client sends TCP keep-alive messages to the server. This keyword is deprecated. Use **ServerAliveInterval** instead. The allowed values are 'yes' and 'no'. The default is 'yes'.

KEXs

Specifies which key exchange algorithms the client supports. Supported values are 'diffie-hellman-group1-sha1' and 'diffie-hellman-group14-sha1'. Multiple algorithms can be specified as a comma-separated list. The default value is 'diffie-hellman-group14-sha1,diffie-hellman-group1-sha1'.

LibGssKrb5

Use this setting if you use GSSAPI (Kerberos 5) authentication. It specifies the fully-qualified path to the Kerberos library called `libgssapi_krb5.so`

LocalForward

Use this keyword to forward connections from an arbitrary port on the client through the secure tunnel. The syntax for configuring this setting is:

```
[protocol/][listening_host:]listening_port:host:hostport
```

When a Secure Shell connection is established, the Secure Shell client opens a socket on the Secure Shell client host using the designated local port (*listening_port*). (On client hosts with multiple interfaces, use *listening_host* to specify which interface.) Configure your application client (the one whose data you want to forward) to send data to the forwarded socket (rather than directly to the destination host and port). When that client establishes a connection, all data sent to the forwarded port is redirected through the secure tunnel to the Secure Shell server, which decrypts it and then directs it to the destination socket (*host*, *hostport*). Unless the gateway ports option is enabled, the forwarded local port is available only to clients running on the same computer as the Secure Shell client. The optional *protocol* can be **tcp** or **ftp**.

Note: If the final destination host and port are not on the Secure Shell server host, data is sent in the clear between the Secure Shell host and the application server host.

The following example uses local forwarding to secure e-mail communications between a mail client running on the same computer as the Secure Shell client and a mail server running on the same computer as the Secure Shell server. The local mail client is configured to send communications to local port 14300. Data received on port 14300 is forwarded through the secure tunnel to the server, where it is redirected to port 143.

```
LocalForward=14300:localhost:143
```

In the following example, FTP communications sent from an FTP client (on the same computer as the Secure Shell client) are forwarded to an FTP server running on `myhost.com`. With this configuration, you would configure the FTP client to connect to `localhost:2121`.

```
LocalForward=ftp/2121:myhost.com:21
```

You can also configure local forwarding on the **ssh** command line using the **-L** option.

LogLevel

Sets the verbosity level used for **ssh** messages. Allowed values are 'fatal', 'error', 'quiet', 'info', 'verbose', 'debug1' ('debug' and 1 are equivalent), 'debug2' (2 is equivalent), 'debug3' (3 is equivalent), and 'trace' ('debug99' and 99 are equivalent). The syslog level associated with these values is CRIT for fatal, ERROR for error and quiet, INFO for info and verbose, and DEBUG for debug1, debug2, debug3, and trace. The default is 'info'.

Note: Setting logging to 'trace' can increase your security risk. At this level, information leakage is a concern, as unencrypted protocol information may be written out. Also, the volume of information written may fill up disk space rapidly, potentially causing the host or Reflection for Secure IT to stop responding.

MACs

Specifies which MACs (message authentication codes) are supported by the client. Allowed values are 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', 'hmac-md5-96', 'hmac-ripemd160', 'hmac-sha256', and 'hmac-sha512'. Use 'AnyMac' to support all of these. Use 'AnyStdMac' to support 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', and 'hmac-md5-96'. Additional options are 'none', 'any' (equivalent to AnyMac plus 'none'), and 'AnyStd' (equivalent to 'AnyStdMac' plus 'none'). Multiple MACs can also be specified as a comma-separated list. When 'none' is the agreed on MAC, no message authentication code is used. Because this provides no data integrity protection, options that include 'none' are not recommended.

You can also configure MACs on the **ssh** command line using the **-m** option. The default is 'AnyStdMac'.

NoHostAuthenticationForLocalHost

This option disables host authentication when the client connects to localhost. It is useful when the home directory is shared across computers. In this situation localhost will refer to a different host on each of the computers, and the client user will get many warnings about changed host keys. Setting this to 'yes' disables authentication for localhost so the user won't see these warnings. The allowed values are 'yes' and 'no'. The default is 'no'.

NumberOfPasswordPrompts

Specifies the number of password prompts to respond to before giving up. Note: The server can also set a maximum number of allowed password attempts. If you set **NumberOfPasswordPrompts** to a larger value than is configured by the server, the connection will fail when the server limit is reached. The default is 3.

PasswordPrompt

Specifies the prompt to display for password authentication. Two variable options are supported: %r is replaced by the user name and %h is replaced by the host name. The default is "%r@%h's password:" (This setting has no effect on keyboard-interactive authentication.)

PkidAddress

Specifies the port used to connect to PKI Services Manager. Use the format *host:port*. The default is `localhost:18081`. If you specify a host and omit the port, the default PKI Services Manager port (18081) is used.

PkidPublicKey

Specifies the name and location of the public key used by to confirm the identity of Reflection PKI Services Manager. The default is `/opt/attachmate/pkid/config/pki_key.pub`.

Port

Specifies the port to connect to on the server. The default is 22, which is the standard port for Secure Shell connections. You can also configure this on the **ssh** command line using the **-p** option.

ProxyCommand

Specifies the command to use to connect to the server. The command string extends to the end of the line, and is executed with the user's shell. Two variable options are supported in the command: '%h' is replaced by the host name and '%p' by the port. The command can be anything that reads from stdin and writes to stdout. The command should eventually connect to a Reflection for Secure IT server. You can use **ProxyCommand** in conjunction with a command such as **nc** (or **netcat**) that provides proxy support. For example, the following command uses **nc** to connect via an HTTP proxy at 198.168.2.1:

```
ProxyCommand /usr/bin/nc -X connect -x 198.168.2.1:8080 %h %p
```

The default is 'none', which disables this option. (This is equivalent to specifying an empty string).

Note: **CheckHostIP**, is not available for connections made with a proxy command.

QuietMode

Enables quiet mode, which causes all warning and diagnostic messages, including banners, to be suppressed. The allowed values are 'yes' and 'no'. The default is 'no'. You can also configure this on the **ssh** command line using the **-q** option.

RekeyIntervalSeconds

Specifies the number of seconds the client waits before initiating a negotiation for a new session key. The value must be an integer. The default is 3600. This key can be used in combination with **RekeyLimit**, in which case the client initiates a new key exchange whenever the first limit is reached.

RekeyLimit

Specifies the maximum amount of data that can be transmitted before the client initiates a negotiation for a new session key. The argument is the number of bytes, with an optional suffix of 'K', 'M', or 'G' to indicate kilobytes, megabytes, or gigabytes, respectively. Set this value to 0 (zero) to use the default value. The default is between '1G' and '4G', depending on the cipher. This key can be used in combination with **RekeyIntervalSeconds**, in which case the client initiates a new key exchange whenever the first limit is reached.

RelaySignals

Specifies which signals the client should relay to the server. **RelaySignals** accepts a comma-separated list of any of the following signals: ABRT, ALRM, FPE, HUP, ILL, INT, PIPE, QUIT, SEGV, TERM, USR1, USR2. The signals KILL and STOP cannot be caught, blocked, or ignored, so these signals are not supported. No signals are relayed by default.

RemoteForward

Use this keyword to forward connections from an arbitrary port on the server through the secure tunnel. The syntax for configuring this setting is:

```
[protocol/][listening_host:]listening_port:host:hostport
```

When the Secure Shell connection is established, the Secure Shell server opens a socket on its host (the computer running the Secure Shell server) using the designated remote port (*listening_port*). (On server hosts with multiple interfaces, use *listening_host* to specify which interface.) Configure your client application (the one whose data you want to forward) to send data to the forwarded socket (rather than directly to the destination host and port). When that client establishes a connection, all data sent to the forwarded port is redirected through the secure tunnel to the Secure Shell client, which decrypts it and then directs it to the destination socket (*host,hostport*). The optional *protocol* can be **tcp** or **ftp**.

In the following example, FTP communications sent from an FTP client (on the same computer as the Secure Shell server) are forwarded to an FTP server (on the same computer as the Secure Shell client). With this configuration, you would configure the FTP client to connect to port 3333.

```
RemoteForward=ftp/3333:localhost:21
```

You can also configure remote port forwarding on the **ssh** command line using the **-R** option.

SendNOOPackets

Specifies whether the client sends NOOP messages through the Secure Shell channel to the server. Setting this to 'yes' is equivalent to setting **ServerAliveCountMax** to 3 and **ServerAliveInterval** to 600. The allowed values are 'yes' and 'no'. The default is 'no'.

ServerAliveCountMax

Use this setting to close sessions to servers that have become unresponsive. It is relevant only when **ServerAliveInterval** is set to a non-zero value. **ServerAliveCountMax** sets the maximum number of server alive messages the client will send without receiving a return message from the server. When this threshold is reached, the client terminates the session. The default is 3. For example, if **ServerAliveInterval** is set to 600, and **ServerAliveCountMax** is 3, the client sends a message to the server every 10 minutes until it has sent 3 messages to the server without response. This means that the client will close an unresponsive connection after about 30 minutes.

ServerAliveInterval

Sets a time interval, in seconds, for sending NOOP messages to the server through the Secure Shell channel. The client sends a message to the server when no data has been received from the server during the specified interval. Setting this to a non-zero value can be used to inform the Secure Shell server and the TCP stack that the client is still alive, inform all networking hardware (such as routers and NATs) that the Secure Shell connection is still active, and detect network problems and application problems. Use this setting in conjunction with **ServerAliveCountMax** to terminate a connection to a server that has become unresponsive. The default is 0; which configures the client to send no messages.

SetRemoteEnv

Specifies an environment variable to set on the server before executing a shell or a command. The value should be in the form: `VAR=val`, where `val` can be empty. The argument must be uppercase.

Note: Values set with this keyword are cumulative; you can set multiple variables by configuring this keyword multiple times in one or more configuration files.

SmartFileCopy

Specifies whether Reflection for Secure IT checks for identical files before doing a file transfer. When this setting is 'yes' (the default), existing files are checked for equality and no data transfer takes place if the files are identical. When this setting is 'no', no check for equality is made and existing files are always overwritten. Note: Smart file copy can also be disabled on the server using the **SmartFileTransfer** keyword.

StrictHostKeyChecking

This keyword determines how the client behaves when a host presents an unknown key for authentication. The possible values are:

'yes' - Connections succeed only when host keys have been manually copied to the user's host key directory (`~/.ssh2/hostkeys`), or the system-wide host key directory (`/etc/ssh2/hostkeys`). The client does not add host keys to the user's computer. This is the most secure option.

'ask' - This is the default. The client displays a prompt asking if the user wants to accept a key from an unknown host. This prompt shows the host key fingerprint, which can be used to verify the host's identity. If the user answers 'yes', the client adds the host key to the known host keys in the user's directory (`~/.ssh2/hostkeys`) and uses this key to verify the host's identity in subsequent connections.

'no' - Unknown host keys are added automatically to the user's host key directory (`~/.ssh2/hostkeys`) and used to verify the host's identity in subsequent connections. The user never knows when an unknown host key is presented.

StrictModes

Specifies file and directory permissions required for public key authentication. The allowed values are 'yes' and 'no'. The default is 'yes'. When set to 'yes', the user directory (`~/.ssh2/`) and all parent directories must be writable and executable only by the user (mode 755 is accepted). Recommended permissions for the user directory = 700. The user identification file (`~/.ssh2/identification` by default) must be configured for user-only read/write access (600 is recommended, 644 is accepted). When set to 'no' these file permissions are not enforced and sensitive files and information could be compromised.

Note: Additional file permission restrictions are enforced for all private keys. Keys must be configured for user-only read access regardless of the current **StrictModes** setting. If access to the private key is not sufficiently restricted, public key authentication will always fail. Recommended permissions for private keys = 600.

SysLogFacility

Specifies the facility code used to log messages for **ssh**, **sftp**, and **scp** connections. The default is 'USER'. When this value is 'none', Reflection for Secure IT disables logging to syslog. Other valid values are platform-dependent. See `syslog(3)` and `syslog.conf(5)`.

TrustAnchor

This keyword is optional and is relevant only if you use certificates for server authentication. By default, Reflection PKI Services Manager validates certificates presented for authentication using all of the trust anchors you have configured. Use this keyword to limit which of the Reflection PKI Services Manager trust anchors can be used for certificate validation. You can specify either Subject DN (Distinguished Name) from a certificate available in the PKI Services Manager store, or use the file name of a certificate. Note that the specified trust anchors must be also be configured for Reflection PKI Services Manager (using the PKI Services Manager **TrustAnchor** keyword).

TrustX11Applications

Specifies whether the X server treats forwarded X11 client applications as trusted. The allowed values are 'yes' and 'no'. The default is 'no'. Set this to 'yes' to give remote X11 clients full access to the X11 display. When this is set to 'no', X11 applications are treated as untrusted. This avoids the risk created when a connection to a compromised host allows applications on that host to "sniff" input operations using the forwarded X11 connection.

User

Specifies the user name for the remote server. You can configure different user names for different hosts by defining this setting in host-specific stanzas. The default is the current value of the environment variable `$USER`.

VerboseMode

Sets the debug level to verbose mode, which is equivalent to setting the debug level to 2. You can also configure this on the **ssh** command line using the **-v** option. The allowed values are 'yes' and 'no'. The default is 'no'.

XauthPath

Specifies the location of the `xauth(1)` program. The default (for example `/usr/X11R6/bin/xauth`) is system-dependent.

APPENDIX D

Server Configuration Keywords

You can configure the following settings in the server configuration file. The default file is `/etc/ssh2/ssh2_config`.

AccountManagement

Configures the account management system that **sshd** uses to validate a user account. Account management services determine if an account is active, and whether or not a password is still valid. The allowed values are 'password', 'pam', and 'none'. The default is 'pam,password', which requires the user account to pass validation by both systems.

pam - Use PAM for account management. PAM account management applies to all sessions, regardless of the authentication method (or methods) used. If an account is locked, the connection is refused.

password - Use the password database to validate the account.

none - Use no account validation. Use this only for troubleshooting.

AddressFamily

This setting is used by the server when it creates a listening, session, or forwarding TCP socket. The allowed values are 'any' (allow the system to decide which address family to use), 'inet' (accept only IPv4), and 'inet6' (accept only IPv6). The default is 'inet'. Note: The current value of **ListenAddress** may also affect whether or not the server accepts connections using IPv4 or IPv6 addresses.

AllowAgentForwarding

Specifies whether agent forwarding is allowed. The allowed values are 'yes' and 'no'. The default is 'yes'.

AllowedAuthentications

Specifies which authentication methods the server supports. The client and server agree on one or more authentication methods during the initial connection process, based on both client and server configuration. (Use **RequiredAuthentications** to require one or more authentication methods. **RequiredAuthentications** overrides **AllowedAuthentications**.)

The supported authentication methods are 'gssapi-keyex', 'gssapi-with-mic', 'publickey', 'keyboard-interactive', and 'password'. The default is 'gssapi-with-mic, publickey, keyboard-interactive, password'.

AllowedPasswordAuthentications

This keyword is no longer supported. If you used it in previous versions, you need to manually migrate your setting. Refer to the following keywords: **AllowedAuthentications**, **RequiredAuthentications**, and **AuthKbdInt.Required**.

AllowGroups

Use this keyword to allow login only for users who are members of a specified group. Regular expressions are supported. For details, see *Configuring Group Access* (page 98). If this keyword is not configured, all groups are allowed to log in.

AllowHosts

Use this keyword to allow login only for specified client hosts. Regular expressions are supported. For details, see *Configuring Client Host Access* (page 98). If this keyword is not configured, all client hosts are allowed.

Notes:

If you configure a host expression using the domain name (rather than IP address), you must also set **ResolveClientHostName** to 'yes'. When **ResolveClientHostName** is 'yes', the resolved name is the fully qualified domain name. This means that when **RequireReverseMapping** is 'yes', you must specify a fully qualified domain name or use a regular expression for the host name to ensure that connections from an IP address are handled correctly.

To configure addresses in any allow or deny list, both IPv4 and IPv6 addresses must be specified. This is particularly important if you are configuring a deny list to ensure that access is blocked. To configure localhost in any allow or deny list, include IP addresses for all external interfaces and also the local loopback address (127.0.0.1 and 0:0:0:0:0:0:1).

AllowSftpCommands

Controls what kinds of operations users can perform using **sftp** and **scp** commands from Reflection for Secure IT clients. This keyword supports a comma-separated list of one or more of the following: 'all', 'none', 'browse', 'download', 'upload', 'delete', 'rename'. The upload option enables users to modify files, create files, create directories, or modify file attributes on the server. The download option enables users to read file contents. The default is 'all'.

Note: This setting affects both **sftp** and **scp** connections from Reflection for Secure IT clients. The **SessionRestricted** keyword also affects access to file transfers. The default value for **SessionRestricted** is 'shell, exec, subsystem'. For Reflection for Secure IT clients, the 'subsystem' session type is required for both **sftp** and **scp** transfers. For OpenSSH-style clients 'subsystem' is required for **sftp** transfers; 'exec' is required for **scp** transfers.

AllowTCPForwarding

Use this keyword to allow or deny port forwarding to all client users. The allowed values are 'yes' and 'no'. The default is 'yes'. This keyword controls both local (client to server) and remote (server to client forwarding). Use **ForwardAcl** for more fine-grained control.

AllowTCPForwardingForGroups

Use this keyword to allow port forwarding only for users who are members of a specified group. Regular expressions are supported.

AllowTCPForwardingForUsers

Use this keyword to allow port forwarding only for specified users. Regular expressions are supported.

AllowUsers

Use this keyword to allow login only for specified users. Regular expressions are supported. For details, see *Configuring User Access* (page [97](#)).

AllowX11Forwarding

Specifies whether X11 forwarding is allowed. The allowed values are 'yes' and 'no'. The default is 'yes'.

AuthFailureErrorMessages

When set to 'no', no information about authentication failures is sent to the client. This complies with SSH convention. When set to 'yes', the client receives information about the reason for the failure. Warning: This increases your security risk by providing this information to potential attackers. The allowed values are 'yes' and 'no'. The default is 'no'.

AuthImmediateDisconnect

The allowed values are 'yes' and 'no'. The default is 'no'. When this setting is 'no', the server responds identically to all failed authentication attempts. This complies with SSH convention. When this setting is 'yes', users with blocked accounts are disconnected as soon as possible, which means they might not see any authentication prompts. If a user is denied access because of Reflection for Secure IT server settings (for example **AllowUsers** or **DenyUsers**), the disconnection always happens immediately. If a user is denied access because of operating system configuration, the timing of the disconnection is affected by the **AccountManagement** setting. When `AccountManagement=pam`, denied users see PAM authentication prompts before being disconnected. This is because PAM authentication happens before PAM account management. If you prefer to have users be disconnected without seeing PAM authentication prompts, set `AccountManagement=pam,password` (the default). In most cases, enabling password account management provides the server with enough information about the user account to reject the connection before PAM authentication starts.

Caution: Enabling this setting increases your security risk by providing clients with information about valid account names.

AuthKbdInt.Required

Specifies which authentication method to use for keyboard-interactive authentication. The specified authentication method must succeed for the user to be successfully authenticated. The allowed values are 'pam', 'password', and 'radius'. The default is 'pam', which specifies that PAM modules are used for authentication and password management. When 'password' is specified, the user response is handled as a standard login password. When 'radius' is specified, one or more RADIUS authentication servers are used for authentication.

AuthKbdInt.Retries

Sets the maximum number of attempts allowed for keyboard interactive authentication. The default is 3.

AuthKbdInt.Verbose

Specifies whether the server uses verbose keyboard interactive prompts. The allowed values are 'yes' and 'no'. The default is 'no'.

AuthorizationFile

Specifies the name of the file used for configuring user keys for public key authentication. For public key authentication to succeed, a key presented by a client user for authentication must be correctly identified in this file. For file syntax, see the FILES section.

The file is assumed to be relative to `~/.ssh2` (or whatever location is set for **UserConfigDirectory**) unless you specify an absolute path. The following macros are recognized: %U = user log-in name, %D = user's home directory, %IU = UID for user, %IG = GID for user. The default file is `%D/.ssh2/authorization`.

AuthPublicKey.MaxSize

Sets the largest public key size allowed for user authentication. The default is 32768, and values larger than this are not allowed. The range of accepted values is 512-32769. Using zero (0) is equivalent to using the default.

AuthPublicKey.MinSize

Sets the smallest public key size allowed for user authentication. The default is 512, and values smaller than this are not allowed. Using zero (0) is equivalent to using the default.

AuthPublicKey.Retries

Specifies the maximum number of attempts the server accepts for public key authentication. Once this number is reached, further attempts to authenticate using a public key are rejected, but the connection is not broken. This allows the client to attempt authentication using the next allowed method. The default is 100.

BannerMessageFile

Identifies a file that contains text for a banner message. The server sends this text to the client before the client authenticates. Note: Some clients do not support banner display. If you configure a banner, you should ensure that your Secure Shell client supports this feature. The default is `/etc/ssh2/ssh_banner_message`.

ChrootSftpGroups

Specifies groups whose users are restricted to their home directory for sftp protocol connections. Any sftp protocol request that operates on a file or directory is checked to ensure it is not outside of the confined directory or any of its child directories. Regular expressions are supported. Patterns match against group names, not GID's.

Note: This setting affects both **sftp** and **scp** connections from Reflection for Secure IT clients. The **SessionRestricted** keyword also affects access to file transfers. The default value for **SessionRestricted** is 'shell, exec, subsystem'. For Reflection for Secure IT clients, the 'subsystem' session type is required for both **sftp** and **scp** transfers. For OpenSSH-style clients 'subsystem' is required for **sftp** transfers; 'exec' is required for **scp** transfers.

When **ChrootSftpUsers** or **ChrootSftpGroups** is enabled, connected users see additional subdirectories (`etc` on all platforms and `dev` on AIX) added to their home directory. These directories cannot be moved or deleted. The `etc` directory contains two required files. The `rsit.conf` file identifies the installation location of files required by Reflection for Secure IT. The `localtime` file is needed so that processes such as logging can get the current time. The system `localtime` file is in a location that cannot be accessed by a chrooted user. Users running on AIX also require `/dev/null`, which is needed for correct logging to syslog.

ChrootSftpUsers

Specifies users who are restricted to their home directory for sftp protocol connections. Any sftp protocol request that operates on a file or directory is checked to ensure it is not outside of the confined directory or any of its child directories. Regular expressions are supported. Patterns match against user names, not UID's.

Note: This setting affects both **sftp** and **scp** connections from Reflection for Secure IT clients. The **SessionRestricted** keyword also affects access to file transfers. The default value for **SessionRestricted** is 'shell, exec, subsystem'. For Reflection for Secure IT clients, the 'subsystem' session type is required for both **sftp** and **scp** transfers. For OpenSSH-style clients 'subsystem' is required for **sftp** transfers; 'exec' is required for **scp** transfers.

When **ChrootSftpUsers** or **ChrootSftpGroups** is enabled, connected users see additional subdirectories (`etc` on all platforms and `dev` on AIX) added to their home directory. These directories cannot be moved or deleted. The `etc` directory contains two required files. The `rsit.conf` file identifies the installation location of files required by Reflection for Secure IT. The `localtime` file is needed so that processes such as logging can get the current time. The system `localtime` file is in a location that cannot be accessed by a chrooted user. Users running on AIX also require `/dev/null`, which is needed for correct logging to syslog.

Ciphers

Specifies one or more (comma separated) encryption algorithms the server supports. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'.

You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.

The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers). The default is AnyStdCipher.

ClientAliveCountMax

The client alive mechanism enables the server to determine when the client has become inactive. **ClientAliveCountMax** sets the maximum number of client alive messages the server sends through the encrypted channel to request a response from the client. If this number is reached with no response from the client, the server ends the session and disconnects the client. Specify the message interval using **ClientAliveInterval**. The default is 3.

Note: These settings affect the SSH connection and messages are sent through the SSH tunnel.

ClientAliveInterval

Sets the interval, in seconds, for sending client alive messages to the client. If the client is unresponsive for this interval, the server sends a message through the encrypted channel to request a response from the client. Use **ClientAliveCountMax** to specify how many messages the server sends without response before it ends the session and disconnects the client. The default is 0 (disabled).

Compat.RSA.HashScheme

Specifies whether the MD5 hash algorithm is supported for verifying the digital signature for RSA keys used in public key or X.509 certificate authentication. The allowed values are 'yes' and 'no'. When this keyword is set to 'no' (the default), only signatures with SHA-1 hashes are accepted. When it is set to 'yes' signatures with either SHA-1 or MD5 hashes are accepted.

Compression

Specifies the level of compression. You can specify compression values 0-9. Increasing the value increases the amount of compression. Using higher values results in the use of less network bandwidth, but at the cost of more CPU cycles. Level 6 is equivalent to 'yes'. Level 0 is equivalent to 'no'. The default is 'yes' (6).

DenyGroups

Use this keyword to deny login for specified user groups. Regular expressions are supported. For details, see *Configuring Group Access* (page 98). If this keyword is not configured, all groups are allowed to log in.

DenyHosts

Use this keyword to deny login for specified client hosts. Regular expressions are supported. For details, see *Configuring Client Host Access* (page 98). If this keyword is not used, all client hosts are allowed.

Notes:

If you configure a host expression using the domain name (rather than IP address), you must also set **ResolveClientHostName** to 'yes'. You should also set **RequireReverseMapping** to 'yes' to prevent access from hosts whose domain name could not be resolved. When **ResolveClientHostName** is 'yes', the resolved name is the fully qualified domain name. This means that when **RequireReverseMapping** is 'yes', you must specify a fully qualified domain name or use a regular expression for the host name to ensure that connections from an IP address are handled correctly.

To configure addresses in any allow or deny list, both IPv4 and IPv6 addresses must be specified. This is particularly important if you are configuring a deny list to ensure that access is blocked. To configure localhost in any allow or deny list, include IP addresses for all external interfaces and also the local loopback address (127.0.0.1 and 0:0:0:0:0:0:0:1).

DenyTCPForwardingForGroups

Use this keyword to deny port forwarding for specified user groups. Regular expressions are supported. For details, see *Configuring Group Access* (page 98).

DenyTCPForwardingForUsers

Use this keyword to deny port forwarding for specified users. Regular expressions are supported. For details, see *Configuring User Access* (page [97](#)).

DenyUsers

Use this keyword to deny login for specified users. Regular expressions are supported. For details, see *Configuring User Access* (page [97](#)). If this keyword is not configured, all users are allowed to log in.

FipsMode

Specifies whether all connections will be made using security protocols and algorithms that meet FIPS 140-2 standards. The allowed values are 'yes' and 'no'. The default is 'no'.

ForwardACL

Use this keyword for detailed control over client access to port forwarding. Regular expressions are supported. The syntax is:

```
ForwardACL allow|deny local|remote user_ex forward_ex [origin_ex]
```

user_ex is a regular expression that determines which users are allowed or denied access to port forwarding. For details, see *Configuring User Access* (page [97](#))."

forward_ex is a regular expression in the form *host%port*. Its meaning depends on whether you are configuring restrictions on local or remote forwards. If you are configuring local forwarding control, it specifies the target host and port. If you are configuring remote forwarding control, the host is the server computer and the port is the port that server is forwarding to the client.

origin_ex is a regular expression that identifies an IP address. Its meaning depends on whether you are configuring restrictions on local or remote forwards. If you are configuring local forwarding control, it specifies the client machine making the forward request. If you are configuring remote forwarding control, it specifies the computer that is connecting to the forwarded port on the server.

GatewayPorts

Specifies whether remote hosts are allowed to connect to ports forwarded for the client. The allowed values are 'yes' and 'no'. The default is 'no'.

HostCertificateFile

Specifies an X.509 certificate to be used for server authentication. Specify the associated private key using **HostKeyFile**.

HostKeyFile

Specifies the filename and location of the private key used to authenticate the server. The default is `/etc/ssh2/hostkey`.

HostSpecificConfig

Specifies a host-specific subconfiguration file. The syntax is:

```
HostSpecificConfig host_expression subconfig_file
```

If the host expression matches the client host, the server uses the specified subconfiguration file.

If you configure a host expression using the domain name (rather than IP address), you must also set **ResolveClientHostName** to 'yes'.

HPNDisabled

Specifies whether Reflection for Secure IT uses HPN dynamic TCP window features to enhance performance. When **HPNDisabled** = 'no' (the default), Reflection for Secure IT adjusts the TCP window and TCP receive buffers to optimize performance. When **HPNDisabled** is 'yes', the receive buffer is set to 64 KB.

IdleTimeout

Specifies how long a connection can remain inactive before the server terminates the connection. To set the time in seconds use an `s` or nothing after the number. You can also specify a time in minutes (m), hours (h), days (d), or weeks (w). Use zero (0) to set no limit. The default is 0.

IgnoreRlogin

This keyword applies only to AIX systems. It specifies whether the 'rlogin' attribute in `/etc/security/user` should be ignored or applied. The allowed values are 'yes' and 'no'. The default is 'no', which means that the server honors the current 'rlogin' value. Note: The 'login' attribute in `/etc/security/user` has no effect on remote logins made using the Secure Shell client. This is true regardless of the value of **IgnoreRlogin**.

KeepAlive

Specifies whether the system should send TCP keep alive messages to the other side. The server uses the system-wide value for how often the message is sent. The allowed values are 'yes' and 'no'. The default is 'yes'. Note: **ClientAliveCountMax** and **ClientAliveInterval** affect the SSH connection and messages are sent through the SSH tunnel. The **KeepAlive** setting affects the TCP connection, and is more vulnerable to spoofing because TCP messages are not sent in the secure tunnel.

KEXs

Specifies which key exchange algorithms the server supports. Supported values are 'diffie-hellman-group1-sha1' and 'diffie-hellman-group14-sha1'. Multiple algorithms can be specified as a comma-separated list. The default value is 'diffie-hellman-group14-sha1,diffie-hellman-group1-sha1'.

LibGssKrb5

Use this setting if you use GSSAPI (Kerberos 5) authentication. It specifies the fully-qualified path to the Kerberos library called `libgssapi_krb5.so`.

LibKrb5

Use this setting if you use GSSAPI (Kerberos 5) authentication. It specifies the fully-qualified path to the Kerberos library called `libkrb5.so`.

Note: The server requires a library named `libkrb5.so` (or `.sl` on HP-UX PARISC). If a library of this name is not present, you need to create a symbolic link named `libkrb5.so` pointing to the actual library.

LibWrap

This keyword provides dynamic support for TCP Wrappers. To enable TCP Wrapper support, specify the fully qualified path to the `libwrap` shared library (for example, `LibWrap=/usr/lib/libwrap.so`). The `libwrap` file must be a shared library and not a static one. By default, this keyword is empty and the TCP Wrappers feature is disabled.

Note: Before using this keyword, you should confirm that the specified file is a valid `libwrap` library. This is important to ensure that only allowed users can connect. If the specified file doesn't exist, the **sshd** server won't start. However, if the file exists, **sshd** starts, but does not confirm that the file is a valid library. For each connection, the **sshd** process tries to load the specified file, and, if the file is not a valid library, the server logs an error message and allows the user to connect.

ListenAddress

Specifies the address of the interface to which the **sshd** server socket is bound. You can specify one or more comma-separated values using either IPv4 or IPv6 format, or use 'any' (the default). The value 'any' configures the server to listen to any available IPv4 or IPv6 address (equivalent to '['::'],0.0.0.0'). If you specify only IPv4 addresses, the client must connect using an IPv4 address. If you specify only IPv6 format, most operating systems will still allow IPv4 clients to connect; this is controlled by the operating system, not the Secure Shell server. You can optionally include a port in the address by adding a colon or space followed by the port number. This port value overrides the **Port** keyword setting. If you are specifying an IPv6 address, you need to surround the address with square brackets.

For example:

IPv4 syntax: `ListenAddress=209.85.171.99:6666`

IPv6 syntax: `ListenAddress=[::D155:AB63]:6666`

ListenAddress interacts with the **AddressFamily** setting. When **AddressFamily**=inet, the **ListenAddress** value 'any' is equivalent to '0.0.0.0'. When **AddressFamily**=inet6, the **ListenAddress** value 'any' is equivalent to '[:::]'. If **AddressFamily** is set to either 'inet' or 'inet6' and **ListenAddress** specifies an address of a different family, **sshd** will fail to start because of a configuration file error. If you specify a host name for **ListenAddress** rather than an IP address, the **AddressFamily** restrictions require that the host name be associated with an address of the appropriate family; and the server will bind to that address.

LogCertificateSubject

Specifies whether the Serial Number and Subject of certificates used for authentication are logged to the system log. Messages are logged for both successful and failed attempts. The allowed values are 'yes' and 'no'. The default is 'yes'.

LoginGraceTime

Sets the number of seconds allowed for client authentication. If the client fails to authenticate the user within the specified number of seconds, the server disconnects and exits. Use zero (0) to set no limit. The default is 120.

LogLevel

Sets the verbosity level used for **sshd** messages logged to syslog. Allowed values are 'fatal', 'error', 'quiet', 'info', 'verbose', 'debug1' ('debug' and 1 are equivalent), 'debug2' (2 is equivalent), 'debug3' (3 is equivalent), and 'trace' ('debug99' and 99 are equivalent). The syslog level associated with these values is CRIT for fatal, ERROR for error and quiet, INFO for info and verbose, and DEBUG for debug1, debug2, debug3, and trace. The default is 'error'.

Note: Setting logging to 'trace' can increase your security risk. At this level, information leakage is a concern, as unencrypted protocol information may be written out. Also, the volume of information written may fill up disk space rapidly, potentially causing the host or Reflection for Secure IT to stop responding.

LogPublicKeyFingerprint

Specifies whether public key fingerprints used for authentication are logged to the system log. Messages are logged for both successful and failed attempts. The allowed values are 'yes' and 'no'. The default is 'yes'.

MACs

Specifies which MACs (hashed message authentication codes) the server allows for verifying data integrity. Allowed values are 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', 'hmac-md5-96', 'hmac-ripemd160', 'hmac-sha256', and 'hmac-sha512'. Use 'AnyMac' to support all of these. Use 'AnyStdMac' to support 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', and 'hmac-md5-96'. Additional options are 'none', 'any' (equivalent to AnyMac plus 'none'), and 'AnyStd' (equivalent to 'AnyStdMac' plus 'none'). Multiple MACs can also be specified as a comma-separated list. When 'none' is the agreed on MAC, no message authentication code is used. Because this provides no data integrity protection, options that include 'none' are not recommended. The default is 'AnyStdMac'.

MaxConnections

Sets the maximum number of client connections allowed. Use zero (0) to set no limit. The default is 50.

MaxStartups

Specifies the maximum number of concurrent unauthenticated connection attempts allowed. After this limit is reached additional connections are dropped until authentication succeeds or the **LoginGraceTime** limit is reached for a connection attempt. The default is 10.

PamServiceName

Specifies the name of the PAM (Pluggable Authentication Modules) service used for authentication and sessions. The default is 'ssh'.

PamServiceNameForInternalProcesses

Specifies the name of an optional PAM service to be used for internal processes. You can use the specified service to provide additional account and session management. For example:

```
PamServiceNameForInternalProcesses ssh-shell
```

In this case, all users still go through the service specified by **PamServiceName** ("ssh" by default). Shell and exec users will also go through the "ssh-shell" service.

Note: The specified PAM service will always support PAM account and session management and may support authentication management on particular platforms (Linux and AIX, but not Solaris). Because authentication management may or may not be used depending on the platform, it should always be set to pam_permit.so so that access to the system can be configured using account and session management.

PamServiceNameForSubsystems

Specifies the name of an optional PAM service to be used for subsystems. You can use the specified service to provide additional account and session management. The syntax is:

```
PamServiceNameForSubsystems subsystem PAMservicename
```

For example, You could use the following to provide additional account and session management for SFTP connections:

```
PAMServiceNameforSubsystems sftp ssh-sftp
```

In this case, all users still go through the service specified by PamServiceName ("ssh" by default). SFTP users will also go through the "ssh-sftp" service.

Note: The specified PAM service will always support PAM account and session management and may support authentication management on particular platforms (Linux and AIX, but not Solaris). Because authentication management may or may not be used depending on the platform, it should always be set to pam_permit.so so that access to the system can be configured using account and session management.

PasswordGuesses

Sets the maximum number of attempts the user is allowed for password authentication. The default is 3.

PermitEmptyPasswords

Specifies whether the server allows password authentication by users with empty (null) passwords. The allowed values are 'yes' and 'no'. The default is 'yes'.

PermitRootLogin

Specifies whether client users with root privileges can log in. The allowed values are 'yes', 'no', and 'without-password'. If you specify 'without-password', a user can log in with root privileges only if 'public key' or 'GSSAPI' authentication methods are used to authenticate the user. The default is 'yes', which allows root login for all authentication methods.

PidFile

Specifies the file that contains the process ID of the **sshd** daemon. Use a fully qualified path. If the file name contains the string %s, the string will be replaced by the server port number.

PkidAddress

Specifies the port used to connect to PKI Services Manager. Use the format *host:port*. The default is localhost:18081. If you specify a host and omit the port, the default PKI Services Manager port (18081) is used.

PkidPublicKey

Specifies the name and location of the public key used by to confirm the identity of Reflection PKI Services Manager. The default is `/opt/attachmate/pkid/config/pki_key.pub`.

Port

Specifies the port on which the server listens. The default is 22, which is the standard port for Secure Shell connections.

PrintMotd

Specifies whether the server prints the message-of-the-day text from the file `/etc/motd` when a user logs into a terminal session. (This setting does not override the display of `/etc/issue`.) The allowed values are 'yes' and 'no'. The default is 'yes'.

ProtocolVersionString

Specifies the software version portion of the string that the server sends to clients during the initial connection protocol. (The first part of the string is always "SSH-2.0-", which indicates the SSH version supported by the server. This is required by the protocol RFC and cannot be edited.) Use double quotation marks if the string includes spaces. When **ProtocolVersionString** is an empty string (the default), the software version portion of the string is generated automatically, and includes the server's version and build number. This number will be updated automatically when you upgrade your server software.

Note: Many clients use the protocol string to identify the server type and enable compatible features. Changing the default value may cause public key authentication to fail, and may also affect the functionality of other features that vary between servers.

QuietMode

This keyword is deprecated. Use **LogLevel**.

RadiusFile

Specifies the name of the file used for configuring RADIUS authentication. The file is assumed to be relative to `/etc/ssh2` unless you specify an absolute path. For file syntax, see `/etc/ssh2/radius_config` in the FILES section. There is no default; this keyword can have no value.

RekeyIntervalSeconds

Specify the interval (in seconds) after which the server initiates a new key exchange. Setting this value too low can make communication between the client and server impossible. To avoid this problem, it is recommended that you avoid specifying an interval of less than 200 seconds. Use 0 (zero) to turn off rekey requests initiated by the server. Using 0 does not prevent the client from requesting a rekey. The default is 3600.

RequiredAuthentications

Use this keyword to require one or more client authentication methods. All specified authentication methods must succeed before a user is considered authenticated. The supported authentication methods are 'gssapi-keyex', 'gssapi-with-mic', 'publickey', 'keyboard-interactive', and 'password'.

Note: **RequiredAuthentications** overrides **AllowedAuthentications**.

RequireReverseMapping

Specifies whether DNS lookup must succeed when checking whether connections from client hosts are allowed. To enable this feature you also need to set **ResolveClientHostName** to 'yes'. The allowed values are 'yes' and 'no'. The default is 'no'.

ResolveClientHostname

Specifies whether the server attempts to resolve the client IP address to a domain name. Setting this to 'yes' may slow down the connection time, but is required if you configure any keywords to match host names based on domain name, rather than IP address. (See **AllowHosts**, **DenyHosts**, **UserSpecificConfig**, and **HostSpecificConfig**.) Setting this keyword to 'yes' also means that DNS names appear in the log rather than IP addresses. The allowed values are 'yes' and 'no'. The default is 'yes'.

Note: When **ResolveClientHostname** is 'yes', the resolved name is always the fully qualified domain name. This means that you must use a fully qualified domain name with any keywords in which you specify a host name, or use a regular expression to ensure that host names are handled correctly.

SessionRestricted

Specifies what session types the server allows. The possible values are 'shell' (which allows terminal shell sessions), 'exec' (which allows the client to execute commands on the server), and 'subsystem' (which is required to support **sftp** and **scp** transfers from Reflection for Secure IT clients). The default is 'shell, exec, subsystem'.

Note: For OpenSSH-style clients 'subsystem' is required for **sftp** transfers; 'exec' is required for **scp** transfers.

SettableEnvironmentVars

Specifies which environment variables can be configured by the client. This value limits the scope of the client **SetRemoteEnv** keyword on the client and the user-specific environment file (`~/.ssh2/environment`). (Note: This setting does not affect variables configured in `/etc/environment`, `/etc/ssh2/environment` or other server files which can be controlled only by root.) The arguments must be uppercase. This keyword is enabled in the default configuration file and set to the following value:
 'LANG,LC_ALL,LC_COLLATE,LC_CTYPE,LC_MONETARY,LC_NUMERIC,LC_TIME,PATH,TERM,TZ,UMASK'

SmartFileTransfer

Specifies whether the server performs checks for file equality before transferring data. When this keyword is 'yes' (the default), the server supports smart file copy (which enables skipping transfer of identical files) and checkpoint resume (which enables interrupted file transfers to resume at the point of interruption). When this keyword is 'no', Reflection for Secure IT always transfers the entire content of every file. Note: Smart file copy can be disabled on the client using **SmartFileCopy**. Checkpoint resume can be disabled on the client using **CheckpointResume**.

SftpLogCategory

Determines which categories of sftp server messages are sent to the facility specified by **SftpSysLogFacility**. Use a comma-separated list. The default is 'loginlogout,directorylistings,downloads,modifications,uploads', which configures logging of all categories. You can specify any of those options, plus 'all', or 'none'.

SftpSysLogFacility

Specifies the facility code used for logging messages from the sftp-server subsystem. This value is empty by default. When this value is empty and **LogLevel** is not empty, logging goes to the AUTH facility. When **SftpSysLogFacility** and **LogLevel** are both empty, the server does no logging to syslog. When this value is 'none', Reflection for Secure IT disables logging to syslog (regardless of the **LogLevel** setting). Other valid values are platform-dependent. See `syslog(3)`. Valid values are platform-dependent. See `syslog(3)`. Setting this to "auth" puts the log messages in the same facility as the default for **sshd**.

StrictModes

Specifies the directory permissions required for public key authentication. The allowed values are 'yes' and 'no'. The default is 'yes'. When set to 'yes', The user's directory (`~/ .ssh2`) and all parent directories must be writable and executable only by the user (mode `744` is accepted). Recommended permissions for the user directory = `700`. If these conditions aren't met, public key authentication fails. When set to 'no' these file permissions are not enforced and sensitive files and information could be compromised.

Note: Additional file permission requirements are enforced for each user's authorization file (`~/ .ssh2/authorization`) regardless of the current **StrictModes** setting. This file must be configured to prevent group and public write access (`600` is recommended, `644` is accepted). If the authorization file is not sufficiently restricted, public key authentication will always fail.

Subsystem

Specifies a subsystem to export to the client. The argument specifies the command to execute when the client requests the subsystem. The separator character following the keyword can be a dash, an equals sign, or a space.

To support **sftp** and **scp** transfers, the `sftp-server` subsystem must be specified. The default configuration shown below executes the `sftp` service internally in the child process.

```
Subsystem-sftp internal://sftp-server
```

SyslogFacility

Specifies the facility code used for logging messages from the server. The default is 'AUTH'. When this value is 'none', Reflection for Secure IT disables logging to syslog. Other valid values are platform-dependent. See `syslog(3)`.

Note: Setting this value to 'none' is not recommended because it means you have no audit log of connection attempts or user logins. In the event of a denial-of-service attack, an audit log can help identify a set of IP addresses connecting excessively. An audit log can also provide important evidence if a user falsely claims to not have accessed your system (non-repudiation).

Note: The debugging level you specify for writing to this log can have security ramifications. For more information see **LogLevel**.

TrustAnchor

This keyword is optional and is relevant only if you use certificates for user authentication. By default, Reflection PKI Services Manager validates certificates presented for authentication using all of the trust anchors you have configured. Use this keyword to limit which of the Reflection PKI Services Manager trust anchors can be used for certificate validation. You can specify either Subject DN (Distinguished Name) from a certificate available in the PKI Services Manager store, or use the file name of a certificate. Note that the specified trust anchors must be also be configured for Reflection PKI Services Manager (using the PKI Services Manager **TrustAnchor** keyword).

UseLogin

Specifies whether `login(1)` is used for interactive login sessions. The allowed values are 'yes' and 'no'. The default is 'no'.

Notes:

`login(1)` is never used for remote command execution.

Enabling this setting disables **X11Forwarding** because `login(1)` does not know how to handle `xauth(1)` cookies.

Using `login(1)` disables privilege separation. By default, **sshd** creates a new process that has the privilege of the authenticated user after a successful authentication. This is done to prevent privilege escalation by containing any corruption within the unprivileged processes. Enabling **UseLogin** disables this functionality.

UsePAM

This setting provides an alternate way to configure the server to use PAM. The allowed values are 'yes' and 'no'. If **UsePam** is not configured, the server uses the current values of **AuthKbdInt.Required**, **AccountManagement**, and **UsePamSessions**. Setting this keyword to 'yes' is equivalent to setting `AuthKbdInt.Required=pam`, `AccountManagement=pam`, and `UsePamSessions=yes`. Setting this keyword to 'no' is equivalent to setting `AuthKbdInt.Required=password`, `AccountManagement=password`, and `UsePamSessions=no`.

Note: If you modify **UsePAM**, be sure that none of the related keywords are set after **UsePAM** in the configuration file. If **AuthKbdInt.Required**, **AccountManagement**, or **UsePamSessions** is set to a conflicting value after **UsePAM**, that value overrides the value configured by **UsePAM** because the last value read by the server is the one it uses.

UsePAMAcctMgmt

This keyword is deprecated. Setting it to 'yes' is equivalent to setting `AccountManagement=pam`.

UsePamSessions

Specifies whether or not PAM is used for session management. The allowed values are 'yes' and 'no'. The default is 'yes'.

UserConfigDirectory

Specifies the directory used for user-specific information. This directory contains the authentication file (required for key authentication) and other user-specific files listed in the FILES section. The following macros are recognized: %U = user log-in name, %D = user's home directory, %IU = UID for user, %IG = GID for user. The default is '%D/.ssh2'.

UserSpecificConfig

Specifies a user-specific configuration file. The syntax is:

```
UserSpecificConfig user_expression subconfig_file
```

If the user expression matches the user attempting a connection, the server uses the specified subconfiguration file.

Note: If you configure the host portion of this expression to match based on host domain name (rather than IP address), you must also set **ResolveClientHostName** to 'yes'.

VerboseMode

This keyword is deprecated. Use **LogLevel**.

X11DisplayOffset

Sets the first display number available for X11 forwarding by the server. The default is 10.

X11UseLocalHost

Specifies whether the server should bind X11 forwarding to the loopback address or to the wildcard address. The allowed values are 'yes' and 'no'. The default is 'yes'.

XAuthPath

Specifies the location of the xauth(1) program. The default (for example /usr/X11R6/bin/xauth) is system-dependent.

APPENDIX E

File and Directory Permissions

To help ensure secure authentication, and prevent tampering, information leakage and spoofing, files and directories used by the client and server must be configured with correct permissions and ownership. If these conditions aren't met, Secure Shell connections and public key authentication may fail.

Notes:

- The **StrictModes** setting helps ensure enforcement of a satisfactory level of security and is enabled by default on both the server and the client.
 - Files must be owned by root or by the owner of the home directory in which the files reside.
 - Where permission requirements are enforced, the permissions must be at the level indicated in the table below, or more restrictive (less than or equal to the octal value shown in brackets).
 - Files and directories shown in parentheses are the defaults.
-

Client-side files and directories

File or Directory	Maximum Security	Required when StrictModes = no	Required when StrictModes = yes
Secure Shell directory (~/.ssh2/)	700	No requirements	User-only write access [755]
User home directory and All parent directories	744 755	No requirements	User-only write access [755]
User's private keys	600	User-only read/write access [600]	User-only read/write access [600]
User's public keys	600	No requirements	No requirements
User's identification file (~/.ssh2/identification)	600	No requirements	User-only write access [644]
User's host keys directory (~/.ssh2/hostkeys)	700	No requirements	No requirements
Host public key files	600	No requirements	No requirements
User's configuration file (~/.ssh2/ssh2_config)	600	No requirements	User-only write access [644]

File or Directory	Maximum Security	Required when StrictModes = no	Required when StrictModes = yes
Client PKI Services Manager public key (specified using PkidPublicKey)	600	No requirements	No requirements
Global configuration directory (<code>/etc/ssh2/</code>)	755	No requirements	No requirements
Global host keys directory (<code>/etc/ssh2/hostkeys</code>)	755	No requirements	No requirements
Global host public key files	644	No requirements	No requirements
Global user configuration file (<code>/etc/ssh2/ssh2_config</code>)	644	No requirements	No requirements

Server-side files and directories (user-specific)

File or Directory	Maximum Security	Required when StrictModes = no	Required when StrictModes = yes
Secure Shell directory (<code>~/.ssh2/</code>)	700	No requirements	User-only write access [755]
User home directory and all parent directories	744 755	No requirements	User-only write access [755]
User's authorization file on the server (<code>~/.ssh2/authorization</code>)	600	User-only write access [644]	User-only write access [644]
User's secure shell environment file on the server (<code>~/.ssh2/environment</code>)	600	No requirements	No requirements
User's login behavior file (<code>~/.hushlogin</code>)	600	No requirements	No requirements

Server-side files and directories (global)

File or Directory	Maximum Security	Required when StrictModes = no	Required when StrictModes = yes
Server configuration directory (/etc/ssh2)	644	No requirements	No requirements
Server private key file (/etc/ssh2/hostkey)	600	Root-only read/write access (600)	Root-only read/write access (600)
Server public key file (/etc/ssh2/hostkey.pub)	600	No requirements	No requirements
Server RADIUS authentication configuration file (/etc/ssh2/radius_config)	600	No requirements	No requirements
Subconfiguration file directory (/etc/ssh2/subconfig)	700	No requirements	No requirements
Subconfiguration files	600	No requirements	No requirements
Global Secure Shell environment file (/etc/ssh2/environment)	600	No requirements	No requirements
Client PKI Services Manager public key (specified using PkidPublicKey)	600	No requirements	No requirements

APPENDIX F

ssh Command Line Options

The syntax for **ssh** is:

```
ssh [-4] [-6] [-a] [-c cipher] [-C] [-d debug_level] [-e character]  
[-f] [-fo] [-F file] [-g] [-h] [-i file] [-l username]  
[-L [[protocol/]listening_port:host:hostport] [-m mac_algorithm] [-n]  
[-o option] [-p port] [-q] [-R [[protocol/]listening_port:host:hostport]  
[-s subsystem] [-S] [-t] [-v] [-V] [-W] [-x] [-X] [-Y]  
[[username@]host[#port]] [remote_command [arguments] ...]
```

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

Caution: All options specified on the command line (including user names, host names, and other sensitive information) will show up in a process status (**ps**) listing. Exercise care when specifying sensitive options and switches so that other users cannot easily see that information. A more secure alternative is to set these options in a configuration file and to protect the configuration file using recommended file permissions (configuration file = 600, directory containing the file = 700).

-4

Forces connections using IPv4 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-6

Forces connections using IPv6 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-a

Disables authentication agent forwarding. Authentication agent forwarding is enabled using the **ForwardAgent** keyword, which is set to 'yes' by default. You can use **-a** to override the configuration file setting.

-c *cipher*

Specifies one or more (comma-separated) encryption algorithms supported by the client. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'.

You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.

The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers).

You can also configure encryption algorithms in the configuration file using the **Ciphers** keyword; the default is 'AnyStdCipher'.

-C

Disables compression. Compression is desirable on modem lines and other slow connections, but will slow down response rates on fast networks. Compression also adds extra randomness to the packet, making it harder for a malicious person to decrypt the packet. Compression can be enabled using the **Compression** keyword in the configuration file. Using **-C** overrides the configuration file setting.

-d *debug_level*

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

Note: Setting logging to 99 can increase your security risk. At this level, information leakage is a concern, as unencrypted protocol information may be written out. Also, the volume of information written may fill up disk space rapidly, potentially causing the host or Reflection for Secure IT to stop responding.

-e *character*

Sets the escape character for the terminal session. The default character is a tilde (~). Setting the escape character to 'none' means that no escape character is available and the tilde acts like any other character. For details, see ESCAPE SEQUENCES below. You can also set the escape character in the configuration file using the **EscapeChar** keyword.

-f

Use this option when you have configured port forwarding and you want the Secure Shell session to run in the background. If at least one port forwarding rule is configured, this option sends the Secure Shell session to the background after authentication is complete. The session remains in the background and continues to accept forward requests indefinitely until you manually kill the process. (This is equivalent to setting `GoBackground=yes` in the configuration file.)

-fo

The options works like **-f**, but in this case the background session waits for only one forwarded connection to occur and exits as soon as the forwarded connection is closed. (This is equivalent to setting `GoBackground=oneshot` in the configuration file.)

-F *file*

Specifies an additional configuration file. Settings are read from this file in addition to the default user-specific file (`~/.ssh2/ssh2_config` and/or the system-wide file (`/etc/ssh2/ssh2_config`). Settings in this file override settings in both the user-specific file and the system-wide file.

-g

Enables gateway ports. Remote hosts are allowed to connect to local forwarded ports. You can also configure this in the configuration file using the **GatewayPorts** keyword.

Caution: This option should be used with extreme caution (and never with Internet-facing network adapters), because the client performs no authentication of remote host connections. If the application to which this connection is forwarded does not perform its own authentication, then all remote hosts connections are allowed unrestricted access to that application.

-h

Displays a brief summary of command options.

-i *file*

Specifies an alternate identification file to use for public key authentication. The file location is assumed to be in the current working directory unless you specify a fully-qualified or relative path. The default identity file is `~/.ssh2/identification`. You can also specify an identity file in the configuration file using the **IdentificationFile** keyword.

-l *username*

Specifies a name to use for login on the remote computer. You can also specify a user name in the configuration file using the **Username** keyword. (Note: If you include the optional `[user@]` as part of your host specification, **-l** overrides the specified user name.)

-L [*protocol/*][*listening_host:*]*listening_port:host:hostport*

Redirects data from the specified local port, through the secure tunnel to the specified destination host and port. When a Secure Shell connection is established, the Secure Shell client opens a socket on the Secure Shell client host using the designated local port (*listening_port*). (On client hosts with multiple interfaces, use *listening_host* to specify which interface.) Configure your application client (the one whose data you want to forward) to send data to the forwarded socket (rather than directly to the destination host and port). When that client establishes a connection, all data sent to the forwarded port is redirected through the secure tunnel to the Secure Shell server, which decrypts it and then directs it to the destination socket (*host,hostport*). Unless the gateway ports option is enabled, the forwarded local port is available only to clients running on the same computer as the Secure Shell client. The optional *protocol* can be **tcp** or **ftp**. Multiple client applications can use the forwarded port, but the forward is active only while **ssh** is running.

Note: If the final destination host and port are not on the Secure Shell server host, data is sent in the clear between the Secure Shell host and the application server host.

You can also configure local forwarding in the configuration file using the **LocalForward** keyword.

-m *mac_algorithm*

Specifies which MACs (message authentication codes) are supported by the client. Allowed values are 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', 'hmac-md5-96', 'hmac-ripemd160', 'hmac-sha256', and 'hmac-sha512'. Use 'AnyMac' to support all of these. Use 'AnyStdMac' to support 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', and 'hmac-md5-96'. Additional options are 'none', 'any' (equivalent to AnyMac plus 'none'), and 'AnyStd' (equivalent to 'AnyStdMac' plus 'none'). Multiple MACs can also be specified as a comma-separated list. When 'none' is the agreed on MAC, no message authentication code is used. Because this provides no data integrity protection, options that include 'none' are not recommended.

You can also configure MACs in the configuration file using the **MACs** keyword; the default is 'AnyStdMac'.

-n

Redirects stdin from `/dev/null`, which prevents reading from stdin. You can also configure this in the configuration file using the **DontReadStdin** keyword.

-o *option*

Sets any option that can be configured using a configuration file keyword. For a list of keywords and their meanings, see `ssh2_config(5)`. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
-o key=value1,value2
-o key="value1, value2"
```

To configure multiple options, use multiple **-o** switches.

```
-o key1=value -o key2=value
```

-p *port*

Specifies the port to connect to on the server. The default is 22, which is the standard port for Secure Shell connections. You can also configure the port in the configuration file using the **Port** keyword.

-q

Enables quiet mode, which causes all warning and diagnostic messages, including banners, to be suppressed. You can also configure this in the configuration file using the **QuietMode** keyword.

-R [*protocol/*][*listening_host:*]*listening_port:host:hostport*

Redirects data from the specified remote port (on the computer running the Secure Shell server), through the secure tunnel to the specified destination host and port. When the Secure Shell connection is established, the Secure Shell server opens a socket on its host (the computer running the Secure Shell server) using the designated remote port (*listening_port*). (On server hosts with multiple interfaces, use *listening_host* to specify which interface.) Configure your client application (the one whose data you want to forward) to send data to the forwarded socket (rather than directly to the destination host and port). When that client establishes a connection, all data sent to the forwarded port is redirected through the secure tunnel to the Secure Shell client, which decrypts it and then directs it to the destination socket (*host,hostport*). The optional *protocol* can be **tcp** or **ftp**.

You can also configure remote forwarding in the configuration file using the **RemoteForward** keyword.

-s *subsystem*

Invokes the specified subsystem on the remote system. Subsystems are a feature of the Secure Shell protocol which facilitates the use of Secure Shell as a secure transport for other applications (such as **sftp**). Subsystems must be defined by the Secure Shell server.

-S

Connects without requesting a session channel on the server. This can be used with port-forwarding requests if a session channel (and tty) is not needed, or the server does not give one.

-t

Forces a tty allocation even if a command is specified. You can also configure this in the configuration file using the **ForcePTYAllocation** keyword.

-v

Sets the debug level to verbose mode, which is equivalent to using `'-d 2'`. You can also configure this in the configuration file using the **VerboseMode** keyword.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-W *password_file*

Specifies a file containing the password to use for the connection. Set permissions on the password file to 600; the file is not accepted if it has read or write permissions for group or other. Also, for a non-root user, the file is not accepted if there has been a change in identity (userid). This option applies only to password authentication. If **AllowedAuthentications** is configured to attempt keyboard-interactive before password authentication (the default), users will receive a password prompt even if a valid password file is present. To prevent this, modify the allowed authentications list to support only password authentication or to attempt password authentication before keyboard-interactive.

Note: Passphraseless public keys provide a more secure way to configure authentication without requiring user interaction, because private keys are not transmitted over the encrypted connection like passwords are.

-X

Enables X11 connection forwarding and treats X11 clients as untrusted. Untrusted remote X11 clients are prevented from tampering with data belonging to trusted X11 clients. You can also configure this in the configuration file using the **ForwardX11** keyword.

-x

Disables X11 connection forwarding. You can also configure this in the configuration file using the **ForwardX11** keyword.

-Y

Enables X11 connection forwarding and treats X11 clients as trusted.

APPENDIX G

ssh Escape Sequences

Use escape sequences to manage your client terminal session. Escape sequences are recognized only after a newline character. If you have just logged in, press Enter before you enter your first escape sequence. You can configure an alternate escape character using `-e` on the command line or **EscapeChar** in the configuration file.

The following escape sequences are available. These are shown with the default escape character, a tilde (~).

- ~. Terminates the connection.
- ~^Z Suspends ssh.
- ~# Lists active forwarded connections. Note: Forwarded connections are listed only when the ports are actually transmitting data.
- ~- Disables use of the escape character for the duration of the session.
- ~? Displays a list of available escape sequences.
- ~~ Sends the escape character to the host. (Type the escape character twice to send one escape character.)
- ~C Execute command mode, which you can use to request port forwarding. The options are:

<code>-L[bind_address:]port:host:hostport</code>	Request local forward
<code>-R[bind_address:]port:host:hostport</code>	Request remote forward
<code>-KL[bind_address:]port</code>	Cancel local forward
<code>-KR[bind_address:]port</code>	Cancel remote forward
- ~V Sends version information to `stderr`.
- ~S Sends connection information to `stderr`.
- ~r Initiates an immediate key exchange to establish new encryption and integrity keys.
- ~l Enters line mode. Keystrokes are stored to a buffer and output when you press Enter.
- ~B Sends a BREAK to the remote system.

APPENDIX H

ssh Exit Values

Exit values are provided to assist in troubleshooting. In scripts we recommend that you use only zero or non-zero for error handling. Looking for specific non-zero values is not reliable because of variability caused by operating systems and servers.

Error codes are limited to values 0-255. Errors 65-79 are disconnect conditions, calculated as 64 + error value returned from host, as defined in RFC4253. Errors values 128-254 are system Signals, calculated as 128 + Signal value. Error value 255 is returned when **ssh** fails after being executed by another process, such as **scp**.

- 0 Success.
- 1 Generic error.
- 2 Remote host connection failure.
- 65 Access denied by host for this client address.
- 66 Protocol error.
- 67 Key exchange failed.
- 68 Authentication failed.
- 69 MAC error.
- 70 Compression error.
- 71 Service not available.
- 72 Protocol version not supported.
- 73 Host key not verifiable.
- 74 Connection lost.
- 75 Disconnected by application.
- 76 Too many connections.
- 77 Cancelled by user.
- 78 No more authentication methods available.
- 79 Unknown user name.

APPENDIX I

ssh-keygen Command Line Options

The **ssh-keygen** syntax is:

```
ssh-keygen [-? file] [-b bits] [-c comment] [-D private_key] [-e private_key]  
[-F key] [-h] [-H key] [-i key] [-k file] [-N new_passphrase] [-o key_name]  
[-O key_file] [-p passphrase] [-P] [-q] [-t key_type] [-V] [-X cert] [key_name1  
key_name2 ...]
```

Use **ssh-keygen** to create RSA and DSA keys for public key authentication, to edit the properties of existing keys, and to convert key file formats for compatibility with other Secure Shell implementations.

When no options are specified, **ssh-keygen** generates a 2048-bit RSA key pair and queries you for a passphrase to protect the private key. If you don't specify a file name on the command line, keys are created in `~/.ssh2/` and given a default name that identifies the key type, size, and host name (for example `/home/joe/.ssh2/id_rsa_2048_myhost_a`). If you specify a file name, keys are saved to the current working directory unless you include a fully qualified path name. For each private key you create, **ssh-keygen** also generates a public key. Public keys are given the same base name as the private key, with an added `.pub` extension (for example `id_rsa_2048_myhost_a.pub`).

Command Line Options

Options are available in both a single-character form (such as **-b**) and a descriptive equivalent (**--bits**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

-? *file*

Extracts certificate(s) and CRL(s) from the specified PKCS#? file.

-b *bits*

Specifies the key size. Up to a point, a larger key size improves security. Increasing key size slows down the initial connection, but has no effect on the speed of encryption or decryption of the data stream after a successful connection has been made. The length of key you should use depends on many factors, including: the key type, the lifetime of the key, the value of the data being protected, the resources available to a potential attacker, and the size of the symmetric key you use in conjunction with this asymmetric key. To ensure the best choice for your needs, we recommend that you contact your security officer. The default for RSA keys is 2048 bits and 1024 bits for DSA keys. The minimum allowed value is 512. The maximum allowed value is 32768.

-c *comment*

Specifies information for the comment field within the key file. Use quotation marks if the string includes spaces. If you do not specify a comment, a default comment is created that includes the key type, creator, date, and time. Note: The comment is displayed when a passphrase-protected key is used for client authentication. Don't store passphrases or other sensitive information in the comment.

-D *private_key*

Uses the specified private key to derive a new copy of the public key.

-e *private_key*

Changes the passphrase of the specified private key. When you use this option alone you will be queried for the old and new passphrase for the specified private key. To edit the passphrase without opening an interactive session, you can use this option in combination with **-p** and **-N**. To change to a null passphrase, you can use this option in combination with **-P**.

-F *key*

Displays the fingerprint of the specified key in Bubble Babble format.

-h

Displays a brief summary of command options.

-H *key*

Uses the specified Reflection public key to generate a public key in OpenSSH format. The converted key is created using the same base file name with an added `.ssh` extension. You can use the key that is created to configure public key client authentication on an OpenSSH server.

-i *key*

Displays information about the specified key.

-k *file*

Extracts certificate(s) and private key(s) from the specified PKCS #12 file.

-N *new_passphrase*

Changes the passphrase to the specified new passphrase. Use this option in combination with **-e**.

-o *key_name*

Specifies the filename for the generated private key. (A public key is also created and is always given the same name as the private key plus a `.pub` file extension.) Note: An alternate way of naming key files is to specify one or more key filenames at the end of the **ssh-keygen** command.

-O *key*

Uses the specified OpenSSH public or private key to create a public or private key in Reflection format. The converted key is created using the same base file name with an added `.ssh2` extension.

-p *passphrase*

Specifies a passphrase. Use quotation marks if the phrase includes spaces. This option creates the initial passphrase when you generate a new key. If you are managing an existing key, use this option to specify the passphrase that protects that key. If a passphrase is required and you don't use **-p**, you'll be prompted for the passphrase. Ensure that you follow your company's security policy for password length and complexity.

-P

Creates a key with no passphrase. You can use this option to create keys for server authentication. Passphrases are strongly recommended for client keys. Passphraseless keys should be used only for accounts that require unattended authentication (such as file transfer scripts). Passphraseless private key files should be protected using operating system file access controls (key file = 400, directory containing the key = 700).

-q

Hides the key generation progress indicator.

-t *key_type*

Specifies the algorithm used for key generation. Possible values are `"rsa"` and `"dsa"`. The default is `"rsa"`.

-V

Displays **ssh-keygen** version information.

-X *cert*

Extracts the public key from the specified X.509 certificate file.

[*key_name1 key_name2...*]

Specifies the file name (or names) to be used for the generated private key (or keys). The public key is created using the same name with a ".pub" file extension.

APPENDIX J

scp Command Line Options

The **scp** syntax is:

```
scp [-4] [-6] [-a [arg]] [-b buffer_size] [-B] [-c cipher] [-d] [-D  
debug_level] [-F file] [-h] [-i file] [-N max_requests] [-o option] [--  
overwrite] [-p] [-P port] [-q] [-Q] [-Q] [-r] [-u] [-v] [-V] [-w]  
[[user@]host[#port]:]file_or_dir ... [[user@]host[#port]:]file_or_dir
```

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

Caution: All options specified on the command line (including user names, host names, and other sensitive information) will show up in a process status (**ps**) listing. Exercise care when specifying sensitive options and switches so that other users cannot easily see that information. A more secure alternative is to set these options in a configuration file and to protect the configuration file using recommended file permissions (configuration file = 600, directory containing the file = 700).

-4

Forces connections using IPv4 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-6

Forces connections using IPv6 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-a [*newline_type*]

Transfers files in ASCII mode. Use the optional argument to handle newline conversion. You can specify either 'unix' or 'dos'. By default, the value you specify for *newline_type* sets the destination newline convention, but you can specify either source or destination conventions by prefixing the argument with 'src:' or 'dest:'. For example:

```
scp -a src:unix -a dest:dos unixhost:src_file winhost:dest_file
```

Defaults are: 'dest:unix', 'src:unix'. If destination and source types are the same, no conversion occurs. Otherwise a conversion occurs based on values you specify for the 'src' and 'dest' newline types.

When **-a** is used without specified source or destination conventions, the client attempts to retrieve the end-of-line convention for source and/or destination from the server(s) to which connections have been established. If the server does not support this functionality, the DOS end-of-line convention is assumed.

-b *buffer_size*

Specifies the buffer size used for data transfer. The default is 32768 bytes. The minimum allowed value is 1024. The maximum allowed value is 4194304 bytes. In most cases the default value provides close to optimal transfer speeds. On some systems, moderate increases to the buffer size can improve performance. Caution: Using very large buffer sizes rarely improves performance and can create problems including: slower transfers, transfer failures with servers that don't support very large buffers, and fatal errors when client or server memory limits are exceeded.

-B

Runs **scp** in batch mode, which disables all queries for user input. This is useful for scripts and batch jobs. Authentication methods that require user interaction are not supported when you use this option. In batch mode **scp** always overwrites existing destination files unless **--overwrite** is set to 'no'.

-c *cipher*

Specifies one or more (comma-separated) encryption algorithms supported by the client. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'.

You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.

The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers).

If no cipher is specified, the cipher is determined by the **Ciphers** keyword in the Secure Shell configuration file `ssh2_config(5)`; the default is 'AnyStdCipher'.

-d

Forces the destination to be a directory that already exists. For example, the following command copies `source_file` to the directory called `destination` if this directory exists. If the directory doesn't exist, `source_file` is copied to the `demo` directory and given the file name `destination`.

```
scp source_file joe@myhost:~/demo/destination
```

With the **-d** flag added, the following command copies `source_file` to the `destination` directory, but fails if this directory doesn't exist.

```
scp -d source_file joe@myhost:~/demo/destination
```

-D *debug_level*

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

-F *file*

Specifies an additional configuration file. Settings are read from this file in addition to the default user-specific file (`~/.ssh2/ssh2_config` and/or the system-wide file (`/etc/ssh2/ssh2_config`). Settings in this file override settings in both the user-specific file and the system-wide file.

-h

Displays a brief summary of command options.

-i *file*

Specifies an alternate identification file to use for public key authentication. The file location is assumed to be in the current working directory unless you specify a fully-qualified or relative path. The default identity file is `~/.ssh2/identification`.

-N *max_requests*

Specifies the maximum number of concurrent requests. Increasing this may slightly improve file transfer speed, but also increases memory use. The default is 256.

-o *option*

Sets any option that can be configured using a configuration file keyword. For a list of keywords and their meanings, see `ssh2_config(5)`. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
-o key=value1,value2
-o key="value1, value2"
```

To configure multiple options, use multiple **-o** switches.

```
-o key1=value -o key2=value
```

--overwrite [yes|no|ask]

Specifies whether or not to overwrite existing destination files. The allowed values are 'yes', 'no', and 'ask'. The default is 'yes'. Note: When the source and destination files are identical, no transfer occurs regardless of the value of this setting.

-p

Preserves the modification times and file attributes of the original file.

-P *port*

Specifies the port to connect to on the server. The default is 22, which is the standard port for Secure Shell connections. You can also configure the port in the configuration file using the **Port** keyword.

-q

Runs in quiet mode. Only fatal errors are displayed.

-Q

Disables display of the progress indicator.

-r

Copies recursively, including all subdirectories.

-u

Deletes the source file after the copy to the destination location is completed.

-v

Sets the debug level to verbose mode, which is equivalent to setting the debug level to 2. You can also configure this in the configuration file using the **VerboseMode** keyword.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-W *password_file*

Specifies a file containing the password to use for the connection. Set permissions on the password file to 600; the file is not accepted if it has read or write permissions for group or other. Also, for a non-root user, the file is not accepted if there has been a change in identity (userid). This option applies only to password authentication. If **AllowedAuthentications** is configured to attempt keyboard-interactive before password authentication (the default), users will receive a password prompt even if a valid password file is present. To prevent this, modify the allowed authentications list to support only password authentication or to attempt password authentication before keyboard-interactive.

Note: Passphraseless public keys provide a more secure way to configure authentication without requiring user interaction, because private keys are not transmitted over the encrypted connection like passwords are.

APPENDIX K

sftp Command Line Options

The **sftp** syntax is:

```
sftp [-4] [-6] [-b buffer_size] [-B batch_file] [-c cipher] [-D debug_level] [-h] [-m mac_algorithm] [-N max_requests] [-o option] [-P port] [-v] [-V] [-W] [[user@]host [#port]]
```

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

Caution: All options specified on the command line (including user names, host names, and other sensitive information) will show up in a process status (**ps**) listing. Exercise care when specifying sensitive options and switches so that other users cannot easily see that information. A more secure alternative is to set these options in a configuration file and to protect the configuration file using recommended file permissions (configuration file = 600, directory containing the file = 700).

-4

Forces connections using IPv4 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-6

Forces connections using IPv6 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-b *buffer_size*

Specifies the buffer size used for data transfer. The default is 32768 bytes. The minimum allowed value is 1024. The maximum allowed value is 4194304 bytes. In most cases the default value provides close to optimal transfer speeds. On some systems, moderate increases to the buffer size can improve performance. Caution: Using very large buffer sizes rarely improves performance and can create problems including: slower transfers, transfer failures with servers that don't support very large buffers, and fatal errors when client or server memory limits are exceeded.

-B *batch_file*

Specifies a file to use for batch processing **sftp** commands. After a successful login, **sftp** executes each command in the specified file until a **bye**, **exit** or **quit** command is found, and then terminates the connection. Authentication methods that require user interaction are not supported in this mode. The batch file can use any of the interactive commands documented below. If a command in the batch file fails, **sftp** continues executing the remaining commands, and returns the error code of the first failed command. However, commands prefixed with "-" (dash) always return 0, even if the command fails.

-c *cipher*

Specifies one or more (comma-separated) encryption algorithms supported by the client. The cipher used for a given session is the cipher highest in the client's order of preference that is also supported by the server. Allowed values are 'aes128-ctr', 'aes128-cbc', 'aes192-ctr', 'aes192-cbc', 'aes256-ctr', 'aes256-cbc', 'blowfish-cbc', 'arcfour', 'arcfour128', 'arcfour256', 'cast128-cbc', and '3des-cbc'.

You can also set this value to 'none'. When 'none' is the agreed on cipher, data is not encrypted. Note that this method provides no confidentiality protection, and is not recommended.

The following values are provided for convenience: 'aes' (all supported aes ciphers), 'blowfish' (equivalent to 'blowfish-cbc'), 'cast' (equivalent to 'cast128-cbc'), '3des' (equivalent to '3des-cbc'), 'Any' or 'AnyStd' (all available ciphers plus 'none'), and 'AnyCipher' or 'AnyStdCipher' (all available ciphers). If no cipher is specified, the cipher is determined by the **Ciphers** keyword in the Secure Shell configuration file `ssh2_config(5)`; the default is 'AnyStdCipher'.

-D *debug_level*

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

-h

Displays a brief summary of command options.

-m *mac_algorithm*

Specifies which MACs (message authentication codes) are supported for this connection. Allowed values are 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', 'hmac-md5-96', 'hmac-ripemd160', 'hmac-sha256', and 'hmac-sha512'. Use 'AnyMac' to support all of these. Use 'AnyStdMac' to support 'hmac-sha1', 'hmac-sha1-96', 'hmac-md5', and 'hmac-md5-96'. Additional options are 'none', 'any' (equivalent to AnyMac plus 'none'), and 'AnyStd' (equivalent to 'AnyStdMac' plus 'none'). Multiple MACs can also be specified as a comma-separated list. When 'none' is the agreed on MAC, no message authentication code is used. Because this provides no data integrity protection, options that include 'none' are not recommended.

-N *max_requests*

Specifies the maximum number of concurrent requests. Increasing this may slightly improve file transfer speed, but also increases memory use. The default is 256.

-o *option*

Sets any option that can be configured using a configuration file keyword. For a list of keywords and their meanings, see `ssh2_config(5)`. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
-o key=value1,value2
-o key="value1, value2"
```

To configure multiple options, use multiple **-o** switches.

```
-o key1=value -o key2=value
```

--overwrite [yes|no|ask]

Specifies whether or not to overwrite existing destination files. The allowed values are 'yes', 'no', and 'ask'. The default is 'yes'. Note: When the source and destination files are identical, no transfer occurs regardless of the value of this setting.

-P *port*

Specifies the port to connect to on the server. The default is 22, which is the standard port for Secure Shell connections. You can also configure the port in the configuration file using the **Port** keyword.

-v

Sets the debug level to verbose mode, which is equivalent to setting the debug level to 2. You can also configure this in the configuration file using the **VerboseMode** keyword.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-W *password_file*

Specifies a file containing the password to use for the connection. Set permissions on the password file to 600; the file is not accepted if it has read or write permissions for group or other. Also, for a non-root user, the file is not accepted if there has been a change in identity (userid). This option applies only to password authentication. If **AllowedAuthentications** is configured to attempt keyboard-interactive before password authentication (the default), users will receive a password prompt even if a valid password file is present. To prevent this, modify the allowed authentications list to support only password authentication or to attempt password authentication before keyboard-interactive.

Note: Passphraseless public keys provide a more secure way to configure authentication without requiring user interaction, because private keys are not transmitted over the encrypted connection like passwords are.

APPENDIX L

Supported sftp Commands

You can use the following commands in interactive **sftp** sessions and in **sftp** batch files.

ascii [-s] [*remote_newline*] [*local_newline*]

Sets the current file transfer mode to ASCII. ASCII mode is useful for translating end-of-line characters. Use *remote_newline* and *local_newline* if you need to override the default handling of new lines. Supported values for *remote_newline* are 'DOS' (\r\n) and 'Unix' (\n). If no explicit value for the remote end-of-line convention is given, the remote host is queried to provide the convention. If the remote host does not support this functionality, the DOS end-of-line convention is assumed. The only supported value for *local_newline* is 'Unix' (\n). Use **-s** to display the current transfer mode.

auto

Sets the transfer mode to 'auto'. In auto mode, the transfer method is determined by file extension. Files with specified file extensions use ASCII transfer; all other files use binary transfer. The default list of ASCII file types is "txt, htm*, pl, php*". To modify this list for a given **sftp** session, use the **setext** command. To change the default file extension list, use the client keyword **FileCopyAsciiExtensions**.

binary

Sets the transfer mode to binary. In this mode, files transfer without any modification. Binary is the default transfer mode. This command is useful for turning off ASCII mode within a batch script.

bye

This is a synonym for **quit**.

cd *directory*

Changes the remote directory to *directory*.

chgrp *group file*

Sets group ownership of files or directories specified by *file* to *group*. The group must be specified as a numeric group id (GID).

chmod[-R]*mode file*

Sets file permissions for the files or directories specified by *file*. The mode must be specified in numeric format (for example, 664). Use **-R** to change files and directories recursively.

chown *owner file*

Sets the owner of the files or directories specified by *file* to *owner*. The owner must be specified as a numeric user id (UID).

close

Closes the connection to the remote server without exiting **sftp**.

debug *debug_level* | **disable** | **no**

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.) Use either 'disable' or 'no' to disable debugging.

dir

This is a synonym for **ls**.

exit

This is a synonym for **quit**.

get [**--preserve**] | [**-p**] *remote-file* [*remote-file ...*]

Copies the specified file or files to the current local working directory. (To copy to a different location, use **lcd** to change the local working directory.) If a file with the same name already exists, the existing file is overwritten. Wildcards are supported, but name substitution occurs on file names only, not directories. Use either **--preserve** or **-p** to preserve file attributes and time stamps.

gettext

Displays the current list of file extensions that use ASCII file transfer when auto mode is enabled. Use **auto** to enable auto mode. Use **setext** to change this list for the current session. Use the client keyword **FileCopyAsciiExtensions** to change the default list.

help | ? [*command*]

Displays **sftp** help. Use *command* to display help on the specified command.

lcd *directory*

Sets the local directory to *directory*.

lls [-1 | -a | -f | -l | -n | -r | -S | -t | --] [*file*]

Displays the local directory listing. The options are:

- 1 (one column)
- a (show hidden files)
- f (do not sort)
- l (long list format)
- n (long list format with numeric user and group ids)
- r (reverse order)
- S (sort by size)
- t (sort by file access time)
- (treat hyphens that follow as ordinary characters.)

lmkdir *directory*

Creates the specified local directory.

ln

This is a synonym for **symlink**.

lpwd

Displays the local working directory.

ls [-1 | -a | -f | -l | -n | -r | -S | -t | --] [*file*]

Displays the remote directory listing. The options are the same as for **lls**, and are described above.

mget

This is a synonym for **get**.

mkdir *directory*

Creates the specified remote directory.

mput

This is a synonym for **put**.

open [-l | [*user@*]*host*]

Opens a connection to the specified host. Use -l to connect to the local host; in which case both local and remote commands act on files on the local file system.

put [--preserve] | [-p] *local_file* [*local_file* ...]

Copies the specified file or files to the current remote working directory. (To copy to a different location, use **cd** to change the remote working directory.) If a file with the same name already exists, the existing file is overwritten. Wildcards are supported, but name substitution occurs on file names only, not directories. Use either **--preserve** or **-p** to preserve file attributes and time stamps.

pwd

Displays the remote working directory.

quit

Exits **sftp** and closes the connection.

rename *source destination*

Renames a file from *source* to *destination*. No rename occurs if the destination file already exists.

rm *file*

Deletes the specified remote file or files. Wildcards are supported.

rmdir *directory*

Deletes the specified remote directory.

setext

Specifies the current list of file extensions that use ASCII file transfer when auto mode is enabled. To specify multiple extensions, use a comma or space-separated list; this command is not cumulative. Wildcard (zsh-glob) characters are supported. Don't precede file extensions with a period. To specify extensions containing spaces, use quotation marks around the extension or use a backslash as an escape character. Use **auto** to enable auto mode. Use **getext** to display the current list. Use the client keyword **FileCopyAsciiExtensions** to change the default list.

symlink *linked_path target_path*

Creates a symbolic link (soft) from *linked_path* to *target_path* on the remote host.

verbose

Sets the debug level to verbose mode, which is equivalent to setting the debug level to 2. To disable verbose mode, use 'debug disable'.

version

Displays the supported SFTP protocol version.

APPENDIX M

ssh-add Command Line Options

The syntax for **ssh-add** is:

```
ssh-add [-c] [-d] [-D] [-h] [-l] [-L] [-p] [-t timeout] [-U] [-V] [file1 file2 ...]
```

Use *file1, file2...* to specify keys to add to the agent. Specifying a key file is optional. If don't specify any key files, **ssh-add** adds all the keys specified in your identification file (which is `~/.ssh2/identification` by default)

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

-c

Specifies that agent should ask for confirmation before using a key.

-d

Removes one or more specified keys from the agent. Use the *file* argument to specify the key file(s).

-D

Removes all identities from the agent.

-h

Displays a brief summary of command options.

-l

Lists all the identities currently loaded in the key agent.

-L

Locks the key agent. You are prompted for a password, which you will need to use to unlock the agent. Use **-U** to unlock.

-p

Reads the passphrase from stdin. This may be done over a pipe.

-t <timeout>

Sets a timeout for the key. Use zero (0) to set no limit. Keys are deleted after the specified timeout.

By default, the timeout value is set in minutes. You can specify other units using this syntax:

n<unit>[n<unit>...]

Where unit can be: s (seconds), m (minutes), h (hours), d (days), or w (weeks). (Upper or lower case units are both accepted with the same meaning.) For example:

3600s = 3600 seconds (1 hour)

2w = 2 weeks

2d4h = 2 days and 4 hours

-U

Unlocks an agent that has been locked using **-L**. You are prompted for the required password.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-x

Specifies that the key files to be added are associated with X.509 certificates. If you use **-x** without specifying a file or files, Reflection for Secure IT reads your identification file (`~/.ssh2/identification` by default) and adds all the keys identified using the **CertKey** keyword. Certificates must be in the same directory as the associated private key and use the same base name with a `.crt` file extension.

APPENDIX N

ssh-agent Command Line Options

The syntax for **ssh-agent** is:

```
ssh-agent [-c] [-d debug_level] [-h] [-k] [-s] [-t timeout] [-v] [command]
```

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

-c

Forces the shell to be csh. By default **ssh-agent** uses the SHELL environment variable to determine which shell to invoke. This option overrides the default behavior.

-d *debug_level*

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

-h

Displays a brief summary of command options.

-k

Kills the current agent (given by the SSH_AGENT_PID environment variable).

-s

Forces the shell to be sh. By default **ssh-agent** uses the SHELL environment variable to determine which shell to invoke. This option overrides the default behavior.

-t *timeout*

Sets a default timeout for any key added to the agent. Keys are deleted after the specified timeout. By default keys have no timeout limit, which is equivalent to setting a timeout value of zero(0). (You can also specify timeouts when you add keys by using the **ssh-add -t** option, which overrides this setting.)

By default, the timeout value is set in minutes. You can specify other units using this syntax:

```
n<unit>[n<unit>...]
```

Where unit can be: s (seconds), m (minutes), h (hours), d (days), or w (weeks). (Upper or lower case units are both accepted with the same meaning.) For example:

```
3600s = 3600 seconds (1 hour)
```

```
2w = 2 weeks
```

```
2d4h = 2 days and 4 hours
```

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

APPENDIX O

sshd Command Line Options

-4

Forces connections using IPv4 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-6

Forces connections using IPv6 addresses only. You can also configure IP address requirements using the **AddressFamily** keyword.

-b

When this option is specified **sshd** doesn't detach and doesn't become a daemon. This can be used for monitoring.

-d *level*

Sets the debug level and sends debug output to stderr. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.) With this option **sshd** logs information for only one client connection, and terminates after the client connection closes.

Note: Setting logging to 99 can increase your security risk. At this level, information leakage is a concern, as unencrypted protocol information may be written out. Also, the volume of information written may fill up disk space rapidly, potentially causing the host or Reflection for Secure IT to stop responding.

-D *level*

Sets the debug level and sends debug output to a file. This setting can only be used by root. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.) With this option, **sshd** does not terminate after a client connection closes.

Note: Setting logging to 99 can increase your security risk. At this level, information leakage is a concern, as unencrypted protocol information may be written out. Also, the volume of information written may fill up disk space rapidly, potentially causing the host or Reflection for Secure IT to stop responding.

The output file is located in `/etc/ssh2` and uses a file name in the form: `debugYYMMDD_HHMMSS_uniqueID`, where YY=year, MM=month, DD=day, HH=hour, MM=minutes, SS=seconds, and uniqueID is a unique value that ensures that servers started at the same time use different logs.

-f *config_file*

Specifies an alternate name and location for the server configuration file. The server reads the specified file instead of the default file. (The default configuration file is `/etc/ssh2/ssh2_config`.)

-g *login_timeout*

Sets the number of seconds allowed for client authentication. If the client fails to authenticate the user within the specified number of seconds, the server disconnects and exits. Use zero (0) to set no limit.

-h *host_key_file*

Specifies the filename and location of the private key used to authenticate the server. If the path is not fully qualified, the path is assumed to be relative to `/etc/ssh2`. The default is `/etc/ssh2/hostkey`.

-o *option*

Sets any option that can be configured using a configuration file keyword. For a list of keywords and their meanings, see `sshd2_config(5)`. Options configured on the command line override options configured in the configuration file. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
-o key=value1,value2
-o key="value1, value2"
```

To configure multiple options, use multiple **-o** switches.

```
-o key1=value -o key2=value
```

-p *port*

Specifies the port on which the server listens. The default is 22, which is the standard port for Secure Shell connections. The command line value overrides any value set in the configuration file. Only one port is allowed; if you configure multiple ports, the last configured port is used.

-q

Enables quiet mode. In this mode only errors are logged to the system log. (Both **-d** and **-D** are ignored if **-q** is used on the same command line.)

-v

Sets the debug level to verbose mode, which is equivalent to using `'-d 2'`.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

APPENDIX P

ssh-certview Command Reference

SYNOPSIS

```
ssh-certview [-C] [-c] [-d debug_level] [-h] [-q] [-v] [-v] [file ...]
```

DESCRIPTION

Use **ssh-certview** to view the contents of X.509 certificates (in either PEM or DER format), CRL lists, or PKCS#10 requests. You can also output sample syntax for use in `pki_mapfile(5)`, which is used by Reflection PKI Services Manager to map certificates to allowed identities.

The **ssh-certview** output for certificate fields is compliant with RFC2253. To be compliant with this standard, Subject and Issuer fields start with the Common Name (for example, "CN = Secure CA, O = Secure Corporation, C = US"). This format is also used by Reflection PKI Services Manager.

Note: Other utilities (including earlier versions of Reflection for Secure IT) reverse the order of the field content in the Subject field output. The reversed format is not equivalent and will not result in a match if used in a PKI Services Manager map file.

OPTIONS

Options are available in both a single-character form (such as **-o**) and a descriptive equivalent (**--option**). Single characters are shown here. To view the descriptive equivalents, use the **-h** command line option.

-C

Specifies that output should include a comment mark (#) at the beginning of each line of output.

-c

Extracts content from a certificate and outputs correct syntax for inclusion in `pki_mapfile(5)`. Unless you also specify **-q**, standard output is also included, and is preceded by comment marks.

-d *debug_level*

Sets the debug level. Increasing the value increases the amount of information displayed. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

-h

Displays a brief summary of command options.

-q

Turns off display of all output except map file syntax. Use this option with **-c** to output just map file syntax without commented certificate information.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-v

Increases the verbosity of the output data.

If you are viewing a certificate with **-v**, the output includes Issuer, Serial Number (hex), Subject, Subject Alternative Name, Validity period, Extensions if set (including Key usage, Constraints, CDP, AIA, Policy OIDs), Public key type, and Public key fingerprint. Without this option, the output shows Issuer, Serial Number (hex), and Subject.

If you are viewing CRL with **-v**, the output includes the entire list of revoked certificates. Without this option, the output shows issuer and update information.

EXAMPLES

To view the contents of the specified certificate, including full information about certificate extensions:

```
ssh-certview -v sample.crt
```

To view the contents of the certificate cacert.pem, the certificate request cacert.pem.p10, and the certification revocation file example.revoke.crl:

```
ssh-certview cacert.pem cacert.pem.p10 example.revoke.crl
```

To extract sample output from the specified certificate for inclusion in a PKI Services Manager map file:

```
ssh-certview -q -c cacert.pem
```

APPENDIX Q

ssh-certtool Command Reference

SYNOPSIS

```
ssh-certtool [-b key-length] [-c comment] [-d debug-level] [-h] [-n algorithm]
[-o output-file] [-p private-key] [-P] [--passphrase passphrase] [-V] [-z
option] pkcs10|pkcs12 [arguments]
```

```
ssh-certtool [options] pkcs10 subject [keyUsage] [extendedKeyUsage]
```

```
ssh-certtool [options] pkcs12 [file1] ... [fileN]
```

DESCRIPTION

You can use **ssh-certtool** to create a PKCS#10 certificate request or to create a PKCS#12 package containing a private key and one or more certificates.

Creating a PKCS#10 certificate request

The general syntax for creating a PKCS#10 file is:

```
ssh-certtool [options] pkcs10 subject [keyUsage] [extendedKeyUsage]
```

Note: **req** is supported as a synonym for **pkcs10**.

The value you specify as subject defines the certificate's Subject field. The subject name is required. Use the distinguished name syntax specified by RFC2253. Use commas to separate Subject elements (RDNs). RDNs can be specified using standard abbreviations (CN) or OIDs (2.5.4.3). Quotation marks are required if the subject name contains embedded white space. For example, "CN=Steve Kille,O=Isode Limited,C=GB".

The filename of the generated certificate request is based on the prefix specified by the **-o** option, with `.pkcs10` appended. The default filename of a generated private key, when **-o** is not specified, is `output.pkcs10`.

To create a request using an existing private key use **-p** to specify the key. To generate a new private key for the request, you must specify either key type (**-n**), key size (**-b**) or both. The filename of the generated private key is based on the prefix specified by the **-o** option, with `.ssh2` appended. The default filename of a generated private key, when **-o** is not specified, is `output.ssh2`. If a key with the same name already exists, you are prompted to overwrite it. If you elect not to overwrite it, **ssh-certtool** exits with a return code of zero.

You can use optional flags to set `keyUsage` and `extendedKeyUsage` fields. Use commas, spaces or tabs to separate items. All Key Usage and Extended Key Usage flags are marked as critical in the PKCS#10 request. Valid `keyUsage` flags are `digitalSignature`, `nonRepudiation`, `keyEncipherment`, `dataEncipherment`, `keyAgreement`, `keyCertSign`, `cRLSign`, `encipherOnly` and `decipherOnly`. If you omit this argument, the `digitalSignature` and `keyEncipherment` flags are set by default. Valid `extendedKeyUsage` flags are `anyExtendedKeyUsage`, `serverAuth`, `clientAuth`, `codeSigning` and `emailProtection`. No extended key usage flags are set by default.

Creating a PKCS#12 package

The general syntax for creating a PKCS#12 package is:

```
ssh-certtool [options] pkcs12 [file1] ... [fileN]
```

This constructs a PKCS#12 package file containing one private key and multiple certificates read from the *file* arguments. The PKCS#12 package file contains one safe, which contains the private key and all the certificates. The filename of the generated package file is based on the prefix specified by the `-o` option, with `.p12` appended. The default filename of the generated PKCS#12 package, when `-o` is not specified, is `output.p12`. The PKCS#12 package is protected by an HMAC, and **ssh-certtool** prompts you for a passphrase before creating the package.

File arguments containing private keys can be read in naked PKCS#8 format, in ssh2 PEM format, or in openSSH PEM format. If the key is protected by a passphrase, **ssh-certtool** prompts for the passphrase. *File* arguments containing certificates are recognized in both DER-encoded and PEM-encoded format.

By default, the individual private key and certificates are saved into the PKCS#12 output file using default PBE protection schemes. The default scheme for key encryption is `pbeWithSHA1And3-KeyTripleDES-CBC`. The default for safe encryption is `pbeWithSHA1And40BitRC2-CBC` format. You can use the `-z` option to configure different PBE protection schemes.

OPTIONS

Options are available in both a single-character form (such as `-o`) and a descriptive equivalent (`--option`). Single characters are shown here. To view the descriptive equivalents, use the `-h` command line option.

-b *bits*

Specifies the key size used for generated keys. The default for RSA keys is 2048 bits and for DSA keys is 1024 bits. The value for a DSA key must be an integral multiple of 64. This option is valid for PKCS#10 file creation only.

-c *comment*

Specifies a comment to include in the private key file. This option is valid for PKCS#10 file creation only.

-d *debug_level*

Enables debug output. Use 1, 2, 3, or 99. (Values 4-98 are accepted, but are equivalent to 3.)

-h

Displays a brief summary of command options.

-n *algorithm*

Specifies the algorithm used for key generation. Possible values are "rsa" and "dsa". The default is "rsa". This option is valid for PKCS#10 file creation only.

-o *output_file_prefix*

Specifies the first portion of the filename for output files. You can include an absolute path to generate the file in a different location. The default is "output". (The filename suffix is generated based on the file type: the suffix for PKCS#10 files is .pkcs10, for PKCS#12 is .p12, and for private keys is .ssh2.)

-p *private_key*

Specifies a private key to use in a certificate request. This option is valid for PKCS#10 file creation only.

-P

Saves the private key with an empty passphrase. This option is valid for PKCS#10 file creation only.

--passphrase *passphrase*

Specifies a passphrase for the private key. This option is valid for PKCS#10 file creation only.

-V

Displays product name and version information and exits. If other options are specified on the command line, they are ignored.

-z *Key=Value*

Specifies certificate options for PKCS#10 requests, and encryption options for PKCS#12 packages.

For PKCS#10 requests, *key* must be either **DNS** or **Email** (case-insensitive). There should be no white space in this option, including before or after the equal sign, unless the value literally contains white space characters in its name. The DNS option sets the DNS Alt Name extension. The Email option sets the EMail Alt Name extension. These extensions are not marked as critical.

For PKCS#12 packages, *key* must be either **KeyPBE** or **SafePBE** (case-insensitive). There should be no whitespace in this option, including before or after the equal sign. KeyPBE sets the key encryption and hmac scheme. SafePBE sets the safe encryption and hmac scheme. Values are listed below. The default for **KeyPBE** is PBE-SHA1-3DES. The default for **SafePBE** is PBE-SHA1-RC2-40. The long names in parentheses are synonyms.

None

PBE-SHA1-RC4-128 (pbeWithSHA1And128BitRC4)

PBE-SHA1-RC4-40 (pbeWithSHA1And40BitRC4)

PBE-SHA1-3DES (pbeWithSHA1And3-KeyTripleDES-CBC)

PBE-SHA1-2DES (pbeWithSHA1And2-KeyTripleDES-CBC)

PBE-SHA1-RC2-128 (pbeWithSHA1And128BitRC2-CBC)

PBE-SHA1-DES (pbeWithSHA1AndDES-CBC)

PBE-SHA1-RC2-40 (pbeWithSHA1And40BitRC2-CBC)

PBE-MD2-RC2-64 (pbeWithMD2AndRC2-CBC)

PBE-MD5-RC2-64 (pbeWithMD5AndRC2-CBC)

EXAMPLES

To create a PKCS#10 request using a newly generated key:

```
ssh-certtool -n RSA -z DNS=steves.dns.server.com -z Email=steved@myorg.org
pkcs10 CN=steved,O=myorg.org,OU=rsit,C=US DigitalSignature,nonRepudiation
ServerAuth,ClientAuth
```

To create a PKCS#12 package file and specify encryption for the key and safe:

```
ssh-certtool -z keyPBE=default -z safePBE=PBE-SHA1-RC4-40 -ofile pkcs12
id_rsa.crt id_rsa
```

APPENDIX R

winpki and pkid Command Reference

Use **winpki** (on Windows) or **pkid** (on UNIX systems) to configure, start, and stop the PKI Services Manager service, and to check certificate validity and allowed identities.

Synopsis

Windows:

```
winpki [command [command args]] [options...]
```

UNIX:

```
pkid [command [command args]] [options...]
```

command = **start** | **stop** | **restart** | **reload** | **ping** | **validate** <cert>

options = [-b *path*] [-c *cert*] [-d *level*] [-f *file*] [-h] [-i] [-k]

[-m *path*] [-p] [-o *key=value*] [-t *host*] [-u *user*] [-V] [-w]

Commands

start

Starts the service.

stop

Stops the service.

restart

Stops and restarts the service.

reload

Reloads the configuration without stopping the service. Note: Some settings require a restart.

ping

Displays service status and the port used by the service.

validate *certificate*

Validates a certificate and optionally provides information about allowed identities. The service must be running. For example, to determine if `sample.crt` is valid (UNIX syntax):

```
pkid validate sample.crt
```

Use **-u**, **-t**, or **-w** after the certificate name to get information about allowed identities for the specified certificate. For example, to determine if the user `joe` can authenticate using `sample.cer` (Windows syntax):

```
winpki validate sample.cer -u joe
```

Options

-b *path* **--baseDir** *path*

Specifies the data directory used for PKI Services Manager configuration.

-c *cert* **--cert** *cert*

Validates the specified certificate. This option is available when the service is not running. Use the **validate** command to validate certificates when the service is running.

-d *level* **--debug** *level*

Specifies the amount of information sent to the log. Allowed values are: 'error', 'warn', 'info', 'debug', and 'trace'. The default is 'error'.

-f *file* **--config_file** *file*

Launches using a non-default configuration file.

-h **--help**

Displays a brief summary of command options.

-i **--init**

This option is rarely needed. It initializes PKI Services Manager, which creates a key pair for the server, and creates user data directories and files. Initialization happens automatically during installation on UNIX systems and on first run on Windows systems. Using this option has no effect if your system is already initialized. Note: You can create new keys by deleting the existing keys (`pki_key` and `pki_key.pub`), and then using this option. Existing configuration files are not affected.

-k **--check-config**

Checks for errors in your configuration and map files and then quits.

-m *path* **--migrate** *path*

Migrates certificate authentication settings from Reflection and F-Secure configuration files. If *path* specifies a directory, PKI Services Manager looks for server (`sshd2_config`) and client (`ssh2_config`) configuration files in that directory and migrates settings from those files. If *path* specifies a file, PKI Services Manager migrates the settings in the specified file. Full path information is required for both files and directories. Note: If the `pki_config` file in the destination folder already has a trust anchor configured, no migration occurs. This helps ensure that the migration won't overwrite modifications you have already configured.

Settings are migrated to the `pki_config` and `pki_map` files used by PKI Services Manager. If you use the `-b` switch, files with your migrated settings are created in the specified directory. If you omit this switch, the files are created in the default PKI Services Manager configuration directory.

A migration log is created in the `logs` directory located in the PKI Services Manager data directory. By default, this log records at a level of 'info' which shows if errors or warnings occurred. The level can be elevated using `-d`.

`-o key=value` **--option** `key=value`

Sets any option that can be configured using a configuration file keyword. Options configured this way override configuration file settings. For a list of keywords and their meanings, see `pki_config`. Syntax alternatives are shown below. Use quotation marks to contain expressions that include spaces.

```
-o key1=value
-o key1="sample value"
-o "key1 value"
-o key=value1,value2
-o key="value1, value2"
```

To configure multiple options, use multiple `-o` switches.

```
-o key1=value -o key2=value
```

-p--showkey

Displays the public fingerprint and shows the full path and key name.

`-t host` **--hostName** `host`

Use this option after the certificate name following a **validate** command. PKI Services Manager reads the map file(s) and reports whether the specified host is an allowed identity for the host certificate being validated.

`-u user` **--userID** `user`

Use this option after the certificate name following a **validate** command. PKI Services Manager reads the map file(s) and reports whether the specified user is an allowed identity for the user certificate being validated. If you include a server name (in the form `user@server`), PKI Services Manager reports on whether the user is allowed to authenticate to the specified server. If you specify only a user name, PKI Services Manager tests whether the user is allowed to authenticate with this certificate without checking for host-specific conditions.

-V --version

Displays the product name and version.

-w [host] --whoAmI [host]

Use this option after the certificate name following a **validate** command. PKI Services Manager reads the identity map file(s) and returns a list of all allowed identities for the certificate being authenticated. If you specify a server name after this option, the list is limited to allowed users for connections to that server. If no server name is specified, PKI Services Manager doesn't check for server-specific conditions.

APPENDIX S

pkid_config Configuration File Reference

The Reflection PKI Services Manager console saves settings to the configuration file. You can also view and edit this file manually. The default file location is:

- UNIX
`/opt/attachmate/pkid/config/pki_config`
- Windows XP, Windows Server 2003:
`\Documents and Settings\all users\Application Data\Attachmate\ReflectionPKI\config\pki_config`
- Windows 7, Windows Vista, Windows Server 2008:
`\ProgramData\Attachmate\ReflectionPKI\config\pki_config`

File Format

The configuration file consists of keywords followed by values. The value can be separated from the keyword by tabs, spaces, or spaces and one '='. Any line starting with a pound sign (#) is a comment. Any empty line is ignored. Some keywords can appear multiple times, and these settings are applied cumulatively. Changes to settings do not take effect until you reload the settings or restart the service. (If a restart is required, that information is given in the keyword description.)

The file includes a global section that contains settings that apply to all validation queries. You can also create stanzas that configure certificate-specific settings. The **TrustAnchor** keyword marks the beginning of each trust anchor stanza. Settings beneath the **TrustAnchor** keyword apply only to that trust anchor. The stanza ends at the next **TrustAnchor** keyword.

Some settings must be configured outside any trust anchor stanzas. These settings apply to all validation queries. Where a setting is supported both globally and within a stanza, the value within the trust anchor stanza overrides the global value.

Keywords

AllowClientStats

Specifies whether PKI Services Manager allows clients to request PKI Services Manager runtime statistics. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'yes'.

AllowMD2Certificates

Allow certificates signed using the MD2 RSA hash. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'no'. You need to restart the service if you modify this setting.

AllowMD5InFipsMode

Allow certificates signed using the MD5 hash, even when FIPS mode is enabled. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'yes'. You need to restart the service if you modify this setting.

AllowVers1

Specifies whether PKI Services Manager allows version 1 certificates for a trust anchor. Note: Intermediate certificates must be version 3 regardless of the value of this setting. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'no'.

AllowWhoAmI

Specifies whether PKI Services Manager allows a client to query for the mapped identity (using **-w** or **--whoAmI**) when using PKI Services Manager to validate certificates. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'yes'.

CertSearchOrder

A comma-separated list that specifies where PKI Services Manager searches for intermediate certificates required to validate a certificate. Listed locations are searched in order. The options are 'local','certserver', 'aia', and 'windows'. The default is 'local, certserver.' (Note: If you select 'windows', PKI Services Manager uses only those certificates that are installed for use by the local computer, not certificates installed for the current user. To view and manage the local computer certificates, use the Microsoft Management Console. Add the Certificates Snap-in and configure it to manage certificates for the computer account.) Configure this keyword once, outside any stanza.

CertServers

Specifies a server from which PKI Services Manager can retrieve intermediate certificates when 'certserver' is included in the **CertSearchOrder** list. You can specify either an HTTP or an LDAP server. (For example: `ldap://certserver:10389` or `http://certserver:1080`) This keyword can be configured multiple times outside any stanza. The values are cumulative.

CRLServers

Specifies a server from which PKI Services Manager can retrieve Certificate Revocation Lists (CRLs) when 'crlserver' is included in the **RevocationCheckOrder** list. You can specify either an HTTP or an LDAP server. (For example: `ldap://crlserver:10389` or `http://crlserver:1080`.) This keyword can be configured multiple times outside of any stanza and multiple times per stanza. The values are cumulative.

ClientDebugging

Specifies whether the application that is requesting certificate validation can request and receive debug messages from PKI Services Manager. Configure this keyword once, outside any stanza. The allowed values are 'yes' and 'no'. The default is 'no'. Note: To view these messages you also need to set a sufficiently detailed debug level in the calling application. For the Reflection for Secure IT Windows server, specify "Protocol details" or higher. For the Reflection for Secure IT UNIX clients and servers, specify debug level 3 or higher.

EnforceDODPKI

Determines whether PKI Manager enforces settings that meet US Department of Defense PKI requirements. The allowed values are 'yes' and 'no'. The default is 'no'. When this setting is 'yes', the service will not start unless the following conditions are met: **FipsMode** = yes; **AllowMD2Certificates** = no; **AllowMD5InFipsMode** = no; **AllowVers1** = no; **CertSearchOrder** does not include 'windows'; and **RevocationCheckOrder** has at least one option specified and does not include 'none'.

ExplicitPolicy

Determines whether PKI Services Manager enforces application policies. This keyword can be configured once outside of any stanza and once per stanza. The allowed values are 'yes' and 'no'. The default is 'no'. If the value is 'yes' you must specify one or more application policies to be enforced using the **PolicyOID** keyword. Each application policy is specified with a Policy Identifier (OID). (Note: Policies may also be required by the certificate being presented or by a certificate within the chain of trust.)

FipsMode

Enforces security protocols and algorithms that meet FIPS 140-2 standards. The allowed values are 'yes' and 'no'. The default is 'yes'. Configure this keyword once, outside any stanza. You need to restart the service if you modify this setting.

KeyFilePath

Specifies the path to the private key used to identify Reflection PKI Services Manager. When no path is specified, the path or file name is relative to the PKI Services Manager configuration directory. Configure this keyword once, outside any stanza. This setting is required. If **KeyFilePath** is not specified, or no key is present, the PKI Services Manager service will not start. The default is 'pki_key'. You need to restart the service if you modify this setting. PKI Services Manager creates a key pair when it initializes the settings, but you can also use a key pair created by **ssh-keygen** (or another tool). Only RSA keys are allowed.

ListenAddress

Specifies the port on which PKI Services Manager listens for validation requests. The syntax is `host:port`. You can specify the host name using either an IP address or a host name. IP addresses can be in either IPv4 or IPv6 format. IPv6 addresses must be enclosed in square brackets, for example `[::D155:AB63]:18081`. The default is `0.0.0.0:18081`, which configures the server to listen on port 18081 using any available network adapter. This setting is required. You need to restart the service if you modify this setting.

LocalStore

The local store is used to hold items that are required for certificate validation. Depending on your configuration, this may include trusted root certificates, intermediate certificates, and/or Certificate Revocation Lists (CRLs). You can specify directories or files. When a directory is specified, all files in the specified directory and any subdirectories are included in the store. Files must be binary or base 64 encoded X.509 certificates or CRLs. This keyword can be configured multiple times outside any stanza. The values are cumulative. This setting is required.

LogFacility

Specifies the output location for log messages. Allowed values are 'file' and 'none'. The default is 'file'. Log files are created daily and saved to a directory called `logs` located in the PKI Services Manager data directory. Configure this keyword once, outside any stanza. You need to restart the service if you modify this setting.

LogLevel

Specifies the amount of information sent to the log. Allowed values are: 'error', 'warn', 'info', 'debug', and 'trace'. The log can contain both auditing messages (labeled "[audit]"), and debug messages (labeled "[debug]"). Auditing messages provide information about both successful and unsuccessful validation attempts. Debug messages are designed to help in troubleshooting. The default log level is 'error'. At this level, auditing messages are sent to the log, but debug messages are sent only if a PKI Services Manager error occurs, generally because PKI Services Manager is not correctly configured. The other options include audit messages plus increasing levels of detail in the debug messages. Configure this keyword once, outside any stanza.

MapFile

Specifies the location of the PKI Services Manager map file. Use the map file to configure which users or computers are allowed to authenticate with a valid certificate. When no path is specified, the path or file name is relative to the PKI Services Manager configuration directory. This setting is required. This keyword can be configured once outside of any stanza and once per stanza.

MaxLogFiles

Specifies the maximum number of log files to create. A new log file is automatically created daily. When the maximum is reached, the oldest log is removed. The default is 10. Configure this keyword once, outside any stanza. You need to restart the service if you modify this setting.

NetworkTimeout

Specifies the timeout for any network download: LDAP, HTTP, or OCSP. Units are milliseconds. The default is 20000. Configure this keyword once, outside any stanza. Configure this keyword once, outside any stanza.

OCSPCertificate

Specifies a certificate that can be used to verify the signature of the OCSP response. This is needed only if the OCSP response does not include the signer's certificate. The value can be either a certificate file or the Subject value of the certificate (for example `OcspCertificate = "CN = Secure CA, O = Secure Corporation, C = US"`). If you use the Subject value, the certificate must be in the local store. This keyword can be configured multiple times outside of any stanza and multiple times per stanza. The values are cumulative.

OCSPResponders

Specifies the address of an OCSP responder to use for checking certificate revocation when 'ocsp' is included in the **RevocationCheckOrder** list. Use an HTTP address to identify the responder. (For example: `http://ocsp.myhost.com:1080`.) This keyword can be configured multiple times outside of any stanza and multiple times per stanza. The values are cumulative.

PolicyOID

Specifies an allowed Policy Identifier (OID) to use when application policies are in force, either because **ExplicitPolicy** is 'yes' or because policies are required by the certificate being presented or by a certificate within the chain of trust. When **ExplicitPolicy** is 'yes', the specified OID must match at least one of the OIDs in the final policy set of the certificate chain. The value 2.5.29.32.0 allows use of any Policy Identifier. (Note: The default value is 'no-policy'. When **ExplicitPolicy** is set to 'yes', you must change **PolicyOID** to indicate which policy or policies are allowed; if **ExplicitPolicy** is set to 'yes' and **PolicyOID** is set to 'no-policy', no certificate can pass validation.) This keyword can be configured multiple times both outside any stanza and within a stanza. Configured values are cumulative.

RevocationCheckOrder

A comma-separated list that specifies which sources are used to check for certificate revocation and the order in which these checks occur. The options are 'ocsp', 'cdp', 'crlserver', 'local', and 'none'. The default is 'local'. Note: If you specify just 'none', no revocation checking occurs. If you specify 'none' with other options, PKI Services Manager attempts to determine the revocation status using the specified options until it reaches 'none'. If the certificate revocation status is still unknown at this point, authentication is allowed. This keyword can be configured once outside of any stanza and once per stanza.

StrictMode

Specifies whether strict checking rules (as defined in RFC 3280) are used when validating certificates. Many certificates cannot pass strict checks. The allowed values are 'yes' and 'no'. The default is 'no'. This keyword can be configured once outside of any stanza and once per stanza.

TrustAnchor

Specifies a certificate to use as the final trust point in a certificate chain of trust that Reflection for Secure IT validates. This can be an intermediate CA certificate, a root CA certificate, or a self-signed certificate (which can only validate itself). It can not be a user certificate or host certificate.. The value can be either a certificate filename or the contents of the Subject field defined in the certificate (for example `TrustAnchor = "CN = Secure CA, O = Secure Corporation, C = US"`). If you specify a certificate filename and include full path information, the trust anchor is used regardless of how you configure the **CertSearchOrder** keyword. If you specify a certificate filename without including full path information, **CertSearchOrder** must include 'local'; and PKI Services Manager looks for the certificate in your local store. If you specify the contents of the certificate's Subject field, **CertSearchOrder** must include 'local' and/or 'windows'; and PKI Services Manager looks for the certificate in your local store and/or Windows certificate store. This setting is required. To configure multiple trust anchors, add additional **TrustAnchor** lines.

Note: On Windows systems, you can view the Subject value of certificates in your store using the PKI Services Manager console. On UNIX systems, you can use `ssh-certview(1)` to view this information.

Any keywords under a **TrustAnchor** setting create a stanza. The values you configure within a trust anchor stanza are specific to that trust anchor.

APPENDIX T

pkimapfile Map File Reference

Reflection PKI Services Manager binds certificates to one or more allowed identities using mapping rules. Typically, allowed identities are users or hosts. To authenticate a user correctly, you need to define a rule that links information in the validated certificate to an allowed user account. The mapper provides flexible options for mapping certificates to names. You can specify allowed names explicitly in your rules, or define rules that extract information, such as user or host name, from a certificate. By using these options, you can bind identities to certificates without having to create a separate rule for each certificate.

The default map filename and location is:

- UNIX
`/opt/attachmate/pkid/config/pki_mapfile`
- Windows XP, Windows Server 2003:
`\Documents and Settings\all users\Application Data\Attachmate\ReflectionPKI\config\pki_mapfile`
- Windows 7, Windows Vista, Windows Server 2008:
`\ProgramData\Attachmate\ReflectionPKI\config\pki_mapfile`

Note: On Windows systems, you can modify the map file from the Reflection PKI Services Manager console using the **Identity Mapper** pane.

File Format

The map file consists of keyword settings and rules. Each rule is a single line and is independent of other rules. The format of a rule is:

```
{Allowed-Identity} [Conditional Expression]
```

After a certificate is determined to be valid, rules are processed in order (based on rule type then sequence). If the certificate meets the requirements defined in the conditional expression (or if the rule has no condition), the allowed identities specified in that rule are allowed to authenticate. No additional rules are applied after the first match.

Within the map file, you can use the **RuleType** keyword to apply different mapping criteria based on whether a user or host presents the certificate. Note: Rule type determines the order in which rules are processed. The order for processing user certificates is: user-address, user, none. The order for processing host certificates is: host, none. Within each rule type, rules are processed in order from top to bottom.

Allowed Identity Set

The allowed identity set is a required component of a rule. Allowed identities can be specified using a combination of constant values and values extracted from the certificate. The set of allowed identities can take multiple constant values, extracted values, or a combination of both.

Using constant values to define allowed identities

Constant values are literal strings. Use white space to delimit separate values. (If an allowed name includes spaces, enclose it in quotes.) For example, the following rule uses literal strings to allow root, joe, and fred smith to authenticate with any valid certificate:

```
{ root joe "fred smith" }
```

The format "*domain\user*" is required for Windows domain users, for example:

```
{ windomain\joe "windomain\fred smith" }
```

Note: After PKI Services Manager determines that a certificate meets the condition defined in a rule, rule processing stops. In both examples above, no conditions are defined. This means the rule will be applied to any valid certificate and no subsequent rules will be processed. To create a similar rule, you would need to include all allowed identities within the same rule.

Two asterisks used alone { ** } act as a wildcard for defining the allowed identity set. This option may be useful for testing, but should otherwise be used only with extreme caution. If you use this wildcard in a user rule, any user presenting a valid certificate is allowed to authenticate to any user account on the server. This creates a major security risk by allowing access to accounts with root, administrator, or power user privileges without requiring a password. If you use this wildcard in a host rule, any server with a valid certificate is accepted by the client. If you do choose to use the wildcard, consider limiting access using other options:

- Use the wildcard only with certificates signed by Certification Authorities that you control.
- Use the wildcard only in rules that have very restrictive conditions.
- Use the wildcard only in server-specific user rules (those whose **RuleType** is **user-address**).
- Limit user account access on the server side. For example, on a Secure Shell server, you might define sftp chroot jails and allow no command shell or remote command access.

Using values extracted from the certificate

Use extracted values to construct the allowed identity set based on the contents of the certificate presented for authentication. Extracted values must be preceded and followed by "%". For example, to allow authentication by the host specified in the Host portion of the UPN field:

```
{ %UPN.Host% }
```

You can also combine literal strings with extracted identities. (You can prepend a literal string to an extracted identity, and/or append a literal string, but you cannot combine more than one extracted value to form a single identity.) The following example adds a Windows domain name to an extracted user identity:

```
{ windomain\%UPN.User% }
```

Note: If the extracted identity evaluates to an empty result, the entire concatenated string is deemed to be empty and is not included in the set of allowed identities. If the entire set of allowed identities is empty, the rule is deemed to have failed and processing continues to the next rule.

Supported certificate fields are:

Subject

The Subject field defined in the certificate. The comparison is done following X.500 rules (not as a string comparison). For a successful match, the format must follow standards described in RFC 2253. To be compliant with this standard, Subject and Issuer fields start with the Common Name (for example, "CN = Secure CA, O = Secure Corporation, C = US"). On UNIX systems, you can use the **ssh-certview** utility to obtain the Subject value in this format. On Windows systems, copy the Subject contents from the Details tab of the certificate viewer, paste to an editor, and then replace new line characters with commas.

Subject.CN

The Common Name portion of the Subject field, if present.

Subject.Email

The email attribute part of the Subject, if present.

DNS

The DNS part of a SubjectAltName, if present.

UPN

The "otherName" representation of the SubjectAltName field, with the OID of 1.3.6.1.4.1.311.20.2.3 (UPN OID), if present.

UPN.User

The userID portion of the UPN field.

UPN.Host

The host portion of the UPN field.

Email

The representation of SubjectAltName as defined in RFC 822.

Email.User

The userID portion of Email.

Email.Host

The host portion of Email.

SerialAndIssuer

The certificate serial number (hex encoded) and value of the certificate's Issuer field in this format:

serial_number Issuer

Use white space to separate the serial number from the issuer. For example:

461D07A8 CN = Secure CA, O = Secure Corporation, C = US

Cert

This indicates the entire certificate. The Operation must be **Equals** and the argument must be a file path to a certificate. Note: The Mapper does not use the certificate store defined by Reflection PKI Services Manager.

subst

This option is available when the conditional expression within a rule uses either **Regex** or **Extern**.

With **Regex**, use **subst** in combination with any regular expression that has a capturing group, which has been identified using round brackets (). If the regular expression includes an exact match to a specified certificate field, the value of the first capturing group in the expression replaces %subst% in the allowed identity set.

With **Extern**, use **subst** as a placeholder for the value returned by the external application.

Conditional Expression

When a conditional expression follows the {Allowed-Identity}, the allowed identities can authenticate only if the conditional expression is true. The use of a conditional expression is optional, but in most cases is recommended. If no conditional expression is included, the allowed identities can authenticate with any valid certificate.

After a certificate is determined to be valid, rules are processed in order (based on rule type then sequence). If the certificate meets the requirements defined in the conditional expression (or if the rule has no condition), the allowed identities specified in that rule are allowed to authenticate. No additional rules are applied after the first match.

The syntax for a conditional expression is:

Field Operation Argument

For *Field*, specify any of these supported certificate fields (described above): Subject, Subject.CN, Subject.Email, DNS, UPN, UPN.User, UPN.Host, Email, Email.Host, SerialAndIssuer, Cert, or subst.

For *Argument*, specify a string value.

For *Operation*, use one of the following:

Equals

Checks for absolute equality between the *Field* value and the *Argument* string. For DNS, UPN and Email options, the comparison is case-insensitive.

Contains

Checks if the *Field* value is contained anywhere within the *Argument* string. For DNS, UPN and Email options, the comparison is case-insensitive.

Regex

Applies the *Argument* as a regular expression to the *Field*. If the regular expression includes an exact match to the *Field* contents, the condition is true. If the set of allowed identities contains the string **%subst%**, the first capturing group (if defined) of the Regex match is inserted.

Extern

Uses an external application to test the condition. Use *Argument* to point to the application. Use **%subst%** in the allowed identity set as a placeholder for the value returned by the external application. If the match within the external application is successful, it should exit with status 0; a non-zero return means an unsuccessful match.

Sample rules with conditional expressions:

```
{ %UPN.Email% } Subject.CN Equals acme.com
{ joep } Subject Contains "Joe Plumber"
```

Rule Type Stanzas

Rule types apply different mapping criteria based on whether the validated certificate is a user certificate or a host certificate. Use the **RuleType** keyword to create a new stanza for each supported type. A stanza ends at the next **RuleType** keyword or the end of the file. The format is:

```
RuleType type
```

Valid rule types are:

none

The rule applies to both hosts and user certificates.

host

The rule applies to host certificates only.

user

The rule applies to user certificates only.

user-address = *server*

The rule applies only to user certificates authenticating to the specified server. Note: When PKI Services Manager evaluates a user-address rule, it uses the server name (not the DNS host name) of the server the user is connecting to. The server sends its name to PKI Services Manager when it requests validation of a user certificate, and PKI Services Manager uses that name when applying the user-address rule. To determine the host name that is sent, you can enter the **hostname** command from a Windows DOS window or from a UNIX terminal session.

For example, to create rules that apply only to users connecting to the server acme:

```
RuleType user-address=acme
```

Note: Rule type determines the order in which rules are processed. The order for processing user certificates is: user-address, user, none. The order for processing host certificates is: host, none. Within each rule type, rules are processed in order from top to bottom.

Keywords

DynamicFile

Specifies whether PKI Services Manager reloads the map file every time it checks for allowed identities. The allowed values are 'yes' and 'no'. The default is 'no'.

ExternTimeout

Sets the timeout for rules that use the **Extern** option.

RuleType

Marks the beginning of a rule type stanza, which can be used to apply different mapping criteria based on whether a user or host presents the certificate. The allowed values are 'user', 'host', 'none', and 'user-address = *server*'. The default is 'none'.

APPENDIX U

Sample Mapping Rules

Rule	What happens
<pre>{ guest }</pre>	Because no condition is included, all valid certificates are mapped to the user "guest". This can serve as a default rule. A rule like this should go at the end of the rule list to ensure that all other rules are processed first.
<pre>{ fred.jones } UPN.user Equals "fred"</pre>	If the UPN representation of SubjectAltName is present, and the user part is equal to "fred", the set of allowed identities is fred.jones.
<pre>{ %UPN.user% } UPN.host Equals "acme.com"</pre>	If a certificate has a UPN representation of SubjectAltName, and the host name part is "acme.com", the user name part of the UPN is returned as the set of allowed identities.
<pre>{ guest %UPN.user% }</pre>	If the UPN is set, the user part is included in the set of allowed identities (along with "guest"). Otherwise the set of allowed identities is "guest". Because there is no condition, this rule applies to any valid certificate.
<pre>{ fred root } Subject.CN Contains "Fred Jones"</pre>	If the CN of the certificate contains "Fred Jones", the set of allowed identities has two values: "fred" and "root".
<pre>{ %subst% } Subject.CN Regex [a-zA-Z\.\.]*([0-9])</pre>	Sets the allowed identity equal to the first numerical string within the common name portion of the Subject field. For example, if the CN is "joe.smith.12345", the allowed identity is set to "12345".
<pre>{ elmer.foo.com } Subject.CN Contains "elmer"</pre>	Sets the allowed identity to the fully-qualified domain name "elmer.foo.com" from a certificate that contains the short name "elmer".

Rule	What happens
{ bob } Cert Equals /temp/certs/bob_cert.crt	Compares the incoming certificate to the one locally stored. If they are equal, the allowed identity set is "bob".
{ %subst% } Subject.CN Extern /bin/myapp	PKI Services Manager sends the Common Name portion of the Subject field to the application "/bin/myapp". If the exit code of the called application equals 0, the allowed identity is set equal to the returned result.
{ %UPN.User% } UPN Extern /bin/ldap-app	In this case, an exit-code of 0 from the external application serves as confirmation that the UPN is an authorized user.
{ %Subject.CN% %DNS% }	Sets the allowed identity set to include the contents of either the Subject.CN field or the DNS part of SubjectAltName.
{ windomain\%UPN.User% }	Allows users from the specified Windows domain name to authenticate if their user name matches the UPN user name.

APPENDIX V

Sample Map File with RuleType Stanzas

```
RuleType user
# the following rules are evaluated for user certificates only:
{ scott } CN Contains acme
{ joe } CN Equals acme
{ guest }

RuleType host
# The following rule is evaluated for host certificates only:
{ elmer.acme } CN Contains elmer

RuleType user-address=myserver
# The following rule is evaluated only when myserver
# requests validation of a user certificate:
{ good %subst% } Regex UPN "([A-Za-z0-9\.-])@[*.]"

RuleType none
# "none" is the default if no RuleType is specified.
# If no rule is successfully applied from "user" or "host",
# this rule is evaluated.
{ good } SerialandIssuer contains 123 CN=foo
```

APPENDIX W

PKI Settings Migration

Review the information below if you configured certificate authentication using Reflection for Secure IT 6.x or F-Secure. Some certificate settings continue to be supported in Reflection for Secure IT UNIX Client and Server settings files. Others need to be migrated to the Reflection PKI Services Manager settings file. You can use the **pkid** command with the **-m** option to migrate settings from Reflection for Secure IT 6.x or F-Secure settings files.

Note: For details about the **-m** option, refer to the **pkid command reference** (page [193](#)).

The following tables summarize how prior versions settings are handled. The entries under **Status** describe the effect of prior version keywords in your current version settings files. These entries have the following meanings:

- **Supported:** The keyword has the same meaning as it did in prior versions.
- **Deprecated:** The keyword continues to have an effect, but it's meaning may have changed.
- **Ignored:** The keyword has no effect in current Reflection for Secure IT settings file. These settings need to be migrated to PKI Services Manager settings files. Refer to the migration log for additional information.
- **Not supported:** The keyword cannot be used in current version settings files. It has no meaning and causes an error if present.

Client Settings

Prior version keyword	Status	Migrated?	Equivalent PKI Services Manager keyword
HostCA	Deprecated	Yes	TrustAnchor
HostCANoCRLs	Deprecated	Yes	TrustAnchor RevocationCheckOrder = none
HostCertNameCheck	Supported	No	--
LDAPServers	Ignored	Yes	CertServers CRLServers (All servers are migrated to both keywords)
LocalPKI	Ignored	Yes	LocalStore

Prior version keyword	Status	Migrated?	Equivalent PKI Services Manager keyword
OCSPResponder	Ignored	Yes	OCSPResponders
RevocationChecks	Ignored	Yes	RevocationCheckOrder
RevocationCA	Ignored	Yes	OcspCertificate

Server Settings

Prior version keyword	Status	Migrated?	Equivalent PKI Services Manager keyword
HostCA	Deprecated	Yes	TrustAnchor
HostCANoCRLs	Deprecated	Yes	TrustAnchor RevocationCheckOrder = none
HostCertificateFile	Supported	No	--
DynamicMapFile	Ignored	Yes	DynamicFile (This keyword is configured in pki_mapfile.)
ExternalMapper	Ignored	Yes	Supported in map file rules by using the Extern option in the conditional expression.
ExternalMapperTimeout	Ignored	Yes	ExternTimeout (This keyword is configured in pki_mapfile.)
LDAPServers	Ignored	Yes	CertServers CRLServers (All servers are migrated to both keywords)
LocalPKI	Ignored	Yes	LocalStore
OCSPResponder	Ignored	Yes	OCSPResponders
RevocationChecks	Ignored	Yes	RevocationCheckOrder
RevocationCA	Ignored	Yes	OcspCertificate
MapFile	Ignored	Yes	MapFile
OcspMode	Ignored	Yes	RevocationCheckOrder
PKI	Ignored	Yes	TrustAnchor
PkiDisableCrls	Ignored	Yes	RevocationCheckOrder =none

Prior version keyword	Status	Migrated?	Equivalent PKI Services Manager keyword
PkiIgnoreBasicConstraints	Ignored	Yes	StrictMode
SocksServer	Not supported	No	--

APPENDIX X

PKI Services Manager Return Codes

Reflection PKI Services Manager returns the following codes to the application requesting validation services.

- Code 0 = No errors, successful validation.
- Codes 1-10 = Command-line errors, either with **winpki** or **pkid**.
- Codes 11-19 = Network or protocol errors.
- Codes 21-29 = Validation errors.
- Codes 31-39 = Mapper errors (certificate is valid but could not be mapped).
- Codes 41-49 = CRL or other revocation errors

Code	Meaning
0	No errors.
1	General error, unknown cause.
2	Syntax error with the command, improper arguments.
3	PKI Services Manager is already running.
4	Error in the configuration file.
5	Timeout occurred while executing the command.
6	Network error (for example, cannot connect to PKI Services Manager).
7	Access denied, user does not have permission to run the command.
8	System error . This is an internal error. Re-run with <code>-d</code> switch to see what happened.
9	Migration or initialization failed. See migration error log.
11	Unknown command was requested by the calling application.
12	An exception was thrown by PKI Services Manager. For more information, see the PKI Services Manager event log.
13	Syntax error with the command or packet sent to PKI Services Manager.
14	Command was ignored (not currently used, internal error).

Code	Meaning
0	No errors.
15	Processing error. The certificate sent to PKI Services Manager is not encoded correctly.
16	Command failed (commands are: stop, reload, reconfigure).
17	Signature mismatch. Sender did not sign with a matching key.
18	Format error. The ASN protocol was not properly formatted
19	PKI Services Manager is in FIPS mode and the certificate is not valid in that mode
21	Certificate is invalid (expired, not signed, bad key, etc.)
22	No path. The issuing certificate could not be located.
23	Certificate is revoked.
24	No trust anchor. The path did not terminate to a known trust anchor.
25	Other validation error. Policy or other constraints failed.
26	Path length to the end certificate exceeded the CA path length constraint.
27	Certificate policy is invalid or does not match assertions in effect.
28	Certificate's signature does not match.
29	Unknown critical extension was encountered in a certificate or CRL.
31	Identity requested did not match allowed identities.
32	No identities are allowed for this certificate (no maps exist that match).
33	Calling application did not send an identity for matching (client-side error).
34	Certificate is valid, but requested WhoAmI processing
41	Unknown CRL processing error
42	No base for a delta CRL.
43	CRL has expired.
44	Cannot verify signature or it is bad.
45	Unknown CRL extension that is marked critical.

Code	Meaning
0	No errors.
46	Mismatch of IDP field in CRL.
47	No CRL available.

Glossary of Terms

A

authentication

The process of reliably determining the identity of a communicating party. Identity can be proven by something you know (such as a password), something you have (such as a private key or token), or something intrinsic about you (such as a fingerprint).

B

bandwidth

The rate of transmission of data across the network; the maximum amount of information (Kbits/second or Mbits/second) that can be transmitted along a channel.

C

cipher

A cipher is an encryption algorithm. The cipher you select determines which mathematical algorithm is used to obscure the data being sent after a successful Secure Shell connection has been established.

D

data integrity

The assurance that data has not been changed from its original source. Methods to preserve data integrity are designed to ensure that data has not been accidentally or maliciously modified, altered or destroyed.

digital signature

Used to confirm the authenticity and integrity of a transmitted message. Typically, the sender holds the private key of a public/private key pair and the recipient holds the public key. To create the signature, the sender computes a hash from the message, and then encrypts this value with its private key. The recipient decrypts the signature using the sender's public key, and independently computes the hash of the received message. If the decrypted and calculated values match, the recipient trusts that the sender holds the private key, and that the message has not been altered in transit.

E

encryption

Encryption is the process of scrambling data by use of a secret code or cipher so it is unreadable except by authorized users. Encrypted data is far more secure than unencrypted data.

G

GSSAPI (Generic Security Services Application Program Interface)

An application programming interface that provides programs with access to security services.

H

hash

Also called a message digest, a hash or hash value is a fixed-length number generated from variable-length digital data. The hash is substantially smaller than the original data, and is generated by a formula in such a way that it is statistically unlikely that some other data will produce the same hash value.

K

Kerberos

A protocol that uses a trusted third party to enable secure communications over a TCP/IP network. The protocol uses encrypted tickets rather than plain-text passwords for secure network authentication.

L

latency

The time delay between when an action is initiated and when its effect is detectable. In a network, a delay in the reception of data packets can be caused by several factors, such as the transmission medium, and the number of network devices between the sending and receiving points. In general, the greater the physical distance between your workstation and your X client host, the greater the chance of encountering latency.

M**MAC (Message Authentication Code)**

Used to verify that data is not changed in transit, a MAC is a hash created using an arbitrary-length packet of data and a shared secret key. The sending and receiving party compute the MAC independently for each packet of transferred data using the shared key and an agreed-upon algorithm. If the message has changed in transit, the hash values are different and the packet is rejected.

P**passphrase**

A passphrase is similar to a password, except it can be a phrase with a series of words, punctuation, numbers, white space, or any string of characters. Passphrases improve security by limiting access to secure objects, such as private keys and/or a key agent.

PKCS

PKCS (Public Key Cryptography Standards) is a set of standards devised and published by RSA laboratories that enable compatibility among public key cryptography implementations. Different PKCS standards identify specifications for particular cryptographic uses. Reflection for Secure IT UNIX Client and Server uses the following PKCS standards:

- PKCS#7 can be used to sign and/or encrypt messages. It can also be used to store certificates and to disseminate certificates (for instance as a response to a PKCS#10 message). You can use **ssh-keygen** to extract certificates from a PKCS#7 file.
- PKCS#10 is used for certificate requests to a Certificate Authority (CA). You can use **ssh-certtool** to create PKCS#10 files.
- PKCS#12 is used for storage and transportation of certificates and associated private keys. Files in this format typically use a *.pfx or *.p12 extension. Reflection for Secure IT supports authentication using certificates and keys stored in this format.

port forwarding

A way to redirect unsecured traffic through a secure SSH tunnel. Two types of port forwarding are available: local and remote. Local (also called outgoing) port forwarding sends outgoing data sent from a specified local port through the secure channel to a specified remote port. You can configure a client application to exchange data securely with a server by configuring the client to connect to the redirected port instead of directly to the computer running the associated server. Remote (also called incoming) port forwarding sends incoming data from a specified remote port through the secure channel to a specified local port.

public key/private key

Public keys and private keys are pairs of cryptographic keys that are used to encrypt or decrypt data. Data encrypted with the public key can only be decrypted with the private key; and data encrypted with the private key can only be decrypted with the public key.

R

regular expression

Often abbreviated as *regex*, a regular expression is a string of characters that describes one or more matching strings. Within a regular expression, some characters have a predefined meaning that determines what qualifies as a match. For example, the regular expression "t.*t" matches any word that starts and ends in the letter *t*, while the regular expression "text" matches only itself.

S

Secure Shell

A protocol for securely logging onto a remote computer and executing commands. It provides a secure alternative to Telnet, FTP, rlogin, or rsh. Secure Shell connections require both server and user authentication, and all communications pass between hosts over an encrypted communication channel. You can also use Secure Shell connections to forward X11 sessions or specified TCP/IP ports through the secure tunnel.

socket

The combination of a host name (IP address or DNS name) and a port number. This creates a unique identifier that a client application uses as an end point of communications.

T

trust anchor

A certificate that can be used as the final trust point in a certificate chain of trust. Note: PKI Services Manager validates certificates using only those trust anchors that have been explicitly configured for use by PKI Services Manager. You can configure a trust anchor using a root CA certificate, an intermediate CA certificate, or a self-signed certificates (one which can only validate itself).

Index

A

- access control
 - access control settings • 95
 - client access • 98
 - directory and file permissions • 151
 - group access • 98
 - subconfiguration files • 30
 - user access • 97
 - using allow and deny keywords • 96
- auditing
 - message logs • 103
 - Solaris (BSM) • 105
- authentication
 - client • 51
 - server • 37

B

- basics
 - getting started • 19
 - understanding secure shell • 24

C

- certificate authentication
 - about server authentication • 41
 - about user authentication • 57
 - configure server authentication • 44
 - configure user authentication • 59
 - obtain authentication certificates • 42
- certificates
 - ssh-certtool • 189
 - ssh-certview • 187
- cipher
 - configure • 34
 - defined • 219
- client
 - authentication methods • 51
 - client configuration files • 27
 - client file list • 112
 - keywords • 118
 - make a client connection • 21

- client host access
 - configure client host access • 98
 - subconfiguration files • 30
- configuration files
 - client configuration files • 27
 - host stanzas • 28
 - server configuration files • 29
 - subconfiguration files • 30

D

- debugging
 - client debugging • 101
 - server debugging • 102
- directory and file permissions • 151

E

- encryption
 - ciphers and macs • 34
 - data encryption • 33

F

- file transfer
 - resume • 76
 - scp • 75
 - secure file transfer • 71
 - sftp • 71
 - sftp batch files • 73
 - sftp command list • 176
 - troubleshooting slow transfer • 109
 - use sftp interactively • 72
- files
 - client file list • 112
 - recommended and required permissions • 151
 - server file list • 114
- fingerprint
 - display host key fingerprint • 41
 - unknown host key prompt • 21
- FIPS Mode • 35
- forwarding
 - configure • 89
 - introduction • 83
 - local • 84
 - remote • 87
 - settings • 92
 - X protocol • 91

G

group access • 98

GSSAPI

configure Kerberos authentication
• 48

Kerberos authentication • 47

system requirements • 47

H

host access

configure client host access • 98

subconfiguration files • 30

host key

add to client known hosts list • 39

create a new host key • 39

display host key fingerprint • 41

migration of existing key • 7

unknown host key prompt • 21

host stanzas • 28

HP-UX Trusted Systems • 69

I

installation

HP-UX • 14

IBM AIX • 15

installed features • 7

Linux • 10

replace existing Secure Shell • 9

Sun Solaris • 12

system requirements • 7

K

Kerberos (Secure Shell connections)

configure Kerberos authentication
• 48

Kerberos authentication • 47

system requirements • 47

key agent

ssh-add command line options •
181

ssh-agent command line options •
183

using the key agent • 56

key authentication

configure public key
authentication • 54

create a new host key • 39

overview • 37

troubleshooting • 107

unknown host key prompt • 21

keywords

client • 118

server • 132

known hosts

add a key to client • 39

L

local port forwarding

configure • 89

overview • 84

settings • 92

logging

client debugging • 101

server auditing • 103

server debugging • 102

M

MACS

configure • 34

defined • 221

MaxConnections • 95

migration

migrate settings • 16

migration of existing key • 7

P

PAM

configure PAM • 63

introduction • 62

passwords

configure password authentication
• 52

permissions

protecting files and directories •
151

PKI

configure server authentication •
44

configure user authentication • 59

install PKI Services Manager • 17

pki_config configuration file
reference • 197

pki_map map file reference • 203

pkid command reference • 193

port forwarding

configure • 89

- introduction • 83
- local • 84
- remote • 87
- settings • 92
- X protocol • 91
- public key authentication
 - configure public key authentication • 54
 - create a new host key • 39
 - overview • 37
 - troubleshooting • 107
 - unknown host key prompt • 21

R

- remote port forwarding
 - configure • 89
 - overview • 87
 - settings • 92
- resume • 76
- RSA SecureID authentication
 - configure • 67
 - overview • 67

S

- scp
 - command line options • 167
 - getting started • 23
 - overview • 75
 - resume • 76
- Secure Shell
 - understanding Secure Shell • 24
- SecurID authentication
 - configure • 67
 - overview • 67
- server
 - authentication • 37
 - keywords • 132
 - server configuration files • 29
 - server file list • 114
 - start and stop • 19
 - subconfiguration files • 30
- sftp
 - command line options • 172
 - getting started • 23
 - overview • 71
 - resume • 76
 - sftp batch files • 73
 - sftp command list • 176

- use sftp interactively • 72
- Solaris
 - auditing (BSM) • 105
- ssh
 - command line options • 154
 - escape sequences • 161
 - exit values • 162
- ssh2_config
 - client configuration files • 27
 - client configuration keywords • 118
 - host stanzas • 28
- ssh-add command line options • 181
- ssh-agent
 - ssh-agent command line options • 183
 - using the key agent • 56
- ssh-certtool
 - command reference • 189
 - create certificate requests • 42
- ssh-certview • 187
- sshd2_config
 - server configuration files • 29
 - server configuration keywords • 132
 - subconfiguration files • 30
- ssh-keygen
 - command line options • 163
 - create a new host key • 39
 - display host key fingerprint • 41
- StrictModes
 - file and directory permissions • 151
- subconfiguration files • 30
- syntax
 - command line • 29
 - configuration file • 28
- system requirements • 7

T

- troubleshooting
 - client debugging • 101
 - public key authentication • 107
 - server debugging • 102
 - slow transfer • 109
- Trusted Systems on HP-UX • 69
- tunneling
 - configure • 89
 - introduction • 83

- local • 84
- remote • 87
- settings • 92
- X protocol • 91

U

uninstall

- HP-UX • 14
- IBM AIX • 15
- Linux • 10
- Sun Solaris • 12

upgrade

- migrate settings • 16
- replace existing Secure Shell • 9

user access

- subconfiguration files • 30
- user access keywords • 97

X

X protocol forwarding

- overview • 91
- settings • 92