

HP TCP/IP Services for OpenVMS

Guide to SSH

OpenVMS 7.3-2

This is a new manual.

Hewlett-Packard Company

PROPRIETARY INFORMATION

Furnished for Field Test Purposes Only

The information contained herein is furnished
in confidence and is subject to the terms and conditions of
a License Agreement for testing
Hewlett-Packard software.



Manufacturing Part Number: AA-xxxxx-TE

Field Test DRAFT

© Copyright 2003 Hewlett-Packard Development Company, L.P.

Legal Notice

Confidential computer software. Valid license from HP required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Neither HP nor any of its subsidiaries shall be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for HP products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

ZKnnnn

1. Secure Shell Overview

Introduction to SSH	14
The Secure Shell Server	14
The Secure Shell Client	14
Introduction to Keys	15
Host Keys	15
User Keys	15
Generating Keys	16
Managing User's Keys	16
Authentication	16
How the SSH Client and Server Communicate	16
Port Forwarding	18

2. Configuring the Secure Shell Software

Running the TCP/IP\$CONFIG Configuration Command Procedure	20
Configuring the SSH Client.	21
Configuring the SSH Server	23

3. Customizing the SSH Run-time Environment

Customizing the User Environment on the SSH Client Host	26
Copying the Server's Public Host Key to the Client	26
Key Naming Conventions for the Server's Public Host Key.	27
Customizing the User Environment on the Server Host.	28
Authentication Methods	28
Customizing an Authentication Method.	29
Customizing Password Authentication	29
Customizing Host-Based Authentication	30
Customizing Public-Key Authentication.	31

4. Managing the SSH Service

Starting and Stopping the SSH Server.	34
Starting and Stopping the SSH Client	34
SSH Logical Names	35

5. SSH Command Reference

Before You Begin	38
Copying Files	38
.	38
Using the SCP Command.	38
Using the SFTP command	40
Login and Remote Command Execution with the SSH	41
Command Synopsis.	42
SSH Key Management.	44
Key Generation	44
Key Manipulation	45
Key Agent	46

Contents

A. SSH Client and Server Files

Client side.....	50
Server side	51

B. SSH Client and Server Configuration Files

Client Configuration File.....	56
Server Configuration File	58

Table 2-1. Files Created During the SSH Configuration Procedure on TCPIP\$SSH_DEVICE . . .	21
Table 4-1.	35
Table 5-1. SCP Command Options	40
Table 5-3. SSH Command Options	42
Table 5-4. SSH_KEYGEN Command Options	45
Table 5-5. SSH_ADD Command Options	46

Preface

The *Guide to SSH for OpenVMS* describes how to configure, customize, manage, and use the Secure Shell Software.

Intended Audience

This guide is intended for the SSH user and for system managers who need to configure, customize, manage, and use SSH.

Document Structure

This guide describes how to configure, customize, manage, and use the Secure Shell Software.

This guide contains the following chapters and appendixes:

Chapter 1 introduces definitions and concepts that are important to understanding the Secure Shell (SSH).

Chapter 2 describes how to run the configuration procedure for SSH, configure the SSH server, and configure the SSH client.

Chapter 3 describes how to customize the SSH run-time environment to meet your organization's specific security needs.

Chapter 4 describes how to stop and start the SSH client and server.

Chapter 5 describes SSH command and utilities that you can use to invoke SSH, copy or transfer files, and manage keys.

Appendix A summarizes information about files and directories that the SSH client and server use.

Appendix B lists the system-wide SSH client and server files that the TCPIP\$CONFIG utility generates during configuration.

The *SSH Guide for OpenVMS* assumes that readers are familiar with OpenVMS concepts and operation, and does not cover basic OpenVMS information.

Related Documents

The following manuals contain OpenVMS information that might be useful for the TCP/IP environment:

- *HP TPC/IP Services for OpenVMS Installation and Configuration*
- *HP TPC/IP Services for OpenVMS Concepts and Planning*

For additional information about HP OpenVMS products and services, see the following World Wide Web address:

<http://h71000.www7.hp.com/>

This manual describes SSH configuration and customization specific to the hp TCP/IP Services for OpenVMS implementation of the Secure Shell software. If you are looking for a comprehensive overview of SSH, you might find the following useful:

SSH, The Secure Shell: The Definitive Guide by Daniel J. Barrett, Richard Silverman

O'Reilly and Associates. January 2001.

Reader's Comments

HP welcomes your comments on this manual.

Please send comments to either of the following addresses::

Internet: openvmsdoc@hp.com

Postal Mail:

Hewlett-Packard Company
OSSG Documentation Group
ZKO3-4/U08
110 Spit Brook Road
Nashua, NH 03062-2698

How to Order Additional Documentation

For information about how to order additional documentation, visit the following World Wide Web address :

<http://h71000.www7.hp.com/>

Conventions

The following conventions may be used in this manual:

Convention	Meaning
Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1 and then press and release another key (x) or a pointing device button.
Return	In examples, a key name in bold indicates that you press that key.
...	A horizontal ellipsis in examples indicates one of the following possibilities: <ul style="list-style-type: none">– Additional optional arguments in a statement have been omitted.– The preceding item or items can be repeated one or more times.– Additional parameters, values, or other information can be entered.
.	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.

Convention	Meaning
	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
<i>italic type</i>	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error <i>number</i>), in command lines (/PRODUCER= <i>name</i>), and in command parameters in text (where (<i>dd</i>) represents the predefined par code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
Example	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX command and pathnames, PC-based commands and folders, and certain elements of the C programming language.
–	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.

1 Secure Shell Overview

This chapter introduces definitions and concepts that are important to understanding the Secure Shell (SSH). It describes:

- Introduction to SSH
- Introduction to Keys

Introduction to SSH

- Authentication
- How the Client and Server Communicate
- Port Forwarding

Introduction to SSH

Secure Shell (SSH) is a combination of client and server software that transparently encrypts and decrypts data flow between hosts on a network. This section introduces the Secure Shell client and server.

SSH provides a suite of secure network commands that you can use in addition to, or in place of, traditional non-secure network commands, like `Telnet` and `FTP`.

Using Secure Shell commands, you create a secure connection between systems running the Secure Shell client and server software by providing:

- Authentication -- Secure Shell servers and clients use an authentication method to reliably determine each other's identity and the user's identity.
- Data encryption -- Secure Shell servers and clients exchange encrypted data. Data encryption is transparent to the user.
- Data integrity -- Secure Shell servers and clients detect whether or not data was intercepted and modified while in transit.
- Nonrepudiation -- Systems can prove the origin of data so that a user or entity cannot deny having performed a particular action related to data or proof of ownership.

The Secure Shell Server

A **Secure Shell server** is a system on which the system administrator installs and runs the Secure Shell server software. Throughout the remainder of this guide, a Secure Shell server is referred to as the **SSH server**.

The SSH server accepts or rejects incoming connections to the server from the SSH clients on remote hosts. The SSH server listens on the port defined for the TCP/IP SSH service (port 22 by default). When a connection request occurs, the auxiliary server creates a new server process that controls all data exchanges over the new connection.

The SSH server provides the following functions:

- Secure remote user login
- Secure file transfers between remote computers
- Remote command execution

For all of these functions, the entire login and data transfer sessions, including user identification information, are secured through user authentication, data encryption, and data integrity.

The Secure Shell Client

A **Secure Shell client** is a system on which the system administrator installs the Secure Shell client software. The remainder of this guide refers to a Secure Shell client as the **SSH client**.

There are several SSH utilities invoked through SSH commands:

- The `SCP` and `SFTP` commands are used to copy files to and from an SSH server.
- The SSH commands to log in on a remote server and to provide remote command execution.
- The SSH key management utilities are used to generate public/private key pairs and to manipulate keys.

These commands and utilities are described in Chapter 5.

NOTE OpenVMS SSH software is based on SSH2 software from SSH Communication Security version 2.4.1. In the OpenVMS implementation, the commands `SSH`, `SCP`, and `SFTP` mean the same as `SSH2`, `SCP2`, and `SFTP2`. You can use either set of commands with OpenVMS SSH.

Introduction to Keys

SSH uses public-key cryptography to verify the identity of hosts as well as the identity of individual users. Public-key cryptography uses a pair of mathematically-related keys. One key is public, which is distributed to anyone who wants it; the other key is private, which remains known only to the owner. When a message is encrypted with a public key, it can be decrypted only using the private key.

Host Keys

The SSH host public and private keys are long-term, asymmetric keys that distinguish and identify hosts. When the SSH client is establishing a connection with the SSH server, the keys are used in two places:

1. The server host provides its public key to connecting clients so that they can verify the identity of the server.
2. The client host provides its public key to the server to allow a server host to verify the identity of the client host during host-based authentication.

Host keys are either created during TCP/IP configuration by the `TCPIP$CONFIG.COM` command procedure or manually by a system manager.

NOTE SSH is designed to configure a single SSH service listening port (22) and a single host key. All incarnations of the SSH server process will use the same host key.

User Keys

Public key authentication requires that a user have a pair of keys, a **public key** and a **private key**. The public key is published and distributed, or copied, to all the SSH servers with which the user communicates. The private key is kept on the local SSH client and is never be revealed to anyone except the key's owner.

As a user, your private and public keys are not the same as the server's private host key and public host key. The user's keys are used during public key authentication, as described in Chapter 3, and require that a user have personal public and private keys. The public key is published and distributed, or copied, to all the

Authentication

servers with which the user communicates. The private key is kept secret on the local SSH client and must never be made known to anyone except the user. The user creates the public-private key pair by using the key generation utility.

Generating Keys

Key generation is performed using the `SSH_KEYGEN` utility as described in Chapter 5. `SSH_KEYGEN` is used to generate both user's keys and host keys. For each key, the `SSH_KEYGEN` utility generates a pair of files: one with a public key and other with a private key. These files are used by cryptographic algorithms.

Managing User's Keys

It is possible for a user to need several keys, even hundreds of keys. For example, you may use one for each remote server to which you connect, or one for each account on a remote server. Two key management utilities help you manage multiple keys. They are discussed in Chapter 5:

- `SSH_AGENT`
- `SSH_ADD`

Authentication

Authentication means verifying the identity of someone or something. Every SSH connection involves two types of authentication:

- The client verifies the identity of the SSH server. This is known as **server authentication**.
The SSH server authentication process uses the server's host public key to ensure that the SSH server is not an imposter.
- The server verifies the identity of the user requesting access. This is known as **user authentication**.
The user authentication process uses system-specific user authentication method to verify the user's identity.

There are three user authentication methods from which you can choose: **password**, **host-based**, and **public-key**. All of these methods require configuration of both the client and the server systems. For more information about these methods, see Chapter 3.

How the SSH Client and Server Communicate

This section provides a brief overview of the client and server communication process. For a detailed description, refer to the book recommended in the Preface.

After the system manager installs and configures the SSH software on all client and server hosts, the system manager (and perhaps the user) customizes authentication methods on the clients and server, creates and distributes key files, and uses TCP/IP commands to start the SSH client and server.

During SSH client and server configuration, two configuration files are installed: a client configuration file, which is read by an SSH client process when the SSH command is invoked, and the server configuration file, which is read by an SSH server process when a connection request arrives from an SSH client. Appendix B lists the SSH client and server configuration files.

When the TCP/IP auxiliary server (`inetd`) starts on an SSH server host, it creates a listening socket for SSH. The server is now ready to accept a remote connection request. When you execute one of the SSH commands on a remote client host, the SSH client is initiated. The client reads the configuration file and initiates a TCP connection to a server host using the specified destination port. On an SSH server host, `inetd` creates a copy of the server process, which reads the server's configuration file. All configuration files are ASCII text files and have either `STREAM_LF` format (for example, if copied directly from UNIX), or variable-length formats (when created with the `TCPIP$CONFIG.COM` procedure or with a text editor.).

To establish a secure connection:

1. The SSH client and server exchange information about supported protocol versions. This enables different implementations to overcome incompatibilities.
2. The SSH server initiates a host public key exchange with the client to prove its identity. Each server host has a public and private key pair, which is created during the SSH server configuration. This pair uniquely identifies the server host. The first time an SSH client connects to a server, SSH prompts the user to accept a copy of the server's public host key with the following message:

```
Host key not found from the list of known hosts.  
Are you sure you want to continue connecting (yes/no)
```
3. If the user response is YES, SSH copies the server's public host key to the SSH client host. The client host uses this public host key to authenticate the SSH server on subsequent connections.
4. If, during subsequent connection attempts the SSH client detects that the SSH server's host key differs from the one stored on the client, the following message is displayed:

```
WARNING: HOST IDENTIFICATION HAS CHANGED!...
```

The message continues with text that warns of a possible "man-in-the-middle" attack. Although this message may not mean that data has been compromised (the host key may have been legitimately changed), the user should copy the server's new key or contact the system manager.
5. The SSH client and server negotiate session parameters by exchanging information about supported parameters including authentication methods, hash functions, and data compression methods.
6. The SSH client and server develop a shared (symmetric) session key to encrypt data using a key exchange method. When both the client and server know the secret data encryption key, a secure connection exists, and the client and server can securely exchange data. The session key expires when the session ends.
7. After the SSH client and server authenticate each other, the session is ready to authenticate the user using one or more of the authentication methods. Then, SSH verifies the user's identity. The user or the system manager on the client and system manager on the server coordinate authentication methods by modifying information in the client and server configuration files.
8. The SSH server checks the user's identity. The user must have a valid user account and home directory on the server. If the server fails to authenticate the user, it refuses the connection.
9. After SSH authenticates the user's identity, the actual secure data transfer between client and server occurs.
10. The SSH server runs in a loop, accepting messages from the client, performing required actions, and returning reply messages to the client.

Port Forwarding

11. When the user closes the connection, the server process terminates. The auxiliary server continues to listen for new SSH connection requests.

Port Forwarding

Port forwarding means encapsulating any TCP-based communication between the client and the server programs within an SSH session. This feature allows any TCP-based application or service to take advantage of all the benefits of SSH. SSH allows you to establish a “secure tunnel” between two hosts. After you have set up a secure tunnel, the participating applications operate transparently. For example, when you forward a regular TELNET connection through SSH, all information, including your user name, password, and actual data, are automatically encrypted and checked for integrity.

SSH forwarding includes additional features for encrypting the X protocol (for X-windows). Using SSH, you can invoke X programs on a remote machine and have them appear on your local display. In this case, all X-protocol data are secured.

2 Configuring the Secure Shell Software

After you install the TCP/IP Services software, as described in the *HP TCP/IP Services for OpenVMS Installation and Configuration* manual, you must configure the SSH service using the menu-driven TCP/IP\$CONFIG.COM command procedure.

This chapter describes:

Running the TCPIP\$CONFIG Configuration Command Procedure

- Running the TCPIP\$CONFIG Configuration Utility for the SSH Service
- Configuring the SSH Server
- Configuring the SSH Client

Running the TCPIP\$CONFIG Configuration Command Procedure

After TCP/IP installation is completed, the SSH service must be configured using the TCP/IP configuration command procedure, TCPIP\$CONFIG.COM. The configuration creates the system-wide SSH environment by setting up various components of SSH, such as configuration files and host keys.

Once you have completed the client and server configuration using TCPIP\$CONFIG, you can customize the configuration with parameters to meet your specific run-time environment. For more information on customizing your run-time environment, refer to Chapter 3, *Customizing the SSH Run-time Environment*.

To run the configuration command procedure:

1. Invoke the TCPIP\$CONFIG configuration command procedure. Refer to the *HP TCP/IP Services for OpenVMS Installation and Configuration Guide* for general configuration procedures. The Main Configuration menu is displayed:

```
hp TCP/IP Services for OpenVMS Configuration Menu
```

```
Configuration options:
```

```
1 - Core environment
2 - Client components
3 - Server components
4 - Optional components

5 - Shutdown hp TCP/IP Services for OpenVMS
6 - Startup hp TCP/IP Services for OpenVMS
7 - Run tests

A - Configure options 1 - 4
[E] - Exit configuration procedure
```

```
Enter configuration option:
```

2. Choose the option `Server components (3)` to configure the SSH server and `Client components (2)` to configure the SSH client. See *Configuring the SSH Client* and *Configuring the SSH Server*.

During the configuration procedure, TCPIP\$CONFIG creates the system-wide environment necessary to run the SSH client and server. TCPIP\$CONFIG:

- Creates the SSH server's account TCPIP\$SSH, and the account's default directory: TCPIP\$SSH_DEVICE:[TCPIP\$SSH]. Note that the default device of the account is defined by the logical name TCPIP\$SSH_DEVICE. This logical name can be assigned by the system manager. If this logical name is not defined, the default is SYS\$SYSDEVICE.
- Creates all subdirectories and files required by the SSH server.
- Copies all necessary files from the distribution kit into the appropriate directories. See Table 2-1, which lists the files created during the SSH configuration procedure.

Table 2-1 Files Created During the SSH Configuration Procedure on TCPIP\$SSH_DEVICE

TCPIP\$SSH_DEVICE:	File	Description	Server/ Client
[TCPIP\$SSH.SSH2]	SSH2_CONFIG.	Configuration file	Client
[TCPIP\$SSH.SSH2]	SSHD2_CONFIG.	Configuration file	Server (Client for host-based authen- tication)
[TCPIP\$SSH]	SHOSTS.EQUIV	This file contains a list of trusted hosts, used by the host-based authentication method.	Server
[TCPIP\$SSH.SSH2]	HOSTKEY. HOSTKEY.PUB	Private (HOSTKEY) and public (HOSTKEY.PUB) server host keys.	Server
[TCPIP\$SSH.SSH2.SEED]	RANDOM_SEED.	Generates pseudo-random numbers for cryptographics operations.	Server
[TCPIP\$SSH.SSH2.KNOWNHOSTS]		Contains public keys of all remote client hosts that may attempt to connect to the server using host-based authentication.	Server
[TCPIP\$SSH.SSH2.HOSTKEYS]		Contains host keys for all remote servers to which the user connects using the SSH client.	Client

Configuring the SSH Client

1. When you choose Client components from the TCPIP\$CONFIG command procedure Main Menu as described in *Running the Configuration Command Procedure* within this chapter, the Client Components Configuration Menu is displayed:

hp TCP/IP Services for OpenVMS Client Components Configuration Menu

Configuring the SSH Client

Configuration options:

```
1 - DHCP Client      Disabled Stopped
2 - FTP Client       Enabled  Stopped
3 - NFS Client       Disabled Stopped
4 - REXEC and RSH   Disabled Stopped
5 - RLOGIN           Disabled Stopped
6 - SMTP             Disabled Stopped
7 - SSH Client       Disabled Stopped
8 - TELNET           Enabled  Stopped
9 - TELNETSYM        Disabled Stopped
```

```
A - Configure options 1 - 9
```

```
[E] - Exit menu
```

Enter configuration option: 7

2. Enter the SSH Client configuration option number (7) at the prompt. The SSH Client Configuration Options menu is displayed:

```
SSH CLIENT Configuration
```

```
Service is not defined in the SYSUAF.
```

```
Service is not enabled.
```

```
Service is stopped.
```

```
SSH CLIENT configuration options:
```

```
1 - Enable service on this node
```

```
2 - Enable & Start service on this node
```

```
[E] - Exit SSH_CLIENT configuration
```

Enter configuration option:

3. Select the appropriate menu option. For example, select configuration option 1 to enable the SSH client on this node.

The configuration procedure copies the system-wide clien configuration file `SSH2_CONFIG`. from the distribution kit into the directory `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]`.

```
Creating TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH]SSH2_CONFIG.
```

The `SSH2_CONFIG` file contains keywords and values that each client process reads when it starts. In many cases, the system manager may want to edit this file to make it user-specific and to provide a secure environment for the client host. You can copy and edit your own version of the configuration file. For more information, refer to Chapter 3.

After the SSH client is configured, the following message and prompt are displayed if, for example, the SSH server is not enabled and has not been configured:

```
The SSH SERVER is not enabled.
```

```
* Do you want to configure SSH SERVER [NO]:
```

4. If you want to configure the SSH server, type YES and continue with Step 2 in the *Configuring the SSH Server* procedure within this chapter

or

Press ENTER to respond NO to the prompt.

NOTE You must restart the SSH client to use the changes in the SSH configuration files. You can restart by using the commands:

```
$ @SYS$STARTUP:TCPIP$SSH_CLIENT_SHUTDOWN
```

```
$ @SYS$STARTUP:TCPIP$SSH_CLIENT_STARTUP
```

Configuring the SSH Server

1. When you choose Server components from the TCPIP\$CONFIG command procedure Main Menu, as described in *Running the Configuration Command Procedure* within this chapter, the Server Components Configuration Menu is displayed:

```
hp TCP/IP Services for OpenVMS Server Components Configuration Menu
```

```
Configuration options:
```

```
1 - BIND           Disabled Stopped      12 - NTP           Disabled Stopped
2 - BOOTP          Disabled Stopped      13 - PC-NFS        Disabled Stopped
3 - DHCP           Disabled Stopped      14 - POP            Disabled Stopped
4 - FINGER         Disabled Stopped      15 - PORTMAPPER    Disabled Stopped
5 - FTP            Disabled Started      16 - RLOGIN         Disabled Started
6 - IMAP           Disabled Stopped      17 - RMT            Disabled Stopped
7 - LBROKER        Disabled Stopped      18 - SNMP           Disabled Started
8 - LPR/LPD        Disabled Stopped      19 - SSH            Enabled Started
9 - METRIC         Disabled Stopped      20 - TELNET         Disabled Started
10 - NFS           Disabled Stopped      21 - TFTP           Disabled Stopped
11 - LOCKD/STATD   Disabled Stopped      22 - XDM            Disabled Stopped

A - Configure options 1 - 22
[E] - Exit menu
```

Configuring the SSH Server

Enter configuration option:

2. Enter the SSH configuration option (19) at the prompt. The SSH Configuration Option menu appears.
3. Select the appropriate menu option. For example, select configuration option 1 to enable SSH on this server. The configuration utility creates the SSH service entry and server configuration file:

```
Creating SSH Service Entry
Creating TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH]SSHD2_CONFIG.
```

4. Respond to the following question:

```
Create a new default Server host key? [YES]
Creating private key file: TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]HOSTKEY
Creating public key file: TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]HOSTKEY.PUB
```

If you type NO to bypass creating new keys, your server may have no host keys (unless the host keys were created at an earlier time). You may need to run the key generation utility, `SSH_KEYGEN`, manually to generate keys before you can run SSH.

After the SSH server is configured, the following message and prompt are displayed if, for example, the SSH client is not enabled and has not been configured:

```
The SSH CLIENT is not enabled.
* Do you want to configure SSH CLIENT [NO]:
```

Type YES or press the ENTER key to create new host key pair files, `HOSTKEY` and `HOSTKEY.PUB`. `TCPIP$CONFIG` creates the default key pair in the directory `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]` and the following message is displayed:

5. If you want to configure the SSH client, type YES and continue with Step 2 in the *Configuring the SSH Client* procedure within this chapter

The configuration procedure copies the system-wide configuration file `SSHD2_CONFIG`. into this directory: `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]`.

This file contains keywords and values that each server process reads when it starts. The system manager may want to edit this file to make it host-specific in order to provide a secure environment for the server host.

NOTE You must restart the SSH client to use the changes in the SSH configuration files. You can restart by using the commands:

```
$ @SYS$STARTUP:TCPIP$SSH_SHUTDOWN
$ @SYS$STARTUP:TCPIP$SSH_STARTUP
```

3 Customizing the SSH Run-time Environment

When the TCP/IP configuration procedure is completed, all required system-wide SSH configuration parameters are established. The host is now prepared to become an SSH server by accepting remote connections, and the SSH client is ready to execute SSH commands. Different environments may have specific security requirements that can be achieved by exercising control over SSH run-time parameters on two levels:

Customizing the User Environment on the SSH Client Host

- A system-wide setup, which is typically the system manager's responsibility and applies to running instances of the client and server processes.
- A user-specific setup, which is typically the account owner's responsibility, from whose account the SSH connections are made on the client host or to which an SSH connection will be requested on the server host.

This chapter describes how to customize the SSH run-time environment to meet your organization's specific security needs.

An important component of the SSH run-time environment is the `.[SSH2]` subdirectory, created by the user or automatically by the SSH software, on the user's login directory (as specified by `SYS$LOGIN`).

SSH uses this subdirectory to store multiple files necessary for SSH to function. For example, if `SYS$LOGIN` is translated into `DKAO:[username]`, then this special subdirectory will be `DKAO:[username.SSH2]`. Through this manual, we refer to this directory as the "user's SSH directory."

Customizing the User Environment on the SSH Client Host

During configuration, the `SSH2_CONFIG` file is copied to `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]`. When the user invokes the SSH command, the SSH client process reads the file and creates the run-time version of the configuration parameters. If the user wants a different set of parameters (user-specific), then he or she needs to create his or her own version of the configuration file in the user's SSH directory.

The SSH client will load this file and will modify the run-time version of the parameters accordingly. You can copy this file from UNIX or OpenVMS and edit it or create a new file. The file can be either `STREAM_LF` or variable length format.

Copying the Server's Public Host Key to the Client

Any connection request from a client to an SSH server requires that the client obtain the server's public key. There are several ways to copy the server's public key to the client:

- During SSH server configuration, the `TCPIP$CONFIG` configuration procedure creates a system-wide directory `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.HOSTKEYS]`. You can copy into this directory the host keys of all the remote servers to which you will connect from the client host.

NOTE

If you copy the keys, they must have `STREAM_LF` record format and have proper access, for example: `S:RWED, O:RWED, G:RE,W:R`.

The key generation utility creates all key files as `STREAM_LF` record format files. When the SSH server transfers the server's host key file to a client host the resulting file is formatted correctly.

However, sometimes, setting up the SSH environment requires that you manually copy public key files (whether host or user's) between the SSH client and server hosts. For example, when using public-key authentication, the key file must be copied to an OpenVMS system. In these cases, FTP, for example, may create a variable-length record file. Then, the user or the system manager must convert this file, into `stream_LF` format using the OpenVMS Convert utility. Failure to convert the file will cause key processing errors.

On a connection request, the SSH client checks this directory for the appropriate server's host key and proceeds with authentication if the key is found.

- If your SSH client host does not have keys from remote servers in the system-wide directory, you can copy them manually:

1. Create the subdirectory [.HOSTKEYS] in the user's SSH directory.
2. Copy the server's public key to this directory using, for example,

```
$ COPY/FTP
```

On a connection request, the SSH client checks this directory for the appropriate server's host key and proceeds with authentication if the key is found.

- If the file is not found in either the system-wide or account-specific [.HOSTKEYS] directory, the first time you attempt to connect from your client to a remote SSH server, you are prompted to accept a copy of the server's public host key:

```
Host key not found from the list of known hosts.
```

```
Are you sure you want to continue connecting (yes/no)?
```

If you respond YES, the SSH client automatically creates the subdirectory SYS\$LOGIN:[SSH2.HOSTKEYS] if it does not exist, and copies the server's public key into this directory.

Key Naming Conventions for the Server's Public Host Key

The server's public and host private key pair files by default are `HOSTKEY` and `HOSTKEY.PUB`. When you copy these files manually, you must rename them following the proper naming conventions. (When SSH copies the files, the proper file name is assigned automatically.) The name of the remote SSH server's public key on the client host must be:

```
KEY_port_hostname.PUB
```

The *port* is typically 22. The *hostname* is the name of the remote SSH server. For example, when you copy the public key from the remote SSH server, MYSERVER, to the client host, the key name becomes: `KEY_22_MYSERVER.PUB`. If the remote server's name uses dot notation in its name, as in `MYSERVER.MYLAB.COM`, SSH replaces the dots with underscores, as in `KEY_22_MYSERVER_MYLAB_COM.PUB`.

Note that the *hostname* corresponds to the form of the SSH server name to which the SSH client connects, with underscores replacing dots if a qualified host name is used. For example, connecting to a server as:

```
$ SSH USER@MYSERVER.MYLAB.COM
```

results in copying the remote SSH server's public key file `HOSTKEY.PUB` into a local directory as a file named `KEY_22_MYSERVER_MYLAB_COM.PUB`. Note that underscores replace the dots.

If you copy these files manually, be sure to name the key files using this format. For example, if the server name is `MYSERVER.MYLAB.COM`, copy its `HOSTKEY.PUB` file to `KEY_22_MYSERVER_MYLAB_COM.PUB` in the appropriate directory.

Customizing the User Environment on the Server Host

During configuration, the `SSHD2_CONFIG` file is copied to `TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]`. When the connection attempt is made from a remote client, the SSH server reads the file and creates the run-time version of the configuration parameters. If the user wants a different set of parameters (user-specific), then he or she needs to create his or her own version of the configuration file in the user's login directory, `SYS$LOGIN:[SSH2]SSHD2_CONFIG`.

The SSH server will load this file and will modify the run-time version of the parameters accordingly. You can copy this file from UNIX or OpenVMS and edit it or create a new file. The file can be either `STREAM_LF` or variable length format.

Authentication Methods

Before you can connect to a remote SSH server, you must choose one or more of the three supported authentication methods introduced in Chapter 1. You configure the SSH client by specifying the authentication methods you choose. The SSH server, which the system manager configures, determines which authentication methods it will use before a connection is possible. Therefore, each of these methods requires configuration on both the SSH client and server.

After the SSH client makes a connection request to a remote SSH server, the server sends the client its permitted authentication methods. The SSH server may require the client to pass multiple authentication tests, like password and host-based authentication, before being connected.

The SSH authentication methods:

1. **Password authentication.** This method requires that the user supply a password to the client, which the client transmits encrypted to the server over the network. Then the SSH server performs authorization, verifying the supplied password using the OpenVMS native password authentication mechanism.
2. **Host-based authentication.** This method allows you to avoid specifying any secret information on the SSH client end. This method trusts the relationships between hosts rather than requiring you to prove your identity.

The SSH server host authenticates by verifying:

- The identity of the client host, using the client's host public key file, which the system manager maintains in the **known hosts database**. The directory `[TCPIP$SSH.SSH2.KNOWNHOSTS]` contains public keys for all client hosts that use the host-based authentication method to connect to the server.
 - The client host belongs to the **trusted hosts** list, which the system manager maintains on the server. This list of trusted hosts enables you to log in on the server without proving your identity.
 - Optionally, you can restrict users to only certain usernames on the client host.
3. If any of the checks fail, the connection is refused. An advantage of this method is that it does not require the client to type any password or passphrases, or to generate, distribute, and maintain keys. This is convenient for batch processing. One disadvantage, however, is a reliance on the identification of the host.

This method requires that the server manager maintain two pieces of information:

- a. The knownhost database, which is a list of public key files of remote hosts
 - b. A trusted hosts file, where the trusted hosts (and optionally, the user names) are listed.
4. **Public-key authentication.** This authentication method uses public key cryptography to verify the client's identity and requires two pieces of data: your private-public key pair, and optionally, a passphrase to encode this key for saving it in a file. This method is flexible because it allows additional control over authorization by providing multiple keys and applying restrictions to each key.

This method requires management actions on both ends of an SSH connection: the user on the client host and the system management on the server host, must create and maintain keys on the client, copy public keys from the client to the server hosts, and remember passphrases.

Customizing an Authentication Method

The type of authentication that the SSH client uses is specified by assigning values to the `AllowedAuthentications` keyword in the client configuration file `SSH2_CONFIG`. The user authentication methods are tried in the order in which they are listed for the `AllowedAuthentications` keyword. For example, if `hostbased` is listed first, the SSH server will try `hostbased` authentication before attempting the next authentication method on the list in the SSH client configuration file.

On the SSH server host, it does not matter in which order the authentication methods are listed. The order is the defining factor. For example, in a case in which client lists

```
AllowedAuthentications hostbased,publickey,password.
```

The server tries each of the methods (`hostbased,publickey,password`) and uses the first successful authentication.

If the `AllowedAuthentications` keyword is missing or has no entries, the server tries the public-key authentication method, and then the password authentication method. In this case, host-based authentication is not tried.

Customizing Password Authentication

Password authentication requires only that you set parameters in the SSH client and server hosts' configuration files. No additional files are required. For more information on the configuration files, refer to Chapter 2, *Configuring the Secure Shell Server* and Appendix B, *SSH Client and Server Configuration Files*.

Customizing Password Authentication on the Client

Set the value of the `AllowedAuthentications` keyword to include the word `password` (or omit the line). For example:

```
AllowedAuthentications password
```

Customizing Password Authentication the Server

Set the following:

1. The value of the `AllowedAuthentications` keyword must contain the word `password` (or omit the line). For example

```
AllowedAuthentications password
```

Customizing an Authentication Method

2. You can define the number of password attempts allowed by assigning a numeric value to the `PasswordGuesses` keyword in this configuration file. For example:

```
PasswordGuesses 4
```

You are allowed three password attempts by default.

Customizing Host-Based Authentication

Host-based authentication requires configuration actions on both client and server hosts.

Customizing Host-Based Authentication on the Client

Set the value of the following keywords:

- `AllowedAuthentications` to include word `hostbased`. For example:

```
AllowedAuthentications hostbased
```
- `DefaultDomain` to be the fully qualified domain name for the local host. For example, if the fully qualified domain name for the local host is `color.art.com`, enter:

```
DefaultDomain color.art.com
```

Customizing Host-Based Authentication on the Server

1. Edit the server configuration file as follows:

- Set the value of `AllowedAuthentications` to include the word `hostbased`. For example:

```
AllowedAuthentications hostbased
```
- To enable use of the user-specific `SHOSTS.` files, you must set the value of the `IgnoreRhosts` keyword to `no`. For example:

```
IgnoreRhosts no
```

This can also be left commented out, as in `#IgnoreRhosts no`. Note that this parameter applies to both `RHOSTS.` and `SHOSTS.` files.

2. Edit the system-wide trusted hosts files, `TCPIP$SSH_DEVICE: [TCPIP$SSH]SHOSTS.EQUIV`, to add the fully-qualified name of every SSH client host that will communicate with the server. You can also list a specific user name to limit access to that user. For example,

```
MYHOST.MYLAB.COM
```

or

```
MYHOST.MYLAB.COM smith
```

If the `IgnoreRhosts` parameter is set to `NO` as in Step 1, you can also add client host and optional usernames to the file `SYS$LOGIN:SHOSTS.` for a specific user.

If usernames are used, those associated with OpenVMS client hosts must be in lowercase; those associated with UNIX client hosts must match the account name case as it exists on the UNIX host.

3. In host-based authentication, the client and server hosts authenticate each other. Therefore, the server host must have the client's host public key. Copy the client's host public key file `TCPIP$SSH_DEVICE: [TCPIP$SSH.SSH2]HOSTKEY.PUB`, to the directory `TCPIP$SSH_DEVICE: [TCPIP$SSH.SSH2.KNOWNHOSTS]`, naming the key file name, using the format *fullyqualifiedhostname_ssh-dss.pub*. For example, if the host name is `green` and its domain name is `color.art.com`, copy it as `green_color_art_com_ssh-dss.pub`

For example:

```
$ COPY SYS$LOGIN:[SSH2.KNOWNHOSTS]green_color_art_com_ssh-dss.pub  
TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.KNOWNHOSTS]green_color_art_com_ssh-dss.pub
```

4. If you want your own version of the host public key files (in addition to the system-wide file specified in Step 4), create a list of key files, using the same naming rules specified in Step 4, in your `SYS$LOGIN:[SSH2.KNOWNHOSTS]` directory. In the event that a file with the same name exists in both directories, the SSH server uses the user-specific key name file.

Customizing Public-Key Authentication

Public key authentication requires configuration actions on both sides of the connection: client and server hosts.

1. Create public/private key pairs on the client host. For more information, refer to the next section, *“Customizing Public-key Authentication on the Client.”*
2. Install your public key in your accounts on all server hosts to which you will want to connect. Your user’s account on each server host might have many public keys for accessing it in different ways.

To install keys, you need two files:

1. On the client host, you need the file `SYS$LOGIN:[SSH2]IDENTIFICATION`, which identifies your private key file.
2. On the server host, you need the file `SYS$LOGIN:[SSH2]AUTHORIZATION`, which contains information about all public keys (the names of the corresponding files) that can be used by remote clients to identify themselves to the server.

Both are ASCII text files that contain keywords and assigned values. They are parsed by the client and the server, allowing, each server host to identify the public key of the user who is connecting to the server. The following two sections describe the steps.

Customizing Public-key Authentication on the Client

1. Edit the client configuration file by setting the value of the `AllowedAuthentications` keyword to include word `publickey`. For example:

```
AllowedAuthentications publickey, password
```

2. Create the subdirectory `SYS$LOGIN:[SSH2]` (if one does not exist).
3. From the user’s account, run the `SSH_KEYGEN` utility, as described in Chapter 5, which creates public-private key files:

- `SYS$LOGIN:[SSH2]ID_DSA_1024_A` - contains your private key, which you must protect so that only you can access it. To protect the file, use the `SET/FILE/PROTECTION DCL` command:

```
$ SET FILE/PROTECTION=(S,W,G,O:RW) ID_DSA_1024_A
```
- `SYS$LOGIN_DEVICE[JONES.SSH2]ID_DSA_1024_A.PUB` - contains your public key, which you can copy to other hosts; ensure that this file is available for world read access.

These file names are generated if you do not specify a file name with the `SSH_KEYGEN` utility. This default name includes identification of the default key generation algorithm, which is usually DSA (Digital Signature Algorithm) or RSA (Rivest, Shamir, and Adleman). Note that DSA is the default on OpenVMS; RSA is an option.

The `SSH_KEYGEN` command allows specification of a custom key name (and if desired, suppressing the passphrase with the `-P` flag). Also, the user can rename these files for convenience. For example, if multiple keys are necessary, the public key to be used with a particular SSH server host can be renamed

Customizing an Authentication Method

into file names in the format: *username-serverhostname.pub* (public key) and *username-serverhostname*. (private key). Using this convention will make it easier to copy designated public key files to the appropriate server hosts.

The examples below assume that the public and private key files have been generated as or renamed to files `MEUSER-MYHOST_MYDOMAIN_COM.*`.

4. Create a file `SYS$LOGIN: [SSH2] IDENTIFICATION`. and add a line that identifies the name of your private key. For example, if this key file name were used:

```
MEUSER-MYHOST_MYDOMAIN_COM.
```

This line would be added:

```
IdKey MEUSER-MYHOST_MYDOMAIN_COM
```

Customizing Public-key Authentication on the Server

1. Set the value of the `AllowedAuthentications` keyword in the server configuration file to include the word `publickey`. For example:

```
AllowedAuthentications publickey
```

2. Create the subdirectory `SYS$LOGIN: [SSH2]` (if one does not exist).
3. Create the file `SYS$LOGIN: [SSH2] AUTHORIZATION`.
4. Add entries in the `SYS$LOGIN: [SSH2] AUTHORIZATION` file as necessary. Each entry is a single line that identifies the user's client public key file name. The format of the entry is:

```
KEY username-hostname.PUB
```

For example, if the user's public key file name is:

```
MEUSER-MYHOST_MYDOMAIN_COM
```

Add this line to the `AUTHORIZATION` file:

```
KEY MEUSER-MYHOST_MYDOMAIN_COM.PUB
```

5. To authenticate you as the user, the server host must access the public key file, which you created on the client host. For more information, refer to the section, "*Customizing Public-key Authentication on the Client.*" Copy this file to your account directory on the server host: `SYS$LOGIN:[SSH2]` as *username-hostname.PUB*. Note that the name must be the same as the one listed on the line with the keyword "KEY."

4 Managing the SSH Service

This chapter describes:

- Starting and stopping the SSH server
- Starting and stopping the SSH client
- Logical names

Starting and Stopping the SSH Server

To start the SSH server, enter:

```
$ @SYS$STARTUP:TCPIP$SSH_STARTUP.COM
```

To stop the server, enter:

```
$ @SYS$STARTUP:TCPIP$SSH_SHUTDOWN.COM
```

If the `Enable service` option is chosen from the menu during SSH service configuration, as shown in the SSH Configuration menu, the SSH server will restart when the TCP/IP service is restarted.

```
SSH Configuration
Service is not defined in the SYSUAF.
Service is defined in the TCPIP$SERVICE database.
Service is not enabled.
Service is stopped.
SSH configuration options:
 1 - Enable service on this node
[E] - Exit SSH configuration
```

Enter configuration option: 1

You can also start and stop the SSH server from `TCPIP$CONFIG`.

Starting and Stopping the SSH Client

To start the SSH client, enter:

```
$ @SYS$STARTUP:TCPIP$SSH_CLIENT_STARTUP.COM
```

To stop the SSH client, enter:

```
$ @SYS$STARTUP:TCPIP$SSH_CLIENT_SHUTDOWN.COM
```

You can also start and stop the SSH client from `TCPIP$CONFIG`.

SSH Logical Names

The logical name described in Table 4–1 can be used to modify the behavior of the SSH service:

Table 4-1

Name	Function
TCPIP\$SSH_DEVICE	Defines the SSH device on which client default directory is located. If you do not define this logical, the default is SYS\$SYSDEVICE.

5 SSH Command Reference

This chapter describes SSH commands that you can use to invoke SSH, copy or transfer files, and manage keys.

Before You Begin

To use SSH client utilities at the DCL prompt, execute the command procedure

```
$ @SYS$MANAGER:TCPIP$DEFINE_COMMANDS.COM . For example:
```

```
$ @sys$manager:TCPIP$DEFINE commands.
```

Copying Files

You can use the following Secure Shell commands to copy files between clients and servers:

- SCP (or SCP2)
- SFTP (or SFTP2)

Using the SCP Command

The SCP command securely copies files between a Secure Shell client and server. This command is intended as a secure replacement for the `rcp` command. When the user enters the SCP command, the client establishes an SSH session. If authentication succeeds and the user's identity has been accepted by the server, the server executes the command. All communication will be automatically encrypted. The session terminates when the command completes. The SCP command does not require special privileges.

Example 5-1 SCP

The following example shows how to copy files from a local system (*localfile*) to a remote system (*destination*):

```
$ SCP localfile user@remotehost:[destination]
```

The following example shows how to copy files from a remote system (*directory*) to a local system (*destination*):

```
$ SCP user@remotehost:[directory]remotefile [destination]
```

Command Synopsis

NOTE When specifying uppercase options, enclose them in quotation marks, for example, “-D”, -“D”, or “D”.

```
SCP [-D debug_level_spec] [-d] [-q] [-Q] [-p] [-u] [-v] [-h] [-r] [-k] [-B] [-a] [-v] [-l]
[-c cipher] [-S ssh-path] [-P ssh-port] [-o ssh_option] [-b] [-N] [-V] [-h]
[[user@]host[#port]:]filename or directoryname...[[user@]host[#port]:]filename or directoryname
```

Parameters

Source: *filename(s) and / or directory name(s)*

Destination: *filename or directory name*

The general format for the source and destination name is `[[user@]host[#port]:]filename or directoryname`

Options

Table 5-1 SCP Command Options

Option	Description
-D <i>debug_level_spec</i>	Displays debug information to SYS\$OUTPUT. The <i>debug_level_spec</i> argument is a number between 0 and 99, where 99 specifies that all debug information should be displayed.
-d	Makes sure that the destination file is a directory. If not, the SCP command exits with an error message.
-q	Makes SCP quiet (only fatal errors are displayed).
-Q	Suppresses progress indicator.
-p	Preserves file attributes and timestamps.
-u	Removes source files after copying.
-k	Removes destination files after copying.
-B	Sets batch-mode on.
-r	Recurse subdirectories.
-a	Transfers files in ASCII mode.
-v	Displays information in verbose mode. This is equal to specifying the -D 2 option.
-1	Engages SCP1 compatibility.
-c cipher	Specifies the encryption algorithm to use. See the <i>ciphers</i> keyword in the SSH2_CONFIG. for more information. Multiple -c options are allowed; a single -c option can specify only one cipher.
-S <i>ssh-path</i>	Specifies location of a non-standard SSH server executable.
-P <i>ssh-port</i>	Tells SCP which port SSHD2 listens on at the server host.
-o	Specifies additional options for SSH2.
-b	Defines maximum buffer size for one request (default 2048 bytes).
-N	Defines maximum buffer size of concurrent requests (default 10).
-V	Displays version of SSH.
-h	Displays help file.

Using the SFTP command

You can use the SFTP command on a client to copy files to and from a server. Some SFTP commands and syntax are similar to those for the FTP command, but SFTP does not use the FTP server or the FTP client for its connections. The SFTP command runs with normal user privileges.

Example 5-2 SFTP

The following example shows how to invoke SFTP:

```
$ SFTP
or
$ SFTP user@host
```

NOTE When specifying uppercase options, enclose them in quotation marks, for example, “-D” , -“D”, or “D”.

Command Synopsis

```
SFTP [-D ] [-b batchfile] [-S path][-h][-v] [-P ssh-port]host
```

At the `sftp>` prompt, use “help” or “help <topic> for more details about SFTP commands. For example, to find more information about the open command, type:

```
sftp> help open
```

Options

Table 5-2 SFTP Command Options

Option	Description
-D <i>debug_level_spec</i>	Displays debug information to SYS\$OUTPUT. The <i>debug_level</i> argument is a number between 0 and 99, where 99 specifies that all debug information should be displayed.
-b <i>batchfile</i>	Reads commands from a file instead of SYS\$INPUT.
-S <i>ssh-path</i>	Specifies location of a non-standard SSH server executable.
-h	Displays help file.
-V	Displays version of SSH.
-P	Tells SFTP which port SSHD2 listens on at the server host.

Login and Remote Command Execution with the SSH

The SSH command creates a secure network connection for remote login and command execution. This command is intended as a secure replacement for the RLOGIN and RSH commands. When the user enters the SSH command, the SSH client establishes a session with the server and proves the user's identity to the server using a chosen authentication method as described in Chapter 3. When the user's identity has been accepted by the SSH server, all communication with the remote SSH server will be automatically encrypted.

You can use the SSH command on the client to securely log in and execute remote commands on an SSH server.

The syntax for the SSH command is:

Login and Remote Command Execution with the SSH

```
$ SSH[options] server_name [command]
```

NOTE When specifying uppercase options, enclose them in quotation marks, for example, “-D” , -“D”, or “D”.

Command Synopsis

```
$ SSH [-l login_name] hostname [options]
```

```
$ SSH [-l login_name] [-i file] [-F file] [-v][-d debug_level_spec] [-V] [-q] [-e char] [-c cipher] [-m MAC] [-p port] [+C] [-C] [-h] [username@]host[#port] [command]
```

Table 5-3 SSH Command Options

Options	Description
-l <i>login_name</i>	Specifies the user for login to the remote system (same as <i>login_name@host</i>)
-i <i>file</i>	Specifies the identity file for public key authentication. This option takes file name as a parameter. It is assumed that file resides in the user’s [.SSH2] directory. This option can also be specified in the configuration file.
-F <i>file</i>	Specifies an alternative client host configuration file instead of the default one. The specified file name must include the directory where the file resides. For example [.SSH2]MY_SSH2_CONFIG. Information from this file will supersede information from TCP/IP\$SSH_DEVICE:[TCP/IP\$SSH]SSH2_CONFIG. and the user’s [.SSH2]SSH2_CONFIG.
-v	Enables verbose mode. Displays verbose debugging messages. Equal to the -d2 option. This option can also be specified in the client’s configuration file.
-d <i>debug_level_spec</i>	Displays debug information. The <i>debug_level_spec</i> argument is a number from 0 to 99, where 99 specifies that all debug information should be displayed or a comma-separated list of assignments.
-q	Disables warning messages. This option can also be specified in the client’s configuration file.
-V	Displays the version of SSH.
-c <i>cipher</i>	Specifies the encryption algorithm to use. See the ciphers keyword in the SSH2_CONFIG. configuration file for more information. A single -c option can specify only one cipher. Multiple -c options are allowed.
-m <i>MAC</i>	Specifies the MAC (Message Authentication Code) algorithm. See the MACs keyword in the SSH2_CONFIG. configuration file for more information. A single -m option can have only one MAC. Multiple -m options are allowed.
-p <i>port</i>	Specifies the port to connect to on the remote system. This option can also be specified in the Client’s configuration file.

Table 5-3 SSH Command Options (Continued)

Options	Description
+C	Enables compression.
-C	Disables compression. (default)
-o <i>option</i>	Specifies an option in the format used in the SSH2_CONFIG configuration file. This is useful for specifying an option for which there is no command-line option. Comment lines are not accepted with this option.
-L [<i>protocol</i> /] <i>port:host:hostport</i>	Specifies that the given port on the local (client) system is to be forwarded to the specified host and port on the remote system. This allocates a socket to listen to the port on the local system. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the <i>host:hostport</i> argument from the remote system. Only root can forward privileged ports. The argument protocol enables the protocol specific forwarding. The protocols implemented are TCP (default, no specific processing) and FTP. Temporary forwardings are created for FTP data channel, effectively securing the whole FTP session. This option can also be specified in the SSH2_CONFIG configuration file. The ftp data channel forwarding works in passive mode. Set Passive On for ftp data channel connections
-R [<i>protocol</i> /] <i>port:host:hostport</i>	Specifies that the given port on the remote (server) system is to be forwarded to the specified host and port on the local system. This allocates a socket to listen to the port on the remote system. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the <i>host:hostport</i> argument from the local system. Only root can forward privileged ports on the remote system. The argument protocol enables the protocol specific forwarding. The protocols implemented are TCP (default, no specific processing) and FTP. Temporary forwardings are created for FTP data channel, effectively securing the whole FTP session. This option can also be specified in the SSH2_CONFIG configuration file. The ftp data channel forwarding works in passive mode. Set Passive On for ftp data channel connections.
-h	Displays help on SSH command options.

To execute remote commands:

```
$ SSH [options] server_name [command]
```

When a user successfully logs in, the SSH server process:

- Runs with the user's privileges.
- Sets up a user environment.
- Sets default to the user's home directory.
- Executes the requested command.

SSH Key Management

SSH key management allows the user to generate public/private key pairs and to manipulate keys when a large number of keys are used.

Key Generation

SSH_KEYGEN is the authentication key pair generation utility.

NOTE When specifying uppercase options, enclose them in quotation marks, for example, “-D” , -“D”, or “D”.

Command Synopsis

```
SSH_KEYGEN [-b bits] [-t key_algorithm] [-c comment_string] [-e file] [-p passphrase] [-P ][ -? ][-h ]  
[-q] [-l file] [-i file] [-D file] [-B number] [-V] [-r file] [-F file] [key1 key2..]
```

Description

SSH_KEYGEN generates and manages authentication keys for SSH. Users who need to use SSH with public-key authentication can run this utility to create authentication keys. The system administrator can also use this utility to generate host keys.

Options**Table 5-4 SSH_KEYGEN Command Options**

Options	Description
-b <i>bits</i>	Number of key in bits (default 2048).
-t <i>key-algorithm</i>	The algorithm used in key generation. DSA (Digital Signature Algorithm) or RSA.
-c <i>comment-string</i>	Specifies the key's comment string.
-e <i>file</i>	Edits the comment/passphrase of the key.
-p <i>passphrase</i>	Specifies the passphrase used to protect the key.
-P	Specifies that the key will be saved with an empty passphrase.
-h -?	Displays a short summary of SSH_KEYGEN options.
-q	Hides the progress indicator.
-D <i>file</i>	Derives the public key from the private key file.
-l	Converts a SSH1.x key.
-i <i>file</i>	Loads and displays information on a file.
-B <i>number</i>	The number base for displaying key information (default 10).
-V	Displays version string and exit.
-r <i>file</i>	Randomizes data from a file to a random pool.
-F <i>file</i>	Dumps the fingerprint of a file.

Key Manipulation

The SSH_ADD utility adds identities for the authentication agent.

NOTE When specifying uppercase options, enclose them in quotation marks, for example, “-D”, “-D” or “D”.

Command Synopsis

```
SSH_ADD [-p] [-l] [-N] [-P] [-I] [-d] [-D] [-L] [-U] [-1] [-u] [-f forwarding steps] [-F forwarding constraint] [-t key timeout in minutes] [-R OpenPGP key ring] [files...]
```

Description

SSH_ADD adds identities to the authentication agent, SSH_AGENT. If a file requires a passphrase, SSH-ADD asks the user for the passphrase. If the -p option is given, the passphrase is read from SYS\$INPUT. The authentication agent must be running and must be an parent of the current process for SSH_ADD to work.

SSH Key Management

The authentication agent initially does not have any private keys. Users start the authentication agent, then enter the `SSH_ADD` command to load the private keys into the authentication agent. You must know the passphrase for each key that you want to load. Passphrases never go over the network.

Options**Table 5-5 SSH_ADD Command Options**

Options	Description
-p	Reads passphrase from SYS\$INPUT
-l	Lists all identities currently represented by the agent.
-d	Removes the identity from the agent.
-D	Deletes all identities from the agent.
-L	Temporarily locks the agent with a password.
-U	Unlocks the locked agent. The password given when the agent was locked must be used to unlock.
-t	Agent must delete the key after timeout. Timeout is given in minutes.

Return Status

`SSH_ADD` returns one of the following exit codes. These may be useful in scripts.

`TCPIP$SSH_ADD2_EXIT_NOAGENT` -- No connection could be made to the authentication agent. Presumably there is no authentication agent active in the execution environment of `ssh-add`.

`TCPIP$SSH_ADD2_EXIT_BADPASS` -- The user did not supply a required passphrase.

`TCPIP$SSH_ADD2_EXIT_NOFILE` -- An identify file could not be found, is unreadable, or contains errors.

`TCPIP$SSH_ADD2_EXIT_NOIDENTITY` -- The agent does not have the requested identity.

`TCPIP$SSH_ADD2_EXIT_ERROR` -- An unspecified error has occurred.

Key Agent

The `SSH_AGENT` utility starts the authentication agent, which holds private keys in memory.

SYNOPSIS

```
SSH_AGENT [-c] [-s] [ -1] [command [args...]]
```

DESCRIPTION

The `SSH_AGENT` utility starts the authentication agent on a user's client system. The authentication agent holds the user's private keys in memory. Secure Shell clients automatically contact the authentication agent for all key-related operations. This allows users to access all their remote accounts that contain their public key file without having to enter their passphrase.

The authentication agent initially does not have any private keys. Users start the authentication agent, then use the `SSH_ADD` utility to load the private keys into the authentication agent. You must know the passphrase for each key that you want to load. Passphrases never go over the network.

Users start the authentication agent in the beginning of a login session. The utility normally starts the X server or is the user shell. All other windows or programs are started as children of the authentication agent process and inherit a connection to the agent.

The connection to the authentication agent is forwarded over SSH remote logins, and the user can use the privileges given by the identities anywhere in the network in a secure way. The authentication agent is automatically used for public key authentication when the user logs in to other machines using SSH.

A SSH Client and Server Files

This appendix summarizes information about files and directories that the SSH client and server use. Text files can use either `STREAM_LF` or variable length record format.

Client side

TCPIP\$SSH_DEVICE: [TCPIP\$SSH]

Purpose: Default directory of the TCPIP\$SSH account

Created: During SSH client configuration

Scope: System-wide

Used: By running instances of the client processes

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2]

Purpose: Contains multiple SSH files and subdirectories

Created: During SSH client configuration

Scope: System-wide

Used: By running instances of the client processes

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2] SSH2_CONFIG.

Purpose: client configuration file.

Created: During SSH client configuration, by extracting a template file from the TCP/IP kit. The file edited by the system manager as necessary.

Scope: System-wide

Used: Read by a starting client process

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2.HOSTKEYS]

Purpose: Contains public host keys of all remote servers that user will connect to using SSH.

Created: Empty during SSH client configuration; files are copied to this directory from all servers by the system manager before initiating connections. (Note that this is not required. If the server's public host key file is not in TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2.HOSTKEYS] when the connection initiated, it is automatically copied to the user-specific directory SYS\$LOGIN:[SSH2.HOSTKEYS] on the client).

Scope: System-wide

Used: For host authentication purposes; files in this directory are searched by the client for the server's key before the user-specific directory.

NOTE An important component of the SSH run-time environment is the [.SSH2] subdirectory, created by the user or automatically by the SSH software, on the user's login directory (as specified by SYS\$LOGIN).

SSH uses this subdirectory to store multiple files necessary for SSH to function. For example, if SYS\$LOGIN is translated into DKAO:[username], then this special subdirectory will be DKAO:[username.SSH2]. Through this manual, we refer to this directory as the "user's SSH directory."

SYS\$LOGIN: [SSH2]

Purpose: Contains multiple SSH files and subdirectories

Created: Either by the user manually, or automatically by running the client

Scope: User-specific

Used: By running the client.

`SYS$LOGIN: [SSH2] SSH2_CONFIG.`

Purpose: client configuration file.

Created: By the user, if necessary

Scope: User-specific

Used: If it exists, used by a starting client process instead of the system-wide configuration file.

`SYS$LOGIN: [SSH2] IDENTIFICATION.`

Purpose: Contains the identification of a user.

Created: By the user when using public-key authentication

Scope: User-specific

Used: To identify a user when using public-key authentication.

`SYS$LOGIN: [SSH2].HOSTKEYS]`

Purpose: Contains the public keys of the server hosts to which client will be connecting.

Created: By the user, if necessary. Files are copied here from a server, either automatically when a connection is requested, or manually before initiating a connection.

Scope: User-specific

Used: For host authentication purposes. First, the client will try to locate the remote host key in this directory. If it is not found, the system-wide directory (described above) is used.

Server side

`TCPIP$SSH_DEVICE: [TCPIP$SSH]`

Purpose: Default directory of TCPIP\$SSH account

Created: During SSH server configuration

Scope: System-wide

Used: By running instances of the server processes.

`TCPIP$SSH_DEVICE: [TCPIP$SSH.SSH2]`

Purpose: Contains multiple SSH files and subdirectories.

Created: During SSH server configuration

Scope: System-wide

Used: By running instances of the server processes.

`TCPIP$SSH_DEVICE: [TCPIP$SSH.SSH2] SSHD2_CONFIG.`

Purpose: server configuration file.

Server side

Created: During SSH server configuration by extracting a template file from the TCP/IP kit. The file edited by the system manager as necessary.

Scope: System-wide

Used: Read by a starting server process; also read by client for host-based authentication.

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2]HOSTKEY.

Purpose: Contains the private part of the host key pair. This file is owned by the system account, readable only by the system, and is not accessible to others.

Created: Together with the public part of the host key pair during SSH server configuration (if requested). The new key can be created any time by a system manager running the key generation utility, SSH_KEYGEN, which creates both keys.

Scope: System-wide

Used: By the server when connection from a client is requested.

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2]HOSTKEY.PUB

Purpose: Contains the public part of the host key. This file is writable by the system account only and readable by world.

Created: Together with the private part of the host key during SSH server configuration (if requested). The new key can be created any time by a system manager running key generation utility, SSH_KEYGEN, which creates both keys).

Scope: System-wide

Used: The server copies this file to a client when a connection is requested by a client.

TCPIP\$SSH_DEVICE: [TCPIP\$SSH]SHOSTS.EQUIV

Purpose: List of trusted hosts.

Created: An empty file is created during SSH server configuration. It is populated by a system manager.

Scope: System-wide

Used: As a system-wide list of trusted hosts checked by a server for host-based authentication.

TCPIP\$SSH_DEVICE: [TCPIP\$SSH.SSH2.KNOWNHOSTS]

Purpose: System-wide directory that contains public keys of all remote client hosts that might make an attempt to connect to the server using “host-based” authentication.

Created: An empty file is created during SSH server configuration. It is populated by the system manager as necessary by copying files from client hosts.

Scope: System-wide

Used: The server gets public keys of remote client hosts from this directory when it is processing a request for host-based authentication connection.

NOTE

An important component of the SSH run-time environment is the [SSH2] subdirectory, created by the user or automatically by the SSH software, on the user's login directory (as specified by SYS\$LOGIN).

SSH uses this subdirectory to store multiple files necessary for SSH to function. For example, if SYS\$LOGIN is translated into DKA0:[username], then

this special subdirectory will be `DKAO:[username.SSH2]`. Through this manual, we refer to this directory as the “user’s SSH directory.”

`SYS$LOGIN:SHOSTS.`

Purpose: List of trusted hosts

Created: By the user if necessary

Scope: User-specific

Used: As a user-specific list of trusted hosts checked by the server for host-based authentication. This list is checked after the system-wide `SHOSTS.EQUIV`, allowing the user to add hosts to the system-wide list.

`SYS$LOGIN:[SSH2]`

Purpose: Contains multiple SSH files and subdirectories

Created: By the user if necessary

Scope: User-specific

Used: By running the server

`SYS$LOGIN:[SSH2.KNOWNHOSTS]`

Purpose: This user-specific directory that contains public keys of all remote client hosts that might make an attempt to connect to the server using host-based authentication.

Created: By the user when necessary. It is populated by the user by copying files from client hosts.

Scope: User-specific

Used: The server gets public keys of remote client hosts from this directory when it is processing a request for host-based authentication connection. The file from this directory is used if another file with the same name exists in the system-wide directory above.

`SYS$SYSLOGIN:[SSH2]AUTHORIZATION`

Purpose: Contains information that allows the server to identify the user for public key-based authentication.

Created: By the user when necessary. The user populates this file by copying files from the client hosts.

Scope: User-specific

Used: The server uses the information in this file to identify the user.

Server side

B SSH Client and Server Configuration Files

This appendix lists the system-wide SSH client and server files that the TCPIP\$CONFIG configuration procedure creates during SSH configuration, as described in Chapter 2.

Client Configuration File

```
#
# File name:      SSH2_CONFIG.
# Product:       HP TCP/IP Services for OpenVMS
# Version:       V5.4
#
# © Copyright 1976, 2003 Hewlett-Packard Development Company, L.P. #

#
# ssh2 client configuration information
#
# Note: "*" is used for all hosts, but you can use other hosts as well. #

*:

#
# HP Tru64 UNIX specific
# Secure the r* utilities (no, yes)
#
#   EnforceSecureRutilsno

## General

    AuthenticationSuccessMsgyes
#   BatchModeyes
#   Compressionyes
#   DontReadStdinno
#   EscapeChar~
#   ForcePTYAllocationyes
#   GoBackgroundyes
#   PasswordPrompt"%U%H's password: "
    PasswordPrompt"%U's password: "
#   QuietModeyes
    VerboseModeno

## Network
```

```
Port22
NoDelayno
KeepAliveyes
# SocksServersocks://mylogin@socks.ssh.com:1080/203.123.0.0/16,198.74.23.0/24

## Crypto

CiphersAnyStdCipher
MACsAnyMAC
# RekeyIntervalSeconds3600
StrictHostKeyCheckingno

## User public key authentication

AuthorizationFileauthorization
IdentityFileidentification
RandomSeedFilerandom_seed

## Tunneling

# ForwardAgentyes
# ForwardX11yes
# GatewayPortsyes

# Tunnels that are set up upon logging in

# LocalForward"110:pop3.ssh.com:110"
# RemoteForward"3000:foobar:22"

## SSH1 Compatibility

Ssh1AgentCompatibilitynone
# Ssh1AgentCompatibilitytraditional
# Ssh1AgentCompatibilityssh2
```

Server Configuration File

```
Ssh1Compatibilityyes
# Ssh1Path/usr/local/bin/ssh1

## Authentication
## Hostbased is not enabled by default.

# AllowedAuthenticationshostbased, publickey, password

## Authentication, OpenVMS-specific

# NumberOfPasswordVerificationPrompts 3
# PubkeyPassphraseGuesses 3

# For ssh-signer2 (only effective if set in the global configuration # file, usually
TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]SSH2_CONFIG, i.e., this file.

# DefaultDomainfoobar.com
SshSignerPath/sys$system/tcpip$ssh_ssh-signer2

## Examples of per host configurations

#alpha*:
# Hostalpha.oof.fi
# Useruser
# PasswordPrompt"%U:s password at %H: "
# Ciphersidea

#foobar:
# Hostfoo.bar
# Userfoo_user
```

Server Configuration File

```
#
# File name: SSHD2_CONFIG.
```

```
# Product:      HP TCP/IP Services for OpenVMS
# Version:     V5.4
#
# © Copyright 1976, 2003 Hewlett-Packard Development Company, L.P. #

#
# ssh2 server configuration information
#

## General

    AllowCshrcSourcingWithSubsystemsno
    ForcePTYAllocationno
    SyslogFacilityAUTH
#   SyslogFacilityLOCAL7
#   QuietModeyes
    VerboseModeno

## Network

    Port22
    ListenAddress0.0.0.0
    RequireReverseMappingno
    MaxBroadcastsPerSecond0
#   MaxBroadcastsPerSecond1
#   NoDelayyes
#   KeepAliveyes
#   MaxConnections50
#   MaxConnections0
# 0 == number of connections not limited

## Crypto

    CiphersAnyCipher
#   CiphersAnyStd
#   CiphersAnyStdCipher
#   Ciphers3des
```

Server Configuration File

```
MACsAnyMAC
# MACsAnyStd
# MACsAnyStdMAC
# RekeyIntervalSeconds3600

## User

CheckMailyes
PrintMotdyes
# LoginGraceTime600
# PermitEmptyPasswordsno
# StrictModesyes
UserConfigDirectory"%Dssh2"
# UserConfigDirectory"/etc/ssh2/auth/%U"
UserKnownHostsyes

## User public key authentication

AllowAgentForwardingyes
AuthorizationFileauthorization
HostKeyFilehostkey
IdentityFileidentification
PublicHostKeyFilehostkey.pub
RandomSeedFilerandom_seed

## Tunneling

AllowTcpForwardingyes
# AllowTcpForwardingForGroupsprivileged_tcp_forwarders
# AllowTcpForwardingForUsersssj1, cowboyneal@slashdot.org
AllowX11Forwardingyes
# DenyTcpForwardingForGroupscoming_from_outside
# DenyTcpForwardingForUsers"2[:isdigit:]*4, peelo"

## Authentication
## Hostbased and PAM are not enabled by default.
```

```
# AllowedAuthenticationspublickey
# AllowedAuthenticationspublickey,pam-1@ssh.com
# AllowedAuthentications      hostbased,publickey,password
# BannerMessageFile          /etc/ssh2/ssh_banner_message
# BannerMessageFile/etc/issue.net
# PasswordGuesses3
# RequiredAuthenticationspublickey,password
# SshPAMClientPathssh-pam-client

## Host restrictions

# AllowHostslocalhost, *
# AllowSHoststrusted.host.org
# DenyHostsevil.org, aol.com
# DenySHostsnot.quite.trusted.org
# IgnoreRhostsno
# IgnoreRootRHostsno
# (the above, if not set, is defaulted to the value of IgnoreRHosts)

## User restrictions

# AllowGroupsstaff,users
# AllowUsers"sj*,s[:isdigit:]##,s(jl|amza)"
# DenyUsersskuuppa,warezdude,31373
# DenyUsersdon@untrusted.org
# DenyGroupsguest
# PermitRootLoginnopwd
# PermitRootLoginyes

## SSH1 compatibility

# Ssh1Compatibility
# Sshd1Path

## Chrooted environment
```

Server Configuration File

```
# ChRootGroupsguest
# ChRootUsersftp, guest

## subsystem definitions

    subsystem-sftp /sys$system/tcpip$ssh_sftp-server2

## OpenVMS auditing and access control

# AllowVmsLoginWithExpiredPw          no
# AllowNonvmsLoginWithExpiredPw      no
# UserLoginLimit                      -1
# AccountingAuthentications           pubkey,password,hostbased
# IntrusionAuthentications            pubkey,password,hostbased
# IntrusionIdentMethod                pubkey,password,hostbased
# IntrusionIdentSsh                   pubkey,password,hostbased
# LogfailAuthentications               pubkey,password,hostbased
# PubkeyPassphraseGuesses             3
```