SSH Secure Shell for Servers Version 3.2.9 Administrator's Guide

September 2003

© 1996 - 2003 SSH Communications Security Corp.

No part of this publication may be reproduced, published, stored in an electronic database, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, for any purpose, without the prior written permission of SSH Communications Security Corp. This software is protected by international copyright laws. All rights reserved. ssh® is a registered trademark of SSH Communications Security Corp. in the United States and in certain other jurisdictions. SSH2, the SSH logo, SSH IPSEC Express, SSH Certifier, SSH Sentinel, SSH NAT Traversal, and Making the Internet Secure are trademarks of SSH Communications Security Corp. and may be registered in certain jurisdictions. All other names and marks are property of their respective owners. THERE IS NO WARRANTY OF ANY KIND FOR THE ACCURACY OR USEFULNESS OF THIS INFORMATION EXCEPT AS REQUIRED BY APPLICABLE LAW OR EXPRESSLY AGREED IN WRITING.

SSH Communications Security Corp.
Fredrikinkatu 42; FIN-00100 Helsinki; FINLAND
SSH Communications Security Inc.
1076 East Meadow Circle; Palo Alto, CA 94303; USA
SSH Communications Security K.K.
House Hamamatsu-cho Bldg. 5F; 2-7-1 Hamamatsu-cho, Minato-ku; Tokyo 105-0013, JAPAN

http://www.ssh.com/
e-mail: ssh-sales@ssh.com (sales), http://www.ssh.com/support/
Tel: +358 20 5007030 (Finland), +1 650 2512700 (USA), +81 3 34596830 (Japan)
Fax: +358 20 5007031 (Finland), +1 650 2512701 (USA), +81 3 34596825 (Japan)

Contents

1 About This Document **9**

1.1 Available Manual Pages 9

1.2 Documentation Conventions 10

2 Introduction to SSH Secure Shell **11**

2.1 Network Security Risks 11

2.2 Protocol Features 12

2.3 Different Versions of the SSH Protocol 13

2.4 Legal Issues with Encryption 13

2.5 Supported Platforms 13

2.6 Support 14

3 Configuring SSH Secure Shell **15**

3.1 Basic Configuration 15

3.1.1 Default Locations of Secure Shell Files 15

3.1.2 Generating the Host Key 16

3.1.3 Ciphers and MACs 17

3.1.4 Compression 19

3.1.5 Configuring Root Logins 20

3.1.6 Restricting User Logins 20

 © 2003 SSH Communications Security Corp.

3.2 Subconfigurations 22
3.2.1 Host-Specific Subconfiguration 22
3.2.2 User-Specific Subconfiguration 23
3.3 Configuring SSH Secure Shell for TCP Wrappers Support 23
3.3.1 Troubleshooting TCP Wrappers 25
3.4 Configuring SSH2 for SSH1 Compatibility 25
3.5 Forwarding 26
3.5.1 Port Forwarding 26
3.5.2 Dynamic Port Forwarding 29
3.5.3 X11 Forwarding 29
3.5.4 Agent Forwarding 30

4 Authentication 31

4.1 Server Authentication 31
4.1.1 Public-Key Authentication 31
4.1.2 Certificate Authentication 33
4.2 User Authentication 35
4.2.1 Password Authentication 35
4.2.2 Public-Key Authentication 36
4.2.3 Host-Based Authentication 39
4.2.4 Certificate Authentication 43
4.2.5 Kerberos Authentication 47
4.2.6 Pluggable Authentication Module (PAM) 48
4.2.7 SecurID 49
4.3 Keyboard-Interactive Authentication 50
4.3.1 Overview 51
4.3.2 Configuring the Server and Client 51

4.3.3 Adding New Authentication Methods 52

5 Log Messages 57

5.1 Log Message Reference 57

5.1.1 User Authentication - Common 57

5.1.2 User Authentication - Host-Based 59

5.1.3 User Authentication - Keyboard-Interactive PAM 63

5.1.4 User Authentication - Keyboard-Interactive Password 65

5.1.5 User Authentication - Keyboard-Interactive 66

5.1.6 User Authentication - Kerberos TGT 66

5.1.7 User Authentication - Kerberos 67

5.1.8 User Authentication - PAM 67

5.1.9 User Authentication - Password 69

5.1.10 User Authentication - Public Key 72

5.1.11 User Authentication - SecurID 75

5.1.12 Certificate-specific code 75

5.1.13 SFTP Server 75

5.1.14 Agent Forwarding 76

5.1.15 Session Channels 77

5.1.16 SSH1 Agent Forwarding 79

5.1.17 Port Forwarding 80

5.1.18 X11 Forwarding 82

5.1.19 Common Code 82

5.1.20 Host Key I/O 83

5.1.21 General Server Log Messages 83

5.1.22 SFTP 89

 © 2003 SSH Communications Security Corp.

6 Using SSH Secure Shell **99**

6.1 Using the Secure Shell Server Daemon (`sshd2`) 99

6.1.1 Manually Starting the Secure Shell Server Daemon 99

6.1.2 Automatically Starting the Server Daemon at Boot Time 100

6.1.3 Operation of the Server Daemon 101

6.1.4 Resetting and Stopping the Server Daemon 102

6.1.5 Daemon Configuration File and Command-Line Options 102

6.1.6 Subsystems 102

6.2 Using the Secure Shell Client (`ssh2`) 103

6.2.1 Starting the Secure Shell Client 103

6.2.2 Client Configuration File and Command-Line Options 104

6.3 Using Secure Copy (`scp2`) 104

6.4 Using Secure File Transfer (`sftp2`) 104

6.5 Using Authentication Agent (`ssh-agent2`, `ssh-add2`) 105

A Tool Syntax **107**

A.1 `ssh-keygen2` 107

A.2 `ssh-certenroll2` 109

B Technical Specifications **111**

B.1 Supported Cryptographic Algorithms and Standards 111

B.1.1 Public-Key Algorithms 111

B.1.2 Data Integrity Algorithms 111

B.1.3 Symmetric Session Encryption Algorithms 111

B.1.4 Certificate Standards 112

B.1.5 Certificate and CRL Publishing Solutions 112

B.1.6 Cryptographic Token Standards 112

B.1.7 Other Supported Standards 113

B.1.8 Additional Information 113

B.2 Authentication Methods 113

 © 2003 SSH Communications Security Corp.

Chapter 1

About This Document

This document, *SSH Secure Shell for Servers Administrator's Guide*, contains instructions on the basic administrative tasks of SSH Secure Shell. The document is intended for the system administrators responsible for the configuration of SSH Secure Shell software.

To use the information presented in this document, you should be familiar with UNIX system administration.

The document contains the following information:

- introduction to SSH Secure Shell
- configuration options
- authentication options
- keyboard-interactive authentication options
- using SSH Secure Shell
- appendix (list of supported standards and authentication methods)

Installation instructions for SSH Secure Shell software can be found in *SSH Secure Shell for Servers Quick Start Guide*, included in the CD-ROM package and available on the SSH Web site (`http://www.ssh.com/support/`). For more information, see the manual pages included in the distribution.

1.1 Available Manual Pages

The following manual pages are included in the distribution:

- `ssh2`: secure shell client (remote login program)

 © 2003 SSH Communications Security Corp.

- `scp2`: secure copy client
- `sftp2`: secure ftp client
- `ssh-add2`: adds identities for the authentication agent
- `ssh-agent2`: authentication agent
- `ssh-certenroll2`: certificate enrollment client (included only in the commercial distribution)
- `ssh-certificates`: describes the configuration files and options needed when using certificates with ssh2 software (included only in the commercial distribution)
- `ssh-dummy-shell`: ultimately restricted shell
- `ssh-externalkeys`: general documentation about using external keys with ssh2 software
- `ssh-keygen2`: authentication key pair generation
- `ssh-probe2`: seeks ssh servers from the local network
- `ssh2_config`: format of configuration file for ssh2
- `sshd-check-conf`: check what your configuration allows or denies based on incoming user and/or host name
- `sshd2`: secure shell daemon
- `sshd2_config`: format of configuration file for sshd2
- `sshd2_subconfig`: advanced configuration of sshd2
- `sshregex`: describes the regular expressions (or globbing patterns) used in filename globbing with scp2 and sftp2 and in the configuration files ssh2_config and sshd2_config

Please refer to the above mentioned man pages for command syntax descriptions and usage instructions.

1.2 Documentation Conventions

The following special conventions are used in this document:

Convention	Usage	Example
Bold	Menus, GUI elements, strong emphasis	Click **Apply** or **OK**.
`Monospace`	Filenames, commands, directories, URLs etc.	Refer to `readme.txt`.
Italics	Reference to other documents or products, emphasis	See *SSH Sentinel User Manual*.

Chapter 2

Introduction to SSH Secure Shell

SSH Secure Shell is a program that allows secure network services over an insecure network, such as the Internet.

The Secure Shell concept originated on UNIX as a replacement for the insecure ”Berkeley services”, that is, the `rsh` , `rcp`, and `rlogin` commands. SSH Secure Shell replaces other, insecure terminal applications (such as Telnet and FTP). It allows you to securely login to remote host computers, to execute commands safely in a remote computer, to securely copy remote files, to forward X11 sessions (on UNIX), and to provide secure encrypted and authenticated communications between two non-trusted hosts. Also arbitrary TCP/IP ports can be forwarded over the secure channel, enabling secure connection, for example, to an email service.

SSH Secure Shell with its array of unmatched security features is an essential tool in today’s network environment. It is a powerful guardian against the numerous security hazards that nowadays threaten network communications.

This chapter gives an overview of some of the security risks facing the Internet user and introduces the SSH2 protocol to combat these risks.

2.1 Network Security Risks

The open architecture of the Internet Protocol (IP) makes it a highly efficient, cost-effective, and flexible communications protocol for local and global communications. It has been widely adopted, not only on the global Internet, but also in the internal networks of large corporations.

The IP protocol suite, including TCP/IP, was designed to provide reliable and scalable communications over real-world networks. It has served this goal well. However, it was designed twenty years ago in a world where the Internet consisted of a few hundred closely controlled hosts. The situation has changed. The Internet now connects dozens of millions of computers, controlled by millions of individuals and organizations. The core network itself is administered by thousands of competing operators, and the network spans the whole globe, connected by fibers, leased lines, dial-up modems, and mobile phones.

 © 2003 SSH Communications Security Corp.

The phenomenal growth of the Internet has peaked the interest of businesses, military organizations, governments, and criminals. Suddenly, the network is changing the way business is done. It is changing the nature of trade and distribution networks and the way individual people communicate with each other.

The upsurge of business, scientific, and political communications on the Internet has also brought out the usual negative elements. Criminals are looking for ways of getting a cut of the emerging business. Industrial espionage has moved online. Intelligence agencies are showing a growing interest towards networked communications, and they often exchange information with domestic commercial interest and political groups. Crackers, exchanging information and source code, make attacks that ten years ago were thought to be within the reach of only the most powerful intelligence agencies.

It has turned out that the IP protocol, while very tolerant of random errors, is vulnerable to a number of malicious attacks. The most common types of attacks include:

- Eavesdropping a transmission, for example looking for passwords, credit card numbers, or business secrets.
- Hijacking, or taking over a communication in such a way that the attacker can inspect and modify any data being transmitted between the communicating parties.
- IP spoofing, or faking network addresses in order to fool access control mechanisms based on them or to redirect connections to a fake server.

The SSH2 protocol is designed to protect network communications against security hazards like these.

2.2 Protocol Features

The SSH Secure Shell products from SSH Communications Security use the Secure Shell version 2 (SSH2) protocol. The SSH2 protocol is being standardized in the Secsh Working Group of the Internet Engineering Task Force (IETF).

The SSH2 protocol contains the following features:

- Secure terminal sessions utilizing secure encryption.
- Full, secure replacement for FTP and Telnet, as well as the UNIX r-series of commands: `rlogin`, `rcp`, `rexec`.
- Multiple high security algorithms and strong authentication methods that prevent such security threats as identity spoofing.
- Multiple ciphers for encryption, including 3DES, Blowfish, and AES.
- Transparent and automatic tunneling of X11 connections and arbitrary TCP/IP-based applications, such as e-mail.

- Automatic and secure authentication of both ends of the connection. Both the server and the client are authenticated to prevent identity spoofing, Trojan horses, and so on.
- Multiple channels that allow multiple terminal windows and file transfers through one secure and authenticated connection.

2.3 Different Versions of the SSH Protocol

The current version of the SSH protocol is version 2 (SSH2). More information on the protocol can be found in the IETF-secsh Internet-Drafts (`http://www.ietf.org/ID.html`).

Note: SSH Communications Security considers the Secure Shell protocol version 1 (SSH1) deprecated and does not recommend the use of it. The SSH statement regarding the vulnerability of SSH1 protocol is available at `http://www.ssh.com/company/newsroom/article/210/`.

Several different versions of the Secure Shell client and server exist. Please note that the different versions may use different implementations of the ssh protocol, and therefore you may not be able to connect to an ssh1 server using ssh2 client software, or vice versa.

However, the SSH Secure Shell server software includes support for fallback functionality if SSH Secure Shell version 1.x is already installed. (SSH Secure Shell for Windows Servers does not include this functionality.) Furthermore, the SSH Secure Shell client now contains internal ssh1 emulation, allowing it to communicate with ssh1 servers without using an external ssh1 program.

For optimal results, however, upgrade all servers and clients to the latest available version of SSH Secure Shell.

Please note that the version number of the SSH Secure Shell software product does not reflect the version number of the secure shell protocol, but the version of the software.

2.4 Legal Issues with Encryption

The encryption software included in SSH Communications Security products is developed in Europe, and therefore these products are not subject to US export regulations.

SSH Secure Shell software can be used in any country that allows encryption, including the United States of America.

2.5 Supported Platforms

The SSH Secure Shell Server software (the software component that allows remote users to connect to your computer) is available for most UNIX and Linux platforms and for Microsoft Windows NT4 (Service Pack 5

 © 2003 SSH Communications Security Corp.

required), Windows 2000, and Windows XP.

The associated client software (the component that remote users run on their computers) is additionally available also for Microsoft Windows 98, Me and the Symbian operating systems.

A list of the officially supported platforms for SSH Secure Shell products is available at `http://www.ssh.com/products/`.

Independent third parties have also ported Secure Shell to other platforms such as OS/2 and VMS. These independent software products are intended to be compatible with the Secure Shell standard. However, SSH Communications Security can only provide support for software developed by SSH Communications Security.

2.6 Support

Commercial users are entitled to technical support from SSH Communications Security for ninety (90) days from the date of purchase of the software. If you have purchased a maintenance agreement, review your agreement for specific terms.

Commercial users and those evaluating the software prior to purchase can contact the SSH Secure Shell technical support by filling out a support request form at `http://www.ssh.com/support/contact/`.

- Commercial support requests `http://www.ssh.com/support/contact/support-request-shell.mpl`

Non-commercial licensees are welcome to submit bug reports and feature requests, but are not entitled to technical support from SSH Communications Security. Please note that SSH Communications Security makes every effort to make available information on issues that impact non-commercial licensees. See the SSH Web site (`http://www.ssh.com/support/`) for more information. Note, for example, the product FAQ pages.

- Bug Reports

  `http://www.ssh.com/support/contact/bug-report-shell.mpl`

- Feature Requests

  `http://www.ssh.com/support/contact/feature-request-shell.mpl`

Note: The above two links are for submissions only - you will not receive a response to emails sent using these forms.

Chapter 3

Configuring SSH Secure Shell

This chapter gives instructions on configuring SSH Secure Shell.

3.1 Basic Configuration

This section covers some basic configuration options that are commonly modified.

3.1.1 Default Locations of Secure Shell Files

The system-wide configuration files are the most important. They are located in the directory `/etc/ssh2`. User and system binaries are stored in `/usr/local/bin` and `/usr/local/sbin`, respectively. In `/usr/local/sbin`, you will find `sshd2`. All other binaries are stored in `/usr/local/bin`.

System-Wide Configuration Files

The system-wide configuration files for the client and server, respectively, consist of the following files:

- `/etc/ssh2/ssh2_config`
- `/etc/ssh2/sshd2_config`

Example files `sshd2_config.example` and `ssh2_config.example` can be found in the same directory.

Users can have their own configuration files. These are stored in `~/.ssh2`.

 © 2003 SSH Communications Security Corp.

Key Files

The system public key pair (DSS only) consists of the following files:

- `/etc/ssh2/hostkey`
- `/etc/ssh2/hostkey.pub`

Host keys that are recognized for all users on the local system should be placed in the `/etc/ssh2/hostkeys` directory.

User-specific host keys should be placed in the `~/.ssh2/hostkeys` directory.

If you are using host-based authentication, the system-wide file for recognized host keys is `/etc/ssh2/knownhosts`.

User-specific known host keys should be located in `~/.ssh2/knownhosts`.

License File

Commercial packages of SSH Secure Shell use the license file `license_ssh2.dat`. Non-commercial packages do not include this file.

To verify the type of the SSH Secure Shell package you are using, display version information with the following command:

```
ssh2 -V
```

When you acquire a license for an evaluation version of the software, the appropriate license file must be copied in the correct location. The license file is typically stored in the `/etc/ssh2/` directory, but a different location can be specified by setting the environment variable `SSH_LICENSE_FILE`.

The license file specifies the type of the license and indicates that the software has been legally purchased. The file should be safely stored.

3.1.2 Generating the Host Key

Host keys are generated during the installation of SSH Secure Shell. You only need to regenerate them if you want to change your host key, or if your host key was not generated during the installation.

To generate the host key, perform the following tasks:

1. Login as `root`.

2. Kill the sshd2 daemon listening to port 22:

   ```
   kill 'cat /var/run/sshd2_22.pid'
   ```

   If the directory `/var/run` does not exist, `sshd2_22.pid` is in directory `/etc/ssh2/`.

3. Generate the host key with the following command:

   ```
   ssh-keygen2 -P /etc/ssh2/hostkey
   ```

   Note: This will generate a DSA 2048-bit host key pair (without a passphrase). For more information on the key generation options, see the `ssh-keygen2` man page.

4. Restart `sshd2`:

   ```
   /usr/local/sbin/sshd2
   ```

Note: Administrators that have other users connecting to their sshd2 daemon should notify the users of the host-key change. If you do not, the users will receive a warning the next time they connect, because the host key the users have saved on their disk for your server does not match the host key now being provided by your sshd2 daemon. The users may not know how to respond to this error. You can run the following to generate a fingerprint for your new public host key which you can provide to your users via some unalterable method (such as digitally signed email):

```
ssh-keygen2 -F hostkey.pub
```

When the users connect and receive the error message about the host key having changed, they can compare the fingerprint of the new key with the fingerprint you have provided in your email, and ensure that they are connecting to the correct sshd2 daemon. Inform your users to notify you if the fingerprints do not match, or if they receive a message about a host-key change and do not receive a corresponding message from you notifying them of the change.

This procedure can help ensure that you do not become a victim of a man-in-the-middle attack, as your users will notify you if the host-key fingerprints do not match. You will also be aware if the users encounter host-key change messages when you have not regenerated your host key pair.

It is also possible to send the public host key to the users via an unalterable method. The users can save the key to the `~/.ssh2/hostkeys` directory as `key_22_machinename.pub`. In this case, manual fingerprint check is not needed.

3.1.3 Ciphers and MACs

The algorithm(s) used for symmetric session encryption can be chosen in the `sshd2_config` and `ssh2_config` files:

```
Ciphers         twofish,blowfish
```

 © 2003 SSH Communications Security Corp.

The system will attempt to use the different encryption ciphers in the sequence specified on the line. Currently supported cipher names are the following:

- `des`
- `3des`
- `blowfish`
- `twofish`
- `cast`
- `arcfour`
- `aes`.

Of these ciphers, Blowfish and Twofish are especially suitable for file transfers.

Special values for this option are the following:

- `Any`: allows all the cipher values including `none`
- `AnyStd`: allows only standard ciphers and `none`
- `none`: forbids any use of encryption
- `AnyCipher`: allows any available cipher apart from the non-encrypting cipher mode `none`
- `AnyStdCipher`: the same as `AnyCipher`, but includes only those ciphers mentioned in *IETF-SecSH-draft* (excluding `none`). This is the default value.

The MAC (Message Authentication Code) algorithm(s) used for data integrity verification can be chosen in the `sshd2_config` and `ssh2_config` files:

```
MACs            hmac-sha1,hmac-md5
```

The system will attempt to use the different MAC algorithms in the sequence they are specified on the line. Supported MAC names are the following:

- `hmac-sha1`
- `hmac-sha1-96`
- `hmac-md5`
- `hmac-md5-96`
- `hmac-ripemd160`

- `hmac-ripemd160-96`.

Special values for this option are the following:

- `Any`: allows all the MAC values including `none`
- `AnyStd`: allows only standard MACs and `none`
- `none`: forbids any use of MACs
- `AnyMac`: allows any available MAC apart from `none`
- `AnyStdMac`: the same as `AnyMac`, but includes only those MACs mentioned in *IETF-SecSH-draft* (excluding `none`). This is the default value.

Both cipher and MAC can also be defined using command line arguments with `ssh2` and `scp2`:

```
$ scp2 -c twofish -m hmac-md5 foobar user@remote:./tmp
```

Note: Algorithm names are case sensitive.

3.1.4 Compression

SSH Secure Shell uses GNU ZLIB (LZ77) for compression. The ”zlib” compression is described in RFC 1950 and in RFC 1951.

By default, compression is disabled. Compression can be enabled in the `ssh2_config` file:

```
Compression     yes
```

Alternatively, compression can be enabled on the command line:

```
$ ssh2 +C username@remote
```

The client can request a compression level with a number after `+C` (from +C1 to +C9). In this case, the compression level is between the levels requested by the client and offered by the server. For example, if the server offers level 6 (the default) and the client asks for level 1, level 2 is used.

Compression is worth using if your connection is slow (for example a modem connection). The efficiency of the compression depends on the type of the file, and varies widely. It is close to 0% for already compressed files like zip and often 50% or even more for text files.

 © 2003 SSH Communications Security Corp.

3.1.5 Configuring Root Logins

If you want to permit someone to login directly to the *root* login account via ssh, you can define three methods of control in the `sshd2_config` file:

```
PermitRootLogin         no
```

This will disable all root logins. To enable root logins with any authentication method, use the following setting:

```
PermitRootLogin         yes
```

You can limit the authentication methods by using the following setting:

```
PermitRootLogin         nopwd
```

This allows root logins only when an authentication method other than password is used.

It is also possible to create a separate subconfiguration file for root. See Section 3.2 (Subconfigurations) for more information.

3.1.6 Restricting User Logins

By default, SSH Secure Shell does not impose any login restrictions in addition to those provided by the operating system. However, you can restrict connections based on host, user name, or group.

The restrictions are defined in the `sshd2_config` file using the following syntax:

```
keyword pattern
```

Note: All the patterns used in the examples below are in accordance with `SSH_REGEX_SYNTAX_EGREP`, which is the default regex syntax in SSH Secure Shell version 3.0 and above. However, the regex syntax can be chosen by using the metaconfig block in the beginning of `ssh2_config` and `sshd2_config`:

```
## SSH CONFIGURATION FILE FORMAT VERSION 1.1
## REGEX-SYNTAX egrep
## end of metaconfig
```

Possible values of `REGEX-SYNTAX` are `ssh`, `egrep`, `zsh_fileglob` and `traditional`. For more information, please see the `sshregex` man pages.

Previous versions of SSH Secure Shell always use `SSH_REGEX_SYNTAX_ZSH_FILEGLOB`.

Available keywords are the following:

- `AllowHosts`/`DenyHosts`

  Login is allowed/denied from hosts whose name matches one of the specified patterns.

  Example 1. Listing complete hostnames.

  ```
  AllowHosts      localhost, foobar\.com, friendly\.org
  ```

  This allows connections only from specified hosts.

  Example 2. Using patterns with hostnames.

  ```
  AllowHosts      h..s.\..*
  ```

  This pattern matches, for example, *house.foobar.com*, *house.com*, but not *house1.com*. Note that you have to input the string "`\.`" when you want to specify a literal dot.

  Example 3. Using patterns with IP-addresses.

  ```
  AllowHosts ([[:digit:]]{1\,3}\.){3}[[:digit:]]{1\,3}
  ```

  This pattern matches any IP address (*xxx.xxx.xxx.xxx*). However, some host's hostname could also match this pattern.

  Example 4 Using `\i`.

  ```
  AllowHosts      "\i192.*\.3"
  ```

  When `\i` is used in the beginning of a pattern, only the host IP addresses are used. The above pattern matches, for example, `192.0.0.3`.

- `AllowSHosts`/`DenySHosts`

  The `.shosts`, `.rhosts`, `/etc/shosts.equiv` and `/etc/hosts.equiv` entries are honored only for hosts whose name matches one of the specified patterns. It is recommended to use these keywords with host-based authentication.

- `AllowUsers`/`DenyUsers`

  Login is allowed/denied as users whose name matches one of the specified patterns.

  Example 1 Using complete user names

  ```
  DenyUsers       devil@evil\.org,warezdude,1337
  ```

  This denies login as *devil* when the connection is coming from *evil.org*. It also denies login (from all addresses) as *warezdude* and as user whose UID is *1337*.

  Example 2 Using patterns with user names

 © 2003 SSH Communications Security Corp.

```
AllowUsers      "sj*,s[:digit:]+,s(jl|amza)"
```

This pattern matches, for example, *sjj*, *sjjj*, *s1*, *s123*, and *samza* but not *s1x* or *slj*.

Example 3 Using `\i`.

```
AllowUsers      "sjl@\i192.*\.3"
```

This would allow login as user *sjl* from only those hosts whose IP address matches the specified pattern.

- `AllowGroups`/`DenyGroups`

  Login is allowed/denied when one of the groups the user belongs to matches one of the specified patterns.

  Example 1

```
AllowGroups     root,staff,users
```

For more information on keywords, please see the `sshd2_config` man pages.

3.2 Subconfigurations

You can also specify configuration options in so-called subconfiguration files, which have the same basic format as the main configuration file. The process forked to handle the user's connection reads these files. They are read at run-time, so if they are modified, it is not necessary to restart the server process.

If parsing of the subconfiguration files fails, the connection is terminated (for host-specific configuration), or access denied (for user-specific configuration) by the server.

The subconfiguration files are divided into two categories: host-specific and user-specific.

Subconfiguration files are very flexible and because of that, dangerous if the logic of the files is not carefully planned. **Note**: Host-specific subconfiguration files are always read before the user-specific subconfiguration files. See the example file `sshd2_config.example` and the host-specific and user-specific files in `/etc/ssh2/subconfig`.

3.2.1 Host-Specific Subconfiguration

The `HostSpecifiConfig` variable specifies host-specific subconfiguration files for sshd2. The syntax is the following:

```
HostSpecificConfig pattern subconfig-file
```

`pattern` will be used to match the client host as specified under `AllowHosts` (see the `sshd2_subconfig` man page). The file `subconfig-file` will then be read, and configuration data amended accordingly.

The file is read before any actual protocol transactions begin, and you can specify most of the options allowed in the main configuration file. You can specify more than one subconfiguration file, in which case the patterns are matched and the files read in the order specified. Later defined values of configuration options will either override or amend the previous value depending on which option it is. The effect of redefining an option is described in the documentation for that option. For example, setting `Ciphers` in the subconfiguration file will override the old value, but setting `AllowUsers` will amend the value.

For more information, please see the `sshd2_subconfig` and `sshd2_config` man pages.

3.2.2 User-Specific Subconfiguration

User-specific subconfiguration files are read when the client has stated the user name it is trying to log in with. At this point, the server will obtain additional information about the user: does the user exist, what is the user's UID, and what groups does the user belong to. With this information, the server can read the user-specific configuration files specified by `UserSpecifiConfig` in the main sshd2 configuration file. The syntax is the following:

```
UserSpecificConfig pattern subconfig-file
```

You can use patterns of the form:

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

where `user` is matched with the user name and UID, `group` is matched with the user's primary and secondary groups, both group name and GID, and `host` is matched as described under `AllowHosts` (see the `sshd2_subconfig` man page).

For example, the following would match any user in group "sftp" connecting from company.com:

```
.*%sftp@company.com
```

See the `sshd2_subconfig` man pages for more information.

3.3 Configuring SSH Secure Shell for TCP Wrappers Support

To enable usage of TCP Wrappers with SSH Secure Shell, perform the following operations:

1. If SSH Secure Shell was previously installed from binaries, you may want to uninstall it before continuing.
2. Compile the source code:

   ```
   ./configure --with-libwrap
   make
   ```

 © 2003 SSH Communications Security Corp.

Then, become root and run

```
make install
```

Note: If `configure` does not find `libwrap.a`, do the following:

- Locate `libwrap.a`
- Run `configure` again:

  ```
  make distclean
  ./configure --with-libwrap=/path_to_libwrap.a/
  ```

  Note: It is only necessary to specify the path to `libwrap.a` if the library and the include files are located in a non-standard directory, i.e. if the library has been compiled to a local directory, or has been installed to somewhere else than the default location.

3. Create or edit the `/etc/hosts.allow` and `/etc/hosts.deny` files.

   When a user tries to connect to the SSH Secure Shell server, the TCP wrapper daemon (`tcpd`) reads the `/etc/hosts.allow` file for a rule that matches the client's hostname or IP. If `/etc/hosts.allow` does not contain a rule allowing access, `tcpd` reads `/etc/hosts.deny` for a rule that would deny access. If neither of the files contains an accept or deny rule, access is granted by default.

   The syntax for the `/etc/hosts.allow` and `/etc/hosts.deny` files is as follows:

   ```
   daemon : client_hostname_or_IP
   ```

   The typical setup is to deny access to everyone listed in the `/etc/hosts.deny` file. (This example shows both ssh1 and ssh2.)

   ```
   sshd1: ALL
   sshd2: ALL
   sshdfwd-X11 : ALL
   ```

   or simply

   ```
   ALL: ALL
   ```

   And then allow access only to trusted clients in the `/etc/hosts.allow`:

   ```
   sshd1 : trusted_client_IP_or_hostname
   sshd2 : .ssh.com foo.bar.fi
   sshdfwd-X11 : .ssh.com foo.bar.fi
   ```

   Based on the `/etc/hosts.allow` file above, users coming from any host in the *ssh.com* domain or from the host *foo.bar.fi* are allowed to access.

3.3.1 Troubleshooting TCP Wrappers

If configuring TCP wrappers causes problems, please check the following:

1. Make sure that you are not experiencing any network connectivity problems.
2. Make sure that SSH Secure Shell server is running:

   ```
   kill -0 ‘cat /var/run/sshd2_22.pid‘
   ```

   or

   ```
   kill -0 ‘cat /etc/ssh2/sshd2_22.pid‘
   ```

   If you receive the message ”*No such process*”, restart the `sshd2` daemon.
3. Check your `/etc/hosts.allow` and `/etc/hosts.deny` files.
   - Ensure that the client’s IP address or host name is correct.
   - If you are using a host name, you must supply the fully qualified domain name.
4. If you changed something in the `sshd2_config` file, you need to `HUP` the `sshd2` daemon.
5. Run `tcpdchk` and `tcpdmatch`. These programs are used to analyze and report problems with your TCP Wrappers setup. Please see the man pages for more information on these commands.

3.4 Configuring SSH2 for SSH1 Compatibility

Note: SSH Communications Security considers the Secure Shell version 1 protocol deprecated and does not recommend the use of it. For more information, see `http://www.ssh.com/company/newsroom/article/210/`).

The SSH1 and SSH2 protocols are not compatible with each other. This inconvenience is necessary, since the SSH2 protocol includes remarkable security and performance enhancements that would not have been possible if protocol-level compatibility with SSH1 had been retained.

However, the current implementations of ssh2 and ssh1 software are designed so that both can be run on the same computer. This makes the transition from the old but well-established SSH1 protocol to the more secure and more flexible SSH2 protocol much easier. The ssh2 server daemon includes a fallback function that automatically invokes the ssh1 server when required. Furthermore, the SSH Secure Shell client now contains internal ssh1 emulation, allowing it to communicate with ssh1 servers without using an external ssh1 program.

With the `Ssh1Compatibility` option, sshd1 is executed when the client supports only SSH 1.x protocols. The argument must be ”yes” or ”no”. The default is ”no”, which means that you have to manually set ssh2 to use ssh1 even if ssh1 is installed.

 © 2003 SSH Communications Security Corp.

`Sshd1Path` Specifies the path to the sshd1 executable to be executed in SSH1 compatibility mode. The arguments for sshd2 are passed on to sshd1.

`Sshd1ConfigFile` specifies the alternate configuration file for sshd1 when sshd2 runs in compatibility mode. It is only used if sshd2 is executed with the `-f` command line option, otherwise the default sshd1 configuration file is used.

See the `sshd2_config` man page for more information.

To set up both ssh1 and ssh2 servers on the same Unix system, you should do the following:

1. Install the latest available version of ssh1, available on the SSH Communications Security FTP site (`ftp://ftp.ssh.com/pub/ssh`). As of this publication of this document, the latest version is `ssh-1.2.33`. SSH1 compatibility fallback requires version 1.2.26 or later.

2. Install ssh2.

3. If you previously had ssh1 installed, please make sure that the old `sshd` is no longer run at boot. Only `sshd2` should be run. If you have the ssh1 version of `sshd` running, you should kill the master daemon. You can find its process id in `/var/run/sshd.pid` or if the directory `/var/run` does not exist, in `/etc/ssh2/sshd2_22.pid`.

4. Make sure that `/usr/local/sbin/sshd2` is run automatically at boot. On most systems, you should add the command to start it to `/etc/rc.local` or under `/etc/rc.d`.

   - When you run `sshd2`, the ssh1 daemon should not be running. When using ssh2 with ssh1 compatibility, you should only run `sshd2`. It will then automatically start the ssh1 daemon as needed.

5. If you do not want to reboot, you should now manually run `/usr/local/sbin/sshd2` (or `/etc/rc.d/init.d/sshd2 start`).

3.5 Forwarding

The SSH2 connection protocol provides channels that can be used for a wide range of purposes. All of these channels are multiplexed into a single encrypted tunnel and can be used for forwarding (”tunneling”) arbitrary TCP/IP ports and X11 connections.

3.5.1 Port Forwarding

Port forwarding, or tunneling, is a way to forward otherwise insecure TCP traffic through SSH Secure Shell. You can secure for example POP3, SMTP and HTTP connections that would otherwise be insecure - see Figure 3.1 (Encrypted SSH2 tunnel).

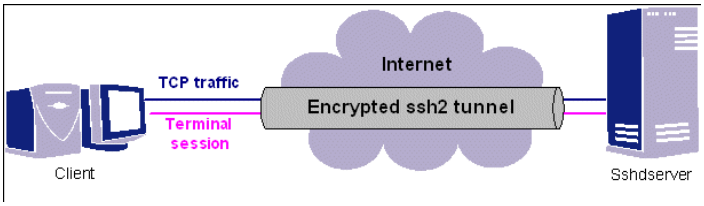

Figure 3.1: Encrypted SSH2 tunnel

The client-server applications using the tunnel will carry out their own authentication procedures, if any, the same way they would without the encrypted tunnel.

The protocol/application might only be able to connect to a fixed port number (e.g. IMAP 143). Otherwise any available port can be chosen for port forwarding.

Privileged ports (below 1024) can be forwarded only with root privileges.

There are two kinds of port forwarding: local and remote forwarding. They are also called outgoing and incoming tunnels, respectively. Local port forwarding forwards traffic coming to a local port to a specified remote port.

For example, if you issue the command

```
ssh2 -L 1234:localhost:23 username@host
```

all traffic coming to port 1234 on the client will be forwarded to port 23 on the server (`host`). Note that `localhost` will be resolved by the sshdserver after the connection is established. In this case `localhost` therefore refers to the server (`host`) itself.

Remote port forwarding does the opposite: it forwards traffic coming to a remote port to a specified local port.

For example, if you issue the command

```
ssh2 -R 1234:localhost:23 username@host
```

all traffic which comes to port 1234 on the server (`host`) will be forwarded to port 23 on the client (`localhost`).

It is important to realize that if you have three hosts, `client`, `sshdserver`, and `appserver`, and you forward the traffic coming to the `client`'s port `x` to the `appserver`'s port `y`, only the connection between the `client` and `sshdserver` will be secured. See Figure 3.2 (Forwarding to a third host). The command you use would be something like the following:

```
ssh2 -L x:appserver:y username@sshdserver
```

 © 2003 SSH Communications Security Corp.

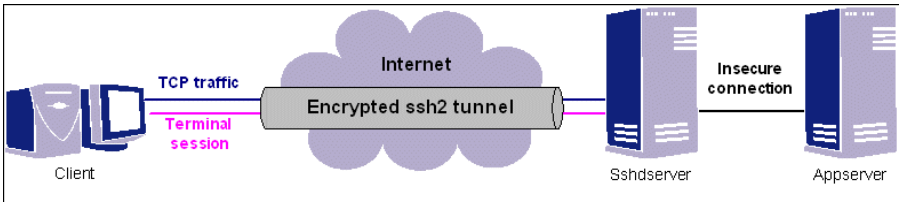

Figure 3.2: Forwarding to a third host

Forwarding FTP

With SSH Secure Shell version 3.0 and above it is possible to easily forward FTP connections by using a command with the following syntax:

```
ssh2 -L ftp/x:ftpdserver:y username@sshdserver
```

FTP forwarding is an extension to the generic port forwarding mechanism. The FTP control channel can be secured by using generic port forwarding, but since the FTP protocol requires creating separate TCP connections for the files to be transferred, all the files would be transferred unencrypted when using generic port forwarding, as these separate TCP connections would not be forwarded automatically.

To protect also the transferred files, use FTP forwarding instead. It works similarly to generic port forwarding, except that the FTP forwarding code monitors the forwarded FTP control channel and dynamically creates new port forwardings for the data channels as they are requested. To see exactly how this is done, two different cases need to be examined: the active mode and the passive mode of the FTP protocol.

FTP in passive mode

In passive mode, the FTP client sends the command 'PASV' to the server, which reacts by opening a listener port for the data channel and sending the IP address and port number of the listener as a reply to the client. The reply is of the form '227 Entering Passive Mode (10,1,60,99,6,12)'.

When the Secure Shell client notices the reply to the PASV command, it will create a local port forwarding to the destination mentioned in the reply. After this the client will rewrite the IP address and port in the reply to point to the listener of the newly created local port forwarding (which exists always in a local host address, 127.0.0.1) and pass the reply to the FTP client. The FTP client will open a data channel based on the reply, effectively tunneling the data through the SSH connection, to the listener the FTP server has opened. The net effect is that the data channel is secure all the way except from the Secure Shell server to the FTP server, if they are on different machines. This sequence of events happens automatically for every data channel.

Since the port forwarding is opened to a local host address, the FTP client must be run on the same machine as the Secure Shell client if passive mode is used.

FTP in active mode

In active mode, the FTP client creates a listener on a local port, for a data channel from the FTP server to the FTP client, and requests the channell by sending the IP address and the port number to the FTP server in a

command of the following form: 'PORT 10,1,60,99,6,12'. The Secure Shell client intercepts this command and creates a remote port forwarding from the Secure Shell server's localhost address to the address and port specified in the PORT command.

After creating the port forwarding, the Secure Shell client rewrites the address and port in the PORT command to point to the newly opened remote forwarding on the Secure Shell server and sends it to the FTP server. Now the FTP server will open a data channel to the address and port in the PORT command, effectively forwarding the data through the SSH connection. The Secure Shell client passes the incoming data to the original listener created by the FTP client. The net effect is that the data channel is secure the whole way except from the Secure Shell client to the FTP client. This sequence of events happens automatically for every data channel.

Since the port forwarding is made to a local host address on the Secure Shell client machine, the FTP client must be run in the same host as the Secure Shell client if passive mode is used.

Where end-to-end encryption of FTP data channels is desired, the FTP server and Secure Shell server need to reside on the same host, and the FTP client and the Secure Shell client will likewise need to reside on the same host. If this is the case, both active or passive mode can be used.

Note: Consider using `sftp2` or `scp2` instead of FTP forwarding to secure file transfers. It will require less configuration than FTP forwarding, since the SSH Secure Shell server already has sftp-server2 as a subsystem, and sftp2 and scp2 clients are included in the distribution. Managing remote user restrictions on the server machine will be easier, since you do not have to do it also for FTP.

3.5.2 Dynamic Port Forwarding

Dynamic port forwarding is a transparent mechanism available for applications, which support the SOCKS4 or SOCKS5 client protocol. Instead of configuring port forwarding from specific ports on the local host to specific ports on the remote server, you can specify a SOCKS server which cab be used by the user's applications. Each application is configured in the regular way except that it is configured to use a SOCKS server on a local host port. The Secure Shell client application opens a port in the local host and mimics a SOCKS4 and SOCKS5 server for any SOCKS client application.

When the applications are connecting to services such as IMAP4, POP3, SMTP, HTTP, and FTP, they provide the necessary information to the SOCKS server, which is actually the Secure Shell client mimicking a SOCKS server. The client will use this information in creating port forwarding to the Secure Shell server and relying the traffic back and forth securely, as with user-specified port forwarding.

For more information, please see the `ssh2(1)` man pages.

3.5.3 X11 Forwarding

To enable X11 forwarding:

 © 2003 SSH Communications Security Corp.

1. Make sure that the SSH Secure Shell software was compiled with X forwarding support. The binary packages contain runtime X detection in SSH Secure Shell version 3.2 and above. However, if X security extensions are wanted, it is necessary to compile from source. When compiling, make sure not to run `./configure` with any X disabling options.

2. Ensure that `xauth` is in the path of the user running `./configure`. Also, make sure that you have the following line in your `/etc/ssh2/sshd2_config` file:

   ```
   AllowX11Forwarding              yes
   ```

   X11 forwarding also needs to be enabled in the client by setting the following line in the `ssh2_config` file:

   ```
   ForwardX11                      yes
   ```

3. Log into the remote system and type `xclock &`. This starts a X clock program that can be used for testing the forwarding connection. If the X clock window is displayed properly, you have X11 forwarding working.

Note: Do *not* set the `DISPLAY` variable on the client. You will most likely disable encryption. (X connections forwarded through Secure Shell use a special local display setting.)

In SSH Secure Shell 3.2 and above, if X11 SECURITY extension is compiled in, the X11 client applications are treated as untrusted by default (the effects of this depend on your Xserver's security policy). For more information, please see the `ssh2_config` man pages.

3.5.4 Agent Forwarding

For information about agent forwarding, see Section 6.5 (Using Authentication Agent).

Chapter 4

Authentication

SSH Secure Shell provides mutual authentication - the client authenticates the server and the server authenticates the client. Both parties are assured of the identity of the other party.

4.1 Server Authentication

The remote host can authenticate itself using either traditional public-key authentication or certificate authentication.

At the beginning of the connection the server sends its public host key to the client for validation. If certificate authentication is used the public key is included in the certificate the server sends to the client.

4.1.1 Public-Key Authentication

Server authentication is done during Diffie-Hellman key exchange through a single public-key operation. When public-key authentication is used to authenticate the server, the first connection is very important. During the first connection the SSH Secure Shell client will display a message similar to the one in Figure 4.1 (Windows client's first connection to a remote host):

At this point, you should verify the validity of the fingerprint, for example by contacting the administrator of the remote host computer (preferably by telephone) and asking her to verify that the key's fingerprint is correct. If the fingerprint is not verified, it is possible that the server you are connecting to is not the intended one (this is known as a *man-in-the-middle attack*).

After verifying the fingerprint, it is safe to continue connecting. The server's public key will then be stored on the client machine. The location depends on the client implemantation (on UNIX it is the `~/.ssh2/hostkeys` directory).

 © 2003 SSH Communications Security Corp.

Figure 4.1: Windows client's first connection to a remote host

After the first connection, the local copy of the server's public key will be used in server authentication.

If you want to avoid the risk associated with the first connection, you can copy the server public key in advance to the `/etc/ssh2/hostkeys` directory on the client computer and set the `StrictHostKeyChecking` keyword in the `ssh2_config` file to `yes`. After this, `ssh2` will refuse to connect if the server's public key is not in the `/etc/ssh2/hostkeys` directory.

The key pair used for server authentication is defined on the server in the `sshd2_config` file with the following parameters:

```
HostkeyFile     <private hostkey>
PublicHostKeyFile <public hostkey>
```

During the installation process, one DSA key pair (with the file names `hostkey` and `hostkey.pub`) is generated and stored in the `/etc/ssh2/` directory. By default this key pair is used for server authentication.

Beginning with SSH Secure Shell version 3.0, each server daemon can have multiple host keys. The daemon supports one DSA and one RSA key pair. You could have, for example, the following set of parameters in your `sshd2_config` file.

```
# RSA key
HostkeyFile     hostkey_rsa
PublicHostKeyFile hostkey_rsa.pub

# DSA key
HostkeyFile     hostkey_dsa
PublicHostKeyFile hostkey_dsa.pub
```

Both keys are stored in memory when the `sshd2` process is started, which means that either one of them can be used to authenticate the server.

4.1.2 Certificate Authentication

Certificate support is included in the commercial version of SSH Secure Shell.

Server authentication is performed using the Diffie-Hellman key exchange. This is what happens when certificates are used:

- The server sends its certificate (which includes its public key) to the client.
- As the server certificate is signed with the private key of a certification authority (CA), the client can verify the validity of the server certificate by using the CA certificate.
- The client checks that the certificate contains the fully qualified domain name of the server.
- The client verifies that the server has a valid private key by using a challenge.

When certificates are used, a man-in-the-middle attack is no longer a threat during key exchange, because the system checks that the server certificate has been issued by a trusted CA.

During authentication the system checks that the certificate has not been revoked. This can be done either by using the Online Certificate Status Protocol (OCSP) or a Certificate Revocation List (CRL), which can be published either to a Lightweight Directory Access Protocol (LDAP) or HTTP repository.

OCSP is automatically used if the certificate contains a valid `authority info access` extension. Correspondingly, CRLs are automatically used if the certificate contains a valid `CRL distribution point` extension. If LDAP is used as the CRL publishing method, the LDAP repository location can also be defined in the `ssh2_config` file (see below).

Server-Side Configuration

To configure the server-side functionality, perform the following tasks:

1. Enroll a certificate for the server. This can be done with the supplied ssh-certenroll2 command line utility.

   Note that the `DNS Address` parameter needs to correspond to the fully qualified domain name of the server.

   Example: Enrollment using `ssh-certenroll2`:

   ```
   $ /ssh-certenroll2 -g -p id:key -e
   -s "C=FI,O=SSH,CN=testserver;dns=testserver.trusted.org"
   -a "http://test-ca.trusted.org:8080/pkix/"
   -o /etc/ssh2/hostcert_rsa /etc/ssh2/test-ca-certificate.crt
   Generated 1024 bit private key saved to file
                                  /etc/ssh2/hostcert_rsa.prv.
   ```

 © 2003 SSH Communications Security Corp.

Remember to also define the SOCKS server (`-S`) or the HTTP proxy (`-P`), if required.

For more information on the `ssh-certenroll2` syntax, see Appendix A.2 (ssh-certenroll2).

2. The server's private host key must be in the ssh2 format. To convert X.509 keys to ssh2 format, use the `-x` flag with `ssh-keygen2`:

```
$ ssh-keygen2 -x <keyname>
Private key imported from X.509 format
Successfully saved private key to <keyname>_ssh2
```

PKCS #12 keys can be converted to ssh2 format by using the `-k` flag with `ssh-keygen2`.

Note: As the server's private key must have an empty passphrase, just press enter when you are prompted for the new passphrase.

```
$ ssh-keygen2 -k <keyname>
Password needed for PFX integrity check :
Integrity check ok.
Got shrouded key.
New passphrase for private key :
Again                          :
Successfully saved private key to <keyname>_ssh2
Safe decrypted succesfully.
Got certificate.
Certificate written to file <keyname>_ssh2.crt
```

3. Define the private key and the server certificate in the `sshd2_config` file:

```
HostkeyFile  <private key>
HostCertificateFile <server-certificate>
```

Client-Side Configuration

To configure the client-side functionality, perform the following tasks:

1. Copy the CA certificate(s) to the client machine.You can either copy the X.509 certificate(s) as such, or you can copy a PKCS #7 package including the CA certificate(s).

   Certificates can be extracted from a PKCS #7 package by specifying the `-7` flag with `ssh-keygen2`.

2. Define the CA certificate(s) to be used in host authentication in the `ssh2_config` file:

```
HostCA <ca-certificate>
```

Only one CA certificate can be defined per a `HostCA` keyword. The client will only accept certificates issued by the defined CA.

You can disable the use of CRLs by using the `HostCANoCrls` keyword instead of `HostCA`:

```
HostCANoCrls <ca-certificate>
```

Note: CRL usage should only be disabled for testing purposes. Otherwise it is highly recommended to always use CRLs.

3. Also define the LDAP server(s) in the `ssh2_config` file.

```
LDAPServers ldap://server1.domain1:port1,ldap://server2.domain2:port2,...
```

4. If necessary, define also the SOCKS server in the `ssh2_config` file.

```
SocksServer socks://socks_server:port/network/netmask,network/netmask...
```

The SOCKS server must be defined if CA services (OCSP,CRLs) are located behind a firewall.

4.2 User Authentication

Different methods can be used to authenticate users in SSH Secure Shell. These authentication methods can be combined or used separately, depending on the level of functionality and security you want.

User authentication methods used by the client by default are, in the following order: public-key, keyboard-interactive, and password authentication (if available). Public-key and certificate authentication are combined into the public-key authentication method. The server allows public-key and password authentication by default.

Different authentication methods are discussed in more detail in the following sections.

4.2.1 Password Authentication

With passwords, it is also possible to use keyboard-interactive authentication. See Section 4.3 (Keyboard-Interactive Authentication) for more information.

The password authentication method is the easiest to implement, as it is set up by default. Password authentication uses the `/etc/passwd` or `/etc/shadow` file on a UNIX system, depending on how the passwords are set up.

To enable password authentication, make sure that the `AllowedAuthentications` field of the `/etc/ssh2/sshd2_config` and `/etc/ssh2/ssh2_config` configuration files contain the parameter `password`:

```
AllowedAuthentications   password
```

Other authentication methods can be listed in the configuration files as well.

 © 2003 SSH Communications Security Corp.

4.2.2 Public-Key Authentication

Per-user configuration information and encryption keys are by default stored in the `.ssh2` subdirectory of each user's home directory.

In the following instructions, *Remote* is the SSH Secure Shell server machine into which you are trying to connect, and *Local* is the machine running an SSH Secure Shell client.

Keys Generated with `ssh-keygen2`

In order to set up user public-key authentication,

do a manual setup according to the following instructions.

The following terms will be used in this example: *Remote* is the SSH Secure Shell server into which you are trying to connect. *RemoteUser* is the user name on the server into which you would like to login. *Local* is the machine running a SSH Secure Shell client. *LocalUser* is the user name on the client machine that should be allowed to login to *Remote* as *RemoteUser.*

1. To make sure that public-key authentication is enabled, the `AllowedAuthentications` field in both the `/etc/ssh2/sshd2_config` file on *Remote* and in the `/etc/ssh2/ssh2_config` file on *Local* should contain the word `publickey`:

   ```
   AllowedAuthentications   publickey
   ```

   Other authentication methods can be listed in the configuration file as well.

2. Create a key pair by executing `ssh-keygen2` on *Local*.

   ```
   Local> ssh-keygen2
   Generating 2048-bit dsa key pair
      1 oOo.oOo.o
   Key generated.
   2048-bit dsa, user@Local, Wed Mar 22 2002 00:13:43 +0200
   Passphrase :
   Again :
   Private key saved to  /home/user/.ssh2/id_dsa_2048_a
   Public key saved to   /home/user/.ssh2/id_dsa_2048_a.pub
   ```

   `Ssh-keygen2` will now ask for a passphrase for the new key. Enter a sufficiently long (20 characters or so) sequence of any characters (spaces are OK). `Ssh-keygen2` creates a `.ssh2` directory in your home directory (if it is not already present), and stores your new authentication key pair in two separate files. One of the keys is your private key which must *never* be made available to anyone but yourself. The private key can only be used together with the passphrase.

In the example above, the private key file is `id_dsa_2048_a`. The other file `id_dsa_2048_a.pub` is your public key, which can be distributed to other computers.

Note: Beginning with version 3.0, SSH Secure Shell includes support for RSA keys. They can be generated by specifying the `-t` flag with `ssh-keygen2`.

```
Local> ssh-keygen2 -t rsa
Generating 2048-bit rsa key pair
   2 oOo.ooOo.oOo
Key generated.
2048-bit rsa, user@Local, Wed May 02 2002 14:15:41 +0300
Passphrase :
Again      :
Private key saved to /home/user/.ssh2/id_rsa_2048_a
Public key saved to /home/user/.ssh2/id_rsa_2048_a.pub
```

3. Create an `identification` file in your `~/.ssh2` directory on *Local*.

   ```
   Local> cd ~/.ssh2
   Local> echo "IdKey id_dsa_2048_a" > identification
   ```

   You now have an `identification` file which consists of one line that denotes the file containing your identification (your private key). For special applications, you can create multiple identifications by executing `ssh-keygen2` again. This is, however, not usually needed.

4. Copy your public key (`id_dsa_2048_a.pub`) to your `~/.ssh2` directory on *Remote*.

5. Create an `authorization` file in your `~/.ssh2` directory on *Remote*. Add the following line to the `authorization` file:

   ```
   Key     id_dsa_2048_a.pub
   ```

   This directs the SSH Secure Shell server to use `id_dsa_2048_a.pub` as a valid public key when authorizing your login. If you want to login to *Remote* from other hosts, create a key pair on the hosts (steps 1 and 2) and repeat steps 3, 4, and 5 on *Remote*. (Remember to specify a different file name for each key pair.)

6. Now you should be able to login to *Remote* from *Local* using SSH Secure Shell.

   Try to login:

   ```
   Local>ssh Remote
   Passphrase for key "/home/user/.ssh2/id_dsa_1024_a
   with comment "2048-bit dsa, created by user@Local
   Wed Mar 22 2002 00:13:43 +0200":
   ```

   After you have entered the passphrase of your private key, a Secure Shell connection will be established.

 © 2003 SSH Communications Security Corp.

Keys Generated with `ssh-keygen1`

Beginning with version 3.0, SSH Secure Shell enables the usage of keys generated with `ssh-keygen1`. However, the keys must be converted from the ssh1 format to ssh2 format.

```
$ ssh-keygen2 -1 <keyname>.pub
Successfully converted public key to <keyname>.pub_ssh2
$ ssh-keygen2 -1 <keyname>
Passphrase :
Successfully converted private key to <keyname>_ssh2
```

PGP Keys

SSH Secure Shell only supports the OpenPGP standard and the PGP programs conforming to it. GnuPG is used in the following instructions. If you use PGP, the only difference is that the file extension is `pgp` instead of GnuPGP's `gpg.`

1. To make sure that user public-key authentication is enabled, the `AllowedAuthentications` field both in the `/etc/ssh2/sshd2_config` file on *Remote* and the `/etc/ssh2/ssh2_config` file on *Local* should contain the word `publickey`:

   ```
   AllowedAuthentications   publickey
   ```

   Other authentication methods can be listed in the configuration file as well.

2. Copy your private key ring (`secring.gpg`) to the `~/.ssh2` directory on *Local*.

3. Create an `identification` file in your `~/.ssh2` directory on *Local* if you do not already have one. Add the following lines to the `identification` file:

   ```
   PgpSecretKeyFile    <filename of the user's private key ring>
   IdPgpKeyName        <name of the OpenPGP key in PgpSecretKeyFile>
   IdPgpKeyFingerprint <fingerprint of OpenPGP key in PgpSecretKeyFile>
   IdPgpKeyId          <id of the OpenPGP key in PgpSecretKeyFile>
   ```

4. Copy your public-key ring (`pubring.gpg`) to the `~/.ssh2` directory on *Remote*

   ```
   scp2 pubring.gpg user@remote_host:.ssh2
   ```

5. Create an `authorization` file in your `~/.ssh2` directory on *Remote*. Add the following lines to the `authorization` file:

   ```
   PgpPublicKeyFile   <filename of the user's public-key ring>
   PgpKeyName         <name of the OpenPGP key>
   PgpKeyFingerprint  <fingerprint the OpenPGP key>
   PgpKeyId           <id of the OpenPGP key>
   ```

6. Now you should be able to login to *Remote* from *Local* using Secure Shell.

   Try to login:

   ```
   Local>ssh Remote
   Passphrase for pgp key "user (comment) <user@Local>":
   ```

   After you have entered the passphrase of your PGP key, a Secure Shell connection will be established.

Optional Additional Configuration Settings

The following configuration steps are optional:

- It is possible to use different settings depending on which key is used in public-key authentication. Your `authorization` file could, for example, contain the following:

  ```
  Key master.pub
  Key maid.pub
  Options allow-from=".*\.trusted\.org"
  Key butler.pub
  Options deny-from=".*\.evilcrackers\.org",deny-from="phoney.org",no-pty
  ```

  When someone now logins using the `master` key, the connection is not limited in any way by the `authorization` file. However, if the `maid` key is used, only connections from certain hosts will be allowed. And if the `butler` key is used, connections are denied from certain hosts, and additionally the allocation of tty is prevented.

  Note: The `options` keyword is available only in SSH Secure Shell version 3.0 and later. In the previous versions, the only key-specific keyword is `command`.

  More information on `options` (and `command`) keywords is available in the `ssh2` man pages.

- The per-user configuration directory can be changed by setting the `UserConfigDirectory` keyword in the `sshd2_config` file and on the client settings.

4.2.3 Host-Based Authentication

The following terms will be used in this example: *Remote* is the SSH Secure Shell server to which you are trying to connect. *RemoteUser* is the user name on the server into which you would like to login. *Local* is the machine running a SSH Secure Shell client. *LocalUser* is the user name on the client machine that should be allowed to login to *Remote* as *RemoteUser.*

1. First, install SSH Secure Shell on the *Local* and *Remote* machines. Do not forget to generate a host key. If your installation took care of this automatically, or if you already have a copy of your `/etc/ssh2/hostkey` and `/etc/ssh2/hostkey.pub`, you can skip the host-key generation. Otherwise, give the following command:

 © 2003 SSH Communications Security Corp.

```
# ssh-keygen2 -P /etc/ssh2/hostkey
```

Note: Beginning with SSH Secure Shell version 3.0, you can also use RSA keys.

2. Copy the *Local* machine's `/etc/ssh2/hostkey.pub` file over to the *Remote* machine and name it like this:

   `/etc/ssh2/knownhosts/`*`hostname.domain`*`.ssh-dss.pub`

   Replace *`hostname.domain`* with the long host name of the *Local* machine (the fully qualified domain name). You will run into problems if the system does not recognize the host name as `hostname.domain.somewhere.com` but recognizes it only as `hostname`. You can find this out while running `sshd2` in verbose mode when trying to establish connections.

   The *Remote* machine now has the *Local* machine's public key, so the *Remote* machine can verify the *Local* machine's identity based on a public-key signature. By contrast, `rsh` only uses the IP address for host authentication.

   Note: If you use RSA keys, the name of *Local*'s `/etc/ssh2/hostkey.pub` file which is copied over to the *Remote* needs to be `/etc/ssh2/knownhosts/`*`hostname.domain`*`.ssh-rsa.pub`.

3. To make sure that SSH Secure Shell finds your complete domain name (not just the host name), edit the following line in the `/etc/ssh2/ssh2_config` file on *Local*:

   ```
   DefaultDomain   yourdomain.com
   ```

   Note: The keyword mentioned in this and the following steps will only be effective in the global ssh2_config file.

4. On *Remote*, create a file in the home directory of *RemoteUser* named `.shosts`. The contents of this file should be the long host name of *Local*, some tabs or spaces, and *LocalUser*'s user name.

   Contents of `~/.shosts`:

   ```
   localhostname.yourdomain.com   LocalUser
   ```

   Be sure to `chown` and `chmod` the `.shosts` file. The `.shosts` file must be owned by *RemoteUser* and should have mode 0400.

5. Check the files `/etc/ssh2/sshd2_config` on *Remote* and `/etc/ssh2/ssh2_config` on *Local*. Make sure that the `AllowedAuthentications` field contains the word `hostbased`. For example, it may read:

   ```
   AllowedAuthentications     hostbased,password
   ```

   It does not matter what other authentication methods are allowed. Just make sure that the `hostbased` keyword is first in the list.

6. Also check that `IgnoreRhosts` is set to `no` in the `/etc/ssh2/sshd2_config` file on *Remote*:

   ```
   IgnoreRhosts     no
   ```

   If you had to modify the `sshd2_config` file, you will have to send a HUP signal to `sshd2` to make the change take effect.

```
# kill -HUP ‘cat /var/run/sshd2_22.pid‘
```

or

```
# kill -HUP ‘cat /etc/ssh2/sshd2_22.pid‘
```

7. Now you should be all set.

   On *Local*, log in as *LocalUser* and give the command

   ```
   ssh RemoteUser@Remote uptime
   ```

   You should now get the results of `uptime` run on *Remote*.

   The first time you run `ssh` to that particular server, you will have to answer *yes* when asked if you want to connect to the server. This is because the local `ssh` does not yet have the remote server's public key. For maximum security, it is highly recommended to verify the fingerprint of the remote host's public key, as described in Section 4.1.1 (Public-Key Authentication) to avoid man-in-the-middle attacks. This will only be necessary when connecting for the first time.

Troubleshooting Host-Based Authentication

If you have problems with host-based authentication, check the following:

- Did you name the local host key file appropriately on *Remote*? It should be either

  `/etc/ssh2/knownhosts/HOSTNAME.ssh-dss.pub`

  or

  `/etc/ssh2/knownhosts/HOSTNAME.ssh-rsa.pub`

  where `HOSTNAME` has to be the long host name (fully qualified domain name).
- Did you copy the host key properly?
- Check that the key on *Remote* is actually the same as `hostkey.pub` on *Local*.

  On *Local*:

  ```
  $ ssh-keygen2 -F /etc/ssh2/hostkey.pub
  ```

  On *Remote*:

  ```
  $ ssh-keygen2 -F /etc/ssh2/knownhosts/hostname.domain.ssh-dss.pub
  ```

  or

  ```
  $ ssh-keygen2 -F /etc/ssh2/knownhosts/hostname.domain.ssh-rsa.pub
  ```

  The key fingerprints should match.

 © 2003 SSH Communications Security Corp.

- Check your spelling in the `.shosts` file.
- Make sure the `.shosts` file is owned by *RemoteUser* and check that its permissions are 0400.

```
# ls -la ~/.shosts
-r--------   1  RemoteUser  users  178 May 28 15:05 /home/RemoteUser/.shosts
```

- Make sure that *RemoteUser*'s home directory on *Remote* is owned by *RemoteUser* and is not writable by anyone byt *RemoteUser*.
- Run the server and the client with the `-d3` option. This is a good way to see if a host key file is missing, or if something is misconfigured.

Using Certificates

It is possible to use a certificate instead of the traditional public key pair to authenticate the *Local* host. To use certificates, perform the following tasks:

1. Enable host-based authentication in `ssh2_config` on *Local* and in `sshd2_config` on *Remote* by using the `AllowedAuthentications` keyword.
2. Define *Local*'s server certificate in `sshd2_config` on *Local*:

   ```
   HostKeyFile <private key>
   HostCertificateFile <server-certificate>
   ```

   The certificate must contain a *dns* extension which contains the fully qualified domain name of *Local*.
3. Set the `DefaultDomain` value in `ssh2_config` on *Local*.
4. Set the `HostCA` and `LdapServers` values in `sshd2_config` on *Remote*.

   ```
   HostCA <trusted-ca-certificate>
   LdapServers ldap://server.domain:port
   ```

5. Make sure that the `~/.shosts` file on *Remote* contains the following:

   ```
   localhostname.yourdomain.com   LocalUser
   ```

More information is available in the `ssh-certificates` man pages and in Sections 4.1.2 (Server) and 4.2.4 (User).

Optional Additional Configuration Settings

To make the host-based authentication more secure, you may want to consider the following optional configuration settings:

- Setting the `AllowSHosts` and `DenySHosts` keywords in the `sshd2_config` file you can filter the `.shosts`, `.rhosts`, `/etc/hosts.equiv` and `/etc/shosts.equiv` entries.
- If you want to allow only global configuration files (`/etc/hosts.equiv` and `/etc/shosts.equiv`), make sure that you have the following entry in your `sshd2_config` file:

  ```
  IgnoreRhosts    yes
  ```

  After this modification the `.shosts` and `.rhosts` files will not be used in host-based authentication.
- To force an exact match between the host name that the client sends to the server and the client's DNS entry, make sure that you have the following definition in your `sshd2_config` file:

  ```
  HostbasedAuthForceClientHostnameDNSMatch yes
  ```

  Please note that with the above-mentioned definition, host-based authentication through NAT will not work.

  Note: This keyword is available in SSH Secure Shell version 3.0 and later.

4.2.4 Certificate Authentication

The commercial version of SSH Secure Shell includes support for certificate authentication.

Also certificate authentication is performed via the public-key authentication method. Instead of the client sending just a public key, it sends a certificate containing a public key.

In brief, certificate authentication works in the following way:

- The client sends the user certificate (which inludes the user's public key) to the server.
- The server uses the CA certificate to check that the user's certificate is valid.
- The server uses the user certificate to check from its mapping file(s) whether login is allowed or not.
- Finally, if connection is allowed, the server makes sure that the user has a valid private key by using a challenge.

Compared to traditional public-key authentication, this method is more secure because the system checks that the user certificate was issued by a trusted CA. In addition, certificate authentication is more convenient because no local database of user public keys is required on the server.

 © 2003 SSH Communications Security Corp.

It is also easy to deny a user's access the system by revoking his or her certificate. The status of a certificate can be checked either by using the Online Certificate Status Protocol (OCSP) or Certificate Revocation Lists (CRLs), which can be published either to a Lightweight Directory Access Protocol (LDAP) or HTTP repository.

OCSP is used if the certificate contains a valid `authority info access` extension. Correspondingly, CRLs are used if the certificate contains a valid `CRL distribution point` extension. If LDAP is used as the CRL publishing method, the LDAP repository location can be also defined in the `sshd2_config` file (see below).

Server-Side Configuration

To configure the server, perform the following tasks:

1. Acquire the CA certificate and copy it to the server machine. You can either copy the X.509 certificate(s) as such or you can copy a PKCS #7 package including the CA certificate(s).

   Certificates can be extracted from a PKCS #7 package by specifying the `-7` flag with `ssh-keygen2`.

2. Certificate authentication is a part of the `publickey` authentication method. Make sure that you have enabled it in the `sshd2_config` file:

   ```
   AllowedAuthentications   publickey
   ```

3. Specify the CA certificate and the mapping file(s) in the `sshd2_config` file:

   ```
   Pki <ca-cert-path>
   MapFile <map-file-path>
   ```

   You can disable the use of CRLs by adding the `PkiDisableCRLs` keyword below the `Pki` keyword.

   ```
   PkiDisableCRLs  yes
   ```

   Note: CRL usage should only be disabled for testing purposes. Otherwise it is highly recommended to always use CRLs.

   You can define several CA certificates by using several `Pki` keywords.

   ```
   Pki <ca-cert-path1>
   MapFile <map-file-path1>
   Pki <ca-cert-path2>
   MapFile <map-file-path1>
   MapFile <map-file-path2>
   ```

   Note that multiple `MapFile` keywords are permitted per `Pki` keyword. Also, if no mapping file is defined, all connections are denied even if user certificates can be verified using the defined CA certificate.

   Server will accept only certificates issued by defined CA(s).

4. Also define the LDAP server(s) in the `sshd2_config` file.

   ```
   LDAPServers ldap://server1.domain1:port1,ldap://server2.domain2:port2,...
   ```

5. If necessary, define also the SOCKS server in the `sshd2_config` file.

   ```
   SocksServer socks://socks_server:port/network/netmask,network/netmask...
   ```

   The SOCKS server must be defined if CA services (OCSP and CRLs) are located behind a firewall.

6. Next you need to create the map file. It specifies which certificates authorize logging into which accounts.

   The format of the file is the following:

   ```
   <account-id> <keyword> <argument>
   ```

   The *keyword* can be either `Email`, `Subject`, `SerialAndIssuer`, `EmailRegex`, or `SubjectRegex`. The *arguments* depend on the *keyword*.

   - `Email`: The argument is the email address which must be present in the certificate.
   - `Subject`: The argument is the required subject name in LDAP DN (distinguished name) string format.
   - `SerialAndIssuer`: The argument is the required serial number and issuer name in LDAP DN string format, separated by spaces or tabs.
   - `EmailRegex`: The argument is the regular expression which must match an email address in the certificate. If `account-id` contains the string `%subst%`, it is substituted with the first parenthesized part of the regular expression.The patterns are matched using `SSH_REGEX_SYNTAX_EGREP`.
   - `SubjectRegex`: The argument is the regular expression which must match a subject name in the certificate. If `account-id` contains the string `%subst%`, it is substituted with the first parenthesized part of the regular expression. The patterns are matched using `SSH_REGEX_SYNTAX_EGREP`.

   Examples

   The following are examples of different map file definitions:

   ```
   testuser email testuser@ssh.com
   testuser subject C=FI,O=SSH,CN=Secure Shell Tester
   testuser serialandissuer 1234 C=FI,O=SSH,CN=Secure Shell Tester
   %subst% subjectregex C=FI, O=SSH, CN=([a-z]+)
   %subst% emailregex ([a-z]+)@ssh\.com
   ```

   The last line permits logging with any email address with only letters in the user name. See `man sshregex` for more information on the regular expression syntax.

 © 2003 SSH Communications Security Corp.

Client-Side Configuration

Configure the client side according to the certificate storage method used; a software or a PKCS #11 token (for example, a smart card).

Software Certificates

To configure the system for software certificates, perform the following tasks:

1. Enroll a certificate for yourself.

   Example: Enrollment using *ssh-certenroll2*

   ```
   $ /ssh-certenroll2 -g -p id:key -e
   -s "C=FI,O=SSH,CN=user;email=user@trusted.org"
   -a "http://test-ca.trusted.org:8080/pkix/"
   -o /home/user/.ssh2/user_rsa /etc/ssh2/test-ca-certificate.crt
   Generated 1024 bit private key saved to file
                                /home/user/.ssh2/user_rsa.prv.
   ```

   Remember to define also the SOCKS server (with the `-S` flag) or an HTTP proxy (with the `-P` flag), if necessary.

2. The private key must be in the ssh2 format. To convert X.509 keys to ssh2 format, specify the `-x` flag with `ssh-keygen2`:

   ```
   $ ssh-keygen2 -x <keyname>
   Private key imported from X.509 format
   Successfully saved private key to <keyname>_ssh2
   ```

   Also PKCS #12 keys can be converted to ssh2 format. This is done by specifying the `-k` flag with `ssh-keygen2`.

   ```
   $ ssh-keygen2 -k <keyname>
   Password needed for PFX integrity check :
   Integrity check ok.
   Got shrouded key.
   New passphrase for private key :
   Again                          :
   Successfully saved private key to <keyname>_ssh2
   Safe decrypted succesfully.
   Got certificate.
   Certificate written to file <keyname>_ssh2.crt
   ```

3. Make sure that public-key authentication is enabled in the `ssh2_config` file.

   ```
   AllowedAuthentications   publickey
   ```

4. Specify the private key of your software certificate in the `~/.ssh2/identification` file.

   ```
   CertKey <private-key-path>
   ```

   The certificate itself will be read from `private-key-path.crt`.

PKCS #11 Tokens

To configure the system for PKCS #11 tokens (for example smart cards), perform the following tasks:

1. Enable public-key authentication in the `ssh2_config` file.

   ```
   AllowedAuthentications   publickey
   ```

2. Also define the external key provider and the initial string.

   ```
   EkProvider <provider-name>
   EkInitString <init-string>
   ```

   The above-mentioned options can be also defined using the `-E` and `-I` flags with `ssh2`, respectively.

4.2.5 Kerberos Authentication

When Kerberos support is enabled, it is possible to authenticate using Kerberos credentials, forwardable TGT (ticket granting ticket) and passing TGT to remote host for single sign-on. It is also possible to use Kerberos password authentication. Please note that SSH Secure Shell only supports Kerberos5.

To enable Kerberos support, perform the following tasks:

1. Compile the source:

   ```
   ./configure --with-kerberos5
   make
   make install
   ```

2. Make sure that you have the following line in your `/etc/ssh2/sshd2_config` file:

   ```
   AllowedAuthentications kerberos-1@ssh.com,kerberos-tgt-1@ssh.com
   ```

   Other authentication methods can be listed in the configuration file as well.

3. Also, make sure that you have the following line in your `/etc/ssh2/ssh2_config` file (for SSH Secure Shell versions 2.3 and 2.4):

 © 2003 SSH Communications Security Corp.

```
AllowedAuthentications kerberos-1@ssh.com,kerberos-tgt-1@ssh.com
```

If you are using SSH Secure Shell version 3.0 or later, make sure that you use the new versions of kerberos authentication methods:

```
AllowedAuthentications kerberos-2@ssh.com,kerberos-tgt-2@ssh.com
```

Note: SSH Communications Security does not provide technical support on how to configure Kerberos. Our support only covers SSH Secure Shell applications and source code.

4.2.6 Pluggable Authentication Module (PAM)

We recommend the use of keyboard-interactive authentication with PAM.

When PAM is used, SSH Secure Shell transfers the control of authentication to the PAM library, which will then load the modules specified in the PAM configuration file. Finally, the PAM library tells SSH Secure Shell whether or not the authentication was successful. SSH Secure Shell is not aware of the details of the actual authentication method employed by PAM. Only the final result is of interest.

To enable PAM support, perform the following tasks:

1. Compile the source:

   ```
   ./configure
   make
   make install
   ```

   By default, the PAM service name is `sshd2`. If you want to change it, you can add the configure flag `--with-daemon-pam-service-name=name`.

2. When using keyboard-interactive PAM submethod, make sure that you have the following lines in the `/etc/ssh2/sshd2_config` file:

   ```
   AllowedAuthentications  keyboard-interactive
   AuthKbdInt.Optional     pam
   ```

   And the following line in the `/etc/ssh2/ssh2_config` file:

   ```
   AllowedAuthentications  keyboard-interactive
   ```

   In case you are not using keyboard-interactive, make sure that you have the following line in both your `/etc/ssh2/sshd2_config` file and `/etc/ssh2/ssh2_config` file:

   ```
   AllowedAuthentications  pam-1@ssh.com
   ```

The PAM configuration settings are located either in `/etc/pam.conf` or in `/etc/pam.d/sshd2`. The modules are usually either in the `/lib/security` directory or in the `/usr/lib/security` directory. Currently, SSH Secure Shell supports PAM on Linux and on Solaris 2.6 or later.

There must be at least one `auth`, one `account`, and one `session` module in the configuration file. Otherwise the connection will be refused. Also, modules which require `PAM_TTY` will not work because TTY allocation is done in SSH Secure Shell after the authentication.

PAM Examples

The following are examples of different PAM configurations.

1. The `/etc/pam.d/sshd2` file on Red Hat Linux:

```
auth     required /lib/security/pam_pwdb.so shadow nullok
auth     required /lib/security/pam_nologin.so
account  required /lib/security/pam_pwdb.so
password required /lib/security/pam_cracklib.so
password required /lib/security/pam_pwdb.so shadow nullok use_authtok
session  required /lib/security/pam_pwdb.so
```

2. The `/etc/pam.conf` entry on Solaris:

```
sshd2  auth      required  /usr/lib/security/pam_unix.so debug
sshd2  account   required  /usr/lib/security/pam_unix.so debug
sshd2  password  required  /usr/lib/security/pam_unix.so debug
sshd2  session   required  /usr/lib/security/pam_unix.so debug
```

See Section 4.3 (Keyboard-Interactive Authentication) for more information on keyboard-interactive authentication.

Note: SSH Communications Security does not provide technical support on how to configure PAM. Our support only covers SSH Secure Shell applications and source code.

4.2.7 SecurID

If SecurID is to be used, the Keyboard-Interactive authentication method must be enabled on the client.

Please familiarize yourself with the *RSA ACE/Server* documentation before reading further.

In the instructions below, the `/top` directory refers to the *RSA ACE/Server* top-level directory.

1. In order to enable SecurID support, you need to compile the source code on a computer where *RSA ACE/Server* (master or slave) or *RSA ACE/Agent* software is already installed, configured, and running.

 © 2003 SSH Communications Security Corp.

```
./configure --with-serversecurid[=/PATH]
make
make install
```

Replace `/PATH` with the absolute PATH to the directory containing the `sdiclient.a` file.

For RSA ACE/Server 4.0 the file is usually located in the `/top/ace/examples` directory, for 5.0 in the `/top/ace/inc` directory.

Note: If you do not want to make the compilation as root, make sure that all the above-mentioned files are readable by the user you are compiling as.

2. When using the Keyboard-Interactive SecurID submethod, make sure that you have the following lines in the `/etc/ssh2/sshd2_config` file:

```
AllowedAuthentications  keyboard-interactive
AuthKbdInt.Optional     securid
```

And the following line in the `/etc/ssh2/ssh2_config` file:

```
AllowedAuthentications  keyboard-interactive
```

In case you are not using Keyboard-Interactive, make sure that you have the following line both in your `/etc/ssh2/sshd2_config` file and in your `/etc/ssh2/ssh2_config` file:

```
AllowedAuthentications  securid-1@ssh.com
```

3. Check that the user's shell is **not** `/top/ace/prog/sdshell`.
4. Start *RSA ACE/Server*.
5. Check that the `VAR_ACE` environment variable is set. It has to be set before starting `sshd2`, and its value must be `/top/ace/data`.
6. Start `sshd2`.

See Section 4.3 (Keyboard-Interactive Authentication) for more information on keyboard-interactive authentication.

Note: SSH Communications Security does not provide technical support on how to configure *RSA ACE/Server*. Our support only covers SSH Secure Shell applications .

4.3 Keyboard-Interactive Authentication

Keyboard-Interactive is a generic authentication method that can be used to implement different types of authentication mechanisms.

4.3.1 Overview

What Is Keyboard-Interactive?

Keyboard-interactive is a relatively new authentication method, designed in the Secure Shell Working Group of the Internet Engineering Task Force (IETF). The Working Group's abstract contains the following introduction to the subject:

This document describes a general-purpose authentication method for the SSH protocol, suitable for interactive authentications where the authentication data should be entered via a keyboard. The major goal of this method is to allow the SSH client to support a whole class of authentication mechanism(s) without knowing the specifics of the actual authentication mechanism(s)

What Can Be Done with It?

Basically, any currently supported authentication method that requires only the user's input, can be performed with Keyboard-Interactive.

Currently, the following methods are supported:

- password
- SecurID
- PAM (but see Section 4.3.1 (What Cannot Be Done With It?)).

What Cannot Be Done with It?

If passing of some binary information is required (as in public-key authentication), keyboard-interactive cannot be used.

PAM has support for binary messages and client-side agents, and those cannot be supported with keyboard-interactive. However, currently there are no implementations that take advantage of the binary messages in PAM, and the specification may not be cast in stone yet.

4.3.2 Configuring the Server and Client

Client Configuration

To enable keyboard-interactive authentication, make sure that you have the following line in the `/etc/ssh2/ssh2_config` file:

```
AllowedAuthentications  keyboard-interactive
```

 © 2003 SSH Communications Security Corp.

Keyboard-interactive is enabled by default on the client.

Note: The client cannot request any specific keyboard- interactive submethod if the server allows several optional submethods. The order in which the submethods are offered depends on the server configuration. However, if the server allows for example two optional submethods SecurID and password, the user can skip SecurID by pressing enter when SecurID is offered by the server. The user will then be prompted for password.

Server Configuration

Keyboard-interactive is not enabled by default on the server. Make sure that you have the following line in the `/etc/ssh2/sshd2_config` file:

```
AllowedAuthentications  keyboard-interactive
```

The submethods and policy for keyboard-interactive are configured as follows:

```
## SSH CONFIGURATION FILE FORMAT VERSION 1.1
## REGEX-SYNTAX egrep
...
AuthKbdInt.Required securid
AuthKdbInt.Optional pam, password
AuthKbdInt.NumOptional 1
AuthKbdInt.FailureTimeout 2
...
```

This allows for maximum configurability without being too hard to implement. See the `sshd2_config` man pages for more information on the keywords.

The default for required submethods is 0, although if no required submethods are specified, the client must always pass at least one optional submethod.

4.3.3 Adding New Authentication Methods

This sections covers advanced implementation of keyboard-interactive.

Please note that SSH Communications Security does not provide technical support on how to implement your own keyboard-interactive submethod. Also, consider carefully the security implications when implementing a submethod of your own.

New submethods are added to the submethods array in `apps/ssh/auths-kbd-int-submethods.c`. You need to create an initialization and uninitialization function for the submethod. The initialization function

will create a method context for the submethod, and start the authentication, using the specified conversation function.

You need to define three functions: `init`, `free` and a callback for the conversation function.

init Type

```
typedef void (*init)(void **method_context,
                     SshAuthKbdIntSubMethods methods,
                     SshKbdIntSubMethodConv conv,
                     void *conv_context);
```

Description The initialization function.

Arguments

`method_context`

Should be assigned to the pointer to the context allocated. Assigning something to `method_context` is mandatory. (This simplifies the main method's bookkeeping on what submethods have actually been initialized.) This context is passed to the `free` function.

`methods`

Should be stored by the submethod. It contains for example the `SshServer` structure and an `SshUser` structure that can be used in the authentication process by the submethod.

`conv`

The conversation function, called by the submethod when it has the requests. The prototype is described below.

`conv_context`

Should be given to the conv function.

free Type

```
typedef void (*free)(void *method_context);
```

Description `free` is called by the main method during uninitialization. The submethod should free all state, including `method_context` at this stage.

 © 2003 SSH Communications Security Corp.

Arguments

`method_context`

Should be assigned to the pointer to the context allocated.

SshKbdIntSubMethodCB Type

```
typedef void (*SshKbdIntSubMethodCB)(size_t num_resp,
                                     char **resps,
                                     Boolean cancel,
                                     void *context);
```

Description A function of this type is passed to the conversation function. It is called after the responses are received from the client.

Arguments

`num_resp`

The size of the `resps` array.

`resps`

The responses received from the client.

`cancel`

If this value is TRUE, the submethod should abort the authentication, and get ready to be destroyed (with the `free` function).

`context`

The allocated context.

SshKbdIntSubMethodConv Type

```
typedef void (*SshKbdIntSubMethodConv)(SshKbdIntResult code,
                                       const char *instruction,
                                       size_t num_reqs,
                                       char **reqs,
                                       Boolean *echo,
                                       SshKbdIntSubMethodCB method_cb,
                                       void *method_ctx,
                                       void *context);
```

Description The submethod calls this function when it has the requests.

Arguments

`code`

Indicates the state of the authentication. If the submethod authentication is a success, this should be `SSH_KBDINT_SUBMETHOD_RESULT_SUCCESS`, if failed, `SSH_KBDINT_SUBMETHOD_RESULT_FAILED`, and if the authentication is not complete, `SSH_KBDINT_SUBMETHOD_RESULT_NONE_YET`.

`instruction`

This field should contain a short description of what the user should do.

`num_reqs`

Indicates the size of reqs and echo arrays. If the code is not `SSH_KBDINT_SUBMETHOD_RESULT_NONE_YET`, this should be `0`, and reqs and echo should be `NULL`.

`reqs`

The requests sent to the client.

`echo`

Tells the client whether the user's response to the corresponding request should be echoed.

`method_cb`

The main method will call this when it has obtained the responses from the client.

`method_ctx`

This is given to `method_cb` by the main method.

`context`

The allocated context.

Please see `auths-kbd-int-submethods.c` for an example of using a 'password' submethod. This submethod is implemented in `auths-kbd-int-passwd.[ch]`.

Please follow the style of those files when creating new submethods. Complete documentation for the API is located in `apps/ssh/auths-kbd-int-submethods.h`.

 © 2003 SSH Communications Security Corp.

Chapter 5

Log Messages

The log messages generated by SSH Secure Shell for Servers are listed in the following sections based on their subject matter.

5.1 Log Message Reference

5.1.1 User Authentication - Common

Authentication failed

```
<methodname> authentication failed. Login to account
<loginname> not allowed or account non-existent.
```

The system administrator has denied logging in for the user, or the user is trying to log in using a non-existent account.

Authentication failed, user does not exist

```
User <login-name> does not exist.
```

Authentication failed because the specified user account does not exist.

Authentication failed, system policy doesn’t allow user’s login

```
User <login-name>’s login is not allowed due to system
policy
```

 © 2003 SSH Communications Security Corp.

Authentication failed because the system policy does not allow the user to log in. The account may be locked or expired, for example.

Authentication failed, username is on deny list

```
User <login-name> is denied login because username matched with
deny list.
```

Authentication failed because the specified user name is included in the server's deny list.

Authentication failed, username not on allow list

```
User <login-name> is denied login because allow list exists, and
username was not matched.
```

Authentication failed because the specified user name is not included in the server's allow list.

Authentication failed, user's group is on deny list

```
User <login-name> is denied login because user's group is on
group deny list.
```

Authentication failed because the user's group is included in the server's group deny list.

Authentication failed, user's group is not on group allow list

```
User <login-name> is denied login because group allow list
exists, and user's group didn't match.
```

Authentication failed because the user's group is not on the server's group allow list.

Authentication not allowed from a host

```
<methodname> authentication failed. Connection from <hostname>
denied. Authentication as user <loginname> was attempted.
```

The server has determined that the user is not allowed to login from the current host.

Authentication not allowed, unable to reverse map hostname

```
Login not allowed because host's IP address (<ip-address>) could
not be mapped to a host name and reverse mapping is required.
```

The server could not map the connecting host's IP address to a host name and the configuration option `RequireReverseMapping` is on.

Authentication not allowed, host is on the deny list

```
Login not allowed because the connecting host (<hostname>) is on
the server's deny list.
```

The connecting host is on the server's deny list, so the authentication is not allowed.

Authentication not allowed, host is not on the allow list

```
Login not allowed because the connecting host (<hostname>) is
not on the server's allow list.
```

The connecting host is not on the server's allow list, so the authentication is not allowed.

5.1.2 User Authentication - Host-Based

Using host-based authentication denied for a host

```
Use of <filename> denied for <hostname>
```

Use of host-based authentication is denied for the host in server configuration.

Configuration files missing for host-based authentication

```
hostbased-authentication (rhosts) refused for <loginname> no .shosts
or .rhosts files and no system-wide files (ie: /etc/shosts.equiv)
```

The required files for host-based authentication are missing.

 © 2003 SSH Communications Security Corp.

Home directory missing in host-based authentication

```
hostbased-authentication (rhosts) refused for <loginname>: no
home directory <directory>
```

The user's home directory does not exist.

Home directory ownership or permissions invalid in host-based authentication

```
hostbased-authentication (rhosts) refused for <loginname>: bad
ownership or modes for home directory.
```

If the `StrictModes` option for the server is switched on, then this message means that the user's home directory is not owned by either the user or the root, or that its permission bits are not correct.

`.rhosts` file ownership or permissions invalid in host-based authentication

```
hostbased-authentication (rhosts) refused for <loginname>: bad
modes for <file>
```

If the `StrictModes` option for the server is switched on, then this message means that the user's `.rhosts` file is not owned either by the user or by the root, or that its permission bits are not correct.

Host-based authentication refused for user

```
hostbased-authentication (rhosts) refused: client user
<loginname>, server user <loginname>, client host <hostname>.
```

The host-based authentication attempt has failed.

Client's host certificate is not valid in host-based authentication

```
Hostbased: User <loginname>'s client host certificate not
valid.
```

The host certificate sent by the client during host-based authentication could not be successfully validated.

Client's host certificate does not contain host names in host-based authentication

```
Certificate does not contain any hostnames.
```

The host certificate sent by the client during host-based authentication does not contain any host names.

Certificate does not contain the client host name in host-based authentication

```
Certificate does not contain client hostname <hostname>.
```

The host certificate sent by the client during host-based authentication does not contain the client's host name.

Certificate could not be decoded in host-based authentication

```
Received certificate could not be decoded
```

The host certificate sent by the client during host-based authentication could not be decoded.

Public key could not be extracted from the received certificate in host-based authentication

```
Could not extract public key from received certificate
```

Extracting the public key from the received certificate failed during host-based authentication.

Cryptographic operation failed in host-based authentication

```
Hostbased: Crypto operation failed for public key (for user
<loginname>).
```

Changing the hashing scheme in the public key failed in host-based authentication.

Verifying the signature failed in host-based authentication

```
Hostbased authentication failed for <loginname>.
```

Verifying the signature failed in host-based authentication.

 © 2003 SSH Communications Security Corp.

Host-based authentication successful

```
Hostbased authentication for user <loginname> accepted.
```

Authentication using the host-based authentication method succeeded for the user.

Root logins not permitted in host-based authentication

```
Root logins are not permitted.
```

The server has been configured not to allow root logins.

Could not decode packet in host-based authentication

```
Error decoding "hostbased" packet.
```

Decoding the host-based authentication protocol packet failed.

The host name given by the client does not match the one given by the client in host-based authentication

```
Client gave us a hostname (<hostname>) which doesn't match the
one we got from DNS (<hostname>) (sending failure)
```

The server has been configured to use strict host name checking and the host name gotten using reverse DNS lookup does not match the one in the host-based authentication packet.

The host name given by client does not match the one given by the client in host-based authentication, ignored

```
Client gave us a hostname (<hostname>) which doesn't match the
one we got from DNS (<hostname>) (trusting that client is valid, if
signature verification succeeds)
```

The host name gotten using reverse DNS lookup does not match the one in the host-based authentication packet, but the server does not require a match.

Invalid packet received in host-based authentication

```
Invalid packet. (extra data at end)
```

The received packet in host-based authentication contains extra data and will be discarded as invalid.

Client is using a public-key algorithm not supported by server in host-based authentication

```
Client's public key algorithms are not supported by us. (client
sent <algorithmname>)
```

The public key sent by the client in host-based authentication uses a public key algorithm not supported by the server.

User's .ssh2 directory is missing in host-based authentication

```
Couldn't find or create user <loginname>'s .ssh2 directory.
```

The user's `.ssh2` directory could not be found or created in host-based authentication.

Received host key and stored host key differ in host-based authentication

```
The public key stored in %s and the one given by the client were
different.
```

The host key sent by the client and the one stored in the known host keys directory differ in host-based authentication.

Invalid client host key in host-based authentication

```
Importing client pubkey failed.
```

There was an error when decoding the host key sent by the client during host-based authentication.

5.1.3 User Authentication - Keyboard-Interactive PAM

Pipe creation failed in Keyboard-Interactive PAM authentication

```
kbd-int:pam: pipe creation failed.
```

 © 2003 SSH Communications Security Corp.

An error occurred when trying to fork and create a pipe.

Error happened during PAM transaction

```
PAM transaction resulted in error.
```

There was an error during PAM transaction.

Malformed packet received from PAM subprocess

```
kbd-int:pam: PAM subprocess sent a malformed packet. (packet
type: <packet-type>)
```

The PAM subprocess sent a malformed packet of the specified type.

Wrong packet received from PAM subprocess

```
kbd-int:pam: PAM subprocess returned packet
SSH_PAM_OP_SUCCESS_WITH_PAYLOAD, expecting SSH_PAM_OP_SUCCESS.
```

The PAM subprocess sent a packet of wrong type.

Error returned by PAM subprocess

```
kbd-int:pam: PAM subprocess returned packet
<packet-type>. (err_num: <errornumber>, err_msg:
<error-message)
```

The PAM subprocess returned an error.

Invalid packet type returned by PAM subprocess

```
kbd-int:pam: PAM subprocess returned invalid packet type
<packet-type>.
```

The packet type returned by the PAM subprocess was invalid.

5.1.4 User Authentication - Keyboard-Interactive Password

Login with empty password denied for user

```
kbd-int: passwd: login with empty password denied for user
'<login-name>'.
```

The server policy forbids login with empty password.

Login with overtly long password denied

```
kbd-int: passwd: user '<login-name>' tried to login with too
long password (256)
```

The server does not allow logins with passwords longer than 256 characters.

Kerberos password accepted

```
kbd-int: passwd: kerberos password accepted for user
<login-name> (<kerberos-name>).
```

The server accepted the user's Kerberos password in Keyboard-Interactive password authentication.

Password accepted

```
kbd-int: passwd: user <login-name>'s local password accepted.
```

The server accepted the user's local password. This message is only generated on Windows platforms.

Password not accepted

```
kbd-int: passwd: wrong password given for user
'<login-name>'.
```

The password supplied by the client was invalid.

 © 2003 SSH Communications Security Corp.

Password must be changed

```
kbd-int: passwd: user '<login-name>' forced to change
password.
```

The server has determined that the user's password must be changed.

5.1.5 User Authentication - Keyboard-Interactive

Client sent fewer responses than server sent requests

```
auth-kbd-int: client sent us fewer responses than we sent
requests (sent: <sent-requests>, recv: <received-responses>).
```

Protocol error: the client sent fewer responses that the server sent requests.

Login denied, faking transaction

```
auth-kbd-int: User '<login-name>' login denied, faking real
transaction.
```

The user is not allowed to login, but the rest of the transaction proceeds normally, except that all results are marked as failures.

5.1.6 User Authentication - Kerberos TGT

Login denied for user

```
Kerberos TGT authentication as <login-name> denied for
<kerberos-name>
```

The server received a negative result from Kerberos authentication module for logging in as the specified user.

Kerberos V5 ticket-granting ticket rejected for user

```
Kerberos V5 TGT rejected for user <login-name>:
<error-message>
```

The server received a result from the Kerberos authentication module: the user's ticket-granting ticket has been rejected.

The Kerberos TGT authentication succeeded

```
Kerberos TGT authentication as user <login-name> accepted for
<kerberos-name>.
```

The user's login has been accepted by the Kerberos authentication.

5.1.7 User Authentication - Kerberos

Login denied for user

```
Kerberos authentication as <login-name> denied for
<kerberos-name>
```

The server received a negative result from Kerberos authentication module for logging in as the specified user.

Login accepted for user

```
Kerberos authentication as user %.100s accepted for
%.100s.
```

The server received a positive result from Kerberos authentication module for logging in as the specified user.

5.1.8 User Authentication - PAM

Malformed packet from PAM subprocess

```
auths-pam: PAM subprocess sent a malformed packet. (packet type:
%d)
```

The server received a malformed packet from the PAM subprocess.

 © 2003 SSH Communications Security Corp.

Wrong packet received from PAM subprocess

```
auths-pam: PAM subprocess returned packet
SSH_PAM_OP_SUCCESS_WITH_PAYLOAD, expecting SSH_PAM_OP_SUCCESS.
```

The PAM subprocess sent a packet of wrong type.

Error returned by PAM subprocess

```
kbd-int:pam: PAM subprocess returned packet
<packet-type>. (err_num: <errornumber>, err_msg:
<error-message)
```

The PAM subprocess returned an error.

Invalid packet type returned by PAM subprocess

```
auths-pam: PAM subprocess returned packet %d, which is
invalid.
```

The packet type returned by the PAM subprocess was invalid.

Client sent response packet prematurely

```
auths-pam: client sent response packet, and we weren't expecting
any (yet).
```

The client sent a response packet prematurely.

Client sent invalid packet

```
auths-pam: client sent invalid packet as a response to our
conversation message.
```

The client sent an invalid packet.

Client sent continuation packet when expecting start packet

```
Client sent continuation packet when expecting start packet for
PAM-auth..
```

The client sent continuation packet when the server was expecting a start packet.

Extra data in SSH_MSG_USERAUTH_REQUEST packet from client

```
Extra data at end of SSH_MSG_USERAUTH_REQUEST packet for PAM
authentication.
```

Extraneous data was included in the `SSH_MSG_USERAUTH_REQUEST` packet received from client.

Pipe creation failed in PAM authentication

```
ssh_server_auth_pam: pipe creation failed.
```

The forking and pipe creation to the subprocess failed in PAM user authentication.

Error happened during the PAM transaction

```
PAM transaction resulted in error.
```

There was an error during the PAM transaction.

5.1.9 User Authentication - Password

User not allowed to log in after password change

```
User '<login-name>' not allowed to log in after password change
(strange).
```

For some reason, the system disallows login for the user after a mandatory password change.

 © 2003 SSH Communications Security Corp.

Password change required after password change

```
User '<login-name>' password must be changed after password
change (strange).
```

For some reason, the system requires the user to change their password even after a mandatory password change.

Password change failed

```
Password change for user '<login-name>' failed (reason:
<error-message>).
```

Password change failed.

The server does not allow root logins

```
root login denied for user '<login-name>'.
```

The server's configuration doesn't allow root logins or doesn't allow root logins with a password.

The server does not allow logins with empty passwords

```
attempt to login as <login-name> with empty password
denied.
```

The server's configuration does not allow logins with empty passwords.

The server does not allow logins with passwords longer than 256 characters

```
attempt to login as <login-name> with password longer than 256
characters denied.
```

The server does not allow logins with passwords longer than 256 characters.

Kerberos password accepted

```
Kerberos password accepted for user <login-name>
(<kerberos-name>).
```

The server accepted the Kerberos password for the user.

Password accepted

```
User <login-name>’s local password accepted.
```

The server accepted the user’s local password.

Password not accepted

```
Wrong password given for user ’<login-name>’.
```

The password supplied by the client was invalid.

Password authentication accepted

```
Password authentication for user <login-name> accepted.
```

The client passed the password authentication, but the server still needs to check whether the password needs to be changed.

Badly formatted password change request

```
Badly formatted password change reques, denying already
successful authentication.
```

The server received a badly formatted password change request, and the authentication will be noted as failed.

Changing user’s password

```
Changing <login-name>’s password.
```

The server is now trying to change the user’s password.

 © 2003 SSH Communications Security Corp.

Password change needed for the user

```
User <login-name> forced to change password.
```

The server has determined that the user's password must be changed.

5.1.10 User Authentication - Public Key

Certificate not authorized for login

```
Authorization check for user <login-name>'s certificate
rejected[, reason: <error-message>].
```

The server's authorization check for the certificate produced a negative result, meaning that public-key authentication with this certificate is denied.

Error in public key decoded from certificate

```
Couldn't get info from certificate public key used by user
<login-name>.
```

After extracting the public key from the received certificate, the server could not get the public key information from the public key.

Key length too small in certificate

```
Use of certificate for user <login-name> is not allowed, because
public key is too small (pk size <key-size>, min size
<minimum-size>).
```

The server has rejected the authentication using the certificate sent by the client, because the public key in the certificate is too short.

Key length too large in certificate

```
Use of certificate for user <login-name> is not allowed, because
public key is too large (pk size <key-size>, max size
<maximum-size>).
```

The server has rejected the authentication using the certificate sent by the client because the public key in the certificate is too long.

Signature check failed

```
Signature verify failed for certificate from user
<login-name>.
```

The signature verify failed for the user in certificate authentication.

Certificate authentication accepted

```
Certificate authentication for user <login-name> accepted.
```

The server has accepted the certificate authentication for the user.

Certificate authentication rejected

```
Certificate authentication for user <login-name> rejected.
```

The server has rejected the certificate authentication for the user.

Public key options prohibit using the key

```
Use of public key <key-name> is not allowed in public key options for
the host the user <login-name> is logging in from.
```

Using the public key for the user and host has been denied based on the public key options set on the server for the key.

Error in public key

```
Couldn’t get info from public key <key-name> used by user
<login-name>.
```

The server could not get the public key information from the public key.

 © 2003 SSH Communications Security Corp.

Key length too small

```
Use of public key <key-name> by user <login-name> is not
allowed, because public key is too small (pk size <key-size>, min size
<minimum-size>).
```

The server has rejected the authentication using the public key sent by the client because the key is too short.

Key length too large

```
Use of public key <key-name> by user <login-name> is not
allowed, because public key is too large (pk size <key-size>, max size
<maximum-size>).
```

The server has rejected the authentication using the public key sent by the client because the key is too long.

Crypto operation failed for RSA key

```
Crypto operation failed for RSA key of user <login-name>.
```

A cryptographic operation failed for a key.

Root logins not permitted

```
root logins are not permitted.
```

The server has been configured to not allow root logins.

Root login permitted for forced command

```
root login permitted for forced command.
```

The server has been configured to not allow root logins, but will allow forced commands as root.

Public-key authentication accepted

```
Public key authentication for user <login-name> accepted.
```

The server has accepted the public-key authentication for the user.

Bad packet when verifying key

```
got bad packet when verifying user <login-name>’s publickey.
```

The server received an invalid packet when trying to verify the user’s public key.

Public key is authorized

```
Public key <filename> authorized for user <login-name>,
verifying signature.
```

The server has determined that the public key supplied by the client is authorized for a login, and will now verify the signature supplied by the client.

5.1.11 User Authentication - SecurID

The server does not allow root logins

```
root login denied for user ’<login-name>’.
```

The server’s configuration does not allow root logins.

5.1.12 Certificate-specific code

Certificate verification failed

```
Certificate verification failed: search-state = { <search-state>
}
```

Verifying a certificate failed in user or host authentication. <search-state> contains the search status bits that give information about what went wrong.

5.1.13 SFTP Server

Starting SFTP subsystem

```
Starting SFTP-subsystem for user ’<login-name>’ parent sshd2 PID
[<pid-number>].
```

 © 2003 SSH Communications Security Corp.

The SFTP server is starting.

Stopping SFTP subsystem

```
Stopping SFTP-subsystem for user '<login-name>' parent sshd2 PID
[<pid-number>].
```

The SFTP server is stopping.

5.1.14 Agent Forwarding

Agent forwarding requested

```
Agent forwarding requested, creating a listener.
```

The client has requested agent forwarding, and the server is creating a listener for it.

Agent forwarding request has been denied

```
Agent forwarding request denied in configuration.
```

The client has requested agent forwarding, but the server configuration forbids it.

Forwarding an agent connection

```
Opening an agent forwarding channel
```

The server has received an agent connection in the agent forwarding listener, and is opening an agent forwarding channel to the client.

Closing an agent connection

```
Closing an agent forwarding channel
```

The agent forwarding channel from the server to the client has been closed.

5.1.15 Session Channels

Session channel open request received

```
Received a channel open request, type <channel-type>, channel id
<channel-number>
```

Received a session channel open request of the specified type with specified channel id.

Closing session channel

```
Destroying session channel <channel-number>
```

The server is destroying the session channel with the specified id number.

Session channel extension request received

```
Received a session channel extension request of type
<extension-type> for channel number <channel-number>
```

Received a session channel extension request of the specified type for the channel with specified id number.

Setting environment variable denied

```
User tried to set environment variable '<environment-variable>'
to '<variable-value>' in "<filename>", but was denied by
policy.
```

Setting the environment variable denied by server policy.

Opening log for internal SFTP server

```
Opening log for internal sftp-server.
```

The server opened the log for the internal SFTP server.

 © 2003 SSH Communications Security Corp.

Chrooting user

```
User ’<login-name>’ will be chrooted to directory
’<directory-name>’.
```

The server is chrooting the user into a directory.

Changed to run on user privileges

```
Now running on <login-name>’s privileges.
```

The server has dropped root privileges and is now running on user’s privileges.

Setting environment variable denied by public key options

```
User tried to set environment variable ’<environment-variable>’
to ’<variable-value>’ in public key options, but was denied by
policy.
```

Setting the environment variable denied by public key options.

Starting internal SFTP server

```
Starting internal sftp-server for user ’<login-name>’, sshd2 PID
[<pid-number>].
```

The internal SFTP server is starting.

Stopping internal SFTP server

```
Stopping internal sftp-server for user ’<login-name>’, sshd2 PID
[<pid-number>].
```

The internal SFTP server is stopping.

Bad data received in environment variable setting

```
Bad data on ssh_channel_request_env().
```

Decoding an environtment variable setting request failed in the `ssh_channel_request_env` function.

Invalid values received in environment variable setting

```
Invalid values on request to set environment variable:
name='<variable-name>',value='<variable-value>'.
```

The environment variable setting request contained invalid values.

Environment variable setting forbidden by policy

```
Client tried to set environment variable '<variable-name>' to
'<variable-value>', but it is forbidden by policy.
```

The environment variable setting failed because of server policy.

Terminal access denied

```
Terminal access is denied for user '<login-name>'.
```

The server configuration doesn't allow a terminal for the user.

5.1.16 SSH1 Agent Forwarding

SSH1 agent forwarding request has been denied

```
SSH1 agent forwarding request denied in configuration.
```

The client has requested SSH1 agent forwarding, but the server configuration forbids it.

SSH1 Agent forwarding requested

```
Agent forwarding requested, creating a listener.
```

The client has requested agent forwarding, and the server is creating a listener for it.

 © 2003 SSH Communications Security Corp.

Forwarding an SSH1 agent connection

```
Opening an agent forwarding channel
```

The server has received an agent connection in the agent forwarding listener, and is opening an agent forwarding channel to the client.

Closing an SSH1 agent connection

```
Closing an agent forwarding channel
```

The agent forwarding channel from the server to the client has been closed.

5.1.17 Port Forwarding

Direct TCP port forwarding request denied

```
Direct TCP/IP forwarding request (<host>:<port>) denied for user
in configuration.
```

A direct TCP port forwarding request has been denied because of system policy.

Direct TCP port forwarding forbidden

```
Direct TCP/IP forwarding request denied in configuration.
```

A direct TCP port forwarding request was denied because the system policy forbids TCP port forwarding.

Connection received to forwarded port

```
Connection to forwarded port <forwarded-port>
from <host>:<port>
```

The forwarded port received a connection.

Remote port forwarding connect received

```
Remote fwd connect from <connection-originator> to local port
<port>
```

A remote port forwarding request was received.

Remote listener creation failed

```
Creating remote listener for <address>:<port> failed.
```

A remote listener creation failed.

Port set up for remote forwarding

```
Port <port> set up for remote forwarding.
```

A port was successfully set up for remote forwarding.

Received a remote forwarding request

```
Remote TCP/IP forwarding request received from host "<host>", by
authenticated user "<login-name>".
```

Received a remote forwarding request.

Forwarding a privileged port

```
Privileged user "<login-name>" forwarding a privileged
port.
```

Forwarding a privileged port (under 1024) for a privileged user.

Remote TCP port forwarding forbidden

```
Remote TCP/IP forwarding request denied in configuration.
```

A remote TCP port forwarding request was denied because the system policy forbids TCP port forwarding.

 © 2003 SSH Communications Security Corp.

Remote TCP port forwarding request denied

```
Remote TCP/IP forwarding (<host>:<port>) request denied for user
in configuration.
```

A remote TCP port forwarding request has been denied because of system policy.

Connection to direct forwarding received

```
direct fwd connect from <connection-originator> to local port
<port>
```

Received a connection to forwarded port.

5.1.18 X11 Forwarding

X11 connection refused by `tcp_wrappers`

```
Fwd X11 connection from <connection-originator> refused by
tcp_wrappers.
```

The X11 connection was refused by `tcp_wrappers`.

X11 connection received

```
X11 connection from <ip-address>:<port> (<host>)
```

An X11 connection was received.

5.1.19 Common Code

Disconnection happened

```
[Local|Remote host] disconnected: <reason-message>
```

The connection was disconnected by the local or the remote end for the reason given.

5.1.20 Host Key I/O

PIN request received

```
PIN requested, PIN label <pin-label> at keypath <keypath>,
ignoring.
```

A PIN request was received when reading host keys, and since the user is not available at this point, the request is ignored.

5.1.21 General Server Log Messages

Disconnect: Host not allowed to connect

```
Disallowed connect from denied host. '<message>'
```

The server has disconnected the client because the client is not allowed to connect.

Disconnect: Protocol error

```
Protocol error in [local|remote]: '<message>'
```

Disconnecting because of a protocol error in local or remote end.

Disconnect: Key exchange failed

```
Key exchange failed in [local|remote]: '<message>'
```

Disconnecting because key exchange failed in local or remote end.

Disconnect: Illegal code

```
RESERVED (this is an illegal code) in [local|remote]:
'<message>'
```

Disconnecting, with illegal code as reason.

 © 2003 SSH Communications Security Corp.

Disconnect: MAC failed

```
MAC failed in [local|remote], disconnecting: '<message>'
```

Disconnecting because MAC failed in local or remote end.

Disconnect: Compression error

```
compression error in [local|remote], disconnecting:
'<message>'
```

Disconnecting because of a compression error.

Disconnect: Service not available

```
service not available: '<message>'
```

Disconnecting because a requested service was not available.

Disconnect: Protocol version not supported

```
protocol version not supported in [local|remote]:
'<message>'
```

Disconnecting because the protocol version was not supported.

Disconnect: Host key not verified

```
host key not verifiable in [local|remote]: '<message>'
```

Disconnecting because the host key could not be verified.

Disconnect: Connection was lost

```
connection lost: '<message>'
```

Disconnecting because the connection was lost.

Disconnect: Disconnected by application

```
disconnected by application in [local|remote]:
'<message>'
```

Disconnected by application.

Disconnect: Too many connections

```
too many connections : '<message>'
```

Disconnected because the internal connection limit was exceeded.

Disconnect: User authentication failed

```
User authentication failed: '<message>'
```

Disconnected because the user authentication failed. This message should only occur when the other end is of version 2.0.13 or older.

Disconnect: Authentication cancelled by user

```
authentication cancelled by user: '<message>'
```

Disconnected because the authentication was cancelled by the user.

Disconnect: No more authentication methods

```
no more authentication methods on [local|remote]:
'<message>'
```

Disconnected because of running out of authentication methods. This generally means that the client failed in all the authentication methods available on the server.

Disconnect: Illegal user name

```
illegal user name : '<message>'
```

Disconnected because the user name was illegal.

 © 2003 SSH Communications Security Corp.

Disconnect: Unknown reason

```
Unknown reason code '<reason-code>' for disconnect. msg:
'<message>'
```

Disconnected, reason code unknown.

Idle timeout exceeded

```
Idle timeout exceeded.
```

The idle timeout limit was exceeded.

Remote protocol version string received

```
Remote protocol version: <version-string>
```

The server has received the remote protocol version string from the client.

User authenticated

```
User <login-name> (uid <uid-number>), coming from <host>,
authenticated.
```

A general message stating that the user has successfully authenticated herself.

User authenticated, root login

```
ROOT LOGIN: User <login-name> (uid <uid-number>), coming from
<host>, authenticated.
```

General message stating that a superuser has successfully authenticated herself.

User authentication failed

```
Authentication failed for user <login-name>,
coming from <host>.
```

An user tried to log in, but failed the user authentication.

User logged out

```
Logout for user <login-name>.
```

The specified user has now logged out or has been logged out.

Login grace time exceeded

```
LoginGraceTime exceeded.
```

The `LoginGraceTime` value was exceeded. This means that the client did not succeed in logging in during time limit defined by the configuration option `LoginGraceTime`.

Connection received

```
connection from "<host>" (listen port: <listen-port>)
```

The server has received a connection from a host.

Connection received in interface

```
connection from "<host>" (listen iface: <address>:<port>)
```

The server has received a connection from a host to a port on specific interface.

Refusing connection: too many open connections

```
Refusing connection from "<host>". Too many open connections
(max <maximum-connections>, now open <open-connections>).
```

The server has refused a connection because it would have otherwise exceeded the limit of simultaneously open connections.

Denied connection because of `tcp_wrappers`

```
Denied connection from <connection-originator> by tcp
wrappers.
```

The server has refused a connection because it was denied by `tcp_wrappers`.

 © 2003 SSH Communications Security Corp.

Forking for new connection failed

```
Forking a server for a new connection failed.
```

The server tried to fork to handle a new connection, but the forking failed.

Server warning

```
WARNING: <message>
```

A warning was received from the server.

Fatal server error

```
FATAL ERROR: <message>
```

The server has encountered a fatal error condition.

`SIGHUP`-handler received an invalid signal

```
Invalid signal received by SIGHUP-handler.
```

Server's `SIGHUP`-handler received an illegal signal.

License file error

```
License file error: <message>
```

The server encountered an error when checking the license file.

Server restarting

```
restarting...
```

The server is restarting.

Server starting in `inetd` mode

```
Starting daemon in inetd mode.
```

The server is starting in `inetd` mode.

Server created a listener

```
Listener created on  <address>:<port>.
```

The server has successfully created a listener.

Server created an UDP listener

```
UDP Listener created on  <address>:<port>.
```

The server has successfully created an UDP listener.

`Setsid` failed

```
setsid: <error-message>
```

A call to `setsid` failed in the server.

Server running

```
Daemon is running.
```

The server process has successfully performed all initialization operations and is now listening for connections.

5.1.22 SFTP

File closed

```
File '%s' closed (written: <bytes-written>, read:
<bytes-read>).
```

 © 2003 SSH Communications Security Corp.

A file has been closed. This is the only log message of a completed transfer of a file. Typically the whole file is read or written, but this is not the only possibility. This log message will report the sum of all reads and the sum of all writes since the opening of the file. (Note: even the same client can open a file multiple times, which will result in multiple closes, each with possibly different number of read and written bytes.)

Bad message received

```
Received bad message <message-number>.
```

The protocol code received a bad message from the client.

Got wrong message before receiving `INIT`

```
Got message type %d before receiving INIT. This is a protocol
error, destroying the file-server.
```

A protocol error occurred, and the file server is being terminated.

Received `SSH_FXP_INIT`

```
Received SSH_FXP_INIT
```

The file server received the `SSH_FXP_INIT` message from the client.

Could not decode request id

```
Couldn't decode request id. Destroying file-server.
```

A protocol error happened, and the file server is being terminated.

Received `SSH_FXP_OPEN`

```
Received SSH_FXP_OPEN
```

The file server received an `SSH_FXP_OPEN` message from the client.

Received SSH_FXP_CLOSE

```
Received SSH_FXP_CLOSE
```

The file server received an SSH_FXP_CLOSE message from the client.

Received SSH_FXP_READ

```
Received SSH_FXP_READ
```

The file server received an SSH_FXP_READ message from the client.

Received SSH_FXP_WRITE

```
Received SSH_FXP_WRITE
```

The file server received an SSH_FXP_WRITE message from the client.

Received SSH_FXP_STAT

```
Received SSH_FXP_STAT
```

The file server received an SSH_FXP_STAT message from the client.

Received SSH_FXP_LSTAT

```
Received SSH_FXP_LSTAT
```

The file server received an SSH_FXP_LSTAT message from the client.

Received SSH_FXP_FSTAT

```
Received SSH_FXP_FSTAT
```

The file server received an SSH_FXP_FSTAT message from the client.

 © 2003 SSH Communications Security Corp.

Received SSH_FXP_SETSTAT

```
Received SSH_FXP_SETSTAT
```

The file server received an SSH_FXP_SETSTAT message from the client.

Received SSH_FXP_FSETSTAT

```
Received SSH_FXP_FSETSTAT
```

The file server received an SSH_FXP_FSETSTAT message from the client.

Received SSH_FXP_OPENDIR

```
Received SSH_FXP_OPENDIR
```

The file server received an SSH_FXP_OPENDIR message from the client.

Received SSH_FXP_READDIR

```
Received SSH_FXP_READDIR
```

The file server received an SSH_FXP_READDIR message from the client.

Received SSH_FXP_REMOVE

```
Received SSH_FXP_REMOVE
```

The file server received an SSH_FXP_REMOVE message from the client.

Received SSH_FXP_MKDIR

```
Received SSH_FXP_MKDIR
```

The file server received an SSH_FXP_MKDIR message from the client.

Received SSH_FXP_RMDIR

```
Received SSH_FXP_RMDIR
```

The file server received an SSH_FXP_RMDIR message from the client.

Received SSH_FXP_REALPATH

```
Received SSH_FXP_REALPATH
```

The file server received an SSH_FXP_REALPATH message from the client.

Received SSH_FXP_RENAME

```
Received SSH_FXP_RENAME
```

The file server received an SSH_FXP_RENAME message from the client.

Received SSH_FXP_READLINK

```
Received SSH_FXP_READLINK
```

The file server received an SSH_FXP_READLINK message from the client.

Received SSH_FXP_SYMLINK

```
Received SSH_FXP_SYMLINK
```

The file server received an SSH_FXP_SYMLINK message from the client.

Received SSH_FXP_EXTENDED

```
Received SSH_FXP_EXTENDED
```

The file server received an SSH_FXP_EXTENDED message from the client.

 © 2003 SSH Communications Security Corp.

Received bad SSH_FXP_INIT

```
Received bad SSH_FXP_INIT
```

The server received a bad SSH_FXP_INIT message.

Negotiated protocol version

```
Negotiated version <version-number>.
```

The server has negotiated to use the specified protocol version with the client.

Uploading file

```
Uploading '<filename>' (flags [append] [trunc] [creat] [excl])
```

The server has received a write request for the specified file with the specified flags.

Downloading file

```
Downloading '<filename>'
```

The server has received a read request for the specified file.

Failed to open file

```
Failed to open file '<filename>'. Error <errornumber>.
```

The server failed to open the specified file.

Closed file

```
[ Closed | Received error when closing ] file '<filename>'
(handle=<handle>)
```

The server closed the file, possibly with an error happening during the close.

Statting file

```
Statting file '<filename>'
```

The server is statting the specified file.

Lstatting file

```
Lstatting file '<filename>'
```

The server is lstatting the specified file.

Fstatting file

```
Fstatting file '<filename>'
```

The server is fstatting the specified file.

Setstatting file

```
Setstat file '<filename>' ([size] [uidgid] [perms]
[acmodtime])
```

The server is setstatting the attributes of the specified file.

Fsetstatting file

```
Fsetstat file '<filename>' ([size] [uidgid] [perms]
[acmodtime])
```

The server is fsetstatting the attributes of the specified file.

Opening directory

```
Opening directory '<filename>'
```

The server is opening the specified directory.

 © 2003 SSH Communications Security Corp.

Failed to open directory

```
Failed to open dir '<filename>'. Error <errornumber>.
```

The server failed to open the specified directory.

Readdir on directory

```
Readdir on directory '<filename>' (handle=<handle>)
```

The server performed a readdir operation on the specified directory.

Removing file

```
Removing file '<filename>'
```

The server is removing the specified file.

Failed to remove file

```
Failed to remove file '<filename>'. Error <errornumber>.
```

The server failed to remove the specified file.

Creating directory

```
Creating directory '<filename>'
```

The server is creating the requested directory.

Failed to create directory

```
Failed to make dir '<filename>'. Error <errornumber>.
```

The server failed to create the requested directory.

Removing directory

```
Removing directory '<filename>'
```

The server is removing the specified directory.

Failed to remove directory

```
Failed to remove dir '<filename>'. Error <errornumber>.
```

The server failed to remove the requested directory.

Resolving path

```
Resolving path to '<filename>'
```

The server is resolving the path to the specified file.

Renaming file

```
Renaming file '<filename>' to '<new-filename>'
```

The server is renaming the specified file.

Failed to rename file

```
Failed to rename '<filename>'. Error <errornumber>.
```

The server failed to rename the specified file.

Reading link

```
Reading link '<linkname>'
```

The server is reading the specified link.

 © 2003 SSH Communications Security Corp.

Readlink failed

```
Readlink failed for '<linkname>'. Error <errornumber>.
```

The readlink operation failed in the server.

Creating link

```
Creating link '<linkname>' with value '<target>'
```

The server is creating the specified link.

Creating link failed

```
Failed to create symlink '<linkname>'. Error <errornumber>.
```

Creating the specified link failed in the server.

Got `EXTENDED` request

```
got EXTENDED request <request-number>
```

The server received an extended request from the client.

Chapter 6

Using SSH Secure Shell

This chapter provides information on how to use the SSH Secure Shell software suite after it has been successfully installed and set up.

6.1 Using the Secure Shell Server Daemon (`sshd2`)

The server daemon program for Secure Shell is called `sshd2`.

`Sshd2` is normally started at boot time from `/etc/rc.local` or its equivalent. It forks a new daemon for each incoming connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange.

The Secure Shell daemon is normally run as root. If it is not run as root, only the user the daemon is running as will be authorized to log in, and password authentication may not work if the system uses shadow passwords. An alternative host key pair must also be used.

6.1.1 Manually Starting the Secure Shell Server Daemon

To manually start the Secure Shell daemon, type the command `/usr/local/sbin/sshd2`.

Note: If the installation was successfully completed, `sshd` is a symbolic link to `sshd2`. If you also have ssh1 installed, the ssh2 installation process modifies the existing link. If ssh1 compatibility is desired, `sshd2` can be configured to execute `sshd1` when the client only supports ssh1. However, it is recommended to use `sshd2` because `sshd` may start an OpenSSH server.

The sshd2 daemon can be configured using command-line options or a configuration file. Command-line options override values specified in the configuration file.

 © 2003 SSH Communications Security Corp.

6.1.2 Automatically Starting the Server Daemon at Boot Time

If you have installed from RPM packages on RedHat or on SuSE, `sshd2` is already starting at boot time. The same is true if you have installed from `depot` on HP-UX.

In the following sections, two different methods for starting the Secure Shell Daemon at boot time are introduced. If neither of these methods work on your system, consult your operating system documentation on how to start services at boot time.

Starting from `/etc/rc.d/rc.local`

In order to start `sshd2` automatically at boot time on System V based operating systems, the startup script `sshd2` should be located in the `/etc/rc.d/init.d` directory, and there should be symbolic links to `sshd2` startup script in `/etc/rc.d/rc?.d`, where "?" is the runlevel. You can either add these links manually or use `chkconfig`.

Note: `chkconfig` is only available on RedHat distributions. With SuSE, add the symbolic links manually.

If you want to use `chkconfig`, check that the first lines in `sshd2` are similar to the following:

```
#!/bin/sh
#
# Author: Sami Lehtinen <sjl@ssh.com>
#
# sshd2    This shell script takes care of starting
#          and stopping sshd2.
#
# chkconfig: 345 34 70
# description: Secure Shell daemon
#
```

This means that `sshd` will be started in runlevels 3, 4 and 5, its starting priority is 34, and its killing priority is 70. You can choose the runlevels and priorities as you want as long as `sshd` is started after the network is up.

After adding the links manually or giving the command

```
chkconfig --add sshd2
```

you should have the links `/etc/rc.d/rc?.d`, similar to

```
lrwxrwxrwx 1 root  root  14 Aug 16 10:07  S34sshd -> ../init.d/sshd
lrwxrwxrwx 1 root  root  14 Aug 16 10:07  K70sshd -> ../init.d/sshd
```

Starting from /etc/rc.local

On BSD based operating systems, you have to add a similar line to the following to the rc.local file in the /etc directory:

```
echo "Starting sshd2..."; /usr/local/sbin/sshd2
```

After this, the Secure Shell daemon will start automatically at boot time.

6.1.3 Operation of the Server Daemon

When sshd2 is started, it begins to listen on a port for a socket. The default port is port 22, now a well-known port for Secure Shell. This can be changed to suit any custom environments, e.g. if you want to run sshd2 from a non-privileged account; however, make sure that no other process is using the port you are planning to use.

The Secure Shell daemon can also be started from the Internet daemon inetd. For the purpose of this text, it is assumed that sshd2 is not invoked through inetd but started on its own.

When the daemon is listening for a socket, it waits until a client initiates a socket connection. Once connected, the daemon forks a child process, which in turn initiates key exchange with the client. The child process handles the actual connection with the client, including authentication, supported cipher negotiation, encrypted data transfer, and termination of the connection. After the connection has been terminated, the child process terminates as well. The parent process continues listening for other connections until explicitly stopped.

Login Process

When a user successfully logs in, sshd2 performs the following operations:

1. Changes to run with normal user privileges.
2. Sets up a basic environment.
3. Reads /etc/environment if it exists.
4. Changes to the user's home directory.
5. Runs the user's shell or specified command.

 © 2003 SSH Communications Security Corp.

6.1.4 Resetting and Stopping the Server Daemon

When the Secure Shell daemon is started, its process identifier (PID) is stored in `/var/run/sshd2_22.pid` or, if the directory `/var/run` does not exist, in `/etc/ssh2/sshd2_22.pid`. This makes it easy to kill the appropriate daemon:

```
kill ‘cat /var/run/sshd2_22.pid‘
```

or send signals to it:

```
kill -SIGNAL ‘cat /var/run/sshd2_22.pid‘
```

The Secure Shell daemon handles signals like `inetd`: you can send it a `SIGHUP` signal to make it reread its configuration file. The daemon can be stopped by sending the `SIGKILL` signal.

All `sshd` processes can be terminated if needed. This should be done only when root is logged in locally, as the server process for the root user who is remotely logged in will also be terminated. Another option is to start a new `sshd2` daemon on a different port before terminating sshd processes:

```
sshd2 -p 1234
```

Only the sshd processes (e.g. `/usr/local/sbin/sshd`) will be terminated:

```
killall sshd
```

6.1.5 Daemon Configuration File and Command-Line Options

`Sshd2` reads configuration data from `/etc/ssh2/sshd2_config` (or the file specified with the `-f` flag on the command line). The file contains keyword-value pairs, one per line. Lines starting with the number sign (#) as well as empty lines are interpreted as comments.

Subconfiguration files can also be specified in the main configuration file. See Section 3.2 (Subconfigurations) for more information.

For detailed information about the options available in the configuration file and on the command line, please refer to the *sshd2_config*(5) and to the *sshd2*(8) manual pages.

6.1.6 Subsystems

Subsystems can be defined in the `sshd2_config` file using the following syntax.

```
subsystem-<name>      argument
```

The `argument` is the command which will be executed when the subsystem is requested.

```
$ ssh2 -s <name> user@remote
```

The `argument` can be a list of commands separated with semicolon (`;`), or it can, for example, refer to a script.

One example of a subsystem is `sftp`.

```
subsystem-sftp      /usr/local/bin/sftp-server
```

The `sftp` subsystem also has an internal alternative. This should be used for example when the user is chrooted and does not have access to the sftp-server binary.

```
subsystem-sftp      internal://sftp-server
```

6.2 Using the Secure Shell Client (ssh2)

The basic Secure Shell client program is called `ssh2`.

`Ssh2` can be used either to initiate an interactive session, resembling `rlogin`, or to execute a command in a way similar to the `rsh` command.

The Secure Shell client connects to the server on port 22, which is a well-known port for Secure Shell.

6.2.1 Starting the Secure Shell Client

`Ssh2` has a very simple syntax:

```
ssh [options] hostname [command]
```

The most common usage is to establish an interactive session to a remote host. This can be done simply by typing `ssh hostname.domain`. A real-world example could be `ssh root@somehost.ssh.com`. As with `rsh` and `rlogin`, the user ID to be used can also be specified with the `-l` option, for example `ssh -l root somehost.ssh.com`.

Note: As shown in the above example, in the normal case, you do not have to type `ssh2`. The installation process creates a symbolic link, `ssh`, that points to the actual `ssh2` executable. If ssh1 was installed before ssh2, you will need to type `ssh1` to run the ssh1 client.

The `ssh2` command-line options are documented in detail on the *ssh2*(1) manual page.

 © 2003 SSH Communications Security Corp.

6.2.2 Client Configuration File and Command-Line Options

Ssh2 reads configuration data from /etc/ssh2/ssh2_config and from $HOME/.ssh2/ssh2_config (or the file specified with the -F option on the command line). The last-obtained value will prevail.

The file contains keyword-value pairs, one per line. Lines starting with the number sign (#) as well as empty lines are interpreted as comments. For detailed information about the options available in the configuration file and on the command line, please refer to the *ssh2_config*(5) and *ssh2*(5) manual pages.

6.3 Using Secure Copy (scp2)

Scp2 is a program for copying files securely over the network. It uses ssh2 for data transfer, and uses the same authentication and provides the same security as ssh2.

Scp2 uses a sftp2 over ssh2 for data exchange between the client and server. This is not to be confused with FTP.

The basic syntax for scp2 is like this:

```
scp user@source:/directory/file user@destination:/directory/file
```

Note: As shown in the example above, in the normal case, you do not have to type scp2. The installation process creates a symbolic link, scp, that points to the actual scp2 executable. If ssh1 was installed before ssh2 was, you will need to type scp1 to run the ssh1 client.

Scp2 can be used to copy files in either direction; that is, from the local system to the remote system or vice versa. Local paths can be specified without the user@system: prefix. Relative paths can also be used; they are interpreted relative to the user's home directory.

The scp2 command-line options are documented in detail on the *scp2*(1) manual page.

6.4 Using Secure File Transfer (sftp2)

Sftp2 is an FTP-like client that works in a similar fashion to scp2. Just like scp2, sftp2 runs with normal user privileges and uses ssh2 for transport. Even though it functions like ftp, sftp2 does not use the FTP daemon or the FTP client for its connections. The sftp2 client can be used to connect to any host that is running the Secure Shell server daemon (sshd2).

The basic syntax for sftp2 is like this:

```
sftp2 [options] [user@]host
```

Note: As shown in the example above, in the normal case, you do not have to type `sftp2`. The installation process creates a symbolic link, `sftp`, that points to the actual `sftp2` executable. `sftp` was not included in ssh1.

Actual usage of `sftp2` is similar to the traditional `ftp` program.

The `sftp2` command-line options and commands are documented in detail on the *sftp2*(1) manual page.

6.5 Using Authentication Agent (ssh-agent2, ssh-add2)

Ssh-agent2 is a program to hold private keys for authentication. With the *Ssh-add2* command, you can add identities to the authentication agent. When you use the authentication agent, it will automatically be used for public-key authentication. This way, you only have to type the passphrase of your private key once to the agent. Authentication data does not have to be stored on any other machine than the local machine, and authentication passphrases or private keys never go over the network.

Start `ssh-agent2` with the command

```
eval ‘ssh-agent2‘
```

or with the command

```
exec ssh-agent $SHELL
```

After that, you can add identities like this:

```
% ssh-add2 id_dsa_1024_a
Adding identity: id_dsa_1024_a
Need passphrase for id_dsa_1024_a (1024-bit dsa,
      user@localhost, Tue Aug 01 2000 19:41:42).
Enter passphrase:
```

When you connect to a remote host and use public-key authentication, you will get straight in.

If you want the connection to the agent to be forwarded over `ssh` remote logins, you should have this line in your `/etc/ssh2/sshd2_config` file:

```
AllowAgentForwarding            yes
```

The `ssh-agent2` and `ssh-add2` command-line options are documented in detail on the *ssh-agent2*(1) and *ssh-add2*(1) manual pages.

 © 2003 SSH Communications Security Corp.

Appendix A

Tool Syntax

The basic syntax of `ssh-keygen2` and `ssh-certenroll2` is explained here. For more information, see the relevant man pages.

A.1 ssh-keygen2

`ssh-keygen2` is a tool that generates and manages authentication keys for ssh2. Each user wishing to use ssh2 with public-key authentication can run this tool to create authentication keys. Additionally, the system administrator may use this to generate host keys for the server.

(Please note that PKI and PKCS #11 support is only available in commercial distributions of SSH Secure Shell for Servers.)

```
SYNOPSIS

  ssh-keygen2 [-b bits] [-t dsa|rsa] [-c comment_string]
  [-e file] [-p passphrase] [-P] [-h] [-?] [-q] [-1 file] [-i file]
  [-D file] [-B number] [-V] [-r file] [-x file] [-k file]
  [-7 file] [-F file] [key1 key2 ...]


OPTIONS

  -b bits
         Length of the key in bits, for example 1024 bits.

  -t dsa | rsa
         Choose the type of the key. Valid options are
         dsa and rsa.
```

 © 2003 SSH Communications Security Corp.

```
-c comment_string
       Specify the key's comment string.

-e file
       Edit the specified key. Makes ssh-keygen2 interactive.
       You can change the key's passphrase or comment.

-p passphrase
       Specify the passphrase used.

-P     Specify that the key will be saved with an empty
       passphrase.

-h | -?
       Print a short summary of ssh-keygen2 commands.

-q     Hide the progress indicator.

-1 file
       Convert key from ssh1 format to ssh2 format.

-i file
       Load and display information on 'file'.

-D file
       Derive the public key from the private key 'file'.

-B number
       The number base for displaying key information (default 10).

-V     Print version string and exit.

-r file
       Stir in data from file to the random pool.

-x file
       Convert private key from X.509 format to ssh2 format.

-k file
       Convert a PKCS #12 file to an ssh2 format certificate
       and private key.

-7 file
       Extract certificates from a PKCS #7 file.
```

```
  -F file
         Dump the fingerprint of a given public key. The finger-
         print is given in the Bubble Babble format, which
         makes the fingerprint look like a string of "real"
         words (making it easier to pronounce).
```

A.2 ssh-certenroll2

`ssh-certenroll2` allows users to enroll certificates. It will connect to a CA (certification authority) and use the CMPv2 protocol for enrolling a certificate. The user may supply an existing private key when creating the certification request, or allow a new key to be generated.

```
SYNOPSIS
       ssh-certenroll2 [-V ] [-S SOCKS-server] [-P proxy-url] [-g
       ] [-t rsa|dsa] [-l key-size]  [-o base-name]  [-p cmp-ref-
       num:cmp-key] [-e ] -a ca-access-url  -s subject-name   ca-
       cert-file [ private-key ]

OPTIONS
       -V     Print version string and exit.

       -S SOCKS-server
              The SOCKS server URL to be used when connecting  to
              the certification authority.

       -P proxy-url
              The  HTTP proxy server URL to be used when connect-
              ing to the certification authority.

       -g     Generate a new private key.

       -t rsa|dsa
              Type of key to be generated.  Valid types are "rsa"
              or "dsa".  Rsa is the default.

       -l key-size
              The  size  of  the  key  to be generated (in bits).
              1024 is the default.

       -o base
              Specify the base prefix  of  the  generated  files.
              The  private  key (if generated) will be <base>.prv
```

 © 2003 SSH Communications Security Corp.

```
       and the certificate will be <base>-num.crt

-p cmp-ref-num:cmp-key
       Specify the CMP enrollment  reference  number  and
       key (the pre-shared secret).

-e     Enable  extensions  in  the  subject name.  If, for
       example, ip, dns, or email extensions are used, the
       -e flag must be present.

-a  ca-access-url
       The full URL to the certification authority.

-s  subject-dn-name
       Specify  the subject name for the certificate.  For
       example,  "c=ca,o=acme,ou=development,cn=Rami Romi"
       would  specify  the common user name "Rami Romi" in
       the organizational unit "Development" in the  orga-
       nization  "Acme"  in  "ca" (Canada).

-u  number
       Optionally gives the key usage bits.
```

EXAMPLES

1. Enroll a certificate and generate a DSA private key:

```
ssh-certenroll2  -g  -t  dsa  -o  mykey  -p  12345:abcd -S
socks://fw.myfirm.com:1080        -a        http://www.ca-
auth.domain:8080/pkix/  -s  "c=fi,o=acme,cn=Rami Romi" ca-
certificate.crt
```

   This will generate a private key called `mykey.prv` and a certificate called `mykey-0.crt`.

2. Enroll a certificate using a supplied private key and provide an email extension:

```
ssh-certenroll2 -o mykey  -p  12345:ab  -a  http://www.ca-
auth.domain:8080/pkix/       -s       "c=ca,o=acme,cn=Rami
Romi;email=rami@acme.ca"    ca-certificate.crt     my_pri-
vate_key.prv
```

   This will generate and enroll a certificate called `mykey-0.crt`.

Appendix B

Technical Specifications

B.1 Supported Cryptographic Algorithms and Standards

This section lists the supported cryptographic algorithms and standards supported by SSH Secure Shell.

B.1.1 Public-Key Algorithms

The following public-key algorithms are the supported:

- RSA (768-, 1024-, 2048- or 3072-bit key)
- DSA (768-, 1024-, 2048- or 3072-bit key)

B.1.2 Data Integrity Algorithms

The following data integrity algorithms are supported:

- SHA-1 (20-byte key, RFC 2104)
- MD5 (16-byte key, RFC 2104)

B.1.3 Symmetric Session Encryption Algorithms

For symmetric session encryption, the following algorithms are supported:

- AES (128-, 192- or 256-bit key)

 © 2003 SSH Communications Security Corp.

- Blowfish (128-bit key)
- Twofish (128-, 192- or 256-bit key)
- CAST-128 (128-bit key)
- Arcfour (128-bit key)
- 3DES (168-bit key)
- DES

B.1.4 Certificate Standards

The supported certificate standards and drafts are:

- **RFC 2315**, **PKCS #7**: Cryptographic Message Syntax Version 1.5, B. Kaliski, March 1998.
- **RFC 2459**, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, R. Housley, W. Ford, W. Polk, and D. Solo, January 1999.
- **PKCS #12 v1.0**: Personal Information Exchange Syntax, RSA Laboratories, June 1999.
- ***draft-ietf-pkix-rfc2510bis-03.txt*** **(RFC 2510bis)**, Internet X.509 Public Key Infrastructure Certificate Management Protocols, C. Adams and S. Farrel, February 2001.

B.1.5 Certificate and CRL Publishing Solutions

For certificate and CRL publishing, the following solutions are supported:

- **RFC 2585**, Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP, R. Housley, SPYRUS, P. Hoffman, IMC, May 1999.
- **RFC 2559**, Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2, S. Boeyen, T. Howes, and P. Richard, April 1999.
- **RFC 2560**, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, M. Myers, VeriSign, R. Ankney, CertCo, A. Malpani, ValiCert, S. Galperin, My CFO, C. Adams, Entrust Technologies, June 1999.

B.1.6 Cryptographic Token Standards

Supported smart card and cryptographic token related supported standards are:

- **PKCS #11 v2.10**: Cryptographic Token Interface Standard, RSA Laboratories, December 1999.

B.1.7 Other Supported Standards

Other supported standards are:

- **RFC 2460**, Internet Protocol, Version 6 (IPv6) Specification, S. Deering, Cisco, R. Hinden, Nokia, December 1998.

B.1.8 Additional Information

Additional information on RFCs and Internet-Drafts is available at the IETF Web site:

`http://www.ietf.org/`

At the RSA Web site you can find additional information on the Public-Key Cryptography Standards (PKCS):

`http://www.rsasecurity.com/rsalabs/pkcs/`

B.2 Authentication Methods

The following table shows the available authentication methods and limitations on their use.

	Available from binaries	Suitable for scripting	Requires additional software	Requires additional hardware
Password	x			
User public key				
- ssh2 keys	x	x[1]		
- ssh1 keys	x	x[1]		
- PGP keys	x	x[1]	x[2]	
Certificates				
- software	x		x	
- PKCS #11 tokens	x			x
Host-based	x	x		
Kerberos5				
- client			x[3]	
- server			x[3]	
RSA securID				
- client	x			x[4]
- server			x[5]	
PAM				
- client	x			
- server			x[6]	

 © 2003 SSH Communications Security Corp.

If *Available from binaries* is not checked, the authentication method in question can be enabled by compiling from source. See Chapter 4 (Authentication) for instructions.

Explanation:

1. Use *ssh-agent2* or NULL passphrase. Note that in the latter case it is **extremely** important that no one else can access your private key.
2. Only the OpenPGP standard and programs using it are supported.
3. Only Kerberos5 is supported
4. SecurID tokens
5. RSA ACE/Server or RSA ACE/Agent software
6. Only PAM on Linux and on Solaris 2.6 or later is supported.

Index

`.rhosts`, 21, 43, 59, 60
`.shosts`, 21, 40, 43, 59
`.ssh2`, 15, 36, 63
`/etc/passwd`, 35
`/etc/shadow`, 35
`/etc/ssh2`, 15, 32
`%subst%`, 45
3DES, 112
`3des`, 18

`account`, 49
`account-id`, 45
ACE/Agent, 49
ACE/Server, 49
active mode, 28
adding authentication methods, 52
AES, 111
`aes`, 18
agent, 105
agent forwarding, 30
agent forwarding log messages, 76
agent forwarding: SSH1, log messages, 79
allow list, 58
allow list: group, 58
`allow list: host`, 59
`AllowedAuthentications`, 35, 36, 38, 42
`AllowGroups`, 22
`AllowHosts`, 21
`AllowSHosts`, 43
`AllowUsers`, 21
`AnyCipher`, 18
`AnyMac`, 19
`AnyStdCipher`, 18
`AnyStdMac`, 19
Arcfour, 112
`arcfour`, 18
auditing, 57
`auth`, 49
authentication, 31
authentication agent, 105
authentication log messages, 57
authentication log messages: host-based, 59
authentication log messages: Kerberos, 67
authentication log messages: Kerberos TGT, 66
authentication log messages: Keyboard-Interactive, 66
authentication log messages: Keyboard-Interactive PAM, 63
authentication log messages: Keyboard-Interactive password, 65
authentication log messages: password, 69
authentication log messages: public-key, 72
authentication log messages: SecurID, 75
authentication methods, 31, 113
authentication: adding new methods, 52
authentication: certificate, 33, 43
authentication: failed, 57–59
authentication: host-based, 16, 39
authentication: Kerberos, 47
authentication: Keyboard-Interactive, 49, 50
authentication: keyboard-interactive, 35, 48
authentication: PAM, 48, 51
authentication: password, 35, 51
authentication: public-key, 31, 36, 43
authentication: SecurID, 49, 51
authentication: server, 31, 33
authentication: user, 35
`authority info access`, 33, 44
`authorization`, 37–39
automatically starting server daemon, 100

basic configuration, 15
Berkeley services, 11
Blowfish, 112
`blowfish`, 18
BSD, 101

CA (certification authority), 33
CA certificate, 44
`cast`, 18
CAST-128, 112
certificate, 33, 42, 43
certificate authentication, 33, 43
certificate revocation list (CRL), 33, 34, 43

certificate-specific log messages, 75
certificate: revoked, 33
certificate: software, 46
certification authority (CA), 33
channel, 26
channel: session, log messages, 77
`chkconfig`, 100
cipher negotiation, 101
`ciphers`, 17
client command-line options, 104
client configuration file, 104
client program, 103
client-side configuration, 34, 46
client: starting, 103
`command`, 39
command-line options: client, 104
command-line options: daemon, 102
commercial user, 14
common code log messages, 82
compatibility: SSH1, 25
compression, 19
configuration, 15
configuration file locations, 15
configuration file: client, 104
configuration file: daemon, 102
configuration: client-side, 34, 46
configuration: root logins, 20
configuration: server-side, 33, 44
configuration: SSH1 compatibility, 25
configuration: TCP wrappers, 23
converting key format, 34, 46
CRL (certificate revocation list), 33, 34, 43
`CRL distribution point`, 33, 44
CRL: disabling, 34, 44
cryptographic algorithms, 111
customer support, 14

daemon, 25, 99
daemon command-line options, 102
daemon configuration file, 102
daemon: operation, 101
daemon: resetting, 102
daemon: starting, 99, 100
daemon: stopping, 102
data transfer, 101
`DefaultDomain`, 42
deny list, 58
deny list: group, 58
`deny list: host`, 59
`DenyGroups`, 22
`DenyHosts`, 21
`DenySHosts`, 43
`DenyUsers`, 21
deprecation of SSH1 protocol, 13
DES, 112
`des`, 18
Diffie-Hellman key exchange, 31, 33
disabling CRL, 34, 44
`DNS Address`, 33
DNS entry, 43
documentation conventions, 10
draft-ietf-pkix-rfc2510bis-03, 112
DSA, 111
DSA key pair, 32
dynamic port forwarding, 29

eavesdropping, 12
`egrep`, 20
`Email`, 45
`EmailRegex`, 45
encrypted data transfer, 101
encryption, 13
encryption algorithm, 17
encryption: legal issues, 13
`environment`, 101
environment variable, 16
evaluation, 14
evaluation version, 16
export regulations, 13
`EXTENDED`, 98

failed authentication, 57–59
faking network addresses, 12
fallback functionality, 13
file locations, 15
File Transfer Protocol (FTP), 11, 12, 104, 112
fingerprint, 31
firewall, 35
forwarding, 26
forwarding: agent, 30
forwarding: agent, log messages, 76, 79
forwarding: FTP, 28
forwarding: local, 27
forwarding: port, log messages, 80
forwarding: remote, 27
forwarding: X11, 29
forwarding: X11, log messages, 82
`free`, 53
FTP (File Transfer Protocol), 11, 12, 104, 112
FTP forwarding, 28

general server log messages, 83
generating host key, 16
GNU ZLIB (LZ77), 19
GnuPGP, 38
group allow list, 58
group deny list, 58

hijacking, 12
`hmac-md5`, 18
`hmac-md5-96`, 18
`hmac-ripemd160`, 18
`hmac-ripemd160-96`, 19
`hmac-sha1`, 18
`hmac-sha1-96`, 18
`host allow list`, 59
`host deny list`, 59
host key, 31
host key generation, 16
host key I/O log messages, 83
host key: multiple, 32
host-based authentication, 16, 39
host-based user authentication log messages, 59
`hostbased`, 40
`HostCA`, 34, 42
`HostCANoCrls`, 34
`hostkey`, 16, 32
`hostkey.pub`, 16, 32
`hostkeys`, 16
`hosts.allow`, 24
`hosts.deny`, 24
`hosts.equiv`, 43
`HostSpecificConfig`, 22
HP-UX, 100
HTTP (HyperText Transfer Protocol), 112
HTTP proxy, 46
HTTP repository, 33, 43
HTTP tunneling, 26
`HUP`, 25, 40
HyperText Transfer Protocol (HTTP), 112

`identification`, 37, 38, 47
IETF (Internet Engineering Task Force), 12
IETF-secsh Internet-Drafts, 13
IETF-SecSH-draft, 18, 19
incoming tunnel, 27
`inetd`, 89, 101
`INIT`, 90
`init`, 53
Internet Engineering Task Force (IETF), 12
Internet Protocol (IP), 11
Internet Protocol version 6 (IPv6), 113
IP (Internet Protocol), 11
IP spoofing, 12
IPv6 (Internet Protocol version 6), 113

Kerberos, 47
Kerberos authentication log messages, 67
Kerberos TGT authentication log messages, 66
key exchange, 31, 33, 101
key fingerprint, 31
key format: converting, 34, 46
key format: ssh2, 34, 46
key format: X.509, 34, 46
key generation, 16
key: host, log messages, 83
key: PGP, 38
key: PKCS #12, 46
key: private, 105
key: ssh1 format, 38
Keyboard-Interactive authentication, 49, 50
keyboard-interactive authentication, 35, 48
Keyboard-Interactive PAM authentication log messages, 63
Keyboard-Interactive password authentication log messages, 65
Keyboard-Interactive user authentication log messages, 66
`knownhosts`, 16

LDAP (Lightweight Directory Access Protocol), 33, 43, 112
LDAP server, 35, 44
`LdapServers`, 42
legal issues, 13
`libwrap.a`, 24
license file, 16
license: commercial, 14
license: non-commercial, 14
`license_ssh2.dat`, 16
Lightweight Directory Access Protocol (LDAP), 33, 43, 112
Linux, 13
list: allow, 58
`list: allow host`, 59
list: deny, 58
`list: deny host`, 59
list: group allow, 58
list: group deny, 58
listening, 101
local forwarding, 27

 © 2003 SSH Communications Security Corp.

location of files, 15
log message reference, 57
logging, 57
login process, 101
login: restricting, 20
login: root, 20
`LoginGraceTime`, 87

MAC (Message Authentication Code), 18
maintenance, 14
man pages, 9
man-in-the-middle attack, 31, 33
manual pages, 9
manually starting server daemon, 99
`MapFile`, 44
MD5, 111
Message Authentication Code (MAC), 18
message reference: log, 57
Microsoft Windows, 13
multiple host keys, 32

NAT (Network Address Translation), 43
negotiation, 101
Network Address Translation (NAT), 43
network security risks, 11
non-commercial user, 14

OCSP (Online Certificate Status Protocol), 33, 43, 112
Online Certificate Status Protocol (OCSP), 33, 43, 112
OpenPGP standard, 38
operating system, 13
operation of server daemon, 101
`options`, 39
OS/2, 14
outgoing tunnel, 27

PAM (Pluggable Authentication Module), 48, 51, 63
PAM authentication log messages: Keyboard-Interactive, 63
`pam.conf`, 49
`PAM_TTY`, 49
passive mode, 28
passphrase, 36
password authentication, 35, 51
password authentication log messages, 69
password authentication log messages: Keyboard-Interactive, 65
password: shadow, 99
`PermitRootLogin`, 20
permitting root logins, 20
PGP keys, 38
PID (process identifier), 102
PKCS #11 token, 46, 47
PKCS #11 v2.10, 112
PKCS #12 key, 46
PKCS #12 v1.0, 112
PKCS #7, 112
PKCS #7 package, 34, 44
`Pki`, 44
`PkiDisableCRLs`, 44
platforms, 13
Pluggable Authentication Module (PAM), 48, 51, 63
policy: system, 57
POP3 tunneling, 26
port 22, 101, 103
port forwarding, 26
port forwarding log messages, 80
port forwarding: dynamic, 29
private key, 105
process identifier (PID), 102
protocol version, 12, 13, 25
public key, 31
public key ring, 38
public-key authentication, 31, 36, 43
public-key authentication log messages, 72
Public-Key Cryptography Standards (PKCS), 113
`publickey`, 38

`rc.local`, 100, 101
`rcp`, 11, 12
RedHat, 100
reference: log messages, 57
remote forwarding, 27
`RequireReverseMapping`, 59
resetting server daemon, 102
restricting user login, 20
reverse mapping, 59
revoked certificate, 33
`rexec`, 12
RFC 1950, 19
RFC 1951, 19
RFC 2315, 112
RFC 2459, 112
RFC 2460, 113
RFC 2510bis, 112
RFC 2559, 112
RFC 2560, 112
RFC 2585, 112
`rhosts`, 21, 59, 60
`rlogin`, 11, 12, 103
root login, 20

RSA, 49, 111
RSA ACE/Agent, 49
RSA ACE/Server, 49
RSA key pair, 32
RSA SecurID, 49
`rsh`, 11, 103

`scp`, 104
`scp1`, 104
`scp2`, 19, 104
Secsh Working Group, 12
secure copy, 104
secure file transfer, 104
Secure Shell, 11
Secure Shell version 1 protocol (SSH1), 13
Secure Shell version 2 protocol (SSH2), 12, 13
SecurID, 49, 51
SecurID authentication log messages, 75
security policy: Xserver, 30
security risks, 11
`SerialAndIssuer`, 45
server authentication, 31, 33
server certificate, 33
server daemon, 25, 99
server daemon: operation, 101
server daemon: resetting, 102
server daemon: starting, 99, 100
server daemon: stopping, 102
server deny list, 58
server log messages, 83
server-side configuration, 33, 44
server: SFTP, log messages, 75
`session`, 49
session channel related log messages, 77
`setsid`, 89
`sftp`, 103, 105
SFTP log messages, 89
SFTP server log messages, 75
`sftp2`, 104
SHA-1, 111
shadow password, 99
`shosts`, 21, 40, 59
`shosts.equiv`, 43, 59
`SIGHUP`, 88, 102
`SIGKILL`, 102
smart card, 46, 47
SMTP tunneling, 26
socket connection, 101
SOCKS server, 29, 35, 45, 46
software certificate, 46
software version, 13
spoofing, 12
`ssh`, 103
SSH Secure Shell, 11
`ssh-add2`, 105
`ssh-agent2`, 105
`ssh-certenroll2`, 33, 46, 109
`ssh-keygen1`, 38
`ssh-keygen2`, 17, 34, 36, 37, 44, 46, 107
`ssh1`, 103
SSH1 agent forwarding log messages, 79
SSH1 compatibility, 25
ssh1 key format, 38
SSH1 protocol, 13, 25
`ssh2`, 19, 63, 103
ssh2 key format, 34, 46
SSH2 protocol, 12, 13, 25
ssh2 server daemon, 25
`ssh2(1)`, 29
`ssh2_config`, 15, 17–20, 32–36, 38, 40, 42, 46, 47, 104
`ssh_channel_request_env`, 79
`SSH_FXP_CLOSE`, 91
`SSH_FXP_EXTENDED`, 93
`SSH_FXP_FSETSTAT`, 92
`SSH_FXP_FSTAT`, 91
`SSH_FXP_INIT`, 90, 94
`SSH_FXP_LSTAT`, 91
`SSH_FXP_MKDIR`, 92
`SSH_FXP_OPEN`, 90
`SSH_FXP_OPENDIR`, 92
`SSH_FXP_READ`, 91
`SSH_FXP_READDIR`, 92
`SSH_FXP_READLINK`, 93
`SSH_FXP_REALPATH`, 93
`SSH_FXP_REMOVE`, 92
`SSH_FXP_RENAME`, 93
`SSH_FXP_RMDIR`, 93
`SSH_FXP_SETSTAT`, 92
`SSH_FXP_STAT`, 91
`SSH_FXP_SYMLINK`, 93
`SSH_FXP_WRITE`, 91
`SSH_LICENSE_FILE`, 16
`SSH_MSG_USERAUTH_REQUEST`, 69
`SSH_PAM_OP_SUCCESS`, 64
`SSH_PAM_OP_SUCCESS_WITH_PAYLOAD`, 64
`SSH_REGEX_SYNTAX_EGREP`, 20, 45
`SSH_REGEX_SYNTAX_ZSH_FILEGLOB`, 21
`sshd`, 99
`sshd2`, 15, 25, 32, 48, 99, 101, 102

 © 2003 SSH Communications Security Corp.

`sshd2_22.pid`, 102
`sshd2_config`, 15, 17, 18, 20, 23, 25, 32, 34–36, 38–40, 42–45, 47, 102
`sshd2_subconfig`, 23
`SshKbdIntSubMethodCB`, 54
`SshKbdIntSubMethodConv`, 54
`sshregex`, 20
standards, 111
starting client, 103
starting server daemon, 99, 100
stopping server daemon, 102
`StrictHostKeyChecking`, 32
`StrictModes`, 60
subconfigurations, 22
`Subject`, 45
`SubjectRegex`, 45
subsystem, 102
support, 14
supported cipher negotiation, 101
supported platforms, 13
supported standards, 111
SuSE, 100
Symbian OS, 14
symmetric encryption, 17
system configuration, 15
system policy, 57
System V, 100

taking over a communication, 12
TCP traffic tunneling, 26
TCP wrappers, 23
TCP/IP, 11
`tcp_wrappers`, 82, 87
`tcpd`, 24
`tcpdchk`, 25
`tcpdmatch`, 25
technical support, 14
Telnet, 11, 12
termination of connection, 101
TGT (ticket granting ticket), 47
third-party products, 14
ticket granting ticket (TGT), 47
`traditional`, 20
transfer, 101
trusted CA, 43
TTY allocation, 49
tunnel: incoming, 27
tunnel: outgoing, 27
tunneling, 26
tunneling log messages, 80
tunneling: agent, 30
tunneling: FTP, 28
tunneling: X11, 29
tunneling: X11, log messages, 82
Twofish, 112
`twofish`, 18

UNIX, 13
usage, 99
user authentication, 35
user authentication log messages, 57
user authentication log messages: host-based, 59
user authentication log messages: Kerberos, 67
user authentication log messages: Kerberos TGT, 66
user authentication log messages: Keyboard-Interactive, 66
user authentication log messages: Keyboard-Interactive PAM, 63
user authentication log messages: Keyboard-Interactive password, 65
user authentication log messages: password, 69
user authentication log messages: public-key, 72
user authentication log messages: SecurID, 75
user certificate, 43
user login: restricting, 20
`UserConfigDirectory`, 39
`UserSpecificConfig`, 23
using authentication agent, 105
using secure copy, 104
using secure file transfer, 104
using server daemon, 99
using the client, 103

`VAR_ACE`, 50
version number, 13
VMS, 14
vulnerability of SSH1 protocol, 13

well-known port, 101
Windows 2000, 13
Windows 98, 14
Windows Me, 14
Windows NT, 13
Windows XP, 13

X.509 certificate, 44
X.509 key format, 34, 46
X11, 11
X11 connection, 26
X11 forwarding, 29
X11 forwarding log messages, 82

X11 SECURITY extension, 30
`xauth`, 30
Xserver: security policy, 30

zlib compression, 19
`zsh_fileglob`, 20

 © 2003 SSH Communications Security Corp.