

F-Secure SSH On-Line Help

Table of Contents

Introduction

[Welcome!](#)
[About This Guide](#)
[System Requirements](#)
[Support](#)
[Web Club](#)

Introducing F-Secure SSH

[Transparent Security](#)
[The SSH Protocol](#)

Installation Guide

[Workstation Setup](#)
[F-Secure SSH Wizard](#)

User's Guide

[Basic Features](#)
[RSA Authentication](#)
[Forwardings](#)
[Appendices](#)

Administrator's Guide

[UNIX Setup](#)
[Tool and Utility Overview](#)
[EDD](#)
[MAKE-SSH-KNOWN-HOSTS](#)
[SCP](#)
[SSH](#)
[SSH-ADD](#)
[SSH-AGENT](#)
[SSHD](#)
[SSH-KEYGEN](#)

Welcome!

Welcome to F-Secure SSH the secure remote login program. F-Secure SSH has been designed to completely replace your existing terminal applications to provide you with secure encrypted, and authenticated connections to your UNIX host computers.

About This Guide

This Guide is divided into three main sections:

- *Introducing F-Secure SSH* introduces some of the most important things you need to know about the F-Secure SSH products and the SSH protocol: how the security has been built, and which cryptographic methods are used.
- *Installation Guide* walks you through the installation process of the F-Secure SSH for Windows client program.
- *User's Guide* describes the various tasks you will face when using remote systems with F-Secure SSH. These include RSA Identity generation, RSA authentication, terminal settings, etc.
- *Administrator's Guide* presents the various UNIX utilities that are shipped with F-Secure SSH UNIX distribution. This section also includes printed manual pages for all F-Secure SSH UNIX programs.

System Requirements

To install and use F-Secure SSH for Windows, you need a laptop or personal computer, running a Windows operating system, with a 386 or higher microprocessor, a minimum of 4 Megabytes of RAM, and about 3 MB of disk space.

Support

If you have a question about F-Secure SSH that the documentation does not address, please feel free to contact your local F-Secure SSH distributor or Data Fellows directly.

Should you need to call the technical support team, please try to be at your computer to facilitate the call.

Having the following information available will make it easier for your technical support representative to answer your questions:

- Any specific error messages.
- F-Secure SSH version number.
- Computer make and model.
- Any other information you may think useful in resolving the problem.

Data Fellows provides technical support through a variety of electronic services:

Support:	F-Secure-SSH-Support@DataFellows.com
World-Wide Web:	http://www.DataFellows.com , http://www.Europe.DataFellows.com
Anonymous FTP:	ftp.DataFellows.com , ftp.Europe.DataFellows.com

You may also contact us by mail or phone and ask to speak to our technical support staff:

USA:	Europe:
Data Fellows Inc.	Data Fellows Ltd.
F-Secure SSH Sales	F-Secure SSH Sales
4000 Moorpark Avenue, Suite 207	Paivantaite 8
San Jose, CA 95117	FIN-02210 Espoo
USA	FINLAND
tel (408) 244 9090	tel +358 9 478 444
fax (408) 244 9494	fax +358 9 478 445 99

Note: Consult the README file for latest information and instructions.

Web Club

One of the most convenient ways to reach Data Fellows Ltd. and to obtain the required information or technical support is through the World-Wide Web. Your F-Secure SSH client program allows you to reach our web site right from your F-Secure SSH Help menu.

To connect to F-Secure SSH Web Club:

1. From the Help menu, choose Web Club.
2. In the Select Web Browser dialog box, enter the path and filename for the Web browser you would like to use to access the Data Fellows WWW server.
3. Select the closest Data Fellows mirror site.
4. Choose the OK button to connect.

Transparent Security

[Overview of the F-Secure Product Family](#)

[Overview of the F-Secure SSH Server](#)

[Overview of the F-Secure SSH Client Products](#)

[Authentication, Privacy, and Integrity Protection](#)

Overview of the F-Secure Product Family

F-Secure products utilize the SSH protocol as a generic transport-layer encryption mechanism, providing both host authentication and user authentication, together with privacy and integrity protection.

The encryption technology has been developed in Europe and thus does not fall under the US ITAR export regulations. F-Secure products can be used in every country where encryption is legal, including the United States of America. F-Secure products are sold with pre-licensed patented encryption algorithms to provide the best possible security.

The F-Secure SSH UNIX Server can be used together with F-Secure SSH Clients for Windows, Macintosh, and UNIX to make secure remote login connections to remote offices. The F-Secure SSH Server for UNIX includes tools for secure systems administration; tools are provided for secure file transfer and to secure backups using public-key encryption.

Overview of the F-Secure SSH Server

F-Secure SSH server for UNIX provides the users with secure login connections, file transfer, X11, and TCP/IP connections over untrusted networks. The server uses cryptographic authentication, automatic session encryption, and integrity protection for all transferred data. RSA is used for key exchange and authentication, and symmetric algorithms, Blowfish or three-key triple-DES (3DES), for encrypting transferred data.

System administrators can use tools provided in the server package to replace existing RSH, RLOGIN, RCP, RDIST, and TELNET protocols. This will enable administrators to perform all remote system administration tasks over secure connections. A tool is also included for making secure backups with RSA based public-key encryption.

The F-Secure SSH server for UNIX supports TCP/IP port forwarding technology to connect arbitrary otherwise insecure connections over a secure channel.

Overview of the F-Secure SSH Client Products

F-Secure SSH clients provide the users with secure login connections over untrusted networks. The F-Secure SSH client acts as a replacement for the TELNET protocol taking advantage of the cryptographic authentication, automatic session encryption, and integrity protection methods that are defined by the SSH protocol. F-Secure SSH clients fully support VT100 terminal emulation and ANSI colors.

The F-Secure SSH clients also support TCP/IP port forwarding technology to connect arbitrary otherwise insecure connections over a secure channel. TCP/IP port forwarding works by creating a proxy server for a source port that a TCP/IP service uses. The proxy server waits on the local machine for a connection from a client program to the source port. F-Secure SSH then forwards the request and the data over the secure channel to the remote system. The F-Secure SSH server on the remote system makes the final connection to the destination host and the destination port.

Most remote services that use TCP/IP can be secured, including custom client-server applications, database systems, and services like HTTP, TELNET, POP, SMTP, etc. FSecure SSH also provides automatic forwarding for the X11 Windowing System commonly used on UNIX machines.

Authentication, Privacy, and Integrity Protection

The industry-standard IP protocol does not in any way guarantee any aspect of information security (e.g., authentication, privacy, data integrity). The higher level protocols are no more reliable since they are customarily based on the assumption that the lower level protocols can be trusted. Therefore, if security is needed, it must be implemented entirely on the application level.

The SSH protocol is an application level protocol used by all F-Secure products. SSH guarantees simultaneous authentication of both ends of the connection, secrecy of transmitted information, and integrity of transmitted data.

See section *The SSH Protocol* for more information.

The SSH Protocol

SSH is a packet-based binary protocol that works on top of any transport that will pass a stream of binary data. Normally, TCP/IP is used as the transport, but the implementation also permits using an arbitrary proxy program to pass data to/from the server.

The packet mechanism and related authentication, key exchange, encryption, and integrity mechanisms implement a transport-layer security mechanism, which is then used to implement the secure connection functionality.

More:

[Host Authentication](#)

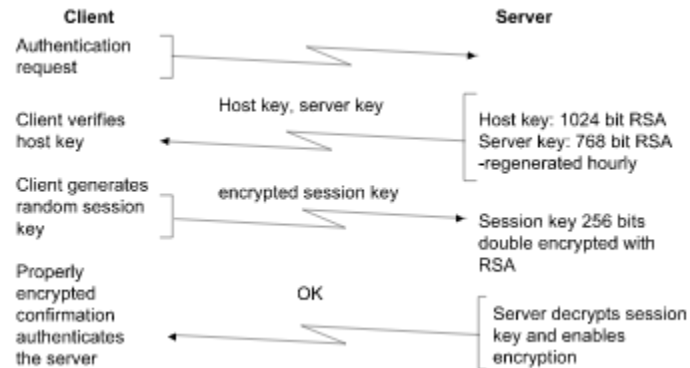
[User Authentication](#)

[Cryptographic Methods](#)

Host Authentication

The server sends its public RSA host key and another public RSA key "server key" that changes every hour. The client compares the received host key against its own database of known host keys.

F-Secure SSH Server will normally accept the key of an unknown host and store it in its database for future reference (this makes use of SSH practical in most environments). However, F-Secure SSH Server can also be configured to refuse access to any hosts whose key is not known.



The client generates a 256 bit random number using a cryptographically strong random number generator, and chooses an encryption algorithm from those supported by the server, normally Blowfish or three-key triple-DES (3DES). The client encrypts the random number (session key) with RSA using both the host key and the server key, and sends the encrypted key to the server.

The purpose of the host key is to bind the connection to the desired server host (only the server can decrypt the encrypted session key). The hourly changed second key, the server key, is used to make decrypting recorded historic traffic impossible in the event that the host key becomes compromised. The host key is normally a 1024 bit RSA key, and the server key is 768 bits. Both keys are generated using a cryptographically strong random number generator.

The server decrypts the RSA encryption and recovers the session key. Both parties start using the session key and the connection is now encrypted. The server sends an encrypted confirmation to the client. Receipt of the confirmation tells the client that the server was able to decrypt the key, and thus holds the proper private keys.

At this point, the server machine has been authenticated, and transport-level encryption and integrity protection are in use.

User Authentication

The user can be authenticated by the server in a number of ways. The user authentication dialogue is driven by the client which sends requests to the server. The first request always declares the user name to log in as. The server responds to each request with either success or failure (further authentication is required).

Supported authentication methods are:

- Traditional password authentication. The password is transmitted over the encrypted channel, and thus cannot be seen by outsiders.
- Pure RSA authentication. The idea is that possession of a particular private RSA key serves as authentication. The server has a list of accepted public keys.

Cryptographic Methods

The SSH protocol provides strong security with cryptographic methods.

SSH uses RSA for host authentication and user authentication. Host keys and user authentication keys are 1024 bits.

The server key that changes every hour is 768 bits by default. It is used to protect intercepted historical sessions from being decrypted if the host key is later compromised. The server key is never saved on disk.

Key exchange is performed by encrypting the 256-bit session key twice using RSA. It is padded with non-zero random bytes before each encryption. Server host authentication happens implicitly with the key exchange (the idea is that only the holder of the valid private key can decrypt the session key, and receipt of the encrypted confirmation tells the client that the session key was successfully decrypted).

Client host authentication and RSA user authentication are done using a challenge-response exchange, where the response is MD5 of the decrypted challenge plus data that binds the result to a specific session (host key and anti-spoofing cookie).

The key exchange transfers 256 bits of keying data to the server. Different encryption methods use varying amounts of the key: Blowfish uses 128 bits and three-key triple-DES (3DES) 168 bits.

All random numbers used in SSH are generated with a cryptographically strong random number generator.

Workstation Setup

Before starting the F-Secure SSH installation process, close all running applications.

1. Insert the F-Secure SSH setup disk into the floppy disk drive.
2. If you're running Windows 95 or Windows NT 4.0, choose Run from the Start menu. Otherwise, choose Run from the File menu.
3. Type "a:setup" and press Enter. If your floppy disk drive is associated to another drive letter, use that instead of "a:".
4. Follow the instructions of the setup program to install F-Secure SSH on your computer.

The F-Secure SSH Wizard will be launched automatically after the setup to help you generate an RSA Identity. See [Creating an RSA Identity with the F-Secure SSH Key Generation Wizard](#) for more information.

Note: You will be able to quit the installation at any time by pressing the Exit button displayed in every dialog box. You can also move back and forth to review your settings by clicking the Back and Next buttons.

F-Secure SSH Wizard

Creating an RSA Identity

The SSH protocol implements RSA authentication as the strongest means for user authentication. To use RSA authentication you must create an RSA key pair with the help of the F-Secure SSH Key Generation Wizard.

To use F-Secure SSH Key Generation Wizard, you can double-click the F-Secure SSH Wizard icon in the F-Secure SSH program group. The wizard is also started automatically during setup.

Below is a short step-by-step guide to the F-Secure SSH Key Generation Wizard and the key generation process.

Please read the information displayed on the first Wizard page and press Next.

Enter the file name you would like to use for your identity files. No file extension is required. The public key file will be distinguished by a .PUB extension. It is recommended that you use the default file name IDENTITY.

Choose the Next button.

Enter a comment for the RSA Identity file. A recommended comment would be your user name and the host name holding the identity files, e.g. "bob@pc1.company.com".

A passphrase will be used to protect your RSA Identity file. Enter your passphrase into the Passphrase and Passphrase Confirmation fields. The fields differentiate between uppercase and lowercase letters. The passphrase should not be less than 6 characters long.

Choose the Next button.

Specify the number of bits you would like to be used as the key length for the RSA key pair. 1024 is recommended, key sizes above that do not greatly improve security. The longer the key is the higher the security is, with the drawback that long keys make the RSA authentication process slower. Minimum length is 512 bits.

Choose the Next button.

If you have created a random seed by moving your mouse cursor on to of a random number generation dialog box, go directly to the next step. Otherwise move your mouse randomly in the dialog box to generate random mouse movements, which will be used to create a random seed for F-Secure SSH's cryptographically strong random number generator.

Choose the Next button.

F-Secure SSH Key Generation Wizard generates two prime numbers, then calculates and tests an RSA key pair to be used as the identity. This can take anything from 1 minute to 15 minutes. Do not reboot your machine even if it does not respond.

Choose Start and then wait for the key generation to finish and press the Next button.

F-Secure SSH Key Generation Wizard is now ready to save the identity files, Choose the Finish button to save the RSA Identity.

Basic Features

F-Secure SSH provides you with a secure, encrypted, authenticated channel for connecting to remote UNIX hosts. This software is intended as a replacement for insecure terminal applications, such as TELNET and RLOGIN.

Furthermore, X11 connections and arbitrary TCP/IP connections can be secured with the port forwarding feature of F-Secure SSH.

More:

[Connecting to a Remote Host](#)

[Creating Connection Templates](#)

[Working with Text in the Terminal Window](#)

[Setting Connection Properties](#)

[Disconnecting](#)

Connecting to a Remote Host

To start F-Secure SSH, double-click the F-Secure SSH icon in the F-Secure SSH program group.

To connect to a remote computer using password authentication:

1. From the Edit menu, choose Properties.
2. In the Connection tab, type the name of the remote host you want to connect to, and the user name for your account.
3. From the options group select Password as your authentication type.
4. Choose the OK button.
5. Press Enter or from the File menu, choose connect.
6. Type in the password for your account to the Connect Using Password Authentication dialog box.
7. Choose the OK button to connect.

If this is the first time you're connecting to the host, SSH may ask you to accept the host's previously unknown host key.

Note: The easiest and quickest way to make a connection is to press Enter once in the empty terminal window. This causes F-Secure SSH to display a Connection dialog box.

Creating Connection Templates

Connection templates allow you to save different sets of settings for your connections. In addition, they can help you open multiple connections with different host names, user names and passwords without visiting the Properties dialog multiple times. You can create a connection template from an existing connection you have open or from a previously saved connection template.

To create a connection template:

1. From the Edit menu, choose Properties.
2. Make the changes you want. You can edit the Connection, RSA Identity, Forward, Font, Terminal, and Keyboard properties.
3. Choose the OK button on the property pages to activate the changes.
4. From the File menu, choose Save. F-Secure SSH saves all changes to a new connection template or to the previous connection template you were editing.

Note: The saved connection templates are associated with the F-Secure SSH program. You can create icons on the Windows desktop for the connection template files (.SSH files) and double click the icons to launch the saved connection.

Working with Text in the Terminal Window

[Copying Text onto the Clipboard](#)

[Selecting All Lines on the Screen](#)

[Selecting All Lines in the Scrollback Buffer](#)

Copying Text onto the Clipboard

To copy text onto the Clipboard:

1. Select the text you want to copy by dragging the mouse across it with the button held down.
2. From the Edit menu, choose Copy.

Once a selection is copied, you can insert it in SSH or another Windows application by using the Paste command.

Note: You can also click the right mouse button to access most of the Edit menu commands from a pop-up menu.

Selecting All Lines on the Screen

From the Edit menu, choose Select Screen.

Using the Select Screen command selects all lines currently visible on the terminal window. After selecting the text, you can copy or paste the selection directly

Selecting All Lines in the Scrollback Buffer

From the Edit menu, choose Select All.

Using the Select All command has the same result as selecting the entire contents of the scrollback buffer by using the mouse. After selecting the text, you can copy or paste the selection directly.

Setting Connection Properties

Properties command (Edit menu) modifies F-Secure SSH settings that control terminal appearance, connection information, keyboard settings, secure TCP/IP connections, and other options.

Property Sheet	Description
Connection	Specifies information about the host, user, authentication type, compression, support for X11 Windowing System, port and cipher used when connecting to a remote system.
RSA Identity	Specifies information about your RSA Identity. This sheet allows you to select the location of the identity files, change the passphrase used to protect your RSA identity, and generate a new RSA key pair.
Forward	Specifies the local and remote TCP/IP connections that are to be secured by forwarding them through the F-Secure SSH connection. Identifies the names and connection parameters for the forwarded connections.
Font	Specifies the font, and the text, background, and inactivity color.
Terminal	Specifies information about the number of lines stored in the scrollback buffer, and also settings that control terminal window behavior.
Keyboard	Specifies the keyboard map file used to translate user key presses and received terminal data to characters displayed in the terminal window. The Keyboard Property Sheet allows you to customize the keyboard behavior to best suite your taste.

More:

[Connection Property Sheet](#)

[RSA Identity Property Sheet](#)

[Forward Property Sheet](#)

[Font Property Sheet](#)

[Terminal Property Sheet](#)

[Keyboard Property Sheet](#)

Connection Property Sheet

Connection tab of Properties command (Edit menu). Specifies options for a remote connection.

Remote System Option	Function
Host Name	Specifies the host name or the IP address of the remote machine to log in to.
User Name	Specifies the user name to log in as, on the remote machine.
Other Options	Function
Compression	Requests compression of all data and data for forwarded X11 and TCP/IP connections. The compression algorithm is the same as used by the gzip program.
Forward X11	Instructs F-Secure SSH to create a proxy X server on the server machine for forwarding the X11 connections over the encrypted channel.
Port	Port to connect to on the remote host.
Cipher	Selects the cipher to use for encrypting the session. 3DES is used by default.
Authentication	Selects the authentication method to be used for the session. RSA authentication is only available if you have specified an identity file to be used on the RSA Identity property page. Password authentication is the default.

RSA Identity Property Sheet

RSA Identity tab of Properties command (Edit menu). Specifies information about your RSA Identity.

Option	Function
RSA Identity File	Selects the file from which the identity (private key) for RSA authentication is read. Default is identity in the installation directory. Different identity files may also be specified on a per-connection basis.
Comment	An optional comment in the identity.pub file. This typically specifies the user and the host that created the RSA Identity file.

Button	Function
Browse	Displays a browse dialog box to browse and select the RSA Identity file from locations other than the default location.
Copy	Copies the RSA public key and the comment to the Clipboard. You can then paste the public key to the authorized_keys file on the server.
New	Runs the Key Generation Wizard to create a new RSA key pair to serve as your RSA Identity.
Passphrase	To prevent unauthorized people from using your RSA Identity and gaining access to your connections, you can use a passphrase to protect the identity file. It is recommended that you change your passphrase periodically.

Forward Property Sheet

Forwardings tab of Properties command (Edit menu). Specifies options for the local and remote TCP/IP connections that are to be secured by F-Secure SSH.

Option	Function
Local Forwardings	Displays the local TCP/IP ports that have been forwarded. The New, Edit, and Delete buttons then apply to local forwardings.
Remote Forwardings	Displays the local TCP/IP ports that have been forwarded. The New, Edit, and Delete buttons then apply to remote forwardings.

Button	Function
New	Define a new TCP/IP connection to be secured. A dialog box to enter source port, destination host and port parameters is displayed.
Edit	Edit a previously defined forwarding. A dialog box to edit source port, destination host and port parameters is displayed.
Delete	Deletes the currently selected forwarding.

Choosing the New or Edit button will display a dialog box called: Forward a TCP/IP Connection.

Forward a TCP/IP Connection dialog box specifies the local and remote ports for the connection, the remote system to forward the connection to, and a descriptive name for the forwarded TCP/IP connection.

Local or Remote Option	Function
Name	Each forwarding can be given a descriptive name that is displayed in the forwardings list box.
Source Port Number	Source port is the port used by the clients to connect to.
Destination host	Specify the host that runs the target TCP/IP service here. The destination host can be specified as 127.0.0.1 (i.e. localhost), if the service is running on the same remote system that is used by the user terminal session.
Destination Port Number	Destination port is the port used by the TCP/IP service on the destination host.
Allow local connections only	This check box should be checked to protect your forwarded connections from being used from any where else except from your PC. Checking this option disallows any connection other than those from the localhost address 127.0.0.1.

Font Property Sheet

Font tab of Properties command (Edit menu). Specifies options for terminal window font and colors.

Font Option	Function
Name	Type or select a font name. The Font property page lists currently available fonts for the terminal window.
Size	Type or select a font size.
Color Option	Function
Font	Type or select one of the 16 predefined colors to be used with the characters displayed on the screen.
Background	Type or select one of the 16 predefined colors to be used as the background color.
Disconnected	Type or select one of the 16 predefined colors to be used to indicate that a connection has been disconnected.

Terminal Property Sheet

Terminal tab of Properties command (Edit menu). Specifies options for the terminal window.

Option	Function
Scrollback Buffer Lines	The number of lines to be allocated for the scrollback buffer.
Scroll to End on Output	This option instructs F-Secure SSH to scroll to the end of the scrollback buffer when data is received from the connection.
Scroll to End on Key Press	This option instructs F-Secure SSH to scroll to the end of the scrollback buffer when you press a key.
Auto Wrap	This option indicates that auto-wraparound should be allowed. The cursor will automatically wrap to the beginning of the next line when it is at the rightmost position of a line and text is output.
Reverse Auto Wrap	This option indicates that reverse-wraparound should be allowed. The cursor will back up from the leftmost column of one line to the rightmost column of the previous line.
Inverse Video	Display the terminal window characters with the background color and the background with the font color.
Status Line	Displays a VT100 status line on the terminal window.

Keyboard Property Sheet

Keyboard tab of Properties command (Edit menu). Specifies options used to translate user key presses and received terminal data to characters displayed in the terminal window.

Option	Function
Keyboard Map File	Selects the file from which the keymap for keyboard remapping is read. Default is keymap.map in the installation directory. Different keymap files may also be specified on a per-connection basis.
Auto Linefeed	This option toggles automatic insertion of linefeeds after each carriage return.
Backspace Sends Delete	Instructs the backspace key to send the delete key code.
Delete Sends Backspace	Instructs the delete key to send the backspace key code.
Lock Function Keys	Locks all VT100 functions keys F1...F20 in order to prevent them from being programmed by VT100 escape codes.
Application Cursor Keys	This option toggles between the application cursor key mode and the VT100 cursor key mode.
Application Numeric Keypad	This option toggles between the application numeric keypad mode and the VT100 numeric keypad mode.
Button	Function
Browse	Displays a browse dialog box to browse and select the keymap file from locations other than the default location.

Disconnecting

Consider saving the connection settings for future use as an F-Secure SSH connection template file. From the File menu, choose the Save As command to save the connection settings.

To disconnect your connection:

1. If you have a terminal session active in the terminal window, enter the command to logout from the remote system. Typically, this command is called logout, exit or quit.
2. If you have any applications running that are using the forwarded TCP/IP connections, close them.
3. From the File menu, choose Disconnect.

If you did not close all forwarded TCP/IP connections, these forwardings will be listed on the terminal screen. The F-Secure SSH client will wait for the forwardings to close after closing the terminal connection.

RSA Authentication

[Creating an RSA Identity](#)

[Setting Up Your Account for RSA Authentication](#)

[Connecting Using RSA Authentication](#)

[Changing Your RSA Identity Passphrase](#)

Creating an RSA Identity

The SSH protocol implements RSA authentication as the strongest means for user authentication. To use RSA authentication you must create an RSA key pair with the help of the F-Secure SSH Key Generation Wizard.

The F-Secure SSH Key Generation Wizard creates two files: the private key is stored in a file called `identity` and the public key in a file called `identity.pub`.

To use F-Secure SSH Key Generation Wizard, you can double-click the F-Secure SSH Wizard icon in the F-Secure SSH program group.

To create an RSA Identity:

1. Please read the information displayed on the first Wizard page and press Next.
2. Enter the file name you would like to use for your identity files. No file extension is required. The public key file will be distinguished by a .PUB extension. It is recommended that you use the default file name `IDENTITY`.
3. Choose the Next button.
4. Enter a comment for the RSA Identity file. A recommended comment would be your user name and the host name holding the identity files, e.g. "bob@pc1.company.com".
5. A passphrase will be used to protect your RSA Identity file. Enter your passphrase into the Passphrase and Passphrase Confirmation fields. The fields differentiate between uppercase and lowercase letters. The passphrase should not be less than 6 characters long.
6. Choose the Next button.
7. Specify the number of bits you would like to be used as the key length for the RSA key pair. 1024 is recommended, key sizes above that do not greatly improve security. The longer the key is the higher the security is, with the drawback that long keys make the RSA authentication process slower. Minimum length is 512 bits.
8. Choose the Next button.
9. If you have created a random seed by moving your mouse cursor on to of a random number generation dialog box, go directly to step 11. Otherwise move your mouse randomly in the dialog box to generate random mouse movements, which will be used to create a random seed for F-Secure SSH's cryptographically strong random number generator.
10. Choose the Next button.
11. F-Secure SSH Key Generation Wizard generates two prime numbers, then calculates and tests an RSA key pair to be used as the identity. This can take anything from 1 minute to 15 minutes. Do not reboot your machine even if it does not respond.
12. Choose the Next button.
13. F-Secure SSH Key Generation Wizard is now ready to save the identity files, Choose the Finish button to save the RSA Identity.

Note: You can also generate the identity files on a UNIX host running `ssh-keygen` version 1.3 or later. You can transfer the generated identity file from `$HOME/.ssh/identity` to the F-Secure SSH installation directory on your computer.

Setting Up Your Account for RSA Authentication

SSH protocol implements RSA authentication automatically. You create your personal RSA key pair by running the F-Secure SSH Key Generation Wizard during setup. The RSA identity files are saved to the installation directory, the private key in a file called `identity` and the public key in a file called `identity.pub`. You can then add the contents of the `identity.pub` file to the `authorized_keys` file on the remote host (normally, `$HOME/.ssh/authorized_keys`).

Adding an RSA public key to the `authorized_keys` file:

1. Connect to your account using password authentication. See [Connecting Using Password Authentication](#).
2. Select `Edit | Properties | RSA Identity`
3. If the `Comment` field is empty, browse for the RSA identity file or create a new RSA identity by pressing the `New` button, then repeat step two. If the `RSA Identity Comment` field is non-empty go to step four.
4. Press the `Copy` button to copy the RSA public key to the clipboard.
5. Change to the `.ssh` directory in your home directory on the remote system.
6. Load the `authorized_keys` file to a text editor. If the file does not exist, create it with a text editor.
7. Paste the RSA public key from the clipboard to the end of the `authorized_keys` file. Be careful if your editor has automatic word wrapping, as the key must not be wrapped to several lines. The file has precisely one line per public key entry.
8. Save the `authorized_keys` file and exit the text editor.

Note: Now the account is ready to be accessed using RSA authentication from a client machine that contains the correct RSA identity file (the private and the public key pair.)

Connecting Using RSA Authentication

RSA authentication is based on public-key cryptography: there are cryptosystems where encryption and decryption are done using separate encryption and decryption keys, and it is not possible to derive the decryption key from the encryption key. RSA is one such system.

During the installation of F-Secure SSH, you generate a public/private key pair to serve as your RSA identity to be used for user authentication. The server knows the public part of your RSA identity, but you are the only one who knows the private key.

When you log in, the F-Secure SSH program tells the server which key pair it would like to use for authentication. The server verifies that this key is permitted, and if so, sends the SSH client a challenge, which is a random number encrypted by your public key. The challenge can only be decrypted using the correct private key.

To respond to the challenge the F-Secure SSH client prompts you to enter your passphrase to unlock the RSA Identity. The F-Secure SSH client then decrypts the challenge using the private key, proving that you have the correct identity but without disclosing the private key to the server or anybody else.

Note: The file `$HOME/.ssh/authorized_keys` on the server lists the public keys of the users that are permitted to log into the system.

To connect to a remote computer using RSA authentication:

1. From the Edit menu, choose Properties.
2. In the Connection tab, type the name of the remote system you want to connect to, and the user name for your account.
3. From the options group select RSA authentication as your authentication type. If RSA authentication is not available, please refer back to section Setting Up Your Account for RSA Authentication.
4. Choose the OK button.
5. From the File menu, choose connect. Wait for F-Secure SSH to establish the connection and ask for your passphrase. The passphrase is the one you gave the F-Secure SSH Key Generation Wizard to protect the RSA identity file.
6. Type your passphrase to the requesting dialog box for user authentication.
7. Choose OK button to connect.

Note: If RSA authentication fails, F-Secure SSH reverts to password authentication and prompts you for a password. The password is sent to the remote host for checking; however, since all communication is encrypted, the password cannot be seen by someone spying the traffic on the network.

Changing Your RSA Identity Passphrase

To prevent unauthorized people from using your RSA Identity and gaining access to your connections, you should use a passphrase to protect the identity file. It is recommended that you change your passphrase periodically.

To change your RSA Identity passphrase:

1. Select Edit | Properties | RSA Identity
2. In the RSA Identity property page, choose the Passphrase button.
3. Your RSA Identity Comment should be displayed in the Change RSA Identity Passphrase dialog box.
4. In the Change RSA Identity Passphrase dialog box, type your current passphrase and new passphrase, and then retype the new passphrase to confirm it.
5. Choose the OK button. Your new RSA Identity passphrase takes effect immediately.
6. In the RSA Identity property page, choose the OK button.

Forwardings

Securing X11 Connections

Securing Arbitrary TCP/IP Connections

Securing X11 Connections

F-Secure SSH can secure X11 connections to a X Window System Server running on your computer.

If you have turned on the Forward X11 option in the Connection tab (Properties command, Edit menu) all connections to the X11 display are automatically forwarded through the encrypted channel.

You should not manually set the DISPLAY environment variable. The DISPLAY value set by F-Secure SSH will point to the server machine, but with a display number greater than zero. This is normal, and happens because F-Secure SSH creates a proxy X server on the server machine for forwarding the connections over the encrypted channel.

F-Secure SSH will also automatically set up Xauthority data on the server machine. For this purpose, it will generate a random authorization cookie, store it in Xauthority on the server, and verify that any forwarded connections carry this cookie and replace it by the real cookie when the connection is opened. The real authentication cookie is never sent to the server machine (and no cookies are sent in the plain).

To use F-Secure SSH terminal connection for secure X11 connections from a remote host to a X Window System Server running in your Windows computer, do the following:

1. Start your PC X Window System Server.
2. Check that connections to the PC X Window System Server are allowed from the IP address 127.0.0.1 (i.e. localhost).
3. From the Edit menu, choose Properties.
4. In the Connection tab, select Forward X11.
5. Choose the OK button.
6. From the File menu, choose connect and make the connections as specified in Connecting Using Password Authentication, or Connecting Using RSA Authentication.
7. Type a command in the terminal windows to start any X11 application (e.g. xterm &).

Note: It is recommended that you use a fast cipher (i.e. Blowfish) for best performance with forwarded X11 connections.

Securing Arbitrary TCP/IP Connections

F-Secure SSH can secure arbitrary TCP/IP connections. The technology used is called TCP/IP port forwarding.

TCP/IP port forwarding works by creating a proxy server for a source port that a TCP/IP service uses. The proxy server waits on the local machine for a connection from a client program to the source port. F-Secure SSH then forwards the request and the data over the secure channel to the remote system. The F-Secure SSH server on the remote systems then makes the connection to the destination host and the destination port.

Most remote services that use TCP/IP can be secured, including custom client-server applications, database systems, and services like HTTP, TELNET, POP, SMTP, etc.

FSecure SSH provides automatic forwarding for the X11 Windowing System commonly used on UNIX machines. To forward other TCP/IP connections, you must configure them manually.

To connect securely to a remote TCP/IP service (a secure HTTP connection is used as a sample):

1. Determine the port numbers used by the service (e.g. port 80 is used by the HTTP protocol and the Web servers.) See the /etc/services file on your remote system for default UNIX service and port configurations.
2. From the Edit menu, choose Properties.
3. In the Forward tab, select Local and press the New to define the TCP/IP connections to be secure.
4. Specify a descriptive name for the forwarded connection (e.g. Secure Intranet Connection.)
5. Specify the source port, the port to be used for the client connection (e.g. 80 for the secure Intranet connection.)
6. Specify the destination host, the host that runs the service. If it is the same host the you will connect to for your terminal session, then use 127.0.0.1 (i.e. localhost) to specify that the connection should be forwarded through the secure terminal connection, and then connect immediately to the service.
If the service is running on another host you must specify the forwarding to that host. Note that connection from the terminal session host onward will not be encrypted.
7. Specify the destination port, the port used by the server to wait for connections (e.g. 80 for Web servers.)
8. If you are specifying a local forwarding, then check that the Allow local connection only checkbox is checked to protect your forwarded connections from being used from any where else except from your PC. See the security note at the end of this section.
9. Choose the OK button.
10. From the File menu, choose connect and make the connections as specified in Connecting Using Password Authentication, or Connecting Using RSA Authentication.

Note: Forwarding a connection to any host other than the terminal session host, will only encrypt the forwarded connection up to the terminal session host. The connection from there on will not be encrypted to the destination host. The destination host should always be in a trusted network or be the host for the terminal session.

Note: SECURITY NOTICE: If you do not check the Allow local connection only check box in the Edit Local Forwarding dialog box, then any host can connect to the forwarded source port on your computer and get a connection to the remote machine through your secured connection.

Appendices

Appendix A: F-SECURE.INI

Appendix B: DEFAULTS.SSH

Appendix A: F-SECURE.INI

F-SECURE.INI is a shared initialization file for all Data Fellows' F-Secure applications. It contains a number of settings such as the default language and directories.

When F-Secure SSH is installed the setup program checks for the existence of the F-SECURE.INI file in the Windows directory. If the file does not exist it is created. If the file exists and contains the settings from a previous installation, setup will check for a previous installation and use its settings as the basis for the new installation.

During the installation process the following entries are written to the F-SECURE.INI file:

Setting	Description
Language	Language setting indicates the preferred language. If the language is not available for one of the F-Secure products US English (ENU) is used by default.
RandomSeed	The seed value for the cryptographically strong random number generator, that is updated every time an F-Secure application closes.
InstallDirectory	Install Directory is the location for the README.TXT file and other additional files like connection template files (.SSH files), RSA identity files (identity and identity.pub), and the SSH.INI file. Install Directory is also the preferred working directory that will be set to SSH.EXE when a program manager icon is created.
ProgramDirectory	Program Directory is the directory that contains the F-Secure SSH executables and libraries (e.g. SSH.EXE, GMP.DLL, KEYGEN.EXE, FSSENU.DLL, etc.)

More:

[Sample F-SECURE.INI file:](#)

Sample F-SECURE.INI file:

```
[Settings]
RandomSeed=f69f64...
[F-Secure SSH]
Language=ENU
DefaultUser=root
InstallDirectory=C:\F-SECURE\SSH
ProgramDirectory=C:\F-SECURE\SSH\PROGRAM
[F-Secure SSH Recent Host List]
DefaultHost=ssh.company.com
NumHosts=3
Host1=ssh.company.com
Host2=www.company.com
Host3=ftp.company.com
[F-Secure SSH Recent File List]
File1=C:\F-SECURE\SSH\CONNECT.SSH
[Web Connection Settings]
BrowserDefault=C:\NETSCAPE\PROGRAMS\NETSCAPE.EXE
BrowserArea=Europe
BrowserConfigured=Yes
```

Note: If the F-SECURE.INI file is not located in the Windows directory, it is looked for in the directory of the executable SSH.EXE file. If it is not found there a new F-SECURE.INI file is created in the WINDOWS directory.

Appendix B: DEFAULTS.SSH

The DEFAULTS.SSH file is a connection template file that is used to load default settings each time SSH is run.

You can edit DEFAULTS.SSH by opening the file as a normal connection template. After making your changes and saving the file back to the disk, all new SSH sessions will have these settings loaded as default.

Note: If the DEFAULTS.SSH file does not exist, "hard-coded" defaults are used. To create a new DEFAULTS.SSH file edit a connection template and save it to the Install Directory (e.g. C:\F-Secure\SSH\DEFAULTS.SSH).

UNIX Setup

[Installation](#)

[Configuration Options](#)

[Configuration Files](#)

[Makefile](#)

[Portability](#)

[Hints and Tips](#)

[Libwrap And Identd](#)

[Starting The Server](#)

[Replacing Rlogin And Rsh](#)

[Client Suid Root, Server Run As Root](#)

[Common Problems](#)

[Reporting Problems And Other Contacts](#)

Installation

For most machines and configurations, the following is all you need to do:

1. Unpack the distribution file with gzip and tar
2. Compile the source code using the following commands in the distribution directory:

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

In a networked environment with shared binary directory, it is enough to do "make install" on one machine, and then "make hostinstall" on others to generate host keys and install configuration files.

Then check (and edit if needed) the following files:

- /etc/sshd_config (server configuration file)
- /etc/ssh_config (client configuration file - defaults for users)

You may also want to create the /etc/ssh_known_hosts for your site and update it periodically. See the manual page for make-ssh-known-hosts on how to do this easily. The file format is documented on the sshd manual page.

You should also edit /etc/rc.local or equivalent to start sshd at boot.

The source is written in ANSI C, and requires an ANSI C compiler or GCC.

A copy of GCC is available on all major FTP sites (e.g., in <ftp://prep.ai.mit.edu/pub/gnu>).

Configuration Options

The package comes with an Autoconf-generated configure script. The script accepts several options All standard options, including:

<code>--prefix=PREFIX</code>	where to install files (default: subdirs of /usr/local)
<code>--exec_prefix=PREFIX</code>	where to install executables (default: same as prefix)
<code>--srcdir=DIR</code>	find sources in DIR (default: where configure is) Specific options:
<code>--with-rsh=PATH</code>	Use rsh specified by PATH when needed
<code>--with-etcdir=PATH</code>	Store system files in the given dir (default: /etc)
<code>--with-path=PATH</code>	Default path to pass to user shell.
<code>--with-rsaref</code>	Use rsaref2 from rsaref2 subdirectory (see below).
<code>--with-libwrap[=PATH]</code>	Use libwrap (tcp_wrappers) and identd (see below).
<code>--with-socks4[=PATH]</code>	Include SOCKS (firewall traversal) support.
<code>--with-socks5[=PATH]</code>	Include SOCKS5 (firewall traversal) support.
<code>--with-securid[=PATH]</code>	Support for the SecurID card (see README.SECURID).
<code>--enable-warnings</code>	Adds -Wall to CFLAGS if using gcc.

You may also want to configure the following variables:

<code>CC=compiler</code>	specify name of the C compiler (default: gcc or cc)
<code>CFLAGS=flags</code>	specify flags to C compiler (default: -O -g or just -O)
<code>LDFLAGS=flags</code>	specify flags to linker (default: none)

Alternate values can be given to configure in the environment, e.g.:

```
CC=xcc CFLAGS="-O2" LDFLAGS="-L/lib/zzz" ./configure
```

(Note that if you have already configured, and later decide to give some values on the command line, you may need to say "make distclean" before reconfiguring.)

Configuration Files

The server has a configuration file `/etc/sshd_config`, which specifies the permitted authentication methods, hosts, port number, etc. The defaults are acceptable for most sites, but you may want to check this file. Its format is documented on the `sshd` manual page.

The client reads a configuration file `/etc/ssh_config`, which gives site-wide defaults for various options. Options in this file can be overridden by per-user configuration files. The file is documented on the `ssh` manual page.

Makefile

The Makefile is generated from Makefile.in by running configure. It supports the following targets:

all:	compile everything
install:	install in \$exec_prefix/bin and \$prefix/man/man1.
hostinstall:	generate host key and install config files
uninstall:	remove installed files
clean:	remove object files and executables
distclean:	remove anything not in the distribution

Portability

This software has been used at least in the following environments.

- 386BSD 0.1; i386
- AIX 3.2.5, 4.1, 4.2; RS6000, PowerPC
- BSD 4.4; several platforms
- BSD/OS 1.1, 2.0.1; i486
- BSD/386 1.1; i386
- ConvexOS 10.1; Convex
- DGUX 5.4R2.10; DGUX
- FreeBSD 1.x, 2.x; Pentium
- HPUX 7.x, 9.x, 10.0; HPPA
- IRIX 5.2, 5.3; SGI Indy
- IRIX 6.0.1; Mips-R8000
- Linux 1.2.x, 2.0.x Slackware 2.x, 3.x, RedHat 2.1, 3.0; i486
- Linux/Mach3, Macintosh(PowerPC)
- Mach3; Mips
- Mach3/Lites; i386
- Machten 2.2VM (m68k); Macintosh
- NCR Unix 3.00; NCR S40
- NetBSD 1.0A, 1.1, 1.2; Pentium, Sparc, Mac68k, Alpha
- NextSTEP 3.3; i486
- OSF/1 3.0, 3.2, 3.2; Alpha
- Sequent Dynix/ptx 3.2.0 V2.1.0; i386
- SCO Unix; i386 (client only)
- SINIX 5.42; Mips R4000
- Solaris 2.3, 2.4, 2.5; Sparc, i386
- Sony NEWS-OS 3.3 (BSD 4.3); m68k
- SunOS 4.1.1, 4.1.2, 4.1.3, 4.1.4; Sparc, Sun3
- SysV 4.x; several platforms
- Ultrix 4.1; Mips
- Unicos 8.0.3; Cray C90

Please report back any other environments where you have used ssh, and send back any patches you had to do so that they can be integrated to the distribution. The proper address is F-Secure-SSH-BUGS@DataFellows.com

Always remember to mention the ssh version number and machine type in your bug reports. Please include also the output of the -v option from the client side, and the output of the -d option from the server, if applicable.

Hints and Tips

Linux note: Some linux systems have a bug which causes an error about libc.so.4 when compiling ssh. This can be solved by any of the following ways:

- Do "ln -s libc.so /usr/lib/libc.so" as root.
- Install gcc-2.7.0.
- Configure ssh with "CFLAGS=-O ./configure" (i.e., without debug info).

More information on this problem is available in <ftp://ftp.netcom.com/pub/ze/zenon/linux>.

BSDI BSD/OS note: Apparently the gcc that comes with BSD/OS is broken. Use "CC=cc ./configure" or "setenv CC cc; ./configure" when configuring to force it to use cc instead of gcc.

ConvexOS note: Convex "make" is broken. Install GNU make first if you have trouble compiling ssh.

Ultrix note: Ultrix /bin/sh is broken. Run configure with "/bin/sh5 configure [options]" if you are on Ultrix.

On alpha, one should edit rsaref2/source/global.h, and make UINT4 "unsigned int" instead of "unsigned long int".

Libwrap And Identd

Sshd does not normally use identd or tcp-wrappers. However, it can be compiled to use these by adding `--with-libwrap` on the command line. This requires that the `tcp_wrappers` `libwrap.a` library and the associated `tcpd.h` have been installed somewhere where the compiler can find them. With libwrap support, ssh will process the `/etc/hosts.allow` and `/etc/hosts.deny` files, and use identd if required by them. The name of the user on the client side (as returned by identd) will be logged if requested by the configuration files. See `tcp_wrappers` documentation for more information.

Starting The Server

The server should be started at boot from `/etc/rc` or equivalent. It need not be given any arguments; however, an optional `"-b bits"` flag may be used to specify RSA key size (default is 768). Key sizes less than 512 can be broken; larger key sizes generally mean more security but require more time to generate and use. 1024 bits is secure for any practical time with current technology.

The server is not started using `inetd`, because it needs to generate the RSA key before serving the connection, and this can take about a minute on slower machines. On a fast machine, and small (breakable) key size (< 512 bits) it may be feasible to start the server from `inetd` on every connection. The server must be given `"-i"` flag if started from `inetd`.

Replacing Rlogin And Rsh

This software has been designed so that it can be installed with the names rlogin, rsh, and rcp, and it will use the SSH protocol whenever the remote machine supports it, and will automatically execute rlogin/rsh (after displaying a warning that there is no encryption) if the remote host does not support SSH.

Rlogin/rsh replacement is done as follows:

```
./configure --with-rsh=RSH-PATH --program-transform-name='s/^s/r/'  
make install
```

where RSH-PATH is the complete pathname of the real rsh program. The rlogin program is searched from the same directory, with the name "rlogin". Note that this means that you need to copy both rsh and rlogin to the same directory, with these names. It is not sufficient to just rename them to e.g. rsh.old.

This will create links for rlogin, rsh, and rcp. If you are installing them in the same directory where rlogin etc. normally are (e.g., /usr/bin), you must first move the original programs to some other directory (e.g., /usr/lib/rsh).

When doing this, you should also build a file containing the host keys of all machines in your organization, and copy this file to /etc/ssh_known_hosts on every machine. This will make .rhosts and /etc/hosts.equiv authentication work for users without any changes to the user configuration, but will be much more secure than conventional .rhosts and /etc/hosts.equiv authentication. This will also protect the users against router attacks where someone (perhaps remotely) reconfigures the routers to direct connections to a certain host to a different machine, which can then grab any passwords which the user types thinking he/she is connected to the real machine.

Client Suid Root, Server Run As Root

This package installs two programs that need special privileges. Ssh is the client program, and it is by default installed as suid root, because it needs to create a privileged port in order to use .rhosts files for authentication. If it is not installed as suid root, it will still be usable, but .rhosts authentication will not be available. Also, the private host key file is readable by root only.

Sshd is the daemon that listens for connections. It should preferably be run as root, because it is by normally listening on a privileged port, and it needs to be able to do setuid(), update utmp, chown ptys etc. when a user logs in. If it is not run as root, explicit "-p port" option must be given to specify an alternate port (same port must also be specified for clients), "-h host_key_file_path" must be given to specify an alternate host key file, and it cannot be used to log in as any other user than the user running it (because it cannot call setuid()). Also, if your system uses shadow passwords, password authentication will not work when running as someone else than root.

Both the server and the client have been carefully screened for possible security problems, and are believed to be secure. However, there can be no guarantee. If you find any problems, please report them immediately.

Common Problems

This section lists some common installation problems.

Shadow passwords

There are many different shadow password schemes. Ssh currently recognizes and supports many of them; however, there are probably still many that it does not understand. This may not be visible at compile time. If your system uses shadow passwords, and password authentication does not work even if sshd is running as root, this is probably your problem. Please contact the author if this happens. Code to recognize (configure.in) and use (auth-passwd.c) the shadow password mechanism on new systems is highly welcome.

login.c does not compile, or logging of logins does not work properly

Mechanisms for updating wtmp, utmp, lastlog, and similar mechanisms are not standardized. Ssh substitutes many of the functions of the conventional login program. These functions are implemented in login.c. You may need to modify this file to make it work on exotic systems. Please send any modifications and bug fixes back to the author for inclusion in the distribution. If you just want to try ssh, and cannot get this file to compile, it is safe to define all of the functions as empty; however, in that case logins will not be logged.

Sshd does not start or dies immediately

The easiest thing to do is to give the -d option to sshd. It will then send debugging output to stderr (and syslog). The -d option also has other side effects, e.g. the daemon will not fork and will only serve a single connection before exiting. However, it is very useful for debugging problems.

Sshd sends debugging output to the system log.

Check your system log (and syslogd configuration) to see why it dies. One possible reason is that your system does not have a proper host key in /etc/ssh_host_key. You can either generate a key with ssh-keygen (it is automatically generated by "make install"), or specify an alternative key with the -h option to the server. Another reason could be that the port which the server tries to listen is already reserved by some other program.

Rhosts authentication does not work

By default, the server does not accept normal .rhosts or /etc/hosts.equiv authentication, because they are fundamentally insecure and can be spoofed by anyone with access to the local network. Rhosts authentication can be enabled at compile time by giving the --with-rhosts option to configure.

The preferred alternative is to collect the public host keys of the entire site to a file, and copy this to /etc/ssh_known_hosts on every machine in the organization running sshd. This will prevent all IP spoofing attacks and provides improved security (provided rshd, rlogind, and rexecd are disabled).

Opening connections is too slow

On very slow machines, encrypting and decrypting the session key may be too slow. For example, on a heavily loaded sun3 it took several minutes to log in with the default key sizes. When we changed it to use shorter host key (512 bits) and server key (384 bits), login time dropped to about a second. A symptom of this problem is that "ssh -v hostname" waits for a long time after printing "Sent encrypted session key".

Shorter host keys can be generated with "ssh-keygen -b 512", giving /etc/ssh_host_key as the file in which to save the key (with empty

passphrase). The server key size can be specified with the -b option on sshd command line (typically, in /etc/rc.local). The server must be restarted for changes to take effect.

The program complains "Could not set controlling tty" or something similar

There are many different styles of pseudo ttys. Ssh currently supports about five different styles (plus variations of them). It is quite possible that there are more variations, some of which are not supported by existing code. Fixing the problem may require adding new code in pty.c and configure.in. You are encouraged to write the needed code and send a patch to the author, or at least report the problem.

General problem solving

The client has -v option, which sends verbose output to stdout. It is very helpful in solving problems.

The server has -d option, which causes it to send verbose debugging output to system log and stderr. This option also causes the server to only serve a single connection and not fork, which helps debugging.

Reporting Problems And Other Contacts

Please report any bugs, problems, and enhancements to
F-Secure-SSH-Bugs@DataFellows.com.

Data Fellows also provides technical support through a variety of electronic services:

Support:	F-Secure-SSH-Support@DataFellows.com
World-Wide Web:	http://www.DataFellows.com , http://www.Europe.DataFellows.com
Anonymous FTP:	ftp.DataFellows.com , ftp.Europe.DataFellows.com

Tool and Utility Overview

EDD

MAKE-SSH-KNOWN-HOSTS

SCP

SSH

SSH-ADD

SSH-AGENT

SSHD

SSH-KEYGEN

EDD

Edd (Encrypting Data Dump) reads data from its input stream (stdin by default), encrypts or decrypts it, and writes the resulting data back to its output stream (stdout by default). Edd is intended to be used as a filter in backup scripts and other similar applications.

Edd generates a random encryption key every time it is called. The encryption key is generated in the same way as in ssh. The random key is encrypted with a public key using RSA. The encrypted key is stored in a fixed-size 4096 byte header at the beginning of the output.

MAKE-SSH-KNOWN-HOSTS

Make-ssh-known-hosts is a perl5 script that helps create the `/etc/ssh_known_hosts` file, which is used by F-Secure SSH to contain the host keys of all publicly known hosts. Ssh does not normally permit login using `rhosts` or `/etc/hosts.equiv` authentication unless the server knows the client's host key. In addition, the host keys are used to prevent man-in-the-middle attacks.

The make-ssh-known-hosts program finds all the hosts in a domain by making a DNS query to the master domain name server of the domain. The master domain name server is located by searching for the SOA record of the domain from the initial domain name server. After getting the hostname list, make-ssh-known-hosts tries to get the public key from every host in the domain and create the `/etc/ssh_known_hosts` file.

SCP

Scp copies files between hosts on a network. It uses ssh for data transfer, and uses the same authentication and provides the same security as F-Secure SSH. Scp asks for passwords or passphrases if they are needed for authentication.

Any file name may contain a host and user specification to indicate that the file is to be copied to/from that host, also copies between two remote hosts are permitted.

SSH

Ssh is a program for logging into a remote machine and for executing commands in a remote machine. It is intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.

Ssh connects and logs into the specified hostname. The user must prove his/her identity to the remote machine using one of several methods. One of these is RSA based authentication. The scheme is based on public-key cryptography.

SSH-ADD

Ssh-add adds identities to the authentication agent, ssh-agent. When run without arguments, it adds the file `$HOME/.ssh/identity`. Alternative file names can be given on the command line. If any file requires a passphrase, ssh-add asks for the passphrase from the user. If the user is using X11, the passphrase is requested using a small X11 program; otherwise it is read from the user's tty.

SSH-AGENT

Ssh-agent is a program to hold authentication private keys. The idea is that ssh-agent is started in the beginning of an X-session or a login session, and all other windows or programs are started as children of the ssh-agent program (the command normally starts X or is the user shell). Programs started under the agent inherit a connection to the agent, and the agent is automatically used for RSA authentication when logging to other machines using ssh.

The agent initially does not have any private keys. Keys are added using ssh-add. When executed without arguments, ssh-add adds the \$HOME/.ssh/identity file. If the identity has a passphrase, ssh-add asks for the passphrase (using a small X11 application if running under X11, or from the terminal if running without X). It then sends the identity to the agent. Several identities can be stored in the agent; the agent can automatically use any of these identities.

The idea is that the agent is run in the user's local PC, laptop, or terminal. Authentication data need not be stored on any other machine, and authentication passphrases never go over the network. However, the connection to the agent is forwarded over ssh remote logins, and the user can thus use the privileges given by the identities anywhere in the network in a secure way.

SSHD

Sshd (F-Secure SSH Daemon) is the daemon program for F-Secure SSH. Together these programs replace rlogin and rsh programs, and provide secure encrypted communications between two untrusted hosts over an insecure network. The programs are intended to be as easy to install and use as possible.

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

Sshd works as follows. Each host has a host-specific RSA key (normally 1024 bits) used to identify the host. Additionally, when the daemon starts, it generates a server RSA key (normally 768 bits). This key is normally regenerated every hour if it has been used, and is never stored on disk.

Whenever a client connects the daemon, the daemon sends its host and server public keys to the client. The client compares the host key against its own database to verify that it has not changed. The client then generates a 256 bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then start to use this random number as a session key which is used to encrypt all further communications in the session. The rest of the session is encrypted using a conventional cipher. Currently, 3DES, Blowfish, DES, and as an option IDEA. 3DES is used by default. The client selects the encryption algorithm to use from those offered by the server.

Next, the server and the client enter an authentication dialog. The client tries to authenticate itself using `.rhosts` authentication, `.rhosts` authentication combined with RSA host authentication, RSA challenge-response authentication, or password based authentication.

SSH-KEYGEN

Ssh-keygen generates and manages authentication keys for ssh. Normally each user wishing to use ssh with RSA authentication runs this once to create the authentication key in \$HOME/.ssh/identity. Additionally, the system administrator may use this to generate host keys.

EDD

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

ENVIRONMENT

DATA FORMAT

RETURN VALUE

DIAGNOSTICS

SEE ALSO

NAME

edd - encrypting data dump

SYNOPSIS

edd [-c cipher] [-e] [-s block_size] [-C stream_comment] [-D diagnostics] [-f input_file] [-o output_file] [-F mag_ical_input_file] [bits mod exp [key_comment]] [keyfile]

edd [-d] [-f input_file] [-o output_file] [-F magi_cal_input_file] [-s block_size] [-B] [-D diagnostics] [-P passphrase] [keyfile]

edd -I

edd -q [-D diagnostics]

DESCRIPTION

Edd (Encrypting Data Dump) reads data from its input stream (**stdin** by default), encrypts or decrypts it, and writes the resulting data back to its output stream (**stdout** by default). **Edd** is intended to be used as a filter in backup scripts and other similar applications.

Edd generates a random encryption key every time it is called. The encryption key is generated in the same way as **ssh**(1) generates its session key. This means that the same random number seed file `$HOME/.ssh/random_seed` is used if available. **Edd** will however work without the seed file. The random key is encrypted with the given public key using RSA. The encrypted key is stored in a fixed-size 4096 byte header at the beginning of the output. **Edd** then reads data from input stream until it receives EOF. It encrypts all data using the selected cipher (3des by default) and writes it to the output stream.

Upon encryption, the RSA public key may be given either as a command line argument *bits*, *mod* and *exp*, or the ssh keyfile containing the key may be given as the argument *keyfile*. If both are omitted, **edd** will try to consult the environment variable **EDD_PUBLIC**. If it contains a valid ssh key file name, the key will be read from the file pointed to by **EDD_PUBLIC**.

Upon decryption, **ssh** reads the RSA private key from the key file (which is in the format used by **ssh**(1) - see **ssh-keygen**(1)), the name of which given in the argument *key_file*. If no *keyfile* argument is present, **edd** will try to consult the environment variable **EDD_PRIVATE**. If it contains a valid ssh key file name, the key will be read from the file pointed to by **EDD_PRIVATE**. The file may optionally be protected by a passphrase; **edd** prompts for the passphrase from the user unless -B (for batch mode) was specified on the command line. It then decrypts the random key used to encrypt the data, and uses it to decrypt the data. **Edd** writes decrypted data to the output stream.

Additionally, **edd** provides the -q option to query which RSA public key was used to encrypt a particular data stream.

OPTIONS

- c** *cipher* Selects the cipher to use for encrypting or decrypting the data stream. *cipher* must be a cipher supported by the implementation. Supported ciphers may be queried with -l command (see below).
- f** *input_file* Makes **edd** use *input_file* as the input stream instead of **stdin**. The file is tried to open with shared lock acquired. If your file system supports locks, and the lock cannot be acquired, **edd** will block.
- o** *output_file* Makes **edd** use *output_file* as the output stream instead of **stdout**. The file is tried to open for writing with explicit lock acquired. If your file system supports locks, and the lock cannot be acquired, **edd** will bail out and not block in this case. This is as to detect the vain attempt to read and write the same file.
- F** *magical_input_file* Makes **edd** use *magical_input_file* as the input stream instead of **stdin**. Moreover, **edd** will try to guess the output file name using the following rules: if *magical_input_file* ends with suffix .edd, **edd** will use that file name, with the suffix stripped off, as the output file name (and choose automatically *decryption*, unless user has specified the mode of operation to be encryption explicitly with -e option). If the file name doesn't end with .edd, the output file name will be the given name with that suffix appended. In this case **edd** will choose automatically the *encryption* mode, unless user has specified the decryption mode explicitly with -d option.
- If **edd** is not able to deduce a good output file name, it will prompt the user for one unless batch mode was selected with -B option (in this case **edd** will bail out).
- d** Selects decryption and forces it to be used even if *magical_input_file* does not end with the suffix .edd.
- e** Selects encryption and forces it to be used even if *magical_input_file* has suffix .edd. Encryption is the default, although it is not implicitly forced.
- C** *stream_comment* Supplies an optional comment to be stored in plain text in the header of the encrypted data. The comment can be queried either with -q or diagnostics flag c (see below). The maximum length of comment is restricted to 1024 bytes.
- s** *block_size* Selects the internal buffer size. This should be a multiple of eight; if it isn't, it will automatically be grown to be a multiple of eight. Larger buffer size will reduce the amount of system calls.
- D** *diagnostics* Selects the diagnostics flags to use. *diagnostics* is a string which may contain any ascii characters. However, only the following are recognized: **n** for normal messages (quite rare), **f** for fatal errors, **e** for other errors, **c** for displaying comments and **t** for trace information (debugging).
- Given characters control which diagnostics messages will be shown. If you want absolutely quiet operation, use -Dq for example. This works because q is not recognized and so it will turn no flags on. -Dcfnt may be abbreviated with -Dall. Default is -Defn. All diagnostics messages will be printed to **stderr**.
- q** Queries information about the incoming data stream. Displays the key used to encrypt the system, and the optional comment supplied when creating the stream.
- B** Specifies batch mode. With this flag, **edd** will never try to read a passphrase or output file name from the user but instead exit if a passphrase or output file name is required but not given.

-P *passphrase*

Specifies a passphrase for the RSA private key. If the passphrase is not specified, the user will be prompted and the passphrase will be read from */dev/tty* (unless **-B** was specified).

-I

Displays information about **edd** and exists. This information includes program version; data quantization; default, minimum and maximum block sizes; default cipher; ciphers supported; header size; maximal length of key and stream comments.

ENVIRONMENT

Edd uses the following environment variables:

EDD_PUBLIC	File where the RSA public key will read if no key or key file is supplied to edd . Note that this file must currently be in binary format.
EDD_PRIVATE	File where the RSA private key will be read if no file is supplied to edd .

DATA FORMAT

Format of the encrypted data is as follows. Data types coding is as given in the ssh protocol specification.

1. The string "EDD FILE V1.0" (only one null character)
2. 32-bit integer: version (must be 0 for now)
3. 32-bit integer: flags (must be 0 for now)
4. 32-bit integer: cipher type (as in ssh)
5. mp-int: public exponent of RSA key
6. mp-int: public modulus of RSA key
7. string: comment of RSA key
8. string: comment supplied with -C
9. mp-int: encrypted random key

Encrypted data bytes immediately follow this header.

The data stream will be padded with some scratch bytes if the length of the stream is not a multiple of eight bytes. After the data stream, one last byte is emitted indicating the amount of padding bytes at the end of the stream. So the total size of the encrypted data stream with header and padding will be $4096 + \text{length of stream} + (8 - \text{length of stream} \bmod 8) \bmod 8 + 1$ bytes.

RETURN VALUE

Edd will return **zero** if it succeeded in the operation requested, **1** if the argument list was inappropriate, **2** if the program has some unknown severe bug and **255** if some other error occurred.

DIAGNOSTICS

Diagnostics messages should be self-explanatory. For control over messages, see -D option (above).

SEE ALSO

`ssh(1)`

MAKE-SSH-KNOWN-HOSTS

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

EXAMPLES

FILES

SEE ALSO

NAME

make-ssh-known-hosts - make ssh_known_hosts file from DNS data

SYNOPSIS

```
make-ssh-known-hosts  
[--initialdns initial_dns]  
[--server domain_name_server]  
[--subdomains comma_separated_list_of_subdomains] [--debug debug_level]  
[--timeout ssh_exec_timeout]  
[--pingtimeout ping_timeout]  
[--passwordtimeout timeout_when_asking_password] [--notrustdaemon]  
[--norecursive]  
[--domainnamesplit]  
[--silent]  
[--keyscan]  
[--nslookup path_to_nslookup_program] [--ssh path_to_ssh_program]  
domain_name [take_regexp [remove_regexp]]
```

DESCRIPTION

Make-ssh-known-hosts is a perl5 script that helps create the `/etc/ssh_known_hosts` file, which is used by **ssh** to contain the host keys of all publicly known hosts. **Ssh** does not normally permit login using `rhosts` or `/etc/hosts.equiv` authentication unless the server knows the client's host key. In addition, the host keys are used to prevent man-in-the-middle attacks.

In addition to `/etc/ssh_known_hosts`, **ssh** also uses the `$HOME/.ssh/known_hosts` file. This file, however, is intended to contain only those hosts that the particular user needs but are not in the global file. It is intended that the `/etc/ssh_known_hosts` file be maintained by the system administration, and periodically updated to contain the host keys for any new hosts.

The **make-ssh-known-hosts** program finds all the hosts in a domain by making a DNS query to the master domain name server of the domain. The master domain name server is located by searching for the SOA record of the domain from the initial domain name server (which can be specified with the `--initialdns` option). The master domain name server can also be given directly with the `--server` option.

After getting the hostname list **make-ssh-known-hosts** tries to get the public key from every host in the domain. It first tries to connect **ssh** port to check if the host is alive, and if so, it tries to run the command `cat /etc/ssh_host_key.pub` on the remote machine using **ssh**. If the command succeeds, it knows the remote machine has **ssh** installed properly, and it then extracts the public key from the output, and prints the `/etc/ssh_known_hosts` entry for it to **STDOUT**. Because **make-ssh-known-hosts** is usually run before remote machines have `/etc/ssh_known_hosts` file you may have to use RSA-authentication to allow access to hosts.

If the command fails for some reason, it checks if the **ssh** client still got the public key from the remote host in the initial dialog, and if so, it will print a proper entry, and if `--notrustdaemon` option is given comment it out.

`Domain_name` is the domain name for which the file is to be generated. By default **make-ssh-known-hosts** extracts also all subdomains of domain. Many sites will want to include several domains in their `/etc/ssh_known_hosts` file. The entries for each domain should be extracted separately by running **make-ssh-known-hosts** once for each domain. The results should then be combined to create the final file.

`Take_regexp` is a perl regular expression that matches the hosts to be taken from the domain. The data matched contains all the DNS records in the form `fieldname=value`. The fields are separated with newline, and the perl match is made in multiline mode and it is case insensitive. The multiline mode means that you can use a regexp like ```^wks=.*telnet.*$``` to match all hosts that have WKS (well known services) field that contains value `telnet`.

`Remove_regexp` is similar but those hosts that match the regexp are not added (it can be used for example to filter out PCs and Macs using the `hinfo` field: ```^hinfo=.*(mac|pc)```).

OPTIONS

- initialdns** *initial_dns*
- i** *initial_dns* Set the initial domain name server used to query the SOA record of the domain.
- server** *domain_name_server*
- se** *domain_name_server* Set the master domain name server of the domain. This host is used to query the DNS list of the domain.
- subdomains** *subdomainlist*
- su** *subdomainlist* Comma separated list of subdomains that are added to hostnames. For example, if subdomainlist is ``*foo, foo.bar, foo.bar.zappa, foo.bar.zappa.hut.fi*'' then when host *foobar* is added to **/etc/ssh_known_hosts** file it has aliases ``*foobar, foobar.foo, foobar.foo.bar, foo_bar.foo.bar.zappa, foobar.foo.bar.zappa.hut.fi*'' . The default action is to take all subparts of the host but the second last on a host by host basis. (The last element is usually the country code, and something like *foobar.foo.bar.zappa.hut* would not make sense.)
- debug** *debug_level*
- de** *debug_level* Set the debug level. Default is 5, bigger values give more output. Using a big value (like 999) will print lots of debugging output.
- timeout** *ssh_exec_timeout*
- ti** *ssh_exec_timeout* Timeout when executing **ssh** command. The default is 60 seconds.
- pingtimeout** *ping_timeout*
- pi** *ping_timeout* Timeout when trying to ping the **ssh** port. The default is 3 seconds.
- passwordtimeout** *timeout_when_asking_password*
- pa** *timeout_when_asking_password* Timeout when asking password for **ssh** command. Default is that no passwords are queried. Use value 0 to have no timeout for password queries.
- notrustdaemon**
- notr** If the **ssh** command fails, use the public key stored in the local known hosts file and trust it is the correct key for the host. If this option is not given such entries are commented out in the generated **/etc/ssh_known_hosts** file.
- norecursive**
- nor** Tell **make-ssh-known-hosts** that it should only extract keys for the given domain, and not to be recursive.
- domainnamesplit**
- do** Split the domainname to get the list of subdomains. Use this option if you don't want hostname to splitted to pieces automatically. Default splitting is done host by host basis. If the domain is *zappa.hut.fi*, and the host name is *foo.bar* then default action adds entries ``*foo, foo.bar, foo.bar.zappa, foo.bar.zappa.hut.fi*'' and this options adds entries ``*foo.bar, foo.bar.zappa, foo.bar.zappa.hut.fi*'').
- silent**

-si Be silent.

--keyscan

-k Output list of all hosts in format ``ipaddr1,ipaddr2,...ipaddrn
hostname.domain.co,host-
name,ipaddr1,ipaddr2,all_other_hostname_entries". The output of this can
be feeded to ssh-keyscan to fetch keys.

--nslookup *path_to_nslookup_program*

-n *path_to_nslookup_program* Path to the **nslookup** program.

--ssh *path_to_ssh_program*

-ss *path_to_ssh_program* Path to the **ssh** program, including all options.

EXAMPLES

The following command:

```
example# make-ssh-known-hosts cs.hut.fi > /etc/ssh_known_hosts
```

finds all public keys of the hosts in cs.hut.fi domain and put them to /etc/ssh_known_hosts file splitting domain names on a per host basis. The command

```
example% make-ssh-known-hosts hut.fi `^wks=.*ssh' > hut-hosts
```

finds all hosts in hut.fi domain, and its subdomains having own name server (cs.hut.fi, tf.hut.fi, tky.hut.fi) that have ssh service and puts their public key to huthosts file. This would require that the domain name server of hut.fi would define all hosts running ssh to have entry ssh in their WKS record. Because nobody yet adds ssh to WKS, it would be better to use command

```
example% make-ssh-known-hosts hut.fi `^wks=.*telnet' > hut-hosts
```

that would take those host having telnet service. This uses default subdomain list.

The command:

```
example% make-ssh-known-hosts hut.fi `dipoli.hut.fi' `^hinfo=.*(mac|pc)' > dipoli-hosts
```

finds all hosts in hut.fi domain that are in dipoli.hut.fi subdomain (note dipoli.hut.fi does not have own name server so its entries are in hut.fi-server) and that are not Mac or PC.

FILES

`/etc/ssh_known_hosts`

Global host public key list

SEE ALSO

ssh(1), **sshd(8)**, **ssh-keygen(1)**, **ping(8)**, **nslookup(8)**, **perl(1)**, **perlre(1)**

SCP

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

DERIVATION

SEE ALSO

NAME

scp - secure copy (remote file copy program)

SYNOPSIS

scp [-**prvC**] [-**P** *port*] [-**c** *cipher*] [-**i** *identity*] [[*user@*]host1:]filename1...
[[*user@*]host2:]filename2

DESCRIPTION

Scp copies files between hosts on a network. It uses **ssh** for data transfer, and uses the same authentication and provides the same security as **ssh**. Unlike **rcp**, **scp** will ask for passwords or passphrases if they are needed for authentication.

Any file name may contain a host and user specification to indicate that the file is to be copied to/from that host. Copies between two remote hosts are permitted.

OPTIONS

-c <i>cipher</i>	Selects the cipher to use for encrypting the data transfer. This option is directly passed to ssh .
-i <i>identity_file</i>	Selects the file from which the identity (private key) for RSA authentication is read. This option is directly passed to ssh .
-p	Preserves modification times, access times, and modes from the original file.
-r	Recursively copy entire directories.
-v	Verbose mode. Causes scp and ssh to print debugging messages about their progress. This is helpful in debugging connection, authentication, and configuration problems.
-B	Selects batch mode (prevents asking for passwords or passphrases).
-C	Compression enable. Passes the -C flag to ssh to enable compression.
-P <i>port</i>	Specifies the port to connect to on the remote host. Note that this option is written with a capital P, because -p is already reserved for preserving the times and modes of the file in rcp .

DERIVATION

Scp is based on the **rcp** program in BSD source code from the Regents of the University of California.

SEE ALSO

ssh(1), **sshd(8)**, **ssh-keygen(1)**, **ssh-agent(1)**, **ssh-add(1)**, **rcp(1)**

SSH

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

CONFIGURATION FILES

ENVIRONMENT

FILES

INSTALLATION

SEE ALSO

NAME

ssh - F-Secure SSH client (remote login program)

SYNOPSIS

ssh [-**I** *login_name*] **hostname** [*command*]

ssh [-**a**] [-**c** 3des|blowfish|des|none] [-**e** *escape_char*] [-**i** *identity_file*] [-**I** *login_name*] [-**n**] [-**o** *option*] [-**p** *port*] [-**q**] [-**t**] [-**v**] [-**x**] [-**C**] [-**L** *port:host:hostport*] [-**R** *port:host:hostport*] *hostname* [*command*]

DESCRIPTION

Ssh is a program for logging into a remote machine and for executing commands in a remote machine. It is intended to replace **rlogin** and **rsh**, and provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.

Ssh connects and logs into the specified hostname. The user must prove his/her identity to the remote machine using one of several methods.

First, if the machine the user logs in from is listed in `/etc/hosts.equiv` or `/etc/shosts.equiv` on the remote machine, and the user names are the same on both sides, the user is immediately permitted to log in. Second, if `.rhosts` or `.shosts` exists in the user's home directory on the remote machine and contains a line containing the name of the client machine and the name of the user on that machine, the user is permitted to log in. This form of authentication alone is normally not allowed by the server because it is not secure.

The second (and primary) authentication method is the **rhosts** or **hosts.equiv** method combined with RSA-based host authentication. It means that if the login would be permitted by `.rhosts`, `.shosts`, `/etc/hosts.equiv`, or `/etc/shosts.equiv`, and additionally it can verify the client's host key (see `$HOME/.ssh/known_hosts` and `/etc/ssh_known_hosts` in the **FILES** section), only then login is permitted. This authentication method closes security holes due to IP spoofing, DNS spoofing and routing spoofing. [Note to the administrator: `/etc/hosts.equiv`, `.rhosts`, and the `rlogin/rsh` protocol in general, are inherently insecure and should be disabled if security is desired.]

As a third authentication method, **ssh** supports RSA based authentication. The scheme is based on public-key cryptography: there are cryptosystems where encryption and decryption are done using separate keys, and it is not possible to derive the decryption key from the encryption key. RSA is one such system. The idea is that each user creates a public/private key pair for authentication purposes. The server knows the public key, and only the user knows the private key. The file `$HOME/.ssh/authorized_keys` lists the public keys that are permitted for logging in. When the user logs in, the **ssh** program tells the server which key pair it would like to use for authentication. The server checks if this key is permitted, and if so, sends the user (actually the **ssh** program running on behalf of the user) a challenge, a random number, encrypted by the user's public key. The challenge can only be decrypted using the proper private key. The user's client then decrypts the challenge using the private key, proving that he/she knows the private key but without disclosing it to the server.

Ssh implements the RSA authentication protocol automatically. The user creates his/her RSA key pair by running **ssh-keygen**(1). This stores the private key in `.ssh/identity` and the public key in `.ssh/identity.pub` in the user's home directory. The user should then copy the `identity.pub` to `.ssh/authorized_keys` in his/her home directory on the remote machine (the `authorized_keys` file corresponds to the conventional `.rhosts` file, and has one key per line, though the lines can be very long). After this, the user can log in without giving the password. RSA authentication is much more secure than `rhosts` authentication.

The most convenient way to use RSA authentication may be with an authentication agent. See **ssh-agent**(1) for more information.

If other authentication methods fail, **ssh** prompts the user for a password. The password is sent to the remote host for checking; however, since all communications are encrypted, the password cannot be seen by someone listening on the network.

When the user's identity has been accepted by the server, the server either executes the given command, or logs into the machine and gives the user a normal shell on the remote machine. All communication with the remote command or shell will be automatically encrypted.

If a pseudo-terminal has been allocated (normal login session), the user can disconnect with ```~.`, and suspend **ssh** with ```~^Z`. All forwarded connections can be listed with ```~#`, and if the session blocks waiting for forwarded X11 or TCP/IP connections to terminate, it can be backgrounded with ```~&` (this should not be used while the user shell is

active, as it can cause the shell to hang). All available escapes can be listed with ```~?`.

A single tilde character can be sent as ```~~` (or by following the tilde by a character other than those described above). The escape character must always follow a newline to be interpreted as special. The escape character can be changed in configuration files or on the command line.

If no pseudo tty has been allocated, the session is transparent and can be used to reliably transfer binary data. On

most systems, setting the escape character to ``none" will also make the session transparent even if a tty is used.

The session terminates when the command or shell in on the remote machine exists and all X11 and TCP/IP connections have been closed. The exit status of the remote program is returned as the exit status of **ssh**.

If the user is using X11 (the DISPLAY environment variable is set), the connection to the X11 display is automatically forwarded to the remote side in such a way that any X11 programs started from the shell (or command) will go through the encrypted channel, and the connection to the real X server will be made from the local machine. The user should not manually set **DISPLAY**. Forwarding of X11 connections can be configured on the command line or in configuration files.

The DISPLAY value set by **ssh** will point to the server machine, but with a display number greater than zero. This is normal, and happens because **ssh** creates a ``proxy" X server on the server machine for forwarding the connections over the encrypted channel.

Ssh will also automatically set up Xauthority data on the server machine. For this purpose, it will generate a random authorization cookie, store it in Xauthority on the server, and verify that any forwarded connections carry this cookie and replace it by the real cookie when the connection is opened. The real authentication cookie is never sent to the server machine (and no cookies are sent in the plain).

If the user is using an authentication agent, the connection to the agent is automatically forwarded to the remote side unless disabled on command line or in a configuration file.

Forwarding of arbitrary TCP/IP connections over the secure channel can be specified either on command line or in a configuration file. One possible application of TCP/IP forwarding is a secure connection to an electronic purse; another is going through firewalls.

Ssh automatically maintains and checks a database containing RSA-based identifications for all hosts it has ever been used with. The database is stored in **.ssh/known_hosts** in the user's home directory. Additionally, the file **/etc/ssh_known_hosts** is automatically checked for known hosts. Any new hosts are automatically added to the user's file. If a host's identification ever changes, **ssh** warns about this and disables password authentication to prevent a trojan horse from getting the user's password. Another purpose of this mechanism is to prevent man-in-the-middle attacks which could otherwise be used to circumvent the encryption. The **StrictHostKeyChecking** option (see below) can be used to prevent logins to machines whose host key is not known or has changed.

OPTIONS

-a	Disables forwarding of the authentication agent connection. This may also be specified on a perhost basis in the configuration file.
-c 3des blowfish des none	Selects the cipher to use for encrypting the session. 3des is used by default. 3des is believed to be very secure (triple-des) is encrypt-decrypt-encrypt triple with three different keys. It is presumably more secure than DES. des is the data encryption standard, but is breakable by governments, large corporations, and major criminal organizations. blowfish is a very fast block cipher algorithm invented by Bruce Schneier. It uses 128 bit keys. none disables encryption entirely; it is only intended for debugging, and it renders the connection insecure.
-e ch ^ch none	Sets the escape character for sessions with a pty (default: ~). The escape character is only recognized at the beginning of a line. The escape character followed by a dot (.) closes the connection, followed by control-Z suspends the connection, and followed by itself sends the escape character once. Setting the character to 'none' disables any escapes and makes the session fully transparent.
-f	Requests ssh to go to background after authentication is done and forwardings have been established. This is useful if ssh is going to ask for passwords or passphrases, but the user wants it in the background. This may also be useful in scripts. This implies -n . The recommended way to start X11 programs at a remote site is with something like ``ssh -f host xterm".
-i identity_file	Selects the file from which the identity (private key) for RSA authentication is read. Default is <code>.ssh/identity</code> in the user's home directory. Identity files may also be specified on a per-host basis in the configuration file. It is possible to have multiple -i options (and multiple identities specified in configuration files).
-l login_name	Specifies the user to log in as on the remote machine. This may also be specified on a per-host basis in the configuration file.
-n	Redirects stdin from /dev/null (actually, prevents reading from stdin). This must be used when ssh is run in the background. A common trick is to use this to run X11 programs in a remote machine. For example, ``ssh -n shadows.cs.hut.fi emacs &" will start an emacs on shadows.cs.hut.fi, and the X11 connection will be automatically forwarded over an encrypted channel. The ssh program will be put in the background. (This does not work if ssh needs to ask for a password or passphrase; see also the -f option.)
-o 'option'	Can be used to give options in the format used in the config file. This is useful for specifying options for which there is no separate command-line flag. The option has the same format as a line in the configuration file.
-p port	Port to connect to on the remote host. This can be specified on a per-host basis in the configuration file.
-q	Quiet mode. Causes all warning and diagnostic messages to be suppressed. Only fatal errors are displayed.
-t	Force pseudo-tty allocation. This can be used to execute arbitrary screen-based programs on a remote machine, which can be very useful e.g. when implementing menu services.
-v	Verbose mode. Causes ssh to print debugging messages about its progress. This is helpful in debugging connection, authentication, and configuration problems.

- x Disables X11 forwarding. This can also be specified on a per-host basis in a configuration file.
- C Requests compression of all data (including **stdin**, **stdout**, **stderr**, and data for forwarded X11 and TCP/IP connections). The compression algorithm is the same used by **gzip**, and the ``level" can be controlled by the **CompressionLevel** option (see below). Compression is desirable on modem lines and other slow connections, but will only slow down things on fast networks. The default value can be set on a host-by-host basis in the configuration files; see the **Compress** option below.
- L *source port: destination host:destination port*
Specifies that the given **source port** on the local (client) host is to be forwarded to the given **destination host** and **destination port** on the remote side. This works by allocating a socket to listen to the **source port** on the local side, and whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the **destination host:destination port** from the remote machine. Port forwardings can also be specified in the configuration file. Only root can forward privileged ports.
- R *source port: destination host:destination port*
Specifies that the given **source port** on the remote (server) host is to be forwarded to the given **destination host** and **destination port** on the local side. This works by allocating a socket to listen to the **source port** on the remote side, and whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the **destination host:destination port** from the local machine. Port forwardings can also be specified in the configuration file. Privileged ports can be forwarded only when logging in as root on the remote machine.

CONFIGURATION FILES

Ssh obtains configuration data from the following sources (in this order): command line options, user's configuration file (**\$HOME/.ssh/config**), and system-wide configuration file (**/etc/ssh_config**). For each parameter, the first obtained value will be used. The configuration files contain sections bracketed by "Host" specifications, and that section is only applied for hosts that match one of the patterns given in the specification. The matched host name is the one given on the command line.

Since the first obtained value for each parameter is used, more host-specific declarations should be given near the beginning of the file, and general defaults at the end.

The configuration file has the following format:

Empty lines and lines starting with **#** are comments.

Otherwise a line is of the format **keyword arguments**. The possible keywords and their meanings are as follows (note that the configuration files are case-sensitive):

Host Restricts the following declarations (up to the next **Host** keyword) to be only for those hosts that match one of the patterns given after the keyword. ***** and **?** can be as wildcards in the patterns. A single ***** as a pattern can be used to provide global defaults for all hosts. The host is the *hostname* argument given on the command line (i.e., the name is not converted to a canonicalized host name before matching).

BatchMode	If set to yes , passphrase/password querying will be disabled. This option is useful in scripts and other batch jobs where you have no user to supply the password. The argument must be yes or no .
Cipher	Specifies the cipher to use for encrypting the session. Currently, <i>3des</i> , <i>blowfish</i> , <i>des</i> , and <i>none</i> are supported. The default is "3des". Using "none" (no encryption) is intended only for debugging, and will render the connection insecure.
Compression	Specifies whether to use compression. The argument must be yes or no .
CompressionLevel	Specifies the compression level to use if compression is enable. The argument must be an integer from 1 (fast) to 9 (slow, best). The default level is 6, which is good for most applications. The meaning of the values is the same as in GNU GZIP.
ConnectionAttempts	Specifies the number of tries (one per second) to make before falling back to rsh or exiting. The argument must be an integer. This may be useful in scripts if the connection sometimes fails.
EscapeChar	Sets the escape character (default: ~). The escape character can also be set on the command line. The argument should be a single character, ^ followed by a letter, or "none" to disable the escape character entirely (making the connection transparent for binary data).
FallBackToRsh	Specifies that if connecting via ssh fails due to a connection refused error (there is no sshd listening on the remote host), rsh should automatically be used instead (after a suitable warning about the session being unencrypted). The argument must be yes or no .
ForwardAgent	Specifies whether the connection to the authentication agent (if any) will be forwarded to the remote machine. The argument must be yes or no .
ForwardX11	Specifies whether X11 connections will be automatically redirected over the secure channel and DISPLAY set. The argument must be yes or no .
GlobalKnownHostsFile	Specifies a file to use instead of <i>/etc/ssh_known_hosts</i> .
HostName	Specifies the real host name to log into. This can be used to specify

nicknames or abbreviations for hosts. Default is the name given on the command line. Numeric IP addresses are also permitted (both on the command line and in **HostName** specifications).

IdentityFile

Specifies the file from which the user's RSA authentication identity is read (default **.ssh/identity** in the user's home directory). Additionally, any identities represented by the authentication agent will be used for authentication. The file name may use the tilde syntax to refer to a user's home directory. It is possible to have multiple identity files specified in configuration files; all these identities will be tried in sequence.

KeepAlive

Specifies whether the system should send keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying.

The default is "yes" (to send keepalives), and the client will notice if the network goes down or the remote host dies. This is important in scripts, and useful to many users too.

To disable keepalives, the value should be set to "no" in both the server and the client configuration files.

LocalForward

Specifies that a TCP/IP port on the local machine be forwarded over the secure channel to given **destination host:destination port** from the remote machine. The first argument must be a port number, and the second must be **destination host:destination port**. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the root can forward privileged ports.

PasswordAuthentication

Specifies whether to use password authentication. The argument to this keyword must be "yes" or "no".

Port

Specifies the port number to connect on the remote host. Default is 22.

ProxyCommand

Specifies the command to use to connect to the server. The command string extends to the end of the line, and is executed with /bin/sh. In the command string, %h will be substituted by the host name to connect and %p by the port. The command can be basically anything, and should read from its **stdin** and write to its **stdout**. It should eventually connect an **ssh** server running on some machine, or execute "ssh -l" somewhere. Host key management will be done using the HostName of the host being connected (defaulting to the name typed by the user).

Note that ssh can also be configured to support the SOCKS system using the --with-socks4 or --with-socks5 compile-time configuration option.

RemoteForward

Specifies that a TCP/IP port on the remote machine be forwarded over the secure channel to given **destination host:destination port** from the local machine. The first argument must be a port number, and the second must be **destination host:destination port**. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the root can forward privileged ports.

RhostsAuthentication

Specifies whether to try rhosts based authentication. Note that this declaration only affects the client side and has no effect whatsoever on security. Disabling rhosts authentication may reduce authentication time on slow connections when rhosts authentication is not used. Most servers do not permit RhostsAuthentication because it is not secure (see RhostsRSAAuthentication). The argument to this keyword must be "yes" or "no".

RhostsRSAAuthentication

Specifies whether to try rhosts based authentication with RSA host authentication. This is the primary authentication method for most sites. The argument must be "yes" or "no".

RSAAuthentication	Specifies whether to try RSA authentication. The argument to this keyword must be " yes " or " no ". RSA authentication will only be attempted if the identity file exists, or an authentication agent is running.
StrictHostKeyChecking	<p>If this flag is set to "yes", ssh will never automatically add host keys to the \$HOME/.ssh/known_hosts file, and refuses to connect hosts whose host key has changed. This provides maximum protection against trojan horse attacks. However, it can be somewhat annoying if you don't have good /etc/ssh_known_hosts files installed and frequently connect new hosts. Basically this option forces the user to manually add any new hosts. Normally this option is disabled, and new hosts will automatically be added to the known host files. The host keys of known hosts will be verified automatically in either case. The argument must be "yes" or "no".</p> <p>User Specifies the user to log in as. This can be useful if you have a different user name in different machines. This saves the trouble of having to remember to give the user name on the command line.</p>
UserKnownHostsFile	Specifies a file to use instead of \$HOME/.ssh/known_hosts .
UseRsh	Specifies that rlogin/rsh should be used for this host. It is possible that the host does not at all support the ssh protocol. This causes ssh to immediately exec rsh . All other options (except HostName) are ignored if this has been specified. The argument must be " yes " or " no ".

ENVIRONMENT

DISPLAY	The DISPLAY variable indicates the location of the X11 server. It is automatically set by ssh to point to a value of the form "hostname:n" where hostname indicates the host where the shell runs, and n is an integer ≥ 1 . Ssh uses this special value to forward X11 connections over the secure channel. The user should normally not set DISPLAY explicitly, as that will render the X11 connection insecure (and will require the user to manually copy any required authorization cookies).
HOME	Set to the path of the user's home directory.
LOGNAME	Synonym for USER ; set for compatibility with systems that use this variable.
MAIL	Set to point the user's mailbox.
PATH	Set to the default PATH , as specified when compiling ssh or, on some systems, <i>/etc/environment</i> or <i>/etc/default/login</i> .
SSH_AUTHENTICATION_FD	This is set to an integer value if you are using the authentication agent and a connection to it has been forwarded. The value indicates a file descriptor number used for communicating with the agent. On some systems, SSH_AUTHENTICATION_SOCKET may be used instead to indicate the path of a unixdomain socket used to communicate with the agent (this method is less secure, and is only used on systems that don't support the first method).
SSH_CLIENT	Identifies the client end of the connection. The variable contains three space-separated values: client ip-address, client port number, and server port number.
SSH_TTY	This is set to the name of the tty (path to the device) associated with the current shell or command. If the current session has no tty, this variable is not set.
TZ	The timezone variable is set to indicate the present timezone if it was set when the daemon was started (e.i., the daemon passes the value on to new connections).
USER	Set to the name of the user logging in. Additionally, ssh reads <i>/etc/environment</i> and <i>\$HOME/.ssh/environment</i> , and adds lines of the format <i>VAR_NAME=value</i> to the environment. Some systems may have still additional mechanisms for setting up the environment, such as <i>/etc/default/login</i> on Solaris.

FILES

\$HOME/.ssh/known_hosts	Records host keys for all hosts the user has logged into (that are not in /etc/ssh_known_hosts). See sshd manual page.
\$HOME/.ssh/random_seed	Used for seeding the random number generator. This file contains sensitive data and should read/write for the user and not accessible for others. This file is created the first time the program is run and updated automatically. The user should never need to read or modify this file.
\$HOME/.ssh/identity	Contains the RSA authentication identity of the user. This file contains sensitive data and should be readable by the user but not accessible by others. It is possible to specify a passphrase when generating the key; the passphrase will be used to encrypt the sensitive part of this file using 3des .
\$HOME/.ssh/identity.pub	Contains the public key for authentication (public part of the identity file in human-readable form). The contents of this file should be added to \$HOME/.ssh/authorized_keys on all machines where you wish to log in using RSA authentication. This file is not sensitive and can (but need not) be readable by anyone. This file is never used automatically and is not necessary; it is only provided for the convenience of the user.
\$HOME/.ssh/config	This is the per-user configuration file. The format of this file is described above. This file is used by the ssh client. This file does not usually contain any sensitive information, but the recommended permissions are read/write for the user, and not accessible by others.
\$HOME/.ssh/authorized_keys	Lists the RSA keys that can be used for logging in as this user. The format of this file is described in the sshd manual page. In the simplest form the format is the same as the .pub identity files (that is, each line contains the number of bits in modulus, public exponent, modulus, and comment fields, separated by spaces). This file is not highly sensitive, but the recommended permissions are read/write for the user, and not accessible by others.
/etc/ssh_known_hosts	<p>System-wide list of known host keys. This file should be prepared by the system administrator to contain the public host keys of all machines in the organization. This file should be world-readable. This file contains public keys, one per line, in the following format (fields separated by spaces): system name, number of bits in modulus, public exponent, modulus, and optional comment field. When different names are used for the same machine, all such names should be listed, separated by commas. The format is described on the sshd manual page.</p> <p>The canonical system name (as returned by name servers) is used by sshd to verify the client host when logging in; other names are needed because ssh does not convert the user-supplied name to a canonical name before checking the key, because someone with access to the name servers would then be able to fool host authentication.</p>
/etc/ssh_config	System-wide configuration file. This file provides defaults for those values that are not specified in the user's configuration file, and for those users who do not have a configuration file. This file must be world-readable.
\$HOME/.rhosts	This file is used in .rhosts authentication to list the host/user pairs that are permitted to log in. (Note that this file is also used by rlogin and rsh , which makes using this file insecure.) Each line of the file contains a host name (in the canonical form returned by name servers), and then a user name on that host, separated by a space. This file must be owned by the user, and must not have write permissions for anyone else. The recommended permission is read/write for the user, and not accessible by

others.

Note that by default **sshd** will be installed so that it requires successful RSA host authentication before permitting **.rhosts** authentication. If your server machine does not have the client's host key in **/etc/ssh_known_hosts**, you can store it in **\$HOME/.ssh/known_hosts**. The easiest way to do this is to connect back to the client from the server machine using **ssh**; this will automatically add the host key in **\$HOME/.ssh/known_hosts**.

\$HOME/.shosts

This file is used exactly the same way as **.rhosts**. The purpose for having this file is to be able to use **rhosts** authentication with **ssh** without permitting login with **rlogin** or **rsh**.

/etc/hosts.equiv

This file is used during **.rhosts** authentication. It contains canonical hosts names, one per line (the full format is described on the **sshd** manual page). If the client host is found in this file, login is automatically permitted provided client and server user names are the same. Additionally, successful RSA host authentication is normally required. This file should only be writable by root.

/etc/shosts.equiv

This file is processed exactly as **/etc/hosts.equiv**. This file may be useful to permit logins using **ssh** but not using **rsh** or **rlogin**.

/etc/sshrd

Commands in this file are executed by **ssh** when the user logs in just before the user's shell (or command) is started. See the **sshd** manual page for more information.

\$HOME/.ssh/rc

Commands in this file are executed by **ssh** when the user logs in just before the user's shell (or command) is started. See the **sshd** manual page for more information.

INSTALLATION

Ssh is normally installed as `suid root`. It needs root privileges only for rhosts authentication (rhosts authentication requires that the connection must come from a privileged port, and allocating such a port requires root privileges). It also needs to be able to read `/etc/ssh_host_key` to perform **RSA** host authentication. It is possible to use **ssh** without root privileges, but rhosts authentication will then be disabled. **Ssh** drops any extra privileges immediately after the connection to the remote host has been made.

Considerable work has been put into making **ssh** secure. However, if you find a security problem, please report it immediately to <F-Secure-SSH-Bugs@DataFellows.com>.

SEE ALSO

sshd(8), ssh-keygen(1), ssh-agent(1), ssh-add(1), scp(1), make-ssh-known-hosts(1), rlogin(1), rsh(1), telnet(1)

SSH-ADD

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

RETURN STATUS

FILES

SEE ALSO

NAME

ssh-add - adds identities for the authentication agent

SYNOPSIS

ssh-add [-l] [-d] [-D] [file...]

DESCRIPTION

Ssh-add adds identities to the authentication agent, **ssh-agent**. When run without arguments, it adds the file ***\$HOME/.ssh/identity***. Alternative file names can be given on the command line. If any file requires a passphrase, **ssh-add** asks for the passphrase from the user. If the user is using X11, the passphrase is requested using a small X11 program; otherwise it is read from the user's tty. (Note: it may be necessary to redirect stdin from /dev/null to get the passphrase requested using X11.)

The authentication agent must be running and must be an ancestor of the current process for **ssh-add** to work.

OPTIONS

- l Lists all identities currently represented by the agent.
- d Instead of adding the identity, removes the identity from the agent.
- D Deletes all identities from the agent.

RETURN STATUS

Ssh-add returns one of the following exit statuses. These may be useful in scripts.

0	The requested operation was performed successfully.
1	No connection could be made to the authentication agent. Presumably there is no authentication agent active in the execution environment of ssh-add .
2	The user did not supply a required passphrase.
3	An identify file could not be found, was not readable, or was in bad format.
4	The agent does not have the requested identity.
5	An unspecified error has occurred; this is a catchall for errors not listed above.

FILES

`$HOME/.ssh/identity`

Contains the RSA authentication identity of the user. This file should not be readable by anyone but the user. It is possible to specify a passphrase when generating the key; that passphrase will be used to encrypt the private part of this file. This is the default file added by `ssh-add` when no other files have been specified.

If `ssh-add` needs a passphrase, it will read the passphrase from the current terminal if it was run from a terminal. If `ssh-add` does not have a terminal associated with it but `DISPLAY` is set, it will open an X11 window to read the passphrase. This is particularly useful when calling `ssh-add` from a `.Xsession` or related script. (Note that on some machines it may be necessary to redirect the input from `/dev/null` to make this work.)

SEE ALSO

`ssh-agent(1)`, `ssh-keygen(1)`, `ssh(1)`, `sshd(8)`

SSH-AGENT

NAME

SYNOPSIS

DESCRIPTION

FILES

SEE ALSO

NAME

ssh-agent - authentication agent

SYNOPSIS

ssh-agent command

DESCRIPTION

Ssh-agent is a program to hold authentication private keys. The idea is that **ssh-agent** is started in the beginning of an X-session or a login session, and all other windows or programs are started as children of the **ssh-agent** program (the command normally starts X or is the user shell). Programs started under the agent inherit a connection to the agent, and the agent is automatically used for RSA authentication when logging to other machines using **ssh**.

The agent initially does not have any private keys. Keys are added using **ssh-add**. When executed without arguments, **ssh-add** adds the **\$HOME/.ssh/identity** file. If the identity has a passphrase, **ssh-add** asks for the passphrase (using a small X11 application if running under X11, or from the terminal if running without X). It then sends the identity to the agent. Several identities can be stored in the agent; the agent can automatically use any of these identities. **Ssh-add -l** displays the identities currently held by the agent.

The idea is that the agent is run in the user's local PC, laptop, or terminal. Authentication data need not be stored on any other machine, and authentication passphrases never go over the network. However, the connection to the agent is forwarded over ssh remote logins, and the user can thus use the privileges given by the identities anywhere in the network in a secure way.

A connection to the agent is inherited by child programs. There are two alternative methods for inheriting the agent. The preferred method is to have an open file descriptor which is inherited, and have an environment variable (**SSH_AUTHENTICATION_FD**) contain the number of this descriptor. This restricts access to the authentication agent to only those programs that are siblings of the agent, and it is fairly difficult even for root to get unauthorized access to the agent.

On some machines, an alternative method is used. A unixdomain socket is created (**/tmp/ssh_agent.***), and the name of this socket is stored in the **SSH_AUTHENTICATION_SOCKET** environment variable. The socket is made accessible only to the current user. This method is easily abused by root or another instance of the same user. The socket is only used if **ssh** is unable to find a file descriptor that would not be closed by shells.

The agent exits automatically when the command given on the command line terminates.

FILES

\$HOME/.ssh/identity

Contains the RSA authentication identity of the user. This file should not be readable by anyone but the user. It is possible to specify a passphrase when generating the key; that passphrase will be used to encrypt the private part of this file. This file is not used by **ssh-agent**, but is normally added to the agent using **ssh-add** at login time.

/tmp/ssh_agent.<pid>

Unix-domain sockets used to contain the connection to the authentication agent. These sockets should only be readable by the owner. The sockets should get automatically removed when the agent exits.

SEE ALSO

`ssh-add(1)`, `ssh-keygen(1)`, `ssh(1)`, `sshd(8)`

SSHD

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

CONFIGURATION FILE

LOGIN PROCESS

AUTHORIZED_KEYS FILE FORMAT

Examples

SSH_KNOWN_HOSTS FILE FORMAT

Examples

FILES

INSTALLATION

SEE ALSO

NAME

sshd - F-Secure SSH daemon

SYNOPSIS

sshd [-b bits] [-d] [-f config_file] [-g login_grace_time] [-h host_key_file] [-i] [-k key_gen_time] [-p port] [-q]

DESCRIPTION

Sshd (F-Secure SSH Daemon) is the daemon program for F-Secure SSH. Together these programs replace **rlogin** and **rsh** programs, and provide secure encrypted communications between two untrusted hosts over an insecure network. The programs are intended to be as easy to install and use as possible.

Sshd is the daemon that listens for connections from clients. It is normally started at boot from **/etc/rc.local** or equivalent. It forks a new daemon for each incoming connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange.

Sshd works as follows. Each host has a host-specific RSA key (normally 1024 bits) used to identify the host. Additionally, when the daemon starts, it generates a server RSA key (normally 768 bits). This key is normally regenerated every hour if it has been used, and is never stored on disk.

Whenever a client connects the daemon, the daemon sends its host and server public keys to the client. The client compares the host key against its own database to verify that it has not changed. The client then generates a 256 bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then start to use this random number as a session key which is used to encrypt all further communications in the session. The rest of the session is encrypted using a conventional cipher. Currently, 3DES, Blowfish, DES, and IDEA as an option. 3DES is used by default. The client selects the encryption algorithm to use from those offered by the server.

Next, the server and the client enter an authentication dialog. The client tries to authenticate itself using .rhosts authentication, .rhosts authentication combined with RSA host authentication, RSA challenge-response authentication, or password based authentication.

Rhosts authentication is normally disabled because it is

fundamentally insecure, but can be enabled in the server configuration file if desired. System security is not improved unless **rshd**(8), **rlogind**(8), **rexecd**(8), and **rexed** (8) are disabled (thus completely disabling **rlogin**(1) and **rsh**(1) into that machine).

If the client successfully authenticates itself, a dialog for preparing the session is entered. At this time the client may request things like allocating a pseudo-tty, forwarding X11 connections, forwarding TCP/IP connections, or forwarding the authentication agent connection over the secure channel.

Finally, the client either requests a shell or execution of a command. The sides then enter session mode. In this mode, either side may send data at any time, and such data is forwarded to/from the shell or command on the server side, and the user terminal in the client side.

When the user program terminates and all forwarded X11 and other connections have been closed, the server sends command exit status to the client, and both sides exit.

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

Sshd rereads its configuration file if it is sent the hangup signal, SIGHUP.

OPTIONS

-b <i>bits</i>	Specifies the number of bits in the server key (default 768).
-d	Debug mode. The server sends verbose debug output to the system log, and does not put itself in the background. The server also will not fork and will only process one connection. This option is only intended for debugging for the server.
-f <i>configuration_file</i>	Specifies the name of the configuration file. The default is <i>/etc/sshd_config</i> .
-g <i>login_grace_time</i>	Gives the grace time for clients to authenticate themselves (default 600 seconds). If the client fails to authenticate the user within this many seconds, the server disconnects and exits. A value of zero indicates no limit.
-h <i>host_key_file</i>	Specifies the file from which the host key is read (default <i>/etc/ssh_host_key</i>). This option must be given if sshd is not run as root (as the normal host file is normally not readable by anyone but root).
-i	Specifies that sshd is being run from inetd. Sshd is normally not run from inetd because it needs to generate the server key before it can respond to the client, and this may take tens of seconds. Clients would have to wait too long if the key was regenerated every time. However, with small key sizes (e.g. 512) using sshd from inetd may be feasible.
-k <i>key_gen_time</i>	Specifies how often the server key is regenerated (default 3600 seconds, or one hour). The motivation for regenerating the key fairly often is that the key is not stored anywhere, and after about an hour, it becomes impossible to recover the key for decrypting intercepted communications even if the machine is cracked into or physically seized. A value of zero indicates that the key will never be regenerated.
-p <i>port</i>	Specifies the port on which the server listens for connections (default 22).
-q	Quiet mode. Nothing is sent to the system log. Normally the beginning, authentication, and termination of each connection is logged.

CONFIGURATION FILE

Sshd reads configuration data from */etc/ssh/sshd_config* (or the file specified with *-f* on the command line). The file contains keyword-value pairs, one per line. Lines starting with *#* and empty lines are interpreted as comments.

AllowHosts	<p>This keyword can be followed by any number of host name patterns, separated by spaces. If specified, login is allowed only from hosts whose name matches one of the patterns. <i>*</i> and <i>?</i> can be used as wildcards in the patterns. Normal name servers are used to map the client's host into a canonical host name. If the name cannot be mapped, its IP-address is used as the host name. By default all hosts are allowed to connect.</p> <p>Note that sshd can also be configured to use <i>tcp_wrappers</i> using the <i>--with-libwrap</i> compile-time configuration option.</p>
DenyHosts	<p>This keyword can be followed by any number of host name patterns, separated by spaces. If specified, login is disallowed from the hosts whose name matches any of the patterns.</p>
FascistLogging	<p>Specifies whether to use verbose logging. Verbose logging violates the privacy of users and is not recommended. The argument must be <i>"yes"</i> or <i>"no"</i> (without the quotes). The default is <i>"no"</i>.</p>
HostKey	<p>Specifies the file containing the private host key (default <i>/etc/ssh_host_key</i>).</p>
IgnoreRhosts	<p>Specifies that <i>rhosts</i> and <i>shosts</i> files will not be used in authentication. <i>/etc/hosts.equiv</i> and <i>/etc/shosts.equiv</i> are still used. The default is <i>"no"</i>.</p>
KeepAlive	<p>Specifies whether the system should send keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying. On the other hand, if keepalives are not sent, sessions may hang indefinitely on the server, leaving <i>"ghost"</i> users and consuming server resources.</p> <p>The default is <i>"yes"</i> (to send keepalives), and the server will notice if the network goes down or the client host reboots. This avoids infinitely hanging sessions.</p> <p>To disable keepalives, the value should be set to <i>"no"</i> in both the server and the client configuration files.</p>
KeyRegenerationInterval	<p>The server key is automatically regenerated after this many seconds (if it has been used). The purpose of regeneration is to prevent decrypting captured sessions by later breaking into the machine and stealing the keys. The key is never stored anywhere. If the value is 0, the key is never regenerated. The default is 3600 (seconds).</p>
LoginGraceTime	<p>The server disconnects after this time if the user has not successfully logged in. If the value is 0, there is no time limit. The default is 600 (seconds).</p>
PasswordAuthentication	<p>Specifies whether password authentication is allowed. The default is <i>"yes"</i>.</p>
PermitEmptyPasswords	<p>When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings. The default is <i>"yes"</i>.</p>
PermitRootLogin	<p>Specifies whether the root can log in using ssh. May be set to <i>"yes"</i>,</p>

```nopwd```, or ```no```. The default is ```yes```, allowing root logins through any of the authentication types allowed for other users. The ```nopwd``` value disables password-authenticated root logins. The ```no``` value disables root logins through any of the authentication methods. (```nopwd``` and ```no``` are equivalent unless you have a `.rhosts`, `.shosts`, or `.ssh/authorized_keys` file in the root home directory.)

Root login with RSA authentication when the ```command``` option has been specified will be allowed regardless of the value of this setting (which may be useful for taking remote backups even if root login is normally not allowed).

<b>PidFile</b>	Specifies the location of the file containing the process ID of the master <code>sshd</code> daemon (default: <code>/etc/sshd.pid</code> or <code>/var/run/sshd.pid</code> , depending on the system).
<b>Port</b>	Specifies the port number that <code>sshd</code> listens on. The default is 22.
<b>PrintMotd</b>	Specifies whether <b>sshd</b> should print <code>/etc/motd</code> when a user logs in interactively. (On some systems it is also printed by the shell, <code>/etc/profile</code> , or equivalent.) The default is <code>``yes``</code> .
<b>QuietMode</b>	Specifies whether the system runs in quiet mode. In quiet mode, nothing is logged in the system log, except fatal errors. The default is <code>``no``</code> .
<b>RandomSeed</b>	Specifies the file containing the random seed for the server; this file is created automatically and updated regularly. The default is <code>/etc/ssh_ran_dom_seed</code> .
<b>RhostsAuthentication</b>	Specifies whether authentication using <code>rhosts</code> or <code>/etc/hosts.equiv</code> files is sufficient. Normally, this method should not be permitted because it is insecure. <code>RhostsRSAAuthentication</code> should be used instead, because it performs RSA-based host authentication in addition to normal <code>rhosts</code> or <code>/etc/hosts.equiv</code> authentication. The default is <code>``no``</code> .
<b>RhostsRSAAuthentication</b>	Specifies whether <code>rhosts</code> or <code>/etc/hosts.equiv</code> authentication together with successful RSA host authentication is allowed. The default is <code>``yes``</code> .
<b>RSAAuthentication</b>	Specifies whether pure RSA authentication is allowed. The default is <code>``yes``</code> .
<b>ServerKeyBits</b>	Defines the number of bits in the server key. The minimum value is 512, and the default is 768.
<b>StrictModes</b>	Specifies whether <code>ssh</code> should check file modes and ownership of the user's home directory and <code>rhosts</code> files before accepting login. This is normally desirable because novices sometimes accidentally leave their directory or files world-writable. The default is <code>``yes``</code> .
<b>SyslogFacility</b>	Gives the facility code that is used when logging messages from <b>sshd</b> . The possible values are: <code>DAEMON</code> , <code>USER</code> , <code>AUTH</code> , <code>LOCAL0</code> , <code>LOCAL1</code> , <code>LOCAL2</code> , <code>LOCAL3</code> , <code>LOCAL4</code> , <code>LOCAL5</code> , <code>LOCAL6</code> , <code>LOCAL7</code> . The default is <code>DAEMON</code> .
<b>X11Forwarding</b>	Specifies whether X11 forwarding is permitted. The default is <code>``yes``</code> . Note that disabling X11 forwarding does not improve security in any way, as users can always install their own forwarders.

## LOGIN PROCESS

When a user successfully logs in, **sshd** does the following:

1. If the login is on a tty, and no command has been specified, prints last login time and /etc/motd (unless prevented in the configuration file or by \$HOME/.hushlogin; see the FILES section).
2. If the login is on a tty, records login time.
3. Checks /etc/nologin; if it exists, prints contents and quits (unless root).
4. Changes to run with normal user privileges.
5. Sets up basic environment.
6. Reads /etc/environment if it exists.
7. Reads \$HOME/.ssh/environment if it exists.
8. Changes to user's home directory.
9. If \$HOME/.ssh/rc exists, runs it (with the user's shell); else if /etc/sshrd exists, runs it (with /bin/sh); otherwise runs xauth. The ``rc" files are given the X11 authentication protocol and cookie in standard input.
10. Runs user's shell or command.

## AUTHORIZED\_KEYS FILE FORMAT

The **\$HOME/.ssh/authorized\_keys** file lists the RSA keys that are permitted for RSA authentication. Each line of the file contains one key (empty lines and lines starting with a '#' are ignored as comments). Each line consists of the following fields, separated by spaces: options, bits, exponent, modulus, comment. The options field is optional; its presence is determined by whether the line starts with a number or not (the option field never starts with a number). The bits, exponent, modulus and comment fields give the RSA key; the comment field is not used for anything (but may be convenient for the user to identify the key).

Note that lines in this file are usually several hundred bytes long (because of the size of the RSA key modulus). You don't want to type them in; instead, copy the identity.pub file and edit it.

The options (if present) consists of comma-separated option specifications. No spaces are permitted, except within double quotes. The following option specifications are supported:

<b>from="pattern-list"</b>	Specifies that in addition to RSA authentication, the canonical name of the remote host must be present in the comma-separated list of patterns ('*' and '?' serve as wildcards). The list may also contain patterns negated by prefixing them with '!'; if the canonical host name matches a negated pattern, the key is not accepted. The purpose of this option is to optionally increase security: RSA authentication by itself does not trust the network or name servers or anything (but the key); however, if somebody somehow steals the key, the key permits an intruder to log in from anywhere in the world. This additional option makes using a stolen key more difficult (name servers and/or routers would have to be compromised in addition to just the key).
<b>command="command"</b>	Specifies that the command is executed whenever this key is used for authentication. The command supplied by the user (if any) is ignored. The command is run on a pty if the connection requests a pty; otherwise it is run without a tty. A quote may be included in the command by quoting it with a backslash. This option might be useful to restrict certain RSA keys to perform just a specific operation. An example might be a key that permits remote backups but nothing else. Notice that the client may specify TCP/IP and/or X11 forwardings unless they are explicitly prohibited.
<b>environment="NAME=value"</b>	Specifies that the string is to be added to the environment when logging in using this key. Environment variables set this way override other default environment values. Multiple options of this type are permitted.
<b>no-port-forwarding</b>	Forbids TCP/IP forwarding when this key is used for authentication. Any port forward requests by the client will return an error. This might be used e.g. in connection with the <b>command</b> option.
<b>no-X11-forwarding</b>	Forbids X11 forwarding when this key is used for authentication. Any X11 forward requests by the client will return an error.
<b>no-agent-forwarding</b>	Forbids authentication agent forwarding when this key is used for authentication.
<b>no-pty</b>	Prevents tty allocation (a request to allocate a pty will fail).

## Examples

1024 33 12121...312314325 ylo@foo.bar

from="\*.niksula.hut.fi,!pc.niksula.hut.fi" 1024 35 23...2334 ylo@niksula

command="dump /home",no-pty,no-port-forwarding 1024 33 23...2323 backup.hut.fi

## SSH\_KNOWN\_HOSTS FILE FORMAT

The ***/etc/ssh\_known\_hosts*** and ***\$HOME/.ssh/known\_hosts*** files contain host public keys for all known hosts. The global file should be prepared by the administrator (optional), and the per-user file is maintained automatically: whenever the user connects an unknown host its key is added to the per-user file. The recommended way to create ***/etc/ssh\_known\_hosts*** is to use the ***make-ssh-known-hosts*** command.

Each line in these files contains the following fields: hostnames, bits, exponent, modulus, comment. The fields are separated by spaces.

Hostnames is a comma-separated list of patterns ('\*' and '?' act as wildcards); each pattern in turn is matched against the canonical host name (when authenticating a client) or against the user-supplied name (when authenticating a server). A pattern may also be preceded by '!' to indicate negation: if the host name matches a negated pattern, it is not accepted (by that line) even if it matched another pattern on the line.

Bits, exponent, and modulus are taken directly from the host key; they can be obtained e.g. from ***/etc/ssh\_host\_key.pub***. The optional comment field continues to the end of the line, and is not used.

Lines starting with '#' and empty lines are ignored as comments.

When performing host authentication, authentication is accepted if any matching line has the proper key. It is thus permissible (but not recommended) to have several lines or different host keys for the same names. This will inevitably happen when short forms of host names from different domains are put in the file. It is possible that the files contain conflicting information;

authentication is accepted if valid information can be found from either file.

Note that the lines in these files are typically hundreds of characters long, and you definitely don't want to type in the host keys by hand. Rather, generate them by a script (see ***make-ssh-known-hosts(1)***) or by taking ***/etc/ssh\_host\_key.pub*** and adding the host names at the front.

## Examples

closenet,closenet.hut.fi,...,130.233.208.41 1024 37 159...93 closenet.hut.fi

## FILES

<b><i>/etc/sshd_config</i></b>	Contains configuration data for <b>sshd</b> . This file should be writable by root only, but it is recommended (though not necessary) that it be worldreadable.
<b><i>/etc/ssh_host_key</i></b>	Contains the private part of the host key. This file is normally created automatically by ``make install'', but can also be created manually using <code>ssh-keygen(1)</code> . This file should only be owned by root, readable only by root, and not accessible to others.
<b><i>/etc/ssh_host_key.pub</i></b>	Contains the public part of the host key. This file is normally created automatically by ``make install'', but can also be created manually. This file should be world-readable but writable only by root. Its contents should match the private part. This file is not really used for anything; it is only provided for the convenience of the user so its contents can be copied to known hosts files.
<b><i>/etc/ssh_random_seed</i></b>	This file contains a seed for the random number generator. This file should only be accessible by root.
<b><i>/var/run/sshd.pid</i></b>	Contains the process id of the <b>sshd</b> listening for connections (if there are several daemons running concurrently for different ports, this contains the pid of the one started last). The contents of this file are not sensitive; it can be world-readable.
<b><i>\$HOME/.ssh/authorized_keys</i></b>	<p>Lists the RSA keys that can be used to log into the user's account. This file must be readable by root (which may on some machines imply it being worldreadable if the user's home directory resides on an NFS volume). It is recommended that it not be accessible by others. The format of this file is described above.</p> <p><i>/etc/ssh_known_hosts</i> and <i>\$HOME/.ssh/known_hosts</i> These files are consulted when using <i>rhhosts</i> with RSA host authentication to check the public key of the host. The key must be listed in one of these files to be accepted. (The client uses the same files to verify that the remote host is the one we intended to connect.) These files should be writable only by root/the owner. <i>/etc/ssh_known_hosts</i> should be world-readable, and <i>\$HOME/.ssh/known_hosts</i> can but need not be worldreadable.</p>
<b><i>/etc/nologin</i></b>	If this file exists, <b>sshd</b> refuses to let anyone except root log in. The contents of the file are displayed to anyone trying to log in, and non-root connections are refused. The file should be worldreadable.
<b><i>\$HOME/.rhosts</i></b>	<p>This file contains host-username pairs, separated by a space, one per line. The given user on the corresponding host is permitted to log in without password. The same file is used by <i>rlogind</i> and <i>rshd</i>. <b>Ssh</b> differs from <i>rlogind</i> and <i>rshd</i> in that it requires RSA host authentication in addition to validating the host name retrieved from domain name servers (unless compiled with the <code>--with-rhosts</code> configuration option). The file must be writable only by the user; it is recommended that it not be accessible by others.</p> <p>If is also possible to use <i>netgroups</i> in the file. Either host or user name may be of the form <code>+@groupname</code> to specify all hosts or all users in the group.</p>
<b><i>\$HOME/.shosts</i></b>	<p>For <b>ssh</b>, this file is exactly the same as for <i>.rhosts</i>. However, this file is not used by <i>rlogin</i> and <i>rshd</i>, so using this permits access using <b>ssh</b> only.</p> <p><i>/etc/hosts.equiv</i> This file is used during <i>.rhosts</i> authentication. In the simplest form, this file contains host names, one per line. Users on those hosts are permitted to log in without a password, provided they have the same user name on both machines. The host name may also be followed</p>



by a user name; such users are permitted to log in as **any** user on this machine (except root). Additionally, the syntax `+@group` can be used to specify netgroups. Negated entries start with `^-`.

If the client host/user is successfully matched in this file, login is automatically permitted provided the client and server user names are the same. Additionally, successful RSA host authentication is normally required. This file must be writable only by root; it is recommended that it be world-readable.

Warning: It is almost never a good idea to use user **names in hosts.equiv**. Beware that it really means that the named user(s) can log in as **anybody**, which includes bin, daemon, adm, and other accounts that own critical binaries and directories. Using a user name practically grants the user root access. The only valid use for user names that I can think of is in negative entries. **Note that this warning also applies to rsh/rlogin.**

**/etc/shosts.equiv**

This is processed exactly as `/etc/hosts.equiv`. However, this file may be useful in environments that want to run both rsh/rlogin and **ssh**.

**/etc/environment**

This file is read into the environment at login (if it exists). It can only contain empty lines, comment lines (that start with `#`), and assignment lines of the form `name=value`. This file is processed in all environments (normal rsh/rlogin only process it on AIX and potentially some other systems). The file should be writable only by root, and should be world-readable.

**\$HOME/.ssh/environment**

This file is read into the environment after `/etc/environment`. It has the same format. The file should be writable only by the user; it need not be readable by anyone else.

**\$HOME/.ssh/rc**

If this file exists, it is run with the user's shell after reading the environment files but before starting the user's shell or command. If X11 spoofing is in use, this will receive the `proto cookie` pair in standard input (and `DISPLAY` in environment). This must call `xauth` in that case.

The primary purpose of this file is to run any initialization routines which may be needed before the user's home directory becomes accessible; AFS is a particular example of such an environment.

This file will probably contain some initialization code followed by something similar to: `if read proto cookie; then echo add $DISPLAY $proto $cookie | xauth -q -; fi`.

If this file does not exist, `/etc/sshr` is run, and if that does not exist either, `xauth` is used to store the cookie.

This file should be writable only by the user, and need not be readable by anyone else.

**/etc/sshr**

Like `$HOME/.ssh/rc`, but run with `/bin/sh`. This can be used to specify machine-specific login-time initializations globally. This file should be writable only by root, and should be world-readable.

## INSTALLATION

**Sshd** 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.

**Sshd** is normally started from **/etc/rc.local** or equivalent at system boot.

Considerable work has been put to making sshd secure. However, if you find a security problem, please report it immediately to <F-Secure-SSH-Bugs@DataFellows.com>.

## SEE ALSO

**ssh(1)**, **make-ssh-known-hosts(1)**, **ssh-keygen(1)**, **sshagent(1)**, **ssh-add(1)**, **scp(1)**, **rlogin(1)**, **rsh(1)**

# SSH-KEYGEN

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

FILES

SEE ALSO

## NAME

**ssh-keygen** - authentication key pair generator

## SYNOPSIS

```
ssh-keygen [-b bits] [-f file] [-N new_passphrase] [-C comment]
ssh-keygen -p [-P old_passphrase] [-N new_passphrase]
ssh-keygen -c [-P passphrase] [-C comment]
ssh-keygen -u [-f file] [-P passphrase]
```

## DESCRIPTION

**Ssh-keygen** generates and manages authentication keys for **ssh**(1). Normally each user wishing to use **ssh** with RSA authentication runs this once to create the authentication key in *\$HOME/.ssh/identity*. Additionally, the system administrator may use this to generate host keys.

Normally this program generates the key and asks for a file in which to store the private key. The public key is stored in a file with the same name but ".pub" appended. The program also asks for a passphrase. The passphrase may be empty to indicate no passphrase (host keys must have empty passphrase), or it may be a string of arbitrary length. Good passphrases are 10-30 characters long and are not simple sentences or otherwise easily guessable (English prose has only 1-2 bits of entropy per word, and provides very bad passphrases). The passphrase can be changed later by using the **-p** option.

There is no way to recover a lost passphrase. If the passphrase is lost or forgotten, you will have to generate a new key and copy the corresponding public key to other machines.

**Using good, unguessable passphrases is strongly recommended. Empty passphrases should not be used.**

There is also a comment field in the key file that is only for convenience to the user to help identify the key. The comment can tell what the key is for, or whatever is useful. The comment is initialized to user@host when the key is created, but can be changed using the **-c** option.

The cipher to be used when encrypting keys with a passphrase is by default 3des. Using the **-u** option, keys encrypted in any supported cipher can be updated to use this default cipher.

## OPTIONS

<b>-b</b> <i>bits</i>	Specifies the number of bits in the key to create. Minimum is 512 bits. Generally 1024 bits is considered sufficient, and key sizes above that no longer improve security but make things slower. The default is 1024 bits.
<b>-c</b>	Requests changing the comment in the private and public key files. The program will prompt for the file containing the private keys, for passphrase if the key has one, and for the new comment.
<b>-f</b>	Specifies the file name in which to load/store the key.
<b>-p</b>	Requests changing the passphrase of a private key file instead of creating a new private key. The program will prompt for the file containing the private key, for the old passphrase, and twice for the new passphrase.
<b>-u</b>	Requests that the key's cipher is changed to the current default cipher (determined at compile-time currently 3des).
<b>-C</b>	Provides the new comment.
<b>-N</b>	Provides the new passphrase.
<b>-P</b>	Provides the (old) passphrase.



## FILES

<b><i>\$HOME/.ssh/random_seed</i></b>	Used for seeding the random number generator. This file should not be readable by anyone but the user. This file is created the first time the program is run, and is updated every time.
<b><i>\$HOME/.ssh/identity</i></b>	Contains the RSA authentication identity of the user. This file should not be readable by anyone but the user. It is possible to specify a passphrase when generating the key; that passphrase will be used to encrypt the private part of this file using 3DES. This file is not automatically accessed by <b>ssh-keygen</b> , but it is offered as the default file for the private key.
<b><i>\$HOME/.ssh/identity.pub</i></b>	Contains the public key for authentication. The contents of this file should be added to <b><i>\$HOME/.ssh/authorized_keys</i></b> on all machines where you wish to log in using RSA authentication. There is no need to keep the contents of this file secret.

## **SEE ALSO**

**ssh(1)**, **sshd(8)**, **ssh-agent(1)**, **ssh-add(1)**

