



# SSH Secure Shell for UNIX Servers Administrator's Guide

May 2001

This document contains instructions on the basic administrator tasks of SSH Secure Shell.

© 1995 - 2001 SSH Communications Security Oyj.

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 Oyj.

This software is protected by international copyright laws. All rights reserved. ssh® is a registered trademark of SSH Communications Security Oyj in the United States and in certain other jurisdictions. SSH2, the SSH logo, SSH IPSEC Express, SSH Certifier, SSH Sentinel and Making the Internet Secure are trademarks of SSH Communications Security Oyj 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 Oyj**

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/ssh/> (support)  
Tel: +358 20 500 7030 (Finland), +1 650 251 2700 (USA), +81 3 3459 6830 (Japan)  
Fax: +358 20 500 7031 (Finland), +1 650 251 2701 (USA), +81 3 3459 6825 (Japan)

# Contents

<b>1</b>	<b>About This Document</b>	<b>7</b>
<b>2</b>	<b>Introduction to SSH Secure Shell</b>	<b>9</b>
2.1	SSH Secure Shell . . . . .	9
2.2	Supported Platforms . . . . .	9
2.3	Different Versions of the SSH Protocol . . . . .	10
2.4	Support . . . . .	10
2.5	Legal Issues with Encryption . . . . .	11
<b>3</b>	<b>Configuring SSH Secure Shell</b>	<b>13</b>
3.1	Basic Configuration . . . . .	13
3.1.1	Default Locations of Secure Shell Files . . . . .	13
3.1.2	Generating the Host Key . . . . .	14
3.1.3	Ciphers and MACs . . . . .	14
3.1.4	Compression . . . . .	15
3.1.5	Configuring Root Logins . . . . .	15
3.1.6	Restricting User Logins . . . . .	16
3.2	Configuring SSH Secure Shell for TCP Wrappers Support . . . . .	18
3.3	Configuring SSH2 for SSH1 Compatibility . . . . .	19
3.4	Forwarding . . . . .	20

---

3.4.1	X11 Forwarding . . . . .	20
3.4.2	Port Forwarding . . . . .	21
3.4.3	Agent Forwarding . . . . .	22
<b>4</b>	<b>Authentication</b>	<b>23</b>
4.1	Server Authentication . . . . .	23
4.1.1	Public-Key Authentication . . . . .	23
4.1.2	Certificate Authentication . . . . .	24
4.2	User Authentication . . . . .	26
4.2.1	Password . . . . .	26
4.2.2	Public-Key Authentication . . . . .	26
4.2.3	Host-Based Authentication . . . . .	30
4.2.4	Certificate Authentication . . . . .	34
4.2.5	Kerberos Authentication . . . . .	37
4.2.6	Pluggable Authentication Modules (PAM) . . . . .	38
4.2.7	SecurID . . . . .	39
<b>5</b>	<b>Using SSH Secure Shell</b>	<b>41</b>
5.1	Using the Secure Shell Server Daemon (sshd2) . . . . .	41
5.1.1	Manually Starting the Secure Shell Server Daemon . . . . .	41
5.1.2	Automatically Starting the Secure Shell Server Daemon at Boot Time . . . . .	42
5.1.3	Operation of the Server Daemon . . . . .	43
5.1.4	Resetting and Stopping the Server Daemon . . . . .	43
5.1.5	Configuration File and Command-Line Options . . . . .	44
5.1.6	Subsystems . . . . .	44
5.2	Using the Secure Shell Client (ssh2) . . . . .	44
5.2.1	Starting the Secure Shell Client . . . . .	45

---

5.2.2 Configuration File and Command-Line Options . . . . .	45
5.3 Using Secure Copy (scp2) . . . . .	45
5.4 Using Secure File Transfer (sftp2) . . . . .	46
5.5 Using Authentication Agent (ssh-agent2, ssh-add2) . . . . .	46
5.6 Using Chroot Manager (ssh-chrootmgr) . . . . .	47
5.7 Using Public-Key Manager (ssh-pubkeymgr) . . . . .	48
<b>6 Appendix</b>	<b>51</b>
6.1 Supported Cryptographic Algorithms and Standards . . . . .	51
6.2 Authentication Methods . . . . .	52



# Chapter 1

## About This Document

This document, *SSH Secure Shell for Unix Servers Administrator's Guide*, contains instructions on the basic administrator tasks of SSH Secure Shell. The document is intended for the system administrators responsible for the configuration of the 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
- using SSH Secure Shell
- appendix (list of supported standards and authentication methods)

Installation instructions for the SSH Secure Shell software can be found in *SSH Secure Shell Quick Start Guide*, included in the CD-ROM package. For more information, see the manual pages included in the distribution.



## Chapter 2

# Introduction to SSH Secure Shell

This chapter provides an introduction to the SSH Secure Shell software suite.

### 2.1 SSH Secure Shell

SSH Secure Shell software allows secure network services over an insecure network such as the public Internet. The Secure Shell concept originated on Unix as a replacement for the insecure "Berkeley services", that is, the `rsh`, `rlogin`, and `rcp` commands. It replaces also Telnet and FTP. SSH Secure Shell can be used for remote terminal connections, remote file copying, and forwarding X11 sessions (on Unix) as well as arbitrary TCP ports through a secure tunnel.

SSH Secure Shell provides strong encryption and authentication. The software has been developed in Europe and can be used in any country that allows encryption.

### 2.2 Supported Platforms

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 required) and Windows 2000. The associated client software (the component that remote users run on their computers) is also available for Microsoft Windows 95 and Windows 98. A list of the officially supported platforms is available at <http://www.ssh.com/products/ssh/portability.html>.

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 SSH Secure Shell products. However, SSH Communications Security can only provide support for its own software.

## 2.3 Different Versions of the SSH Protocol

**Note:** SSH Communications Security considers the SSH protocol version 1 (SSH1) deprecated and does not recommend the use of it. SSH statement regarding the vulnerability of SSH1 protocol is available at <http://www.ssh.com/products/ssh/cert/>.

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

Several different versions of the Secure Shell client and server exist. Please note that the different versions use different implementations of the SSH protocol, and therefore you cannot normally connect to an SSH1 server using SSH2 client software, or vice versa. However, the SSH Secure Shell Unix server software includes support for fallback functionality if SSH Secure Shell version 1.x is already installed. The Windows server software does not include this functionality.

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

## 2.4 Support

If you are a commercial user, you are entitled to support from SSH Communications Security for ninety days (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 customer support by filling out a proper support request form at <http://www.ssh.com/support/ssh/>.

- Pre-sales support  
[http://www.ssh.com/support/ssh/pre-sales\\_support.html](http://www.ssh.com/support/ssh/pre-sales_support.html)
- Warranty and maintenance support  
[http://www.ssh.com/support/ssh/commercial\\_support.html](http://www.ssh.com/support/ssh/commercial_support.html)

Non-commercial users are welcome to submit bug reports and feature requests, but are not entitled to support from SSH Communications Security.

- Bug Reports  
<http://www.ssh.com/support/toolkits/bug-report.html>
- Feature Requests  
<http://www.ssh.com/support/toolkits/feature-request.html>

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

## **2.5 Legal Issues with Encryption**

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

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



# Chapter 3

## Configuring SSH Secure Shell

This chapter gives instructions on configuring SSH Secure Shell.

### 3.1 Basic Configuration

This section goes into some of the basic configuration options that many administrators like to have set up (or not set up, depending on the scenario).

#### 3.1.1 Default Locations of Secure Shell Files

The system files for SSH2 are in `/etc/ssh2`. The user and system binaries are stored in `/usr/local/bin` and `/usr/local/sbin`, respectively. In `/usr/local/sbin`, you will find `sshd2`. All the other binaries are stored in `/usr/local/bin`.

The system-wide configuration files are the most important. They are in `/etc/ssh2/`.

The system public key pair (DSS only):

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

The configuration files for the client and server, respectively:

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

Users can have their own configuration files (and other files as well). These are stored in `~/.ssh2`.

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

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

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

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

### **3.1.2 Generating the Host Key**

You only need to do this if you want to change your host key, or if your host key was not generated during the installation.

1. Login as `root`
2. Kill any instances of `sshd2` or `sshd`:

```
killall sshd
```

3. Generate the host key with the following command:

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

4. Restart `sshd2`:

```
sshd2
```

### **3.1.3 Ciphers and MACs**

The algorithm(s) used for symmetric encryption of the session, can be specified in the `sshd2_config` and `ssh2_config` file.

Ciphers	twofish,blowfish
---------	------------------

Currently supported ciphers are `des`, `3des`, `blowfish`, `twofish`, `idea`, `cast`, `arcfour`, and `aes`, of which all other than IDEA are included in all distributions. Of these ciphers, Blowfish and Twofish are especially suitable for file transfers.

The MAC (Message Authentication Code) algorithm(s) used for data integrity verification, can also be specified in the `sshd2_config` and `ssh2_config` file.

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

Currently `hmac-sha1`, `hmac-sha1-96`, `hmac-md5`, `hmac-md5-96`, `hmac-ripemd160`, and `hmac-ripemd160-96` are supported.

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:** The name of the algorithms are case sensitive.

### 3.1.4 Compression

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

As default compression is disabled. It can be enabled either in the `ssh2_config` file

```
Compression      yes
```

or in the command line.

```
$ ssh2 +C user@remote
```

Compression is worth using if your connection is slow (for example modem connection). Efficiency of the compression depends on the type of the file. It is close to 0% for already packed files like MP3 and 50% or even more for text files.

### 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      nöpwd
```

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

### 3.1.6 Restricting User Logins

As default all connections are allowed. 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`

Available keywords:

- `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      t..l.\..*
```

This pattern matches with, for example, `taulu.foobar.com`, `tuoli.com`, but not `tuoli1.com`. Note that you have to input string "\ ." when you want it to match only a literal dot.

**Example 3.** Using patters with IP-addresses.

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

This pattern matches with any IP address (*x.y.z.q*). 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 host IP-addresses are used. The above pattern matches, for example, with 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,31373
```

This denies login as `devil` when connection is coming from `evil.org`. It also denies login (from all addresses) as `warezdude` and as user whose UID is `31373`.

**Example 2** Using patterns with user names

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

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

**Example 3** Using \i.

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

This would allow login as user `sjl` to connect only from those hosts whose IP-address matches with 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
```

More information on keywords is available in the `sshd2_config` man pages.

**Note:** All the patterns used in above examples are in accordance with `SSH_REGEX_SYNTAX_EGREP`, which is the default regex syntax in SSH Secure Shell version 3.0. 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`. More information is available in the `sshregex` man pages.

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

## 3.2 Configuring SSH Secure Shell for TCP Wrappers Support

To enable usage of TCP Wrappers with SSH Secure Shell, do the following as root:

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
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
```

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 file 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 in the `/etc/hosts.deny` (This example shows both SSH1 and SSH2):

```
sshd1: ALL
sshd2: ALL
sshd fwd-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
sshd fwd-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 get in.

### Troubleshooting

1. Make sure that you are not having any network 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 get a 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.

## 3.3 Configuring SSH2 for SSH1 Compatibility

**Note:**SSH Communications Security considers the version 1 protocol deprecated and does not recommend the use of it (<http://www.ssh.com/products/ssh/cert/>).

The SSH2 and SSH1 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 are designed so that they can both be run on the same machine. 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.

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. (As of this printing, the latest version is `ssh-1.2.30`.)  
The 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.
4. Make sure that /usr/local/sbin/sshd2 is run automatically at boot. On most systems, you should add running it into /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 SSH1 as needed.
5. If you do not want to reboot, you should now manually run /usr/local/sbin/sshd2.

## **3.4 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.

### **3.4.1 X11 Forwarding**

To enable X11 forwarding, make sure that the SSH Secure Shell software was compiled with X (you did not run ./configure with any X disabling options). Also, make sure that you have this line in your /etc/ssh2/sshd2.config file:

```
ForwardX11 yes
```

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 fine.

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.)

To forward X11 traffic on the SSH Secure Shell for Workstations Windows client:

1. Install an X server (X emulation) program on Windows (eXceed, Reflection, or the like)
2. Start the SSH Secure Shell for Workstations Windows client
3. Select **Edit -> Settings... -> Tunneling** and make sure that the **Forward X11 connections** checkbox is checked
4. Save your settings for the SSH Secure Shell for Workstations Windows client

5. Quit the Windows client, start it again and log into the remote host
6. Start the X server (X emulation) program
7. Run `xterm` or `xclock` from SSH Secure Shell, and it should work.

### 3.4.2 Port Forwarding

Port forwarding, in other words tunneling, is a way to forward otherwise insecure TCP traffic through SSH Secure Shell. For example, you can secure POP3, SMTP and HTTP connections that would otherwise be insecure (see Figure 3.1 (Encrypted ssh2 tunnel)).

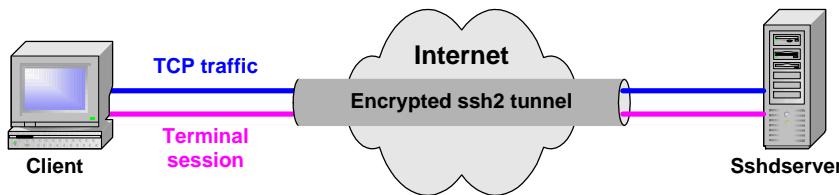


Figure 3.1: Making insecure TCP connections secure using channels inside the encrypted ssh2 tunnel

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 local_port:remote:remote_port user@remote
```

all traffic which comes to port `local_port` on the local host will be forwarded to port `remote_port` on the remote host.

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 remote_port:local:local_port user@remote
```

all traffic which comes to port `remote_port` on the remote host will be forwarded to port `local_port` on the local host.

If you have three hosts, `client`, `sshdserver`, and `appserver`, and you forward the traffic coming to `client`'s port `x` to `appserver`'s port `y` but you connect to `sshdserver` only, the connection between `client` and `sshdserver` is secure. See Figure 3.2 (Forwarding to a third host). The command you use would look like the following:

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

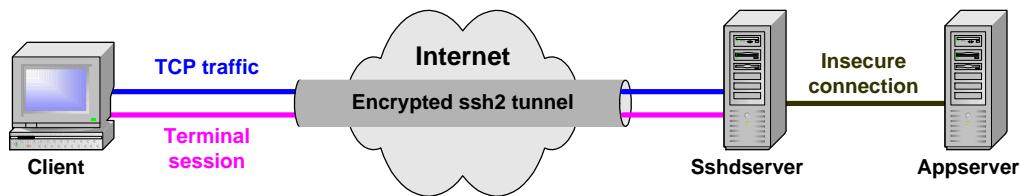


Figure 3.2: Forwarding to a third host

### Forwarding FTP

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

```
ssh2 -L FTP/x:ftpdserver:y user@sshdserver
```

Both the ftp data and control connection will be secured.

If `sshd2` and `ftpd` are on different machines, FTP forwarding works only if `ftp` is running in passive mode. However, if `sshd2` and `ftpd` are on the same machine, the forwarding works regardless of whether `ftp` is running in passive or active mode.

### 3.4.3 Agent Forwarding

See Section 5.5 (Using Authentication Agent).

# Chapter 4

## Authentication

This chapter contains instructions on using different authentication methods with SSH Secure Shell. Please refer to Section 6.2 (in the Appendix) for the list of the available authentication methods.

### 4.1 Server Authentication

#### 4.1.1 Public-Key Authentication

When public-key authentication is used to authenticate the server, the first connection is very important. During that you will get a message similar to following.

```
Host key not found from database.  
Key fingerprint:  
xezop-fomas-lifot-pisoc-zyvik-hutoz-bafaf-zapyc-lubev-riked-dexax  
You can get a public key's fingerprint by running  
% ssh-keygen -F publickey.pub  
on the keyfile.  
Are you sure you want to continue connecting (yes/no)?
```

At this point, you should verify that the fingerprint is correct. If the fingerprint is not verified, it is possible that the server you are connecting to is not the real one (*man-in-the-middle attack*). After verifying the fingerprint, it is safe to continue connecting. The server's public key will then be stored in the `~/.ssh2/hostkeys` directory on the client machine. After the first connection the local copy of server's public key will be used in server authentication which is done during Diffie-Hellman key exchange through a single public-key operation.

If you want to avoid the risk of the first connection, you can copy the server public key in advance to the `/etc/ssh2/hostkeys` directory on the client machine and set the `StrictHostKeyChecking` key-

word in the `ssh2_config` file to yes. After this, `ssh2` refuses to connect if server's public key is not in the `/etc/ssh2/hostkeys` directory.

The key pair used for server authentication is defined in the `sshd2_config` file.

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

During installation one DSA key pair (`hostkey` and `hostkey.pub`) is generated to the `/etc/ssh2/` directory. As default this key pair is used for server authentication.

Starting from SSH Secure Shell version 3.0 server can have multiple host keys. It can have one DSA and one RSA key pair. You could have, for example, the following set of settings 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 read in memory when the `sshd2` is started which means that either one of them can be used to authenticate the server.

### **4.1.2 Certificate Authentication**

Server authentication is done during the Diffie-Hellman key exchange and in brief 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 has 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, it is impossible to fall victim of the man-in-the-middle attack, because the server certificate is checked to have been issued by a trusted CA.

To use certificates for server authentication, do the following:

1. Enroll a certificate for the server. Note that the DNS Address needs to be 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.
```

2. Server's private key must be in ssh2 format. To convert X.509 keys to ssh2 format, use -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 using -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 successfully.
Got certificate.
Certificate written to file <keyname>_ssh2.crt
```

3. 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 using -7 flag with *ssh-keygen2*.

4. Define the CA certificate(s) used in host authentication in the *ssh2\_config* file:

`HostCA <ca-certificate>`

Only one CA certificate can be defined per `HostCA` keyword.

You can disable usage of CRLs by using `HostCANoCrls` keyword instead of `HostCA`.

`HostCANoCrls <ca-certificate>`

**NOTE:** CRL usage should only be disabled for testing purposes. Otherwise it's always highly recommended to use CRLs.

5. Define also the LDAP server(s) in the `ssh2_config` file.

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

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

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

**Note:** Only the commercial version of SSH Secure Shell includes certificate support.

## 4.2 User Authentication

There are several different methods 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.

### 4.2.1 Password

This 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 your UNIX system, depending on how your passwords are set up.

To make sure password authentication is enabled, the `AllowedAuthentications` field both in `/etc/ssh2/sshd2_config` and `/etc/ssh2/ssh2_config` files should contain the word `password`:

```
AllowedAuthentications password
```

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

### 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, either use the Public-Key Manager, ssh-pubkeymgr, or do a manual setup according to the following instructions. See Section 5.7 (Using Public-Key Manager) if you want to know more about ssh-pubkeymgr.

1. To make sure that public-key authentication is enabled, the AllowedAuthentications field both in /etc/ssh2/sshd2\_config file on *Remote* and in /etc/ssh2/sshd2\_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 1024-bit dsa key pair
1 oOo.oOo.o
Key generated.
1024-bit dsa, user@Local, Wed Mar 22 2000 00:13:43 +0200
Passphrase :
Again :
Private key saved to /home/user/.ssh2/id_dsa_1024_a
Public key saved to /home/user/.ssh2/id_dsa_1024_a.pub
```

Ssh-keygen2 will ask you for a passphrase for the new key. Enter a sufficiently long (20 characters or so) sequence of any characters (white spaces are OK). Ssh-keygen2 creates a .ssh2 directory in your home directory, and stores your new authentication key pair in two separate files. One 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 above example, the private key file is id\_dsa\_1024\_a. The other file id\_dsa\_1024\_a.pub is your public key, which can be distributed to other computers.

**Note:** SSH Secure Shell 3.0 includes support for RSA keys. They can be generated by using the -t flag with ssh-keygen2.

```
Local> ssh-keygen2 -t rsa
Generating 1024-bit rsa key pair
2 oOo.oOo.oOo
Key generated.
1024-bit rsa, user@Local, Wed May 02 2001 14:15:41 +0300
Passphrase :
Again :
Private key saved to /home/user/.ssh2/id_rsa_1024_a
Public key saved to /home/user/.ssh2/id_rsa_1024_a.pub
```

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

```
Local> cd ~/.ssh2
Local> echo "IdKey id_dsa_1024_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-keygen` again. This is, however, not needed in the most common cases.

4. Copy your public key (`id_dsa_1024_a.pub`) to the `~/.ssh2` directory on *Remote*.
5. Create an authorization file in your `~/.ssh2` directory on *Remote*. Add the following line to `authorization`:

```
Key      id_dsa_1024_a.pub
```

This directs the SSH Secure Shell server to use `id_dsa_1024_a.pub` as a valid public key when authorizing your login. If you want to login to *Remote* from other hosts, create authorization keys on the hosts (steps 1 and 2) and repeat steps 3 and 4 on *Remote*.

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

Try to login:

```
Local>ssh Remote
Passphrase for key "/home/user/.ssh2/id_dsa_1024_a
with comment "1024-bit dsa, created by user@Local
Wed Mar 22 2000 00:13:43 +0200":
```

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

### Keys Generated with `ssh-keygen1`

SSH Secure Shell version 3.0 enables the usage of keys generated with `ssh-keygen1`. However, the keys need to be first converted from 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

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

- To make sure that user public-key authentication is enabled, the `AllowedAuthentications` field both in `/etc/ssh2/sshd2_config` file on *Remote* and `/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.

- Copy your private key ring (`secring.gpg`) to the `~/.ssh2` directory on *Local*.
- Create an identification file in your `~/.ssh2` directory on *Local* if you do not already have one. Add the following lines to `identification`:

```
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>
```

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

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

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

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

- 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 Extra Configuration

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

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

When someone now logins using master key, the connection is not limited in anyway by authorization file. However, if maid key is used, only connections from certain hosts will be allowed. And if butler key is used, connections are denied from certain hosts and in addition 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) keyword 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.

### 4.2.3 Host-Based Authentication

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. First, install SSH Secure Shell on the *Local* and *Remote* machines. Do not forget to generate a host key. If your installation did this, 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, do this:

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

**Note:** In SSH Secure Shell version 3.0 and later 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 /etc/ssh2/knownhosts/hostname.domain.ssh-dss.pub

In the place of *hostname.domain* above, you must use 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 make 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 the *Local* machine's /etc/ssh2/hostkey.pub file over to the *Remote* machine need 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
```

4. On the *Remote* machine, 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 the user name of *LocalUser*.

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,passwd
```

It does not matter what else is in there. Just make sure that the `hostbased` keyword is first in the list.

6. Also check that `IgnoreRhosts` is set to `no` in your `/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. You should be all set.

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

```
ssh RemoteUser@Remote uptime
```

You should get back 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. This will only happen when connecting for the first time.

## Troubleshooting

- Did you name the host key file appropriately? 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.

- 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 the home directory is owned by *RemoteUser*.
- Run the server 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. In brief, this is what you need to do:

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` in `ssh2_config` on *Local*.
4. Set `HostCA` and `LdapServers` in `sshd2_config` on *Remote*.

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

5. Make sure that the contents of `~/.shosts` on *Remote* is

```
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 Extra Configuration

To make the host-based authentication more secure, you might want to consider the following items:

- 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 in your `sshd2_config` file.

```
IgnoreRhosts yes
```

After this `.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 definition host-based authentication through NAT will not work.

**Note:** This keyword is available in version 3.0 and later.

## 4.2.4 Certificate Authentication

In brief, certificate authentication works the following way:

- Client sends the user certificate (which includes user's public key) to the server.
- Server checks using the CA certificate that the user's certificate is valid.
- Server checks from its mapping file(s) whether login is allowed or not using the user certificate in question.
- Finally, if connection is allowed, 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 it is checked 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.

**Note:** Only the commercial version of SSH Secure Shell includes certificate support.

### Server-Side Configuration

Configure the server-side the following way:

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 using -7 flag with *ssh-keygen2*.
2. Certificate authentication is part of 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 usage of CRLs, by adding *PkiDisableCRLs* keyword below the *Pki* keyword.

```
PkiDisableCRLs    yes
```

**NOTE:** CRL usage should only be disabled for testing purposes. Otherwise it's always highly recommended to 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.

4. Define also the LDAP server(s) in the `sshd2_config` file.

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

5. Next you need to create the map file. It specifies which certificates authorize logging in with which accounts.

The format of the file is the following:

```
<account-id> <keyword> <arguments>
```

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

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

### Examples

```
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% emailregex ([a-z]+)@ssh\..com
```

The last line permits logging with any email address with only letters in the user name to the corresponding account.

### Client-Side Configuration

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

## Software Certificates

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.
```

2. The private key must be in ssh2 format. To convert X.509 keys to ssh2 format, use **-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 using **-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 successfully.
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

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

```
AllowedAuthentications publickey
```

2. Define also the external key provider and initial string.

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

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

### 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, please do the following:

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:

```
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 covers only SSH Secure Shell applications and source code.**

## 4.2.6 Pluggable Authentication Modules (PAM)

When PAM is used, SSH Secure Shell transfers the control of authentication to the Linux-PAM library, which will then load the modules specified in the PAM configuration file. Finally, the Linux-PAM library tells SSH Secure Shell whether or not the authentication was successful. SSH Secure Shell neither knows or cares of the actual authentication method employed by Linux-PAM. Only the final result is of interest.

To enable PAM support, please do the following:

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

```
AllowedAuthentications pam-1@ssh.com  
SshPamClientPath /full/path/to/ssh-pam-client
```

**Note:** By default, `SshPamClientPath` is `/usr/local/bin/ssh-pam-client`.

3. Edit your `/etc/ssh2/sshd2_config` file so that the `pam-1@ssh.com` authentication method is allowed.

The PAM configuration is 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.

### Examples

1. `/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_authok  
session   required /lib/security/pam_pwdb.so
```

---

2. /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
```

---

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

#### 4.2.7 SecurID

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 have SecurID support, you need to compile the source on a machine where RSA Ace/Server (master or slave) or RSA Ace/Agent software is already installed, configured and running.

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

Replace /PATH with the absolute PATH to the directory containing the following files:

- sdclient.a
- sdacmvls.h
- sdconf.h
- sdi\_athd.h
- sdi\_size.h
- sdi\_type.h
- sdi\_defs.h

The above files are normally in /top/ace/examples.

**Note:** If you do not want to make the compilation as root, make sure that all the above files are readable.

2. 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 the 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.

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

## Chapter 5

# 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.

### 5.1 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, it can only log in as the user it is running as, and password authentication may not work if the system uses shadow passwords. An alternative host key file must also be used.

#### 5.1.1 Manually Starting the Secure Shell Server Daemon

To manually start the Secure Shell daemon, type the command `sshd`. (**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.)

`Sshd2` can be configured using command-line options or a configuration file. Command-line options override values specified in the configuration file.

### 5.1.2 Automatically Starting the Secure Shell 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 in HP-UX.

In the following, two different ways of starting Secure Shell Daemon at boot time are introduced. If neither of these work in your system, refer to 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, there should be symbolic links to its startup script in /etc/rc.d/rc?.d, where "?" is the runlevel. You can either add these links manually or use chkconfig. The startup script sshd2 should be in the /etc/rc.d/init.d directory.

**Note:** Chkconfig is only available on RedHat. In SuSE, add the symbolic links manually.

If you want to use chkconfig, check that the first lines in sshd2 are similar to the following ones:

```
#!/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, and that 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 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 `/etc` directory:

```
echo "Starting sshd2..."; /usr/local/sbin/sshd2
```

After this, the Secure Shell daemon will start automatically at boot time.

### 5.1.3 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 remains listening for other connections until explicitly stopped.

#### Login Process

When a user successfully logs in, `sshd2` does the following:

1. Changes to run with normal user privileges.
2. Sets up 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.

### 5.1.4 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.

### **5.1.5 Configuration File and Command-Line Options**

`Sshd2` reads configuration data from `/etc/ssh2/sshd2_config` (or the file specified with `-f` on the command line). The file contains keyword-value pairs, one per line. Lines starting with a 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 `sshd2_config(5)` and to the `sshd2(8)` manual pages.

### **5.1.6 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 `;` or it can, for example, refer to a script.

One example of a subsystem is `sftp`.

```
subsystem-sftp      /usr/local/bin/sftp-server
```

## **5.2 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.

### 5.2.1 Starting the Secure Shell Client

Ssh2 has a very simple syntax:

```
ssh2 [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 somehost.ssh.com`. As with `rsh` and `rlogin`, the user ID to be used can be specified with the `-l` option.

**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 you also have SSH1 installed, 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.

### 5.2.2 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 `-F` on the command line). The file contains keyword-value pairs, one per line. Lines starting with a 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 to the `ssh2(5)` manual pages.

## 5.3 Using Secure Copy (**scp2**)

`Scp2` is a program for copying files over the network securely. It uses `ssh2` for data transfer, and uses the same authentication and provides the same security as `ssh2`.

`Scp2` uses a special file transfer protocol for the 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 above example, 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 you also have SSH1 installed, 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.

## **5.4 Using Secure File Transfer (`sftp2`)**

`Sftp2` is a 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 `f tp`, `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:

```
sftp [options] hostname
```

**Note:** As shown in the above example, 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 `f tp` program.

The `sftp2` command-line options are documented in detail on the `sftp2(1)` manual page.

## **5.5 Using Authentication Agent (`ssh-agent2`, `ssh-add2`)**

`Ssh-agent2` is a program to hold private keys for authentication. With `Ssh-add2`, 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.

## 5.6 Using Chroot Manager (`ssh-chrootmgr`)

*Ssh-chrootmgr* is a helper application to be used in instances where you would like to restrict users to their own home directory when they use `ssh2` and `sftp2`. Note that this works only for static builds, because they do not use any shared libraries. Also, this functionality is not available directly in the binaries, and does not work on Solaris.

1. First, compile the source.

```
./configure --enable-static
make
make install
```

2. Run `ssh-chrootmgr` with root privileges and specify the appropriate user names on the command line.

```
ssh-chrootmgr user1 user2 user3
```

If you want, you can run `ssh-chrootmgr` with the `-v` option to get more information, or with the `-q` option to suppress any output.

3. Edit the following line in the configuration file `/etc/ssh2/sshd2_config`:

---

```
ChRootUsers      user1,user2,user3
```

If all the users are in the same group, edit the following instead:

```
ChRootGroups      group1,group2,group3
```

4. Edit the /etc/passwd file so that the user's shell is /bin/ssh-dummy-shell.
5. Try to connect with sftp, for example as user1, and verify that the environment is chrooted.

The ssh-chrootmgr command-line options are documented in detail on the *ssh-chrootmgr(1)* manual page.

If you want to establish a chrooted environment manually without using ssh-chrootmgr, do the following after compiling static binaries:

1. Create a bin directory under the user's home directory
2. Copy ssh-dummy-shell.static and sftp-server2.static from the /usr/local/bin directory to the \$HOME/bin directory
3. Create the following symbolic links:

```
ln -s sftp-server2.static sftp-server
ln -s ssh-dummy-shell.static ssh-dummy-shell
```

4. As root, edit the /etc/passwd file so that the user's shell is /bin/ssh-dummy-shell.

## 5.7 Using Public-Key Manager (**ssh-pubkeymgr**)

*Ssh-pubkeymgr* creates the user files needed to use public-key authentication with ssh2. After all the required files have been created, it provides an interface that can upload your user public key to a remote host using scp2.

In the following usage example, it is assumed that user *Et* has not yet generated any keys. *Et* is currently logged on host *Earth* and wants to use public-key authentication between the hosts *Earth* and *Home*. The user name is *Et* in both hosts, *Earth* and *Home*.

1. *Ssh-pubkeymgr* is started by giving the command

```
ssh-pubkeymgr
```

2. *Ssh-pubkeymgr* runs ssh-keygen2 and prompts *Et* for a passphrase:

```
Checking for existing user public keys..  
Couldn't find your DSA keypair.. I'll generate you a new set..  
Running ssh-keygen2... don't forget to give it a passphrase!  
Generating 1024-bit dsa key pair  
    4 .oOo.oOo.ooo  
Key generated.  
1024-bit dsa, Et@Earth, Fri Aug 18 2000 15:48:38 +0300  
Passphrase :  
Again :  
Private key saved to /home/Et/.ssh2/id_dsa_1024_a  
Public key saved to /home/Et/.ssh2/id_dsa_1024_a.pub  
Creating your identity file..  
Creating your authorization file..  
...
```

3. Next, `ssh-pubkeymgr` asks if any hosts need to be added to the authorization file. In order to use public-key authentication when connecting from *Home* to *Earth*, the answer must be yes.

```
Do you want to add any hosts to your authorization file?  
(Default: yes)
```

4. After this, the system asks for the required information:

```
Type in their hostname, press return after each one.  
Press return on a blank line to finish.
```

```
Add which user?  
Et  
Add which host?  
Home  
You added Et at Home as a trusted login.  
Press return to continue or Ctrl-D to exit.
```

5. Next, the user public key can be uploaded to a remote host:

```
Do you want to upload Et@Earth key to a remote host? (Default: yes)  
Upload to which host?  
Home  
Which user account?  
Et  
Where is the Et's home directory?  
(e.g. /home/anne, /u/ahc, etc.)  
/home/Et  
Now running scp2 to connect to ssh2-test3...  
Most likely you'll have to type a password :)  
Et@Home's password:
```

```
Et-Earth.pub | 738B | 0.7 kB/s | TOC: 00:00:01  
| 100%
```

Press return to upload to more hosts or Ctrl-D to exit.

Everything is now set up on host *Earth*. Now, user *Et* has to connect to host *Home* with ssh2 and run the command `ssh-pubkeymgr` on host *Home*. After this, user *Et* can use public-key authentication.

If you are not prompted for a passphrase after setting up the public-key authentication, check that you have all the keys listed in the authorization file in your `$HOME/.ssh2` directory.

# Chapter 6

## Appendix

### 6.1 Supported Cryptographic Algorithms and Standards

This section lists the supported cryptographic algorithms and standards in SSH Secure Shell.

The supported public-key algorithms are:

- RSA (768-, 1024-, 2048- or 3072-bit key)
- DSA (768-, 1024-, 2048- or 3072-bit key)

The supported data integrity algorithms are:

- SHA-1 (20-byte key, RFC 2104)
- MD5 (16-byte key, RFC 2104)

For symmetric encryption, the following algorithms are supported:

- AES (128-, 192- or 256-bit key)
- Blowfish (128-bit key)
- Twofish (128-, 192- or 256-bit key)
- CAST-128 (128-bit key)
- Arcfour (128-bit key)
- 3DES (168-bit key)

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.

For certificate and CRL publishing, the following solutions are supported:

- **RFC 2559,** Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2, S. Boeyen, T. Howes, and P. Richard, April 1999.

Supported smart card and cryptographic token related supported standards are:

- **PKCS #11 v2.10:** Cryptographic Token Interface Standard, RSA Laboratories, December 1999.

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/>

## 6.2 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 |
|------------------------|-------------------------|------------------------|------------------------------|------------------------------|
| <b>Password</b>        | x                       |                        |                              |                              |
| <b>User public key</b> |                         |                        |                              |                              |
| - SSH2 keys            | x                       | x <sup>1</sup>         |                              |                              |
| - SSH1 keys            | x                       | x <sup>1</sup>         |                              |                              |
| - PGP keys             | x                       | x <sup>1</sup>         | x <sup>2</sup>               |                              |
| <b>Certificates</b>    |                         |                        |                              |                              |
| - software             | x                       |                        | x                            |                              |
| - PKCS #11 tokens      | x                       |                        |                              | x                            |
| <b>Host-based</b>      | x                       | x                      |                              |                              |
| <b>Kerberos5</b>       |                         |                        |                              |                              |
| - client               |                         |                        | x <sup>3</sup>               |                              |
| - server               |                         |                        | x <sup>3</sup>               |                              |
| <b>RSA securID</b>     |                         |                        |                              | x <sup>4</sup>               |
| - client               | x                       |                        |                              |                              |
| - server               |                         |                        | x <sup>5</sup>               |                              |
| <b>PAM</b>             |                         |                        |                              |                              |
| - client               | x                       |                        |                              |                              |
| - server               |                         |                        | x <sup>6</sup>               |                              |

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

- 3DES, 51
- AES, 51
- agent forwarding, 22
- Arcfour, 51
- authentication, 23
  - authentication methods, 23, 52
  - authentication: host-based, 30
  - authentication: Kerberos, 37
  - authentication: PAM, 38
  - authentication: password, 26
  - authentication: public-key, 26
  - authentication: SecurID, 39
- basic configuration, 13
- Blowfish, 51
- CAST-128, 51
- Certificate Management Protocol (CMP), 52
- client program, 44
- CMPv2, 52
  - command-line options, 44, 45
  - compatibility between versions, 19
  - configuration file, 13, 44, 45
  - configuring Secure Shell, 13
  - configuring SSH1 compatibility, 19
  - configuring support for TCP wrappers, 18
  - cryptographic algorithms, 51
  - customer support, 10
- daemon, 41
- draft-ietf-pkix-rfc2510bis-03, 52
- DSA, 51
- encryption, 11
- export regulations, 11
- fallback functionality, 10
- file locations, 13
- forwarding connections, 20
- generating the host key, 14
- host-based authentication, 30
- introduction to Secure Shell, 9
- Kerberos, 37
- key generation, 14
- legal issues, 11
- Lightweight Directory Access Protocol (LDAP), 52
- location of files, 13
- login process, 43
- MD5, 51
- operating systems, 9
- password, 26
- permitting root logins, 15
- PGP keys, 28
- PKCS #11 v2.10, 52
- PKCS #12 v1.0, 52
- PKCS #7, 52
- Pluggable Authentication Modules (PAM), 38
- port forwarding, 21
- protocol versions, 10
- public-key authentication, 26
- Public-Key Cryptography Standards (PKCS), 52
- resetting the server daemon, 43
- RFC 2315, 52
- RFC 2459, 52
- RFC 2510bis, 52
- RFC 2559, 52
- root logins, 15
- RSA, 51
- scp2, 45
- secure copy, 45
- secure file transfer, 46
- SecurID, 39
- server daemon, 41
- sftp2, 46
- SHA-1, 51

ssh-add2, 46  
ssh-agent2, 46  
ssh-chrootmgr, 47  
ssh-keygen, 27  
ssh-pubkeymgr, 48  
SSH1, 10  
SSH2, 10  
ssh2, 44  
sshd2, 41  
standards, 51  
starting the server daemon, 41  
stopping the server daemon, 43  
support, 10  
supported platforms, 9  
supported standards, 51  
system configuration, 13  
  
TCP wrappers, 18  
technical support, 10  
third-party products, 9  
tunneling, 21  
Twofish, 51  
  
using authentication agent, 46  
using chroot manager, 47  
using public-key manager, 48  
using secure copy, 45  
using secure file transfer, 46  
using the client, 44  
using the server daemon, 41  
  
X11 forwarding, 20