PSCSSH (SSH for OpenVMS) Administration & User's Guide

July 2025

This manual provides the system manager with the procedures for installing, managing, and using the PSCSSH family of software products.

Operating System/Version: OpenVMS VAX V5.5-2 or later

OpenVMS Alpha V6.2 or later

OpenVMS Itanium V8.2 or later

OpenVMS X86_64 V9.2-3 or later

TCP/IP Services Version: V5.5 or higher

Software Version: PSCSSH 3.0

Process Software
Framingham, Massachusetts
USA

Preface

Introducing This Guide

This guide describes the PSCSSH software. It covers the following topics: software installation, server and client configuration, server startup and shutdown, using the SSH clients, utilities, and server monitoring and control.

What You Need to Know Beforehand

Before using PSCSSH, you should be familiar with:

- Computer networks in general
- OpenVMS operating system and file system
- HPE/VSI's OpenVMS TCP/IP software

How This Guide Is Organized

This guide has the following contents:

- Chapter 1, *Before You Begin*, explains what you need to prepare for an installation.
- Chapter 2, *Installing PSCSSH*, provides a step-by-step procedure for executing the software installation.
- Chapter 3, *Configuring PSCSSH*, explains how to configure PSCSSH.
- Chapter 4, *Configuring the Secure Shell (SSH) V1 Server,* describes how to configure and maintain the PSCSSH SSH V1 server.
- Chapter 5, *Configuring the Secure Shell (SSH) V2 Server*, describes how to configure and maintain the PSCSSH SSH V2 server.
- Chapter 6, *Accessing Remote Systems with the Secure Shell (SSH) Utilities*, explains how to configure and maintain the PSCSSH Secure Shell (SSH) client.
- Chapter 7, *Secure File Transfer*, describes using SCP, SFTP2, and FTP over SSH for transferring files in a secure manner.
- Chapter 8, *Monitoring and Controlling SSH*, describes the utilities used for monitoring and controlling the SSH server environment.

Online Help

You can use `HELP` at the DCL prompt to access SSH topical help:

```
$ HELP SSH [topic]
```

The topic entry is optional. You can also enter topics and subtopics at the following prompt and its subprompts:

```
SSH Subtopic?
```

Obtaining Customer Support

You can use the following customer support services for information and help about PSCSSH and other Process Software products if you subscribe to our Product Support Services. (If you bought SSH through an authorized reseller, contact your reseller for technical support.) Contact Technical Support directly using the following methods:

Electronic Mail

E-mail relays your question to us quickly and allows us to respond as soon as we have information for you. Send e-mail to `support@process.com`. Be sure to include your:

- Name
- Telephone number
- Company name
- Process Software product name and version number
- Operating system name and version number
- Process Software support contract number

Describe the problem in as much detail as possible. You should receive an immediate automated response telling you that your call was logged.

Telephone

If calling within the continental United States or Canada, call Process Software Technical Support toll-free at (800) 394-8700. If calling from outside the continental United States or Canada, dial +1 (508) 628-5074. Please be ready to provide your name, company name, Process Software support contract number, and telephone number.

Web

There is a variety of useful technical information available on our website, https://www.process.com/

Conventions Used

Convention	Meaning
host	Any computer system on the network. The local host is your computer. A remote host is any other computer.
`monospaced type`	System output or user input. User input is in **`reversed bold`** `type`. Example: `Is this configuration correct?` **`YES`** Monospaced type also indicates user input where the case of the entry should be preserved.
italic type	Variable value in commands and examples. For example, *username* indicates that you must substitute your actual username. Italic text also identifies documentation references.
[*directory*]	Directory name in an OpenVMS file specification. Include the brackets in the specification.
[*optional-text*]	(Italicized text and square brackets) Enclosed information is optional. Do not include the brackets when entering the information. Example: `START/IP` *`line address`* [*`info`*] This command indicates that the *`info`* parameter is optional.
{value | value}	Denotes that you should use only one of the given values. Do not include the braces or vertical bars when entering the value.
Note	Information that follows is particularly noteworthy.
Caution	Information that follows is critical in preventing a system interruption or security breach.
`key`	Press the specified key on your keyboard.
`Ctrl+key`	Press the control key and the other specified key simultaneously.
`Return`	Press the Return or Enter key on your keyboard.

1. Before You Begin

This chapter introduces you to and prepares you for SSH product installation, configuration, startup, and testing. It is for the OpenVMS system manager or technician responsible for product installation and configuration.

Steps to Get SSH Up and Running

To get SSH up and working, you must perform the following steps:

1. Load the license pack.
2. Install the software. See Chapter 2, *Installing PSCSSH.*
3. Configure the PSCSSH environment. See Chapter 3, *Configuring PSCSSH.*
4. Configure the PSCSSH SSH V1 server. See Chapter 4, *Configuring the Secure Shell (SSH) V1 Server.*
5. Configure the PSCSSH SSH V2 server. See Chapter 5, *Configuring the Secure Shell (SSH) V2 Server*.
6. Configure the PSCSSH client. See Chapter 6, *Accessing Remote Systems with the Secure Shell (SSH) Utilities.*

Prepare for Installation

PSCSSH installation involves using the VMSINSTAL procedure. Preparing for installation involves:

- Understanding the hardware and software requirements
- Determining if you have sufficient disk space and global pages for the installation
- Determining where to install the software

Hardware Requirements

PSCSSH has no special hardware requirements beyond those stated in the Software Product Description for OpenVMS's TCP/IP Services.

Software Requirements

PSCSSH supports OpenVMS VAX version 5.5-2 and later; OpenVMS Alpha version 6.2 and higher; OpenVMS Integrity 8.2 and later; and TCP/IP Services version 5.0 and later.

Disk Space and Global Pages

The destination device for your PSCSSH software must have enough disk space so that you can install and run the software. Your system must meet the following approximate parameters:

System	Number of Blocks Needed to Install	Number of Blocks Needed After Installation	Free GBLPAGES Needed to Run
VAX	150,000	about 42,000	35,000
Alpha	250,000	about 88,000	45,000
Itanium	280,000	about 135,000	45,000
X86_64	280,000	about 135,000	45,000

The runtime values for disk space are slightly higher once you configure and start PSCSSH.

> **Note:** Insufficient `GBLPAGES` can abort the installation and leave your system command tables disconnected. The only way to recover is through a system reboot.

General Requirements

Check at this point that you:

- Have OPER, SYSPRV, or BYPASS privileges
- Can log in to the system manager's account
- Are the only user logged in (recommended)
- Backed up your system disk on a known, good, current, full backup (recommended)
- Need to reinstall PSCSSH after performing a major VMS upgrade
- If PSCSSH is currently running, shut it down using the `PSCSSH SHUTDOWN` command. This is mandatory.
- Ensure TCP/IP Services is currently running.

Where to Install PSCSSH

Install PSCSSH in a location depending on the following:

- Generally, on your system disk, but you can install PSCSSH anywhere, just answer the question when it appears. This is also where you would keep your "common" files. Node-specific files should always be on your system disk.
- If the machine is in a single platform cluster, on a common disk.
- If the machine is in a mixed platform cluster, once on the Alpha system disk (or disks), once on the Integrity system disk (or disks), and once on the VAX common system disk.

Release Notes and Online Documentation

The PSCSSH *Release Notes* provide important information on the current release.

The *Release Notes*, along with the rest of the documentation set for this product, are always available for viewing and download on the Process Software website at https://www.process.com/ under the Support area.

2. Installing PSCSSH

This chapter takes you through the PSCSSH product installation procedure and certain post-installation tasks. It is for the OpenVMS system manager, administrator, or technician responsible for product installation.

To prepare for installation, see Chapter 1, *Before You Begin*.

> **Note:** Once you have installed PSCSSH, you need to reinstall it after you have done a major OpenVMS upgrade.

To install PSCSSH:

1. Load the software.
2. Run the `VMSINSTAL` procedure.
3. Install other products, if needed, and perform post-installation tasks.

Load the Software

PSCSSH is available as a download.

There are three steps to loading the PSCSSH software:

1. Log in to the system manager's account.
2. If PSCSSH is currently running, shut it down:

```
$ SSHCTRL SHUTDOWN
```

3. If you are installing on a VMScluster, shut down PSCSSH on each node in the cluster.

> **Note:** If you install PSCSSH on a VMS cluster that has a common system disk, install the software on only one node in the cluster. **If reinstalling or upgrading PSCSSH, first shut down PSCSSH on all nodes in the cluster.**
>
> Be sure to configure PSCSSH on all systems in a VMS cluster that has a common system disk, even though it only needs to be installed once.

Start VMSINSTAL

VMSINSTAL is the OpenVMS installation program for layered products. VMSINSTAL prompts you for any information it needs. The below table shows the steps to follow.

1. Make sure that you are logged in to the system manager's account, and invoke VMSINSTAL
2. Determine if you are satisfied with your system disk backup
3. Determine where the distribution volumes will be mounted
4. Enter the products you want processed from the first distribution volume set
5. Enter the installation options you wish to use (such as obtaining the *Release Notes*)
6. Specify the directory where you want the files installed.
7. Specify the directory where you want the system-specific files installed

Sample Installation

```
$ @SYS$UPDATE:VMSINSTAL PSCSSH030 DKA600:[PSCSSH030]

     OpenVMS  Software Product Installation Procedure V9.2-3


It is 13-MAY-2025 at 11:38.

Enter a question mark (?) at any time for help.

%VMSINSTAL-W-NOTSYSTEM, You are not logged in to the SYSTEM account.
%VMSINSTAL-W-ACTIVE, The following processes are still active:
       Goat Busters


The following products will be processed:

  PSCSSH V3.0


     Beginning installation of PSCSSH V3.0 at 11:38

No signature manifests found for PSCSSH030

* Do you want to install this product [NO]? yes
```

```
%VMSINSTAL-I-RESTORE, Restoring product save set A ...
%VMSINSTAL-I-RELMOVED, Product's release notes have been moved to SYS$HELP.
* Where do you want to install PSCSSH for OpenVMS [SYS$SYSDEVICE:[PSCSSH]]:
RETURN
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
_X86$DKA0:[PSCSSH].
%VMSINSTAL-I-RESTORE, Restoring product save set E ...
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
_X86$DKA0:[PSCSSH.X86].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_COMMON_ROOT:[COMMON].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_COMMON_ROOT:[X86_EXE].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_LOCAL_ROOT:[LOG].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_LOCAL_ROOT:[SSH].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_LOCAL_ROOT:[SSH2].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_LOCAL_ROOT:[SSH2.HOSTKEYS].
%VMSINSTAL-I-SYSDIR, This product creates system disk directory
PSCSSH_LOCAL_ROOT:[SSH2.KNOWNHOSTS].

  Installation Complete.

  Please report any problems to support@process.com


  Before starting PSCSSH, you must configure it using this command:

                  $ @sys$startup:pscssh_startup.com logicals
                  $ @pscssh:pscssh configure

  Once configured, execute this command and add it to your system
  startup produre:

                  $ @sys$startup:pscssh_startup.com

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target
directories...

     Installation of PSCSSH V3.0 completed at 11:38
$
```

Installing PSCSSH for the First Time on a Common VMScluster System Disk

After installing PSCSSH on one node of a VMScluster with a common system disk, you must perform the following steps on each additional cluster node that shares the common system disk:

1. Log in (telnet/set host/etc.) to the next node of the cluster.
2. Create the SSH logicals by using the following command:
   ```
   $ @SYS$STARTUP:PSCSSH_STARTUP LOGICALS
   ```
3. Make the node-specific SSH root and configure SSH for this node:
   ```
   $ @PSCSSH:SSH_MAKE_ROOT
   ```
4. Start PSCSSH:
   ```
   $ @SYS$STARTUP:PSCSSH_STARTUP
   ```
5. Repeat steps 1-4 for each remaining node of the cluster except for the one where SSH was originally installed.

3. Configuring PSCSSH

This chapter describes how to configure the SSHD Master process, which controls access to the SSH servers for the PSCSSH software.

For a basic configuration, accept the default values for each component, which appear after a prompt. This also helps you step through the process more quickly.

After performing the basic configuration, you must perform the advanced configuration for the SSH1 and SSH2 servers, and for the SSH clients as desired. Chapters 4 through 7 describe the configuration and use of these components.

The SSH Configuration Utility

SSH is the Secure Shell protocol. PSCSSH provides support for both SSH Version 1 protocol and SSH Version 2 protocol.

Please note that in addition to the configuration performed via `CNFSSH` as described below, there are configuration files for both the SSH1/SSH2 servers and SSH client which must be modified as appropriate to meet the security requirements of your organization. Refer to chapters 4 and 5 of this manual for details on the configuration files.

You can use the `CNFSSH` utility to configure the SSH server as shown in the below example.

```
$ @PSCSSH:PSCSSH CONFIGURE


PSCSSH Version V3.0A SSH Configuration procedure

This procedure helps you define the parameters needed to get
PSCSSH running on this system.

This procedure creates the configuration data file,
PSCSSH_LOCAL:SSH_CONFIGURE.COM,
to reflect your system's configuration.

For detailed information on the following parameters, refer to the
PSCSSH Administration and User Guide.

Do you want to enable the SSH2 server [NO]? yes
```

```
You may specify an alternate configuration file for the SSH2 server. If you
have already specified an alternate configuration file, enter a single space
and hit RETURN at the prompt to reset it to the default file name.

Enter an alternate SSH2 configuration filename []: RETURN

Specify the level of debug for the SSH2 server.

The level is a value from 0 to 50, where zero is no debug and 50 is
the maximum level of debug.  Note that at levels exceeding debug level 8,
there may be a substantial impact on SSH2 server (and possibly, the system,
too) performance due to the amount of information logged.

Enter the debug level [0 - 50, 0]: RETURN

You may specify the number of seconds a user has to enter a password during
user authentication (default = 600).  In addition, you may allow this
to default to the value used by OpenVMS when a user is logging into a
non-SSH session.  To specify an infinite wait time, enter 0 for the timeout
value.

Do you want to change the default login grace time [NO]? RETURN

Specify the address for the SSH server to listen on, if you wish to use
an address other than the default listen_address of ANY (0.0.0.0).  Any
valid IPV4 or IPV6 address may be specified, or ANY to listen on all
addresses.

Enter address to listen on [ANY]: RETURN

Specify the port for the SSH server to listen on, if you wish to use
a port other than the default port of 22.

Enter port to use [22]: RETURN
Do you want to suppress SSH server logging (/QUIET mode) [YES]? RETURN
Do you want verbose logging by the SSH server [NO]? RETURN

You may specify the maximum number of concurrent SSH sessions to be
allowed on the server. The default is 1000 sessions.

Enter maximum number of concurrent SSH sessions [1-1000, 1000]: RETURN

You may permit the server to log a brief informational message when a
user is allowed or denied access to a system.

- For SSH2 sessions, an ACCEPT or REJECT event will be logged when the
  user is either successfully authenticated or fails authentication.  The
  message will be of the form:

  <date><time> SSH2 (accepted) from user "foo" at [192.168.0.1,111]
  (my.server.com)
```

```
You may specify the name and location of the log file to record accepted
and/or rejected connections.  If you simply hit RETURN, this information
will be logged to OPCOM as opposed to a disk file.

By default, this file will be in the SSH_DIR: directory.  You may
override this by specifying a complete filename, including the directory
specification; or by specifying a logical name that translates to a
full filename specification.

Do you want to log accepted sessions [NO] RETURN
Do you want to log rejected sessions [NO] RETURN

When generating user keys, a passphrase may be used to further protect
the key.  No limit is normally enforced for the length of the passphrase.
However, you may specify a minimum length the passphrase may be.

What you want the minimum passphrase length to be for SSH2 [0-1024, 0]?


The SSH2 host key has not yet been generated.  Answer YES to the
following question to generate the key now.  Answer NO to generate
the key manually later by issuing the command:

   $ PSCSSH SSHKEYGEN /SSH2/HOST/KEYTYPE=ECDSA/BITS=521

Generating a host key can take a few minutes on slow systems.

Do you want to generate the SSH2 host key now [YES]? RETURN
Generating 521-bit ecdsa key pair

Key generated.
521-bit ecdsa, hunter@x86.goatley.com, Tue May 13 2025 11:40:41 -0400
Private key saved to PSCSSH_ssh2_hostkey_dir:hostkey_ecdsa
Public key saved to PSCSSH_ssh2_hostkey_dir:hostkey_ecdsa.pub

Public key digest for DNS:
x86.goatley.com.  IN  SSHFP  3  1  0d0b90403716af7d8191e5eecd67f18cc23bdd1c
x86.goatley.com.  IN  SSHFP  3  2
22aada6b5aa93d362699a85b7ee1d33ee6d885b793589
5090ea34aeed19efeb1



************************************************************************
************************************************************************

                            PLEASE NOTE

The following VERB definitions, provided by TCP/IP Services,
will be deleted, as they will conflict with corresponding
PSCSSH commands:
```

```
    SSH, SSH2, SCP, SCP2, SFTP, SFTP2

Note: %CDU-W-NOSUCHVERB messages may be ignored

The CLI table does not contain verb name SCP
The CLI table does not contain verb name SFTP2

The following file, supplied by TCP/IP Services, should be edited:

    SYS$MANAGER:TCPIP$DEFINE_COMMANDS.COM

Comment out the command definitions for the foreign TCP/IP Services
commands that are listed under the "ssh2 utilities" heading,
which may include:

    scp, sftp, ssh, ssh_add, ssh_agent, ssh_keygen

Failure to remove these commands may result in the incorrect SSH
utility being run instead of the intended PSCSSH utility.

**********************************************************************
**********************************************************************



SSH Configuration completed.

Review the additional steps you may need to perform as described in
the configuration chapters of the PSCSSH Administration and
User Guide before starting SSH.

Refer to the "Monitoring and Controlling SSH" chapter of the SSH for
OpenVMS Administration and User Guide for information on starting SSH.

$
```

4. Configuring the Secure Shell (SSH) v1 Server

This chapter describes how to configure and maintain the PSCSSH Secure Shell (SSH) v1 server.

> **Note:** SSH1 is deprecated and not configurable via the PSCSSSH CONFIGURE utility. Though its use is not recommended, it is possible to enable the SSH1 server by editing PSCSSH_LOCAL:SSH_CONFIGURE.COM and setting ENABLE_SSH1 to 1:
>
> ```
> $ ENABLE_SSH1 == 1
> ```

This is the server side of the software that allows secure interactive connections to other computers. The SSH server has been developed to discriminate between SSH v1 and SSH v2 protocols, so the two protocols can coexist simultaneously on the same system.

SSH1 and SSH2 Differences

SSH1 and SSH2 are different, and incompatible, protocols. The SSH1 implementation is based on the V1.5 protocol, and the SSH2 implementation is based on the V2 protocol. While SSH2 is generally regarded to be more secure than SSH1, both protocols are offered by PSCSSH, and although they are incompatible, they may exist simultaneously on an OpenVMS system. The PSCSSH server front-end identifies what protocol a client desires to use and will create an appropriate server for that client.

> **Note:** You must install the DEC C 6.0 backport library on all OpenVMS VAX v5.5-2 and v6.0 systems prior to using SSH. This is the `AACRT060.A` file. You can find the ECO on the PSCSSH CD in the following directory: `VAX55_DECC_RTL.DIR`.

When using SSH1 to connect to a VMS server, if the VMS account is set up with a secondary password, SSH1 does not prompt the user for the secondary password. If the VMS primary password entered is valid, the user is logged in, bypassing the secondary password.

When using SSH1 to execute single commands (in the same manner as RSHELL), some keystrokes like **CTRL**+**Y** are ignored. In addition, some interactive programs such as `HELP` may not function as expected. This is a restriction of SSH1. If this behavior poses a problem, log into the remote system using SSH1 in interactive mode to execute the program.

Understanding the Secure Shell Server

Secure Shell daemon (SSHD) is the daemon program for SSH that listens for connections from clients. The server program replaces rshell and telnet programs. The server/client programs provide secure encrypted communication between two untrusted hosts over an insecure network. A new daemon is created for each incoming connection. These daemons handle key exchange, encryption, authentication, command execution, and data exchange.

Servers and Clients

An SSH server is an OpenVMS system server that acts as a host for executing interactive commands or for conducting an interactive session. The server software consists of two processes (for future reference, “SSHD” will refer to both SSHD_MASTER and SSHD, unless otherwise specified):

- SSHD_MASTER, recognizes the differences between SSH v1 and SSH v2 and starts the appropriate server. If the request is for SSH v1, then a new SSH v1 server is run; if the request is for SSH v2, then a new SSH v2 server is run.
- SSHD, a copy of which is spawned for each time a new connection attempt is made from a client. SSHD handles all the interaction with the SSH client.

A client is any system that accesses the server. A client program (SSH) is provided, but any SSH client that uses SSH version 1 protocol may be used to access the server. Examples of such programs are FISSH, MultiNet SSH, and TCPware SSH on OpenVMS systems; TTSSH, SecureCRT, F-Secure SSH Client, and PuTTY on Windows-based systems; and other SSH programs such as OpenSSH on UNIX-based systems.

Security

Each host has a host-specific RSA key (normally 1024 bits) that identifies the host. Additionally, when the SSHD daemon starts, it generates a server RSA key (normally 768 bits). This key is regenerated every hour (the time may be changed in the configuration file) if it has been used and is never stored on disk. Whenever a client connects to the SSHD daemon,

- SSHD 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 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.
- The client and the server 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, IDEA (the default), DES, 3DES, Blowfish, and ARCFOUR are supported.

- The client selects the encryption algorithm to use from those offered by the server.
- The server and the client enter an authentication dialog.
- The client tries to authenticate itself using any of the following methods:
    - `.rhosts` authentication
    - `.rhosts` authentication combined with RSA host authentication
    - RSA challenge-response authentication
    - password-based authentication

> **Note:** 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 the RLOGIN and RSHELL services are disabled. When the client authenticates itself successfully, a dialog is entered for preparing the session. At this time the

client may request things such as:

- forwarding X11 connections
- forwarding TCP/IP connections
- forwarding the authentication agent connection over the secure channel

Finally, the client either requests an interactive session or execution of a command. The client and the server enter session mode. In this mode, either the client or the server may send data at any time, and such data is forwarded to/from the virtual terminal 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.

Care must be exercised when configuring the SSH clients and server to minimize problems due to intrusion records created by OpenVMS security auditing. The SSH user should consult the system

manager to determine the authentication methods offered by the SSH server. The client should then be configured to not attempt any authentication method that is not offered by the server.

If a client attempts authentication methods not offered by the server, the OpenVMS security auditing system may log several intrusion records for each attempt to create a session to that server. The result being that the user could be locked out and prevented from accessing the server system without intervention from the server's system manager.

The authentication methods to be offered by the server are determined by the configuration keywords `RhostsAuthentication`, `RhostsRSAAuthentication`, `RSAAuthentication`, and `PasswordAuthentication`. The number of intrusion records to be logged for any attempted SSH session is determined by the `StrictIntrusionLogging` configuration keyword.

When `StrictIntrusionLogging` is set to `YES` (the default), each method that is tried and fails causes an intrusion record to be logged:

- When Rhosts, RhostsRSA or RSA authentications are attempted and fail, one intrusion record will be logged for each failed method.
- When password authentication is attempted, one intrusion record will be logged for each failed password.

Example 1

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. If all methods fail, five intrusion records are logged:

1 for the failed Rhosts
1 for the failed RSA
3 for the failed password attempts, one per attempt

When `StrictIntrusionLogging` is set to `NO`, it has the effect of relaxing the number of intrusions logged. Overall failure of all authentication methods simply counts as a single failure, except for password authentication. The following rules apply:

- When password authentication is attempted, one intrusion record is logged for each failed password.
- When any of Rhosts, RhostsRSA, or RSA authentication fails, and password authentication is not attempted, exactly one intrusion record is logged, as opposed to one for each failed method.
- When any of Rhosts, RhostsRSA, or RSA authentication fails, but password authentication is attempted and succeeds, the only intrusion record(s) logged is one for each failed password attempt.

Example 2

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. If all methods fail, three intrusion records are logged:

0 for the failed Rhosts
0 for the failed RSA
3 for the failed password attempts, one per attempt

Example 3

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. Rhosts and RSA fail, but password authentication is successful after 1 failed password. Therefore, one intrusion record is logged:

0 for the failed Rhosts
0 for the failed RSA
1 for the failed password attempt

Example 4

The server is set up to allow Rhosts, RhostsRSA, and RSA authentication, but not password authentication. If all methods fail, one intrusion record is logged.

Example 5

The server is set up to allow Rhosts, RhostsRSA, and RSA authentication, but not password authentication. Rhosts and RSA authentication both fail, but RhostsRSA succeeds. No intrusion records are logged.

The SSH v1 protocol does not provide a method for changing an expired VMS password. When an expired password is encountered by the SSH1 server, it will do one of two things.

1. If the logical name `PSCSSH_SSH_ALLOW_EXPIRED_PW` is defined for allowing access for passwords that have exceeded the UAF value for `PWDLIFETIME`, or if the logical name `PSCSSH_SSH_ALLOW_PREEXPIRED_PW` is defined for allowing access for users that have a pre-expired password, the server will allow the user to log in. In the logical name table `LNM$SSH_LOGICALS`, the logical name `PSCSSH_SSH_`*`pid`*`_PWDEXP` (where *`pid`* is the process ID for the user process) will be defined. The system manager can look for this logical to be defined, and if so, take action such as executing the DCL `SET PASSWORD` command.

2. If the appropriate logical is not set as described above, the user will be denied access to the system. In that case, the user must log in interactively via another mechanism such as telnet and change the password, or the system manager must reset the password.

When a user is allowed access to the system with an expired password, the `LOGIN_FLAGS` for the process will reflect this. The values of the `LOGIN_FLAGS` will be as follows:

- new mail has been received (`JPI$M_NEW_MAIL_AT_LOGIN`)
- the password is about to expire (`JPI$M_PASSWORD_WARNING`)
- the password has expired (`JPI$M_PASSWORD_EXPIRED`)

The DCL lexical function `F$GETJPI` may be used to examine these flags, as can the `$GETJPI(W)` system service or `LIB$GETJPI` RTL function. When an expired password value is detected, the user may then execute a `SET PASSWORD` command in the command procedure run for the account.

For example:

```
$!
$! Login_flags:
$!    1 = new mail messages waiting (JPI$M_NEW_MAIL_AT_LOGIN)
$!    4 = password expired during login (JPI$M_PASSWORD_EXPIRED)
$!    5 = password expires within 5 days (JPI$M_PASSWORD_WARNING)
$!
$ flags = f$getjpi("", "LOGIN_FLAGS")
$ new_flags = (flags/2)*2
$ if new_flags .ne. flags then write sys$output "New mail waiting"
$!
$!Note - new_flags is used below because it has the NEW_MAIL_AT_LOGIN$
$!   bit stripped.  The rest of the possible values are all
$!   discrete; i.e., you can't have combinations of them at the
$!   same time.
$!
$ if new_flags .eq. 4 then write sys$output "Password expired during login"
$ if new_flags .eq. 5 then write sys$output "Password expires within 5 days"
$!
```

Configuration File

SSHD reads configuration data from `PSCSSH_LOCAL_ROOT:[SSH]SSHD_CONFIG`. The file contains keyword value pairs, one per line. The following keywords are possible. Keywords are case insensitive

Keyword	Value	Default	Description

AllowForwardingPort	Port list		Permit forwarding for the specified ports
AllowForwardingTo	Host/port list		Permit forwarding for hosts
AllowGroups	List		Access control by UAF rights list entries
AllowHosts	Host list		Access control by hostname
AllowShosts	Host list		Access control by hostname
AllowTcpForwarding	Y/N	Y	Enable TCP port forwarding
AllowUsers	User list		Access control by username
DenyForwardingPort	Port list		Forbid forwarding for ports
DenyForwardingTo	Host/port list		Forbid forwarding for hosts
DenyGroups	Rights list		Deny access for UAF rightslist identifiers
DenyHosts	Host list		Deny access for hosts
DenySHosts	Host list		Deny access for hosts
DenyUsers	User list		Access control by username
FascistLogging	Y/N	Y	Verbose logging
Hostkey	Filename	Ssh_host_key.	Host key filename
IdleTimeout	Time	0 (infinite)	Set idle timeout
IgnoreRhosts	Y/N	N	Ignore local rhosts
IgnoreRootRhosts	Y/N	Y	Ignore system rhosts
KeepAlive	Y/N	Y	Send keepalives
ListenAddress	IP address	0.0.0.0	Listen on given interface
LoginGraceTime	Time	600	Time limit for authentication in seconds
PasswordAuthentication	Y/N	Y	Permit password authentication

PermitEmptyPasswords	Y/N	N	Permit empty (blank) passwords
PermitRootLogin	Y/N	N	SYSTEM can log in
QuietMode	Y/N	N	Quiet mode
RandomSeed	Filename	Random_seed	Random seed file
RhostsAuthentication	Y/N	N	Enable rhosts authentication
RhostsRSAAuthentication	Y/N	Y	Enable rhosts with RSA authentication
RSAAuthentication	Y/N	Y	Enable RSA authentication
StrictIntrusionLogging	Y/N	Y	Determine how intrusion records are created by failed authentication attempts
StrictModes	Y/N	N	Strict checking for directory and file protection
SyslogFacility	Syslog level	“DAEMON”	Syslog log facility
VerboseLogging	Y/N	Y	Verbose logging (also known as FacistLogging)
X11Forwarding	Y/N	Y	Enable X11 forwarding
X11DisplayOffset	#offset	10	Limit X displays for SSH

Starting the SSH Server for the First Time

Follow these instructions to configure the SSH server. You must define the SSH logicals by using:

```
$ @SYS$STARTUP:PSCSSH$STARTUP LOGICALS
```

1. Use the CNFSSH utility to configure the SSH server. It is recommended that the host keys be

generated when executing the CNFSSH procedure, by answering Y to the question “Do you want to generate the SSH1 host key now?”

2. Use SSHKEYGEN to create the file SSH_HOST_KEY in the SSH_DIR: directory if it has not been created as a result of executing @PSCSSH:PSCSSH CONFIGURE.

```
$ PSCSSH SSHKEYGEN/SSH1/HOST
Initializing random number generator...
Generating p:
.....................................................................
.++ (distance 1088)
Generating q:
.....................................................................
.....................................++ (distance 1740)
Computing the keys...
Testing the keys...
Key generation complete.
Key file will be PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY.
Your identification has been saved in PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY..
Your public key is:
1024 35
109063762180373766639976433607825468151696193384697555192852820112461975
383752379934485653144020088702805389287419366851316107097504046574784394895
30538
605753838835403334156385162571585234870131829834266831040946766291651087530
75494
501144617818928909935245497229091318193823986540398801616376199266702131407
13 SY
STEM@x86.kingkong.com
Your public key has been saved in PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY.pub
$
```

3. Edit the default configuration file at SSH_DIR:SSHD_CONFIG (if you wish to change the default settings). This default configuration is the same as contained in the file PSCSSH_COMMON:SSHD_CONFIG.TEMPLATE.

> **Note:** As delivered, the template file provides a reasonably secure SSH environment. However, Process Software recommends this file be examined and modified appropriately to reflect the security policies of your organization.

4. Restart SSH. This creates the SSH server process and defines the SSH logical names.

```
$ SSHCTRL RESTART
$ SHOW PROCESS "SSHD Master"
22-JUL-2025 09:03:06.42  User: SYSTEM      Process ID:   00000057
Node: PANTHR      Process name: "SSHD Master"
```

```
Terminal:
User Identifier:    [SYSTEM]
Base priority:      4
Default file spec:  Not available
Number of Kthreads: 1
Devices allocated:  BG1:
                    BG2:
$ SHOW LOGICAL/SYSTEM SSH*
(LNM$SYSTEM_TABLE)
  "SSH2_DIR" = "PSCSSH_LOCAL_ROOT:[SSH2]"
  "SSHLEI" = "PSCSSH_EXE:SSHLEI"
  "SSHSHR" = "PSCSSH_EXE:SSHSHR"
  "SSH_DIR" = "PSCSSH_LOCAL_ROOT:[SSH]"
  "SSH_EXE" = "PSCSSH_EXE:"
  "SSH_FSCLM" = "PSCSSH_EXE:SSH_FSCLM.EXE"
  "SSH_LOG" = "PSCSSH_LOCAL_ROOT:[LOG]"
  "SSH_MAX_SESSIONS" = "1000"
  "SSH_TERM_MBX" = "MBA43:"
  "SSH_ZLIB" = "PSCSSH_EXE:SSH_ZLIB.EXE"
```

Changing the SSH1 Configuration File After Enabling SSH1

If you make a change to the SSH1 configuration file after you have enabled SSH1, you must restart SSH for these changes to take effect.

```
$ SSHCTRL RESTART
```

> **Note:** When issuing the `RESTART` command for SSH, all active SSH server sessions are terminated. Active client sessions are not affected.

Connection and Login Process

To create a session, SSHD does the following:

1. SSHD_MASTER process sees the connection attempt. It creates an SSHD v1 or v2 process, depending on the protocol version presented to it by the client. SSHD_MASTER then passes necessary information to the SSHD process, such as the server key and other operating parameters.

2. SSHD process performs validation for the user.

3. Assuming the login is successful, SSHD process creates a pseudoterminal for the user (an `FTA`*`nn`*`:` device). This device is owned by the user logging in.

4. SSHD process creates an interactive process on the pseudoterminal, using the username, priority, and privileges of the user logging in. If a command was specified, it is executed and the session is terminated.

5. SSH generates the file `SSHD.LOG` in the directory `PSCSSH_LOCAL:` for each connection to the SSH server. Many connections result in many log files. Instead of purging the files on a regular basis, use the following DCL command to limit the number of versions:

```
$ SET FILE /VERSION_LIMIT=x PSCSSH_LOCAL:SSHD.LOG
```

> **Note:** The value for `/VERSION_LIMIT` must not be smaller than the maximum number of simultaneous SSH sessions anticipated. If the value is smaller, SSH users may be prevented from establishing sessions with the server.

FILES

PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY.
Contains the private part of the host key. This file does not exist when PSCSSH is first installed.

The SSH server starts only with this file. This file must be created using PSCSSH:PSCSSH CONFIGURE or manually using the command:

```
$ PSCSSH SSHKEYGEN /SSH1 /HOST
```

This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others.

To create a host key with a name that is different than what `SSHKEYGEN` creates, do one of the following:

- Generate with `PSCSSH SSHKEYGEN /SSH1 /HOST` and simply rename the file.
- Generate a public/private key pair using `SSHKEYGEN` without the `/HOST` switch, then copy and rename the resulting files appropriately.

PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY.PUB

Contains the public part of the host key. This file should be world-readable but writeable only by SYSTEM. Its contents should match the private part of the key. This file is not used for anything; it is only provided for the convenience of the user so its contents can be copied to known hosts files.

PSCSSH_LOCAL_ROOT:[SSH]SSH_KNOWN_HOSTS
SYS$LOGIN[1]:[.SSH]KNOWN_HOSTS
Checks the public key of the host. These files are consulted when using `rhosts` with RSA host authentication. The key must be listed in one of these files to be accepted. (The client uses the same files to verify that the remote host is the one you intended to connect.) These files should be writeable only by SYSTEM (the owner). `PSCSH_LOCAL_ROOT:[SSH]SSH_KNOWN_HOSTS` should be world-readable, and `SYS$LOGIN:[.SSH]KNOWN_HOSTS` can, but need not be, world-readable.

SSH2_DIR:SSH_RANDOM_SEED
SYS$LOGIN:[.SSH]RANDOM_SEED
Contains a seed for the random number generator. This file should only be accessible by system.

PSCSSH_LOCAL_ROOT:[SSH]SSHD_CONFIG
Contains configuration data for SSHD. This file should be writeable by system only, but it is recommended (though not necessary) that it be world-readable.

AUTHORIZED_KEYS
In the user's `SYS$LOGIN[.SSH]` directory, this file lists the RSA keys that can be used to log into the user's account. This file must be readable by system. It is recommended that it not be accessible by others. The format of this file is described below.

AUTHORIZED_KEYS File Format

The `SYS$LOGIN:[.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 comments and ignored). Each line consists of the following fields, separated by spaces:

Key	Description
`bits`	Is the length of the key in bits.
`comment`	Not used for anything (but may be convenient for the user to identify the key).
`exponent`	Is a component used to identify and make up the key.

[1] . In this chapter, the [.SSH] subdirectory in the user's login directory displays as SYS$LOGIN:[.SSH]

`modulus`	Is a component used to identify and make up the key.
`options`	Optional; its presence is determined by whether the line starts with a number or not (the option field never starts with a number.)

> **Note:** Lines in this file are usually several hundred characters long (because of the size of the RSA key modulus). You do not want to type them in; instead, copy the `IDENTITY.PUB` file and edit it. The options (if present) consist of comma-separated option specifications. No spaces are permitted, except within double quotes. Option names are case insensitive.

The following RSA key file `AUTHORIZED_KEYS` option specifications are supported:

`Allowforwardingport="ports"`
Can be followed by any number of port numbers, separated by spaces. Remote forwarding is allowed for those ports whose number matches one of the patterns.

You can use `*` as a wildcard entry for all ports.

You can use these formats '`>x`', '`<x`', and '`x_y`' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.

The quotes (" ") are required. For example: `allowforwardingport "2,52,2043"`

`Allowforwardingto="host_port_list"`
Can be followed by any number of hostname and port number patterns, separated by spaces. A port number pattern is separated from a hostname pattern by a colon. For example: *`hostname`*`:`*`port`*

Forwardings from the client are allowed to those hosts and port pairs whose name and port number match one of the patterns.

You can use `*` and `?` as wildcards in the patterns for host names. Normal name servers are used to map the client's host into a fully-qualified host name. If the name cannot be mapped, its IP address is used as the hostname.

You can use `*` as a wildcard entry for all ports.

You can use these formats '`>x`', '`<x`', and '`x_y`' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.

command="*command*"
Specifies the command to be executed whenever this key is used for authentication. The user-supplied command (if any) is ignored. You may include a quote in the command by surrounding it with a backslash (\). Use this option 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 prohibited explicitly.

Denyforwardingport="ports"
Can be followed by any number of port numbers, separated by spaces. Remote forwardings are disallowed for those ports whose number matches one of the patterns.

You can use * as a wildcard entry for all ports.

You can use these formats '>x', '<x', and 'x_x' to specify greater than, less than, or inclusive port range.

Denyforwardingto="*host_port_list*"
Can be followed by any number of hostname and port number patterns, separated by spaces. A port number pattern is separated from a hostname by a colon. For example: *hostname*:*port_number_pattern*

Forwardings from the client are disallowed to those hosts and port pairs whose name and port number match one of the patterns.

You can use * and ? as wildcards in the patterns for host names. Normal name servers are used to map the client's host into a fully-qualified host name. If the name cannot be mapped, its IP address is used as a host name.

You can use * as a wildcard entry for all ports.

You can use these formats '>x', '<x', and 'x_x' to specify greater than, less than, or inclusive port range.

from="*pattern-list*"
In addition to RSA authentication, specifies that the fully-qualified name of the remote host must be present in the comma-separated list of patterns. You can use * and ? as wildcards.

The list may contain patterns negated by prefixing them with ! - if the fully-qualified host name matches a negated pattern, the key is not accepted.

This option increases security. RSA authentication by itself does not trust the network or name servers (but the key). However, if somebody steals the key, the key permits login from anywhere in the world.

This option makes using a stolen key more difficult because the name servers and/or routers would have to be comprised in addition to just the key.

idle-timeout=*time*
Sets the idle timeout limit to a time in seconds (`s` or nothing after the number), in minutes (`m`), in hours (`h`), in days (`d`), or in weeks (`w`). If the connection has been idle (all channels) for that time, the process is terminated and the connection is closed.

no-agent-forwarding
Forbids authentication agent forwarding when used for authentication.

no-port-forwarding
Forbids TCP/IP forwarding when used for authentication. Any port forward requests by the client will return an error. For example, this might be used in connection with the command option.

no-X11-forwarding
Forbids X11 forwarding when used for authentication. Any X11 forward requests by the client will return an error.

SSH_KNOWN_HOSTS File Format

The `PSCSSH_LOCAL_ROOT:[SSH]SSH_KNOWN_HOSTS` and `SYS$LOGIN:[.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. 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 is matched against the fully-qualified host names (when authenticating a client) or against the user-supplied name (when authenticating a server). A pattern may be preceded by `!` to indicate negation; if the hostname 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 from PSCSSH_LOCAL_ROOT:[SSH]`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 permissible (but not recommended) to have several lines or different host keys for the same names. This happens 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:** The lines in these files are hundreds of characters long. Instead of typing in the host keys, generate them by a script or by copying `PSCSSH_LOCAL_ROOT:[SSH]SSH_HOST_KEY.PUB` and adding the host names at the front.

SSH Logicals

These logicals are used with the SSH server in the system logical name table.

```
$ SHOW LOGICAL/SYSTEM *SSH*
```

SSH_DIR
Points to the directory where the SSH1 configuration, master server log file, and host key files are kept. Normally, this is `PSCSSH_LOCAL_ROOT:[SSH]`. It is defined in `START_SSH.COM`.

SSH_EXE
Points to the directory where SSH executables are kept. Normally, this is `PSCSSH_COMMON_ROOT:[xxx_EXE]`. It is defined in `START_SSH.COM`.

SSH_LOG
Points to the directory where the log files are kept. Normally, this is `PSCSSH_LOCAL_ROOT:[LOG]`. It is defined in `START_SSH.COM`.

SSH_TERM_MBX
Mailbox used by SSHD_MASTER to receive termination messages from SSHD daemon processes. **Do not change this logical name.** This is created by the SSHD_MASTER process.

PSCSSH_SSH_ACC_REJ_LOG_FILE
If the user has set a log file to log connection accept and reject messages, this logical will be defined and will provide the name of the log file. This logical is set by using the `@PSCSSH:PSCSSH CONFIGURE` command or by editing `PSCSSH_LOCAL:SSH_CONFIGURE.COM`.

PSCSSH_SSH_ALLOW_EXPIRED_PW
Allows logging in to an account when the account's password has expired due to pwdlifetime elapsing. This applies to all users and circumvents normal VMS expired-password checking, and therefore should be used with caution. An entry is made into the SSH_LOG:SSHD.LOG file when access is allowed using this logical name.

When access is allowed by way of this logical, the logical name table LNM$SSH_LOGICALS contains a logical name constructed as PSCSSH_SSH_*pid*_PWDEXP (where *pid* is the PID for the process). The system manager can use this to execute, for example, the DCL SET PASSWORD command in the site SYLOGIN.COM file.

PSCSSH_SSH_ALLOW_PREEXPIRED_PW
Allows logging in to an account when the password has been pre-expired. This applies to all users and circumvents normal VMS expired-password checking, and therefore should be used with caution. An entry is made into the SSH_LOG:SSHD.LOG file when access is allowed using this logical name.

When access is allowed by way of this logical, the logical name table LNM$SSH_LOGICALS contains a logical name constructed as PSCSSH_SSH_*pid*_PWDEXP (where *pid* is the PID for the process). The system manager can use this to execute, for example, the DCL SET PASSWORD command in the site SYLOGIN.COM file.

PSCSSH_SSH_DISPLAY_SYS$ANNOUNCE
The SSH v1 protocol does not allow for the display of SYS$ANNOUNCE prior to logging in. If this logical is set, the contents of SYS$ANNOUNCE is displayed immediately after successful authentication and prior to the display of the contents of SYS$WELCOME.

PSCSSH_SSH_ENABLE_SSH1_CONNECTIONS
Set to enable SSH V1 sessions.

PSCSSH_SSH_KEYGEN_MIN_PW_LEN
Defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero.

PSCSSH_SSH_LOG_ACCEPTS
When set, causes the server to log successful connection requests as either an OPCOM message or a line in a log file. Note that a successful connection request doesn't equate to a successful authentication request.

PSCSSH_SSH_LOG_MBX
Points to the OpenVMS mailbox used to log connection accept and reject messages. This must not be modified by the user.

PSCSSH_SSH_LOG_REJECTS
When set, causes the server to log rejected connection requests as either an OPCOM message or a line in a log file.

PSCSSH_SSH_MAX_SESSIONS
Set this to the maximum number of concurrent SSH sessions you want to allow on the server system. If `PSCSSH_SSH_MAX_SESSIONS` is not defined, the default is 1000. Setting `PSCSSH_SSH_MAX_SESSIONS` to zero (0) causes an error. The value must be between 1 and 1000. The suggested place to set this is in `START_SSH.COM`. You must restart SSH for these changes to take effect.

PSCSSH_SSH_PARAMETERS_*n*
These values are set by PSCSSH and must not be modified by the user.

PSCSSH_SSH_USE_SYSGEN_LGI
If defined, causes SSHD to use the VMS `SYSGEN` value of `LGI_PWD_TMO` to set the login grace time, overriding anything specified in the command line or the configuration file.

5. Configuring the Secure Shell (SSH) Server v2

This chapter describes how to configure and maintain the PSCSSH Secure Shell (SSH) server v2.

This is the server side of the software that allows secure interactive connections to other computers in the manner of rlogin/rshell/telnet. The SSH server has been developed to discriminate between SSH v1 and SSH v2 protocols, so the two protocols can coexist simultaneously on the same system.

SSH1 and SSH2 Differences

SSH1 and SSH2 are different, and incompatible, protocols. The SSH1 implementation is based on the version 1.5 protocol, and the SSH2 implementation is based on the V2. While SSH2 is generally regarded to be more secure than SSH1, both protocols are offered by PSCSSH, and although they are incompatible, they may exist simultaneously on an OpenVMS system. The PSCSSH server front-end identifies what protocol a client desires to use and will create an appropriate server for that client.

The cryptographic library used by the SSH2 server (***this does not apply to SSH1 sessions***) is FIPS 140-2 level 2 compliant, as determined by the Computer Security Division of the National Institute of Science and Technology (NIST).

> **Note:** You must install the DEC C 6.0 backport library on all OpenVMS VAX v6.0 and earlier systems prior to using SSH. This is the `AACRT060.A` file. You can find the ECO on ftp.multinet.process.com in the following directory: PATCHES.`DIR`.

When using SSH2 to connect to a VMS server, if the VMS account is set up with a secondary password, SSH2 does not prompt the user for the secondary password. If the VMS primary password entered is valid, the user is logged in, bypassing the secondary password.

When using SSH2 to execute single commands (in the same manner as RSHELL), some keystrokes like **CTRL**+**Y** are ignored. In addition, some interactive programs such as `HELP` may not function as expected. This is a restriction of SSH2. If this behavior poses a problem, log into the remote system using SSH2 in interactive mode to execute the program.

Understanding the Secure Shell Server

Secure Shell daemon (SSHD) is the daemon program for SSH2 that listens for connections from clients. The server program replaces the rshell and telnet programs. The server/client programs provide secure encrypted communications between two untrusted hosts over an insecure network. A new daemon is created for each incoming connection. These daemons handle key exchange, encryption, authentication, command execution, and data exchange.

Servers and Clients

An SSH server is an OpenVMS system that acts as a host for executing interactive commands or for conducting an interactive session. The server software consists of two pieces of software (for future reference, “SSHD” will refer to both SSHD_MASTER and SSHD, unless otherwise specified):

- SSHD_MASTER, recognizes the differences between SSH v1 and SSH v2 and starts the appropriate server. If the request is for SSH v1, then the existing SSH v1 server is run; if the request is for SSH v2, then the SSH v2 server is run.
- SSHD, a copy of which is spawned for each connection instance. SSHD handles all the interaction with the SSH client.

A client is any system that accesses the server. A client program (SSH) is provided with PSCSSH, but any SSH client that uses SSH version 2 protocol may be used to access the server. Examples of such programs are MultiNet SSH, TCPware SSH, SecureCRT, and puTTY for Windows, MacSSH for Apple systems, and other SSH programs on Linux and UNIX-based systems.

Each host has a key using DSA encryption and is usually 1024 bits long (although, the user may create a different-sized key, if desired). The same key may be used on multiple machines. For example, each machine in a VMScluster could use the same key.

When a client connects to the SSHD daemon:

- The client and server together, using the Diffie-Hellman key-exchange method, determine a 256-bit random number to use as the "session key". This key is used to encrypt all further communications in the session.

Note that this key may be renegotiated between the client and the server on a periodic basis by including the `RekeyIntervalSeconds` keyword in the server configuration file (`SSH2_DIR:SSHD2_CONFIG`). This is desirable because during long sessions, the more data that is exchanged using the same encryption key, the more likely it is that an attacker who is watching the encrypted traffic could deduce the session key.

- The server informs the client which encryption methods it supports. See the description of the `CIPHERS` configuration keyword for the encryption methods supported.
- The client selects the encryption algorithm from those offered by the server.
- The client and the server then enter a user authentication dialog. The server informs the client which authentication methods it supports, and the client then attempts to authenticate the user by using some or all of the authentication methods. The following authentication algorithms are supported:
    - public-key (DSA keys)
    - host-based
    - password keyboard-interactive
    - Kerberos V5 (password, kerberos-tgt, kerberos-1, kerberos-tgt-1, kerberos-2, kerberos-tgt-2)
    - Certificate

System security is not improved unless the RLOGIN and RSHELL services are disabled.

If the client authenticates itself successfully, a dialog is entered for preparing the session. At this time the client may request things like:

- forwarding X11 connections
- forwarding TCP/IP connections
- forwarding the authentication agent connection over the secure channel

Finally, the client either requests an interactive session or execution of a command. The client and the server enter session mode. In this mode, either the client or the server may send data at any time, and such data is forwarded to/from the virtual terminal 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.

Expired Password Handling

The SSH2 server supports expired password changing for interactive accounts without the `CAPTIVE` or `RESTRICTED` flags set and, via the DCL `SET PASSWORD` command. When an expired password is detected, the server will behave as if a `SET PASSWORD` command was specified by the user as a

remotely executed command (e.g., `$ ssh host set password`), and the user will be logged out after changing the password. The user may then log in again using the changed password.

For `CAPTIVE` or `RESTRICTED` accounts, or for those accounts where `LGICMD` is set in the UAF record, the scenario is different. In these cases, the server can't directly execute `SET PASSWORD` command, because the command procedure specified in the `LGICMD` field of the UAF record will override the SSH server attempting to do a `SET PASSWORD` command. For these types of accounts, the system manager and/or user can use the value of the `LOGIN_FLAGS` for the process (normal interactive sessions may also examine these flags). For SSH logins, these flags will reflect:

- new mail has been received (`JPI$M_NEW_MAIL_AT_LOGIN`)
- the password is about to expire (`JPI$M_PASSWORD_WARNING`)
- the password has expired (`JPI$M_PASSWORD_EXPIRED`)

The DCL lexical function `F$GETJPI` may be used to examine these flags, as can the `$GETJPI(W)` system service or `LIB$GETJPI` RTL function. When an expired password value is detected, the user may then execute a `SET PASSWORD` command in the command procedure run for the account.

```
For example:
$!
$! Login_flags:
$!    1 = new mail messages waiting (JPI$M_NEW_MAIL_AT_LOGIN)
$!    4 = password expired during login (JPI$M_PASSWORD_EXPIRED)
$!    5 = password expires within 5 days (JPI$M_PASSWORD_WARNING)
$!
$ flags = f$getjpi("", "LOGIN_FLAGS")
$ new_flags = (flags/2)*2
$ if new_flags .ne. flags then write sys$output "New mail waiting"
$!
$! Note - new_flags is used below because it has the NEW_MAIL_AT_LOGIN$
$!   bit stripped.  The rest of the possible values are all
$!   discrete; i.e., you can't have combinations of them at the
$!   same time.
$!
$ if new_flags .eq. 4 then write sys$output "Password expired during login"
$ if new_flags .eq. 5 then write sys$output "Password expires within 5 days"
$!
```

When an account in the SYSUAF has an expired password and the system `syslogin.com` or user's `login.com` has a `SET TERM` command, a warning message will be displayed prior to prompting to change the password as shown in the following example:

```
Your password has expired; you must set a new password to log in

% SET-W-NOTSET, error modifying DKA0:
-SET-E-INVDEV, device is invalid for requested operation
```

```
Old password:
```

The way to suppress these warning messages would be to check for the appropriate login flag, ignoring any `SET TERM` commands. For example:

```
$ flags = $getjpi("", "LOGIN_FLAGS")
$ new_flags = (flags/2)*2
$ if new_flags.eq.4 then goto skip_the_inquiry
```

Break-In and Intrusion Detection

Care must be exercised when configuring the SSH clients and server to minimize problems due to intrusion records created by OpenVMS security auditing. The SSH user should consult the system manager to determine the authentication methods offered by the SSH server. The client should then be configured to not attempt any authentication method that is not offered by the server.

If a client attempts authentication methods not offered by the server, the OpenVMS security auditing system may log several intrusion records for each attempt to create a session to that server. The result being that the user could be locked out and prevented from accessing the server system without intervention from the server's system manager.

The authentication methods to be offered by the server are determined by the configuration keywords `AllowedAuthentications` and `RequiredAuthentications`. The number of intrusion records to be logged for any attempted SSH session is determined by the `StrictIntrusionLogging` configuration keyword.

When `StrictIntrusionLogging` is set to `YES` (the default), each method that is tried and fails causes an intrusion record to be logged. The following rules apply:

- When `HostBased` or `PublicKey` authentications are attempted and fail, one intrusion record is logged for each failed method.
- When password authentication is attempted, one intrusion record is logged for each failed password.

Example 1:

The server is set up to allow *HostBased* and password authentication; also, up to three password attempts are allowed. If all methods fail, four intrusion records are logged:

1 for the failed `HostBased`
3 for the failed password attempts, one per attempt

When StrictIntrusionLogging is set to NO, it has the effect of relaxing the number of intrusions logged. Overall failure of all authentication methods simply counts as a single failure, except for password authentication. The following rules apply:

- When password authentication is attempted, one intrusion record is logged for each failed password.
- When any of HostBased or PublicKey authentication fails, and password authentication is not attempted, exactly one intrusion record is logged, as opposed to one for each failed method.
- When any of HostBased or PublicKey authentication fails, but password authentication is attempted and succeeds, the only intrusion record(s) logged is one for each failed password attempt.

The server is set up to allow HostBased and password authentication; also, up to three password attempts are allowed. If all methods fail, three intrusion records are logged:

0 for the failed HostBased
3 for the failed password attempts, one per attempt

The server is set up to allow HostBased and password authentication; also, up to three password attempts are allowed. HostBased and RSA fail, but password authentication is successful after 1 failed password. Therefore, one intrusion record is logged:

0 for the failed HostBased
1 for the failed password attempt

The server is set up to allow HostBased and PublicKey authentication, but not password authentication. If all methods fail, one intrusion record is logged.

The server is set up to allow HostBased and PublicKey authentication, but not password authentication. HostBased authentication fails, but PublicKey succeeds. No intrusion records are logged.

Configuring SSHD Master

The SSHD Master is configured via CNFSSH. See Chapter 3 of the *PSCSSH Administration and User's Guide* for details on using CNFSSH to configure SSH.

> **Note:** The only supported methods for starting SSH are to use the @SYS$STARTUP:PSCSSH_STARTUP command if SSH isn't running, or to use the SSHCTRL RESTART command if SSH is currently running.

SSH2 Configuration File

SSHD reads configuration data from its configuration file. By default, this file is `SSH2_DIR:SSHD2_CONFIG`. The file contains keyword value pairs, one per line. Lines starting with `#` and empty lines are interpreted as comments. The following keywords are possible.

Keywords are case insensitive.

Keyword	Value	Default	Description
`AllowedAuthentications`	List	Publickey, Password	Permitted techniques. Valid values are: `keyboard-interactive, password, publickey, hostbased, kerberos-1, kerberos-tgt-1, kerberos-2, kerberos-tgt-2.` Along with *Required Authentications*, the system administrator can force the users to complete several authentications before they are considered authenticated.
`AllowedPasswordAuthentications`	List	`kerberos, local`	Specifies the different password authentication schemes that are allowed. Only `kerberos` and `local` are acceptable.
`AllowGroups`	List		Access control by UAF rights list entries
`AllowHosts`	Host list		Access control by hostname
`AllowShosts`	Host list		Access control by hostname
`AllowTcpForwarding`	Y/N	Y	Enable TCP port forwarding

`AllowTcpForwardingForUsers`	User list		Per-User forwarding
`AllowTcpForwardingForGroups`	Rights list		Per-rights list ID forwarding
`AllowUsers`	User list		Access control by username
`AllowX11Forwarding`	Y/N	Y	Enable X11 forwarding
`AuthInteractiveFailureTimeout`	Seconds	2	Delay, in seconds, that the server delays after a failed attempt to log in using `keyboard-interactive` and `password` authentication.
`AuthKbdInt.NumOptional`	Number	0	Specifies how many optional sub methods must be passed before the authentication is considered a success. (Note that all reported sub methods must always be passed.) See `AuthKbdInt.Optional` for specifying optional sub methods, and `AuthKbdInt.Required` for required sub methods. The default is 0, although if no required sub methods are specified, the client must always pass at least one optional sub method.
`AuthKbdint.Optional`	List	None	Specifies the optional sub methods keyboard-interactive will use. Currently only the sub method `password` is defined. `AuthKbdInt.NumOptional` specifies how many optional sub methods must be

			passed. The `keyboard-interactive` authentication method is considered a success when the specified amount of optional sub methods and all required sub methods are passed.
`AuthKbdInt.Required`			Specifies the required sub methods that must be passed before the `keyboard-interactive` authentication method can succeed.
`AuthKbdInt.Retries`	Number	3	Specifies how many times the user can retry `keyboard-interactive`.
`AuthorizationFile`	Filename	Authorization	Authorization file for public key authentication.
`AuthPublicKey.MaxSize`	Number	0	Specifies the maximum size of a public key that can be used to log in. Value of 0 disables the check.
`AuthPublicKey.MinSize`	Number	0	Specifies the minimum size of a public key that can be used to log in. Value of 0 disables the check.
`Cert.RSA.Compat.HashScheme`	`md5` or `sha`	md5	Previous clients and servers may use hashes in RSA certificates incoherently (sometimes SHA-1 and sometimes MD5). This specifies the hash used when a signature is sent to old versions during the initial key exchanges.
`BannerMessageFile`	Filename	`SYS$ANNOUNCE`	Message sent to the client before authentication begins.

CheckMail	Y/N	Y	Display information about new mail messages when logging in
Ciphers	Cipher list		Encryption ciphers offered
DenyGroups	Rights list		Deny access for UAF rights list identifiers
DenyHosts	Host list		Deny access for hosts
DenySHosts	Host list		Deny access for hosts
DenyTcpForwardingForUsers	User list		Forbid forwarding for listed users
DenyTcpForwardingForGroups	Rights list		Forbid forwarding for listed rights list names
DenyUsers	User list		Access control by username
FascistLogging	Y/N	Y	Verbose logging
ForwardACL	Pattern	None	With this option, you can have more fine-grained control over what the client is allowed to forward, and to where. See *ForwardACL Notes* below.
ForwardAgent	Y/N	Y	Enable agent forwarding
HostCA	Certificate	None	Specifies the CA certificate (in binary or PEM format) to be used when authenticating remote hosts. The certificate received from the host must be issued by the specified CA and must contain a correct alternate name of type DNS (FQDN). If no CA certificates are specified in the configuration file, the protocol tries to do key exchange with ordinary public keys. Otherwise,

			certificates are preferred. Multiple CAs are permitted.
`HostCANoCRLs`	Certificate	None	Similar to `HostCA`, but disables CRL checking for the given ca-certificate.
`HostCertificateFile`	Filename	None	This keyword works very much like `PublicHostKeyFile`, except that the file is assumed to contain an X.509 certificate in binary format. The keyword must be paired with a corresponding `HostKeyFile` option. If multiple certificates with the same public key type (DSS or RSA) are specified, only the first one is used.
`HostbasedAuthForceClient HostnameDNSMatch`	Y/N	N	Host name given by client.
`Hostkeyfile`	Filename	Hostkey	Host key filename
`HostSpecificConfig`	Pattern	None	Specifies a sub-configuration file for this server, based on the hostname of the client system.
`IdentityFile`	Filename	Identification	Identity filename
`IdleTimeout`	Time	0 = none	Set idle timeout (in seconds)
`IgnoreRhosts`	Y/N	N	Don’t use `rhosts` and `shosts` for host-based authentication for all users
`IgnoreRootRhosts`	Y/N	Y	Don’t use `rhosts` and `shosts` files for authentication of SYSTEM
`KeepAlive`	Y/N	Y	Send keepalives

`LdapServers`	Server URL	None	`Specified as ldap://`*`server.domainname`*`:389` CRLs are automatically retrieved from the CRL distribution point defined in the certificate to be checked if the point exists. Otherwise, the comma-separated server list given by option `LdapServers` is used. If intermediate CA certificates are needed in certificate validity checking, this option must be used or retrieving the certificates will fail.
`ListenAddress`	IP address	0.0.0.0	Listen on given interface
`Macs`	Algorithm		Select MAC (Message Authentication Code) algorithm
`MapFile`	Filename	None	This keyword specifies a mapping file for the preceding `Pki` keyword. Multiple mapping files are permitted per one `Pki` keyword. The mapping file format is described below.
`MaxBroadcastsPerSecond`	#broadcasts	0	Listen for UDP broadcasts
`NoDelay`	Y/N	N	Enable Nagel Algorithm
`PasswordAuthentication`	Y/N	Y	Permit password authentication
`PasswordGuesses`	#guesses	3	Limit number of password tries to specified number
`PermitEmptyPasswords`	Y/N	N	Permit empty (blank) passwords

PermitRootLogin	Y/N	N	SYSTEM can log in
Pki	Filename	None	This keyword enables user authentication using certificates. Cacertificate must be an X.509 certificate in binary format. This keyword must be followed by one or more MapFile keywords. The validity of a received certificate is checked separately using each of the defined Pki keywords in turn until they are exhausted (in which case the authentication fails), or a positive result is achieved. If the certificate is valid, the mapping files are examined to determine whether the certificate allows the user to log in. A correct signature generated by a matching private key is always required.
PkiDisableCrls	Y/N	Y	This keyword disables CRL checking for the Pki keyword, if set to “Y”.
PrintMotd	Y/N	Y	Display SYS$WELCOME when logging in
PublicHostKeyFile	Filename	Hostkey.pub	Host key file location
QuietMode	Y/N	N	Quiet mode
RandomSeedFile	Filename	Random_seed	Random seed file
RekeyIntervalSeconds	#seconds	0	Frequency of rekeying
RequiredAuthentication	Authentication list		Authentications client must support

`RequireReverseMapping`	Y/N	N	Remote IP address must map to hostname
`ResolveClientHostName`	Y/N	Y	Controls whether the server will try to resolve the client IP address at all, or not. This is useful when you know that the DNS cannot be reached, and the query would cause additional delay in logging in. Note that if you set this to "no", you should not set `RequireReverseMapping` to "Y".
`RSAAuthentication`	Y/N	Y	Enable RSA authentication
`SendKeyGuess`	Y/N	Y	This parameter controls whether the server will try to guess connection parameters during key exchange, or not. Some clients do not support key exchange guesses and may fail when they are present.
`SftpSysLogFacility`	log facility	None	Defines the log facility the SFTP server will use
`StrictIntrusionLogging`	Y/N	Y	Determine how intrusion records are created by failed authentication attempts.
`StrictModes`	Y/N	N	Strict checking for directory and file protection.
`SyslogFacility`	Facility	AUTH	Defines what log facility to be used when logging server messages.
`Terminal.AllowUsers`	pattern	All users	List users that are allowed terminal (interactive) access to the server.
`Terminal.DenyUsers`	pattern	None	List users that are denied terminal (interactive) access to the server.

`Terminal.AllowGroups`	pattern	All groups	Similar to `Terminal.AllowUsers` but matches groups instead of usernames.
`Terminal.DenyGroups`	pattern	None	Similar to `Terminal.DenyUsers` but matches groups instead of usernames
`UserConfigDirectory`	Directory	`SYS$LOGIN:`	Location of user SSH2 directories
`UserKnownHosts`	Y/N	Y	Respect user `[.ssh2]` known hosts keys
`UserSpecificConfig`	Pattern	None	Specifies a sub-configuration file for this server, based on user logging in.
`VerboseMode`	Y/N	N	Verbose mode

The keywords `MACS` and `CIPHERS` have discrete values, plus there are values that denote a grouping of 2 or more of the discrete values. Each of these values may be put in the configuration file `SSH2_DIR:SSHD2_CONFIG`.

MACs	discrete values: `hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160, none`
	group `ANYMAC` consists of: `hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160`
	group `ANY` consists of: `hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160, none`
	group `ANYSTD` consists of: `rhmac-sha1, hmac-md5, none`
	group `ANYSTDMAC` consists of: `hmac-sha1, hmac-md5`
Ciphers	discrete values:

	3des, aes, blowfish, aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, des-cbc@ssh.com, rc2-cbc@ssh.com, none
	group ANYSTDCIPHER consists of: aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc
	group ANY consists of: aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, des-cbc@ssh.com, rc2-cbc@ssh.com, none
	group ANYCIPHER aes128-cbc, 3des-cbc, twofish128-cbc, cast128-cbc, twofish-cbc, blowfish-cbc, aes192-cbc, aes256-cbc, twofish192-cbc, twofish256-cbc, arcfour, des-cbc@ssh.com, rc2-cbc@ssh.com
	group ANYSTD aes128-cbc, 3des-cbc, twofish128-cbc, cast128-cbc, twofish-cbc, blowfish-cbc, aes192-cbc, aes256-cbc, twofish192-cbc, twofish256-cbc, arcfour, none

A discrete value or a group identifier may be used with MACs and Ciphers. For example, in the configuration file, the following examples could be used:

```
Ciphers    ANYCIPHER

Ciphers    3des, aes128-cbc

MACs       ANYMAC

MACs       hmac-sha1
```

Aliases may be used for some standard ciphers:

Alias	Value
aes	aes128-cbc
3des	3des-cbc
blowfish	blowfish-cbc

The global server file (SSH_DIR:SSHD2_CONFIG) now can use the keyword HostSpecificConfig to allow the specification of a configuration file based on the client system. These lines are specified as:

```
HostSpecificConfig       hostname        subconfig-file
```

hostname will be used to match the client host, as specified under option *AllowHosts*. The file *subconfig-file* will then be read, and configuration data amended accordingly. The file is read before any actual protocol transactions begin, and you can specify most of the options allowed in the main configuration file. You can specify more than one sub-configuration file; in which case the patterns are matched and the files read in the order specified. Later defined values of configuration options will either override or amend the previous value, depending on which option it is. The effect of redefining an option is described in the documentation for that option. For example, setting `Ciphers` in the sub-configuration file will override the old value, but setting `AllowUsers` will amend the value.

The *subconfig-file* will be assumed by default to exist in the `SSH2_DIR` directory. However, this may be overridden by specifying a complete directory/file specification. For example:

```
HostSpecificConfig     foo.example.com       dka0:[sshconfigs]fooconfig.dat
HostSpecificConfig     lima.example.com      limaconfig.dat
```

In the first instance, an incoming connection from foo.example.com will use the sub-configuration file `dka0:[sshconfigs]fooconfig.dat`. In the second example, an incoming connection from `lima.beans.com` will use `ssh2_dir:limaconfig.dat`.

Unlike `ssh2_config`, the sub-configuration files may have configuration blocks, or stanzas, in them. They are used per-host. The sub-configuration heading is interpreted identically to what is described above (i.e, with `UserSpecificConfig`, the pattern is of the format *hostname*.)

> **Note:** If the sub-configuration file cannot be found or cannot be parsed successfully for any reason, access to the system will be denied for the system to which the sub-configuration file applies.

The global server file (`SSH2_DIR:SSHD2_CONFIG`) can use the keyword `UserSpecificConfig` to allow the specification of a configuration file based on the username of the user who's logging into the server. These keywords are of the form:

```
UserSpecificConfig  user[@host]   subconfig-file
```

The *user* and *host* fields will be used to match the username, as specified under the option `AllowUsers`. The file *subconfig-file* will then be read, and configuration data amended accordingly. The file is read before any actual protocol transactions begin, and you can specify most of the options allowed in the main configuration file. You can specify more than one sub-configuration file, in which case the patterns are matched and the files read in the order specified. Later defined values of configuration options will either override or amend the previous value, depending on which option it is. The effect of redefining an option is described in the documentation for that option. For example, setting

`Ciphers` in the sub-configuration file will override the old value, but setting `AllowUsers` will amend the value.

Unlike `sshd2_config`, the sub-configuration files may have configuration blocks, or stanzas, in them. They are used per user. The sub-configuration heading is interpreted identically to what is described above (i.e., with `UserSpecificConfig`, the pattern is of the format *`user`*`[@`*`host`*`]`.

The *`subconfig-file`* will be assumed by default to exist in the `SSH2_DIR` directory. However, this may be overridden by specifying a complete directory/file specification. For example:

```
UserSpecificConfig   dilbert               dka0:[sshconfigs]dilbert.dat

UserSpecificConfig   boss@lima.example.com   pointyhair.dat
```

In the first instance, an incoming connection for user dilbert will use the sub-configuration file `dka0:[sshconfigs]dilbert.dat`. In the second example, an incoming connection from user boss at system lima.example.com will use `ssh2:dir:pointyhair.dat`.

> **Note:** If the sub-configuration file cannot be found or cannot be parsed successfully for any reason, access to the system will be denied for the user to which the sub-configuration file applies.

`KEYBOARD-INTERACTIVE` mode is simply another form of password authentication. The user won't notice anything different with this mode.

With this option, you can have more fine-grained control over what the client is allowed to forward, and to where. Format for this option is:

```
[allow|deny]   [local|remote]    user-pat   forward-pat  [originator-pat]
```

`user-pat` will be used to match the client-user, as specified under the option `UserSpecificConfig`. *`forward-pat`* is a pattern of format *`host-id`*`[%`*`port`*`]`. This has different interpretations, depending on whether the ACL is specified for local or remote forwards. For local forwards, the *`host-id`* will match with the target host of the forwarding, as specified under the option `AllowHosts`. *`port`* will match with the target port. Also, if the client sent a host name, the IP address will be looked up from the DNS, which will be used to match the pattern. For remote forwardings, where the forward target is not known (the client handles that end of the connection); this will be used to match with the listen address specified by the user (and as such is not as usable as with local forwards). *`port`* will match the port the server is supposed to be listening to with this forward. With local forwards, *`originator-pat`* will match with the originator address that the client has reported. Remember, if you do not administer the client machine, users on that machine may use a modified copy of ssh that can be used to lie about the originator address. Also, with NATs (Network

Address Translation), the originator address will not be meaningful (it will probably be an internal network address). Therefore, you should not rely on the originator address with local forwards, unless you know exactly what you are doing. With remote forwards, originator-pat will match with the IP address of the host connecting to the forwarded port. This will be valid information, as it is the server that is checking that information.

If you specify any allow directives, all forwards in that class (local or remote) not specifically allowed will be denied (note that local and remote forwards are separate in this respect, e.g., if you have one "allow remote" definition, local forwards are still allowed, pending other restrictions). If a forward matches with both `allow` and `deny` directives, the forwarding will be denied. Also, if you have specified any of the options [`Allow.Deny`]`TcpForwardingForUsers.Groups`] or `AllowTcpForwarding`, and the forwarding for the user is disabled with those, an allow directive will not re-enable the forwarding for the user. Forwarding is enabled by default.

Mapping File Format

When certificates are used in user authentication, one or more mapping files determine whether the user can log to an account with a certificate. The mapping file must contain one or more lines in the following format:

```
account-id keyword arguments
```

Keyword must be one of the following: `Email`, `EmailRegex`, `Subject`, `SerialAndIssuer`, or `SubjectRegex`.

Arguments are different for each keyword. The following list describes each variation:

`Email`
arguments: an email address in standard format. If the certificate contains the email address as an alternate name, it is good for logging in as user *`account-id`*.

`Subject`
arguments: a subject name in DN notation (LDAP style). If the name matches the one in the certificate, the certificate is good for logging in as user *`account-id`*.

`SerialAndIssuer`
arguments: a number and an issuer name in DN notation (LDAP style), separated by whitespace. If the issuer name and serial number match those in the certificate, the certificate is good for logging in as user *`account-id`*.

EmailRegex
arguments: a regular expression (egrep syntax). If it matches an alternate name (of type `email-address`) in the certificate, the certificate is good lor logging in as user *`account-id`*. As a special feature, if *`account-id`* contains a string `%`*`subst`*`%`, it is replaced by the first parenthesized substring of the regular expression before comparing it with the account the user is trying to log into.

SubjectRegex
Works identically to `EmailRegex`, except it matches the regular expression to the canonical subject name in the received certificate.

Empty lines and lines beginning with `#` are ignored.

```
guest email guest@domain.org guest subject C=Fl,O=Company Ltd., CN-Guest
User guest SerialAndUser 123 C=Fl, O=Foo\Ltd., CN=Test CA %subst% EmailRegex
([a-z]+)@domain.\org
%subst% Subjectregex ^C=Fl,O=Company,CN=([a-z]+)$
```

The example `EmailRegex` permits in users with email addresses with domain domain.org and usernames that contain only letters, each user to the account that corresponds to the username part of the email address.

The example `SubjectRegex` lets in all users with fields `C=Fl` and `O=Company` in the subject name if their `CN` field contains only letters and is the account name they are trying to log into.

Note the `^` and `$` at the beginning and end of the regular expression; they are required to prevent the regular expression from matching less than the whole string (subject name).

Note also that all characters interpreted by the regular expression parser as special characters must be escaped with a backslash if they are a part of the subject name. This also means that the backslash in the `SerialAndIssuer` example would have to be escaped with another backslash if the same subject name was used in a `SubjectRegex` rule.

Starting the SSH Server for the First Time

Follow these instructions to configure the SSH server. If SSH isn't currently running, you must define the `PSCSSH` logicals by using:

```
$ @SYS$STARTUP:PSCSSH_STARTUP LOGICALS
```

1. Use the PSCSSH utility to enable the SSH2 server. It is recommended that the host keys be generated when executing the PSCSSH procedure by answering **Y** to the question "Do you want to generate the SSH2 host key now?" For more information, see Chapter 3 in this manual.

2. If you answer no to that question, use SSHKEYGEN /SSH2/HOST to generate an SSH2 key and to create the server key in the PSCSSH_SSH2_HOSTKEY_DIR directory if it has not previously been created as part of the PSCSSH configuration:

```
$ DEFINE PSCSSH_SSH2_HOSTKEY_DIR -
_$ PSCSSH_LOCAL_ROOT:[SSH2.HOSTKEYS]
$ PSCSSH SSHKEYGEN /SSH2/HOST
Generating 1024-bit dsa key pair
  8 .oOo.oOoo.oO Key generated.
1024-bit dsa, lillies@flower.example.com, Mon Mar 03 2022 09:19:47
Private key saved to PSCSSH_ssh2_hostkey_dir:hostkey.
Public key saved to PSCSSH_ssh2_hostkey_dir:hostkey.pub
```

3. Copy the template server configuration file to the ssh2_dir: directory, renaming it SSHD2_CONFIG.:

```
$ COPY PSCSSH_COMMON:SSHD2_CONFIG.TEMPLATE -
_$ SSH2_DIR:SSHD2_CONFIG.
```

4. Copy the template client configuration file to the ssh2_dir: directory, renaming it SSH2_CONFIG.:

```
$ COPY PSCSSH_COMMON:SSH2_CONFIG.TEMPLATE -
_$ SSH2_DIR:SSH2_CONFIG.
```

> **Note:** As delivered, the template files provide a reasonably secure SSH environment. However, Process Software recommends these files be examined and modified appropriately to reflect the security policies of your organization.

5. Start SSH. This creates the SSH server process and defines the SSH logical names.

```
$ @SYS$STARTUP:PSCSSH_STARTUP
$ SHOW PROCESS "SSHD Master"

3-MAR-2022 09:03:06.42  User: SYSTEM      Process ID:   00000057
Node: PANTHR      Process name: "SSHD Master"
```

```
Terminal:
User Identifier:    [SYSTEM]
Base priority:      4
Default file spec:  Not available
Number of Kthreads: 1
Devices allocated:  BG1:
                    BG2:

$ SHOW LOGICAL/SYSTEM *SSH*
    "PSCSSH" = "PSCSSH_ROOT:[COMMON]"
  "PSCSSH_COMMON" = "PSCSSH_ROOT:[COMMON]"
  "PSCSSH_EXE" = "PSCSSH_ROOT:[xxx_EXE]"
  "PSCSSH_HELP" = "PSCSSH_ROOT:[PSCSSH.HELP]"
  "PSCSSH_LOCAL" = "PSCSSH_ROOT:[node]"
  "PSCSSH_LOCAL_ROOT" = "_X86$DKA0:[PSCSSH.node.]"
  "PSCSSH_ROOT" = "_X86$DKA0:[PSCSSH.]"
  "PSCSSH_SSH2_HOSTKEY_DIR" = "PSCSSH_LOCAL_ROOT:[SSH2.HOSTKEYS]"
  "PSCSSH_SSH2_KNOWNHOSTS_DIR" = "PSCSSH_ROOT:[node.SSH2.KNOWNHOSTS]"
  "PSCSSH_SSH_ENABLE_SSH2_CONNECTIONS" = "1"
  "PSCSSH_SSH_LOG_MBX" = "MBA44:"
  "PSCSSH_SSH_PARAMETERS_0" = " /BITS=768/PORT=22"
  "PSCSSH_SSH_PARAMETERS_1" = " /LISTEN_ADDRESS=ANY/KEY_GEN_TIME=3600"
  "PSCSSH_SSH_PARAMETERS_2" = " /DEBUG=50"
  "PSCSSH_SSH_PARAMETERS_3" = " "
  "SSH2_DIR" = "PSCSSH_LOCAL_ROOT:[SSH2]"
  "SSHLEI" = "PSCSSH_EXE:SSHLEI"
  "SSHSHR" = "PSCSSH_EXE:SSHSHR"
  "SSH_DIR" = "PSCSSH_LOCAL_ROOT:[SSH]"
  "SSH_EXE" = "PSCSSH_EXE:"
  "SSH_FSCLM" = "PSCSSH_EXE:SSH_FSCLM.EXE"
  "SSH_LOG" = "PSCSSH_LOCAL_ROOT:[LOG]"
  "SSH_MAX_SESSIONS" = "1000"
  "SSH_TERM_MBX" = "MBA43:"
  "SSH_ZLIB" = "PSCSSH_EXE:SSH_ZLIB.EXE"
```

Modifying the SSH2 Configuration File

If you make a change to the SSH configuration file after you have enabled SSH, you must restart SSH for these changes to take effect.

```
$ SSHCTRL RESTART
```

> **Note:** When issuing the `RESTART` command for SSH, all active SSH server sessions are terminated. Active client sessions are not affected.

Connection and Login Process

To create a session, SSHD does the following:

1. SSHD_MASTER sees the connection attempt. It creates an SSHD process, passing the operating parameters to it. SSHD performs validation for the user.
2. Assuming the login is successful, SSHD creates a pseudo terminal for the user (an `_FTA`*`nn`*`:` device). This device is owned by the user attempting to log in.
3. SSHD creates an interactive process on the pseudo terminal, using the username, priority, and privileges of the user who is attempting to log in. If a command was specified, it is executed and the session is terminated.
4. SSH generates the file `SSHD.LOG` for each connection to the SSH server. Many connections result in many log files. Instead of purging the files on a regular basis, use the following DCL command to limit the number of versions:

```
$ SET FILE /VERSION_LIMIT=x SSH_LOG:SSHD.LOG
```

> **Note:** The value for `/VERSION_LIMIT` must not be smaller than the maximum number of simultaneous SSH sessions anticipated. If the value is smaller, SSH users may be prevented from establishing sessions with the server.

Files

`PSCSSH_SSH2_HOSTKEY_DIR:HOSTKEY`
Contains the private part of the host key. This file does not exist when PSCSSH is installed. The SSH server starts only with this file. This file must be created manually using the command:

```
$ PSCSSH SSHKEYGEN /SSH2 /HOST.
```

This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others.

To create a host key with a name that is different than what `SSHKEYGEN` creates, do one of the following:

- Generate with `SSHKEYGEN /SSH2/HOST` and simply rename the file(s).
- Generate without the `/HOST` switch and then name the file(s) whatever you want.

By default the logical name `SSH2_DIR` points to the `PSCSSH_LOCAL_ROOT:[SSH2]` directory.

PSCSSH_SSH2_HOSTKEY_DIR:HOSTKEY.PUB
Contains the public part of the host key. This file should be world-readable but writable only by SYSTEM. Its contents should match the private part. This file is not used for anything; it is only provided for the convenience of the user so its contents can be copied to known hosts files.

SSH2:SSH_RANDOM_SEED
SYS$LOGIN:[.SSH]RANDOM_SEED
Contains a seed for the random number generator. This file should only be accessible by SYSTEM.

SSH2_DIR:SSHD2_CONFIG
Contains configuration data for the SSHv2 server. This file should be writable by SYSTEM only, but it is recommended (though not necessary) that it be world-readable.

SYS$LOGIN:[.SSH2]AUTHORIZATION
This file contains information on how the server verifies the identity of a user.

SYS$LOGIN:[.SSH2.KNOWNHOSTS]*xxxxyyyy*.pub
These are the public host keys of hosts that a user wants to log in from using host-based authentication (equivalent to SSH1's `RhostsRSAAuthentication`). Also, a user must set up their individual `.SHOSTS` or `.RHOSTS` file. If the username is the same in both hosts, it is adequate to put the public host key in `SSH2_DIR:KNOWNHOSTS` and add the host's name to the systemwide `SHOSTS.EQUIV` or `RHOSTS.EQUIV` file.

xxxx is the hostname (FQDN) and *yyyy* denotes the public key algorithm of the key (`ssh-dss` or `ssh-rsa`).

For example flower.example.com's host key algorithm is `ssh-dss`. The host key would then be `flower_example_com_ssh-dss.pub` in the `[.SSH2.KNOWNHOSTS]` directory.

SSH2 AUTHORIZATION File Format

The authorization file contains information on how the server verifies the identity of a user. This file has the same general syntax as the SSH2 configuration files. The following keywords may be used:

Keyword	Description
`KEY`	The filename of a public key in the `[.SSH2]` directory in the user's `SYS$LOGIN` directory. This key is used for identification when contacting the host. If there are multiple `KEY` lines, all are acceptable for login.

`COMMAND`	This keyword, if used, must follow the `KEY` keyword above. This is used to specify a "forced command" that executes on the server side instead of anything else when the user is authenticated. This option might be useful for restricting certain public keys to perform certain operations.

SSH2 Logicals

These logicals are used with the SSH server in the system logical name table.

SSH_DIR
Points to the directory where the master server log file is kept. Normally, this is PSCSSH_LOCAL_ROOT:[SSH]. It is defined in START_SSH.COM.

SSH_EXE
Points to the directory where SSH executables are kept. Normally, this is PSCSSH_EXE:. It is defined in START_SSH.COM.

SSH_LOG
Points to the directory where the log files are kept. Normally, this is PSCSSH_LOCAL_ROOT:[LOG]. It is defined in START_SSH.COM.

PSCSSH_LOG_MBX
Points to the OpenVMS mailbox used to log connection accept and reject messages. This must not be modified by the user.

PSCSSH_SSH_ACC_REJ_LOG_FILE
If the user has set a log file to log connection accept and reject messages, this logical will be defined and will provide the name of the log file. This logical is set by using the @PSCSSH:PSCSSH CONFIGURE command or by editing PSCSSH_LOCAL:SSH_CONFIGURE.COM.

PSCSSH_SSH_LOG_ACCEPTS
When set, causes the server to log successful connection requests as either an OPCOM message or a line in a log file. Specified by the line "SSH_LOG_ACCEPTS = 1" in PSCSSH_LOCAL:SSH_CONFIGURE.COM. Note that the server uses the AllowHosts and DenyHosts keywords in the SSH server configuration file. Also, a successful connection request doesn't equate to a successful authentication request. This logical should not be modified directly by the user.

PSCSSH_SSH_LOG_REJECTS
When set, causes the server to log rejected connection requests as either an OPCOM message or a line in a log file. Specified by the line "SSH_LOG_REJEPTS = 1" in PSCSSH_LOCAL:SSH_CONFIGURE.COM. Note that the server uses the AllowHosts and

DenyHosts keywords in the SSH server configuration file. This logical should not be modified directly by the user.

PSCSSH_SSH_MAX_SESSIONS
Set this to the maximum number of concurrent SSH sessions you want to allow on the server system. If PSCSSH_SSH_MAX_SESSIONS is not defined, the default is 1000. Setting PSCSSH_SSH_MAX_SESSIONS to zero (0) will cause an error. The value must be between 1 and 1000. The suggested place to set this is in START_SSH.COM. SSH must be restarted to use the new value if it is changed.

SSH_TERM_MBX
Mailbox used by SSHD_MASTER to receive termination messages from SSHD daemon processes. Do not change this logical name. This is created by the SSHD_MASTER process.

PSCSSH_SSH_KEYGEN_MIN_PW_LEN
Defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero.

PSCSSH_SSH_PARAMETERS_*n*
These values are set by PSCSSH and must not be modified by the user.

PSCSSH_SSH_USE_SYSGEN_LGI
If defined, causes SSHD to use the VMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding anything specified in the command line or the configuration file.

PSCSSH_SSH_ENABLE_SSH2_CONNECTIONS
Enables SSHD Master to accept SSH2 sessions.

PSCSSH_SSH2_HOSTKEY_DIR
Directory containing the host keys for the SSH2 server. Normally set to PSCSSH_LOCAL_ROOT:[SSH2.HOSTKEYS]

PSCSSH_SSH2_KNOWNHOSTS_DIR
Directory containing the public keys for known systems. Normally set to PSCSSH_LOCAL_ROOT:[SSH2.KNOWNHOSTS].

SSH2_DIR
Contains all SSH V2-specific files, such as configuration files. Normally set to `PSCSSH_LOCAL_ROOT:[SSH2]`

SSH daemon Files

These files are used by or created by SSH when you log into a daemon. These files are not to be altered in any way.

SSH_LOG:SSHD.LOG
This log file is created by each SSHD daemon.

SSHD_MASTER.LOG
This log file is created by SSHD_MASTER.

SSH_START.COM
This file is used to start SSH.

6. Accessing Remote Systems with the Secure Shell (SSH) Utilities

PSCSSH provides the client software for allowing secure interactive connections to other computers, replacing TELNET.

The following topics describe how to configure, maintain, and use the following PSCSSH client and utilities:

- Secure Shell Client (remote login program)
- SSHKEYGEN
- SSHAGENT (authentication agent)
- SSHADD
- CERTTOOL
- CERTVIEW
- CMPCLIENT
- Public key subsystem

SSH Protocol Support

The SSH client software supports both the SSH1 and SSH2 protocols. SSH1 and SSH2 are different, and incompatible protocols. While SSH2 is generally regarded to be more secure than SSH1, both protocols are offered by PSCSSH, and although they are incompatible, they may exist simultaneously on server systems, including PSCSSH servers. The SSH client identifies the protocol(s) offered by any given server. If both SSH2 and SSH1 protocols are offered, the client will always use SSH2. Otherwise, the client will use the correct protocol based on the server's capability.

The cryptographic library used by PSCSSH SSH2 is FIPS 140 Level 2 compliant, as determined by the Computer Security Division of the National Institute of Science and Technology (NIST).

Secure Shell Client

```
$ SSH hostname[#port] [qualifiers] [command]
$ SSH "user@hostname[#port]" [qualifiers] [command]
```

SSH (Secure Shell) is a program for logging into and executing commands on a remote system. It replaces insecure protocols like rlogin, rsh, and telnet, and provides secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can be forwarded over the secure channel. SSH connects and logs into the specified hostname.

Qualifier	Description
/ALLOW_REMOTE_CONNECT	Allow remote hosts to connect local port forwarding ports. The default is only localhost; may connect to locally bound ports.
/CIPHER=(*cipher-1,…,cipher-n*)	Select encryption algorithm(s).
/COMPRESS	Enable compression.
/CONFIG_FILE=*file*	Read an alternative client config file.
/DEBUG=*level*	Set debug level.
/ESCAPE_CHARACTER=*char*	Set escape character; none = disable (default: ~).
/HELP	Display help text.
/IDENTITY_FILE=*file*	Identity file for public key authentication.
/IPV4	Use IPv4 protocol to connect.
/IPV6	Use IPv6 protocol to connect.
/LOCAL_FORWARD=([*protocol*/]*listenport:host:port,…*)	Causes the given port on the local (client) host to be forwarded to the given host and port on the remote side. The system to which SSH connects acts as the intermediary between the two endpoint systems. Port forwardings can be specified in the configuration file. Only SYSTEM can forward privileged ports. See the *Port Forwarding* section for more details.
/LOG_FILE=*logfilename*	Log all terminal activity to the specified log file. Defaults to SYS$DISK:[]SSH.LOG if *logfilename* is not specified.
/MAC=*(mac-1,…,mac-n)*	Select MAC algorithm(s).

/NO_AGENT_FORWARDING	Disable authentication agent forwarding.
/NO_X11_FORWARDING	Disable X11 connection forwarding.
/OPTION=*(option-1,…option-n)*	Gives options in the format used in the configuration file. This is useful for specifying options for which there is no separate command line flag. The options have the same format as a line in the configuration file and are processed prior to any keywords in the configuration file. For example: /OPTION=(CompressionLevel=6)
/PORT=*port*	Connect to this port on server system. Server must be listening on the same port.
/QUIET	Quiet Mode. Causes all warning and diagnostic messages to be suppressed. Only fatal errors display.
/REMOTE_FORWARD= ([*protocol*/]*listenport:host:port,…*)	Forward remote port to local address. These cause SSH to listen for connections on a port and forward them to the other side by connecting to host port.
/USE_NONPRIV_PORT	Use a non-privileged (>1023) source port.
/USER=*user*	Log in to the server system using this username.
/VERBOSE	Display verbose debugging messages. Equal to /DEBUG=2
/VERSION	Display version number of the client.

Initial Server System Authentication

When an initial connection is made from the client system to the server system, a preliminary authentication of the server is made by the client. To accomplish this, the server system sends its public key to the client system.

SSH maintains a directory containing the public keys for all hosts to which it has successfully connected. For each user, this is the [.SSH2.HOSTKEYS] directory under the individual SYS$LOGIN directory. In addition, a system-wide directory of known public keys exists in the system directory pointed to by the logical name PSCSSH_SSH2_HOSTKEY_DIR, and this may be populated by the system manager. Both directories are searched as needed when establishing a connection between

systems. Any new host public keys are added to the user's HOSTKEYS directory. If a host's identification 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 that could be used to circumvent the encryption. The SSH configuration option StrictHostKeyChecking can be used to prevent logins to a system whose host key is not known or has changed.

Host-Based Authentication

Host-based authentication relies on two things: the existence of the user's system and username in either SSH_DIR:HOSTS.EQUIV or in the individual user's SYS$LOGIN:.RHOSTS or SYS$LOGIN:.SHOSTS file; and the server system having prior knowledge of the client system's public host key.

When a user logs in to SSH2:

1. The server checks the SSH_DIR:HOSTS.EQUIV file, and the user's SYS$LOGIN:.RHOSTS and SYS$LOGIN:.SHOSTS files for a match for both the system and username. Wildcards are not permitted.
2. The server checks to see if it knows of the client's public host key (SSH2_DIR:HOSTKEY.PUB on VMS client systems) in either the user's SYS$LOGIN:[SSH2.KNOWNHOSTS] directory or in the system-wide directory pointed to by the PSCSSH_SSH2_KNOWNHOSTS_DIR logical name. The key file is named *FQDN_algorithm*.PUB. For example, if the client system is foo.example.com and its key uses the DSS algorithm, the file that would contain its key on the server would be FOO_EXAMPLE_COM_SSH-DSS.PUB. This key file must exist on the server system before attempting host-based authentication.
3. If the key file is found by the server, the client sends its digitally signed public host key to the server. The server will check the signature for validity.

When a user logs in to SSH1, host-based authentication alone is 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, SSH_DIR:HOSTS.EQUIV, or SSH_DIR:SHOSTS.EQUIV file, and if the client's host key can be verified (see SYS$LOGIN:[.SSH]KNOWN_HOSTS and SSH_DIR:SSH_KNOWN_HOSTS), only then is login permitted. This authentication method closes security holes due to IP spoofing, DNS spoofing, and routing spoofing.

> **Caution!** SSH_DIR:HOSTS.EQUIV,.RHOSTS, and the rlogin/rshell protocol are inherently insecure and should be disabled if security is desired.

Public Key Authentication

The SSH client supports DSA-based authentication for SSH2 sessions, and RSA-based authentication for SSH1 sessions. 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.

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 (`SYS$LOGIN:[.SSH]AUTHORIZED_KEYS` lists the public keys permitted for log in), and only the user knows the private key.

When the user logs in:

1. The SSH client program tells the server the key pair it would like to use for authentication.
2. The server checks if this key pair is permitted.
3. If it is permitted, the server sends the SSH client 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.
4. 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.
5. SSH implements the RSA authentication protocol automatically.

The key identity files are created with `SSHKEYGEN`. To create the RSA key pair files: run `SSHKEYGEN` to create the RSA key pair: `IDENTITY.` and `IDENTITY.PUB`. Both of these files are stored in the user's `SYS$LOGIN:[.SSH]` directory. `IDENTITY.` is the private key; `IDENTITY.PUB` is the public key.

Once you have created your identity files:

1. Transfer the `IDENTITY.PUB` file to the remote machine.
2. Update the `AUTHORIZED_KEYS` file on the remote machine by appending the contents of the public key file to the `SYS$LOGIN:[.SSH]AUTHORIZED_KEYS` file on the remote host. The format of the `AUTHORIZED_KEYS` file requires that each entry consists of a single long line.

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 *Public Key Authentication* for more information.

When the user logs in:

1. The client reads possible keys to be used for authentication from its `IDENTIFICATION` file.
2. Note that this file does not contain the actual keys; rather, it contains the name of the key files.
3. The client sends to the server its list of keys.
4. The server compares each key that it received to see if it can match this key with one of those specified in the `AUTHORIZATION` file.
5. The server tells the client the key that was accepted. The client then signs the key with a digital signature that only the server with the proper key could verify, and sends the signature to the server.
6. The server verifies the signature.

Password Authentication

The password is sent to the remote host for checking. The password cannot be seen on the network because all communications are encrypted. When the server accepts the user's identity it either executes the given command or logs into the system and gives the user a normal shell on the remote system. All communication with the remote command or shell will be encrypted automatically.

Using Public Key Authentication with SSH

When a parameter such as a username or hostname is quoted, it’s always passed verbatim to the other side. When it’s not quoted, it’s lowercased. The username entered is used when constructing the digital signature for a key.

On the host side, the uppercase username will be used, and on the server side, the lowercased username (the default on the server since VMS isn’t case-sensitive) will be used to generate the digital signature of the public key that’s being used, as shown in the following examples:

```
$ PSCSSH SSH2 "USER@HOSTNAME" command
```

USER is the username that was specified in all uppercase letters. Public key authentication fails.

```
$ PSCSSH SSH2 "user@HOSTNAME" command
```

user is the username that was specified in all lowercase letters. Public key authentication is successful.

Break-in and Intrusion Detection

Care must be exercised when configuring the client to minimize problems due to intrusion records created by OpenVMS security auditing. The SSH user should consult the system manager to determine the authentication methods offered by the SSH server. Examples of such authentication methods include `HostBased`, `PublicKey`, and `Password`. The client should be configured to not attempt any authentication method that is not offered by the server.

If a client attempts authentication methods not offered by the server, the OpenVMS security auditing system may log several intrusion records for each attempt to create a session to that server. The result being that the user could be locked out and prevented from accessing the server system without intervention from the server's system manager.

Session Termination

The user can disconnect by typing "`~.`". All forwarded connections can be listed with "`~#`". All available escapes can be listed with "`~?`". A single tilde character can be sent as "`~~`" (or by following the tilde with a character other than those described above). The escape character must always follow a carriage return to be interpreted as special. The escape character "`~`" can be changed in configuration files or on the command line.

The session terminates when the command or shell on the remote system exits, or when the user logs out of an interactive session, 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.

X11 Forwarding

With X11 in use, the connection to the X11 display forwards to the remote side any X11 programs started from the interactive session (or command) through the encrypted channel. Also, the connection to the real X server is made from the local system. The user should not set `DECW$DISPLAY` manually. Forwarding of X11 connections can be configured on the command line or in configuration files.

The `DECW$DISPLAY` value set by SSH points to the server system with a display number greater than zero. This is normal and happens because SSH creates a "proxy" X server on the server system for forwarding the connections over the encrypted channel.

SSH sets up "fake" Xauthority data on the OpenVMS server, as OpenVMS does not support Xauthority currently. It generates a random authorization cookie, stores it in Xauthority on the server, and verifies 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 system (and no cookies are sent in plain text).

Configuring the SSH Client

The SSH client uses only SSH2 configuration keywords. There are no SSH1-specific configuration keywords for the SSH client.

The SSH client obtains configuration data from the following sources (in this order):

1. Command line options. See the below table for details.
2. User's configuration file (`SYS$LOGIN [.SSH2]SSH2_CONFIG`). See the following table for details.
3. System-wide configuration file (`SSH2_DIR:SSH2_CONFIG`). See the following table for details.

For each parameter, the first obtained value is used. The configuration files contain sections bracketed by "Host" specifications. That section applies only 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.

Keyword	Value	Default	Description
`AllowedAuthentications`	List	All methods except for `HostBased`	Permitted techniques, listed in desired order of attempt. These can be the following: `keyboard-interactive`, `password`, `publickey`, `kerberos-1@ssh.com`, `kerberos-tgt-1@ssh.com`, `kerberos-2@ssh.com`, `kerberos-tgt-2@ssh.com`, and `hostbased`. Each specifies an authentication method. The authentication methods are tried in the order in which they are specified with this configuration parameter.
`AuthenticationSuccessMsg`	Y/N	Y	Print message on successful authentication
`AuthorizationFile`	Filename	Authorization	Authorization file for `publickey` authentication.

			See below for more information on the contents of this file.
BatchMode	Y/N	N	Don’t prompt for any input during session
Ciphers	Cipher list	None	Supported encryption ciphers
ClearAllForwardings	Y/N	N	Ignore any specified forwardings
Compression	Y/N	N	Enable data compression
DebugLogFile	Filename	None	Specify the file to hold debug information. If used with the QuietMode keyword turned on as well, only the first part of the log information will be written to SYS$ERROR, until the DebugLogFile keyword is parsed. If QuietMode is not used, all debug output will go to both SYS$ERROR and the log file.
DefaultDomain	Domain		Specify domain name
EscapeChar	Character	“~”	Set escape character (^ = CTRL key)
ForwardAgent	Y/N	Y	Enable agent forwarding
ForwardX11	Y/N	Y	Enable X11 forwarding
GatewayPorts	Y/N	N	Allow connection to locally forwarded ports
Host	Pattern		Begin the per-host configuration section for the specified host
HostCA	Certificate	None	Specifies the CA certificate (in binary or PEM [base64] format) to be used when authenticating remote hosts. The certificate received from the host must be issued by the specified CA and must contain

			a correct alternate name of type DNS (FQDN). If the remote host's name is not fully qualified, the domain specified by configuration option `DefaultDomain` is not fully qualified, the domain specified by configuration option `DefaultDomain` is appended to it before comparing it to certificate alternate names. If no CA certificates are specified in the configuration file, the protocol tries to do key exchange with ordinary public keys. Otherwise, certificates are preferred. Multiple CAs are permitted.
`HostCANoCRLs`	Certificate	None	Similar to `HostCA` but disables CRL checking for the given ca-certificate.
`IdentityFile`	Filename	Identification	Name of identification file for public key authentication
`KeepAlive`	Y/N	Y	Send keepalives
`LdapServers`	ServerURL	None	Specified as `ldap://` *`server.domainname`*`:389` CRLs are automatically retrieved from the CRL distribution point defined in the certificate to be checked if the point exists. Otherwise, the comma-separated server list given by option `LdapServers` is used. If intermediate CA certificates are needed in certificate validity checking, this option must be used or

			retrieving the certificates will fail.
`LocalForward`	Port, Socket		Local port forwarding
`Macs`	Algorithm	None	Select MAC (Message Authentication Code) algorithm
`NoDelay`	Y/N	N	Disable Nagle (TCP_NODELAY)
`NumberOfPasswordPrompts`	Number	3	Number of times the user is prompted for a password before the connection is dropped
`PasswordPrompt`	String	"%U's password:"	Password prompt. The following substitutions may be made within the prompt string: `%U` = insert user's username `%H` = insert user's system name
`Port`	Port	22	Server port number
`QuietMode`	Y/N	Y	Quiet mode - only fatal errors are displayed
`RandomSeedFile`	Filename	Random_seed	Random seed file
`RekeyIntervalSeconds`	Seconds	3600	Number of seconds between doing key exchanges during a session. 0 = disable
`RemoteForward`	Port, Socket		Remote port forwarding
`SendNOOPPackets`	Y/N	N	Send NOOP packets through the connection. Used typically to prevent a firewall from closing an interactive session
`StrictHostKeyChecking`	Y/N/Ask	Y	Behavior on host key mismatch

`TryEmptyPassword`	Y/N	N	Attempt an empty password first when doing password authentication. **Note:** Doing so may result in an extra intrusion being logged.
`User`	Username		Remote username
`VerboseMode`	Y/N	N	Verbose mode

The user may specify default configuration options, called “stanzas”, for different destination systems. The format of this within the configuration file is:

```
hostname:
  keyword        value
  keyword        value

hostname2:
   keyword       value
   keyword       value
```

For example:

```
petunia:
  port   17300
  user    jdoe
  host   petunia.example.com

rose:
  port   16003
  user    alice
  host   rose.example.com
  allowedauthentications password

*.beans.com:
  user           limabean
  keepalive      no
  ciphers        3des,twofish
```

In the preceding example:

- When a user types **`SSH PETUNIA`**, the client will connect to port 17300 on petunia.example.com, and will use the default username of `jdoe`.
- When a user types **`SSH ROSE`**, the client will connect to port 16003 on host rose.example.com, and will use the default username of `alice`, and only allow password authentication.

- When a user types **SSH *anything*.BEANS.COM**, the client will use the default username of `limabean`, will not send keepalives, and will only allow 3DES or TWOFISH encryption.

The user may override defaults specified in configurations. Options that are specified on the command line override any like options in the configuration file. For example, if the user wants to use a username of `bob` when connecting to host rose instead of the default username of `alice`, this would be specified as:

```
$ SSH /USER=BOB ROSE
```

Authorization File Options

The authorization file has the same general syntax as the configuration files. The following keywords may be used.

Key
This is followed by the filename of a public key in the `[.SSH2]` directory file that is used for identification when contacting the host. If there is more than one key, they are all acceptable for login.

Options
This keyword, if used, must follow the `Key` keyword above. The various options are specified as a comma-separated list. See below for documentation of the options.

Command
Deprecated - use `Options` instead.

Available Options

allow-from
deny-from
Specifies that in addition to public-key authentication, the canonical name of the remote host must match the pattern(s). These parameters follow the logic of `AllowHosts/DenyHosts` described in detail in `sshd2_config`. Specify one pattern per keyword, and multiple keywords can be used.

command="*command*"
This is used to specify a "forced command" that will be executed on the server side instead of anything else when the user is authenticated. This option might be useful for restricting certain public 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 forwarding unless they are explicitly prohibited.

`idle-timeout=`*`time`*
Sets idle timeout limit to time in seconds (`s` or nothing after number), in minutes (`m`), in hours (`h`), in days (`d`), or in weeks (`w`). If the connections have been idle (all channels) for the specified period of time, the connection is closed down.

`no-port-forwarding`
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, for example, in connection with the command option.

`no-x11-forwarding`
Forbids X11 forwarding when this key is used for authentication. An X11 forward request by the client will return an error.

Authentication Configuration Examples

Hostbased Authentication Example

The following is an example of how to set up the SSH client and SSH2 server for `Hostbased` authentication:

```
$!
$! First, generate the host key - ONLY if it doesn't exist!
$!
$ PSCSSH SSHkeygen /ssh2 /host
Generating 1024-bit dsa key pair
4 oOo.oOo.oOo

Key generated.
1024-bit dsa, myname@myclient.foo.com, Thu MAR 04 2022 13:43:54
Private key saved to PSCSSH_ssh2_hostkey_dir:hostkey.
Public key saved to PSCSSH_ssh2_hostkey_dir:hostkey.pub

$ directory PSCSSH_ssh2_hostkey_dir:hostkey*.*
Directory PSCSSH_LOCAL_ROOT:[SSH2.HOSTKEYS]
HOSTKEY_ECDSA.;1          HOSTKEY_ECDSA.PUB;1
```

```
Total of 2 files $!
$! Copy the client system public key to the user directory on the server
$!
$! DECnet must be running before you execute the following commands:
$!
$ copy PSCSSH_ssh2_hostkey_dir:hostkey.pub -
_$ myserv"myname myuser"::[.ssh2.knownhosts]myclient_foo_com.pub
$!
MYSERV_$
MYSERV_$ logout
  MYNAME      logged out at  1-MAR-2022 13:46:58.91 %REM-S-END, control
returned to node MYCLIENT::
```

Publickey Authentication Example

The following is an example of how to set up the SSH client and SSH2 server for `Publickey` authentication:

```
$!
$! First, generate a key tuple $!
$ PSCSSH SSHkeygen /ssh2
Generating 1024-bit dsa key pair
1 oOo.oOo.oOo.

Key generated.
1024-bit dsa, myname@myclient.foo.com, Thu Mar 04 2022 14:06:10
Passphrase :
Again      :

Private key saved to DISK$USERDISK:[MYNAME.SSH2]id_dsa_1024_a.
Public key saved to DISK$USERDISK:[MYNAME.SSH2]id_dsa_1024_a.pub
$ directory [.ssh2]id*.*/since = TODAY

Directory DKA0:[MYNAME.SSH2]

ID_DSA_1024_A.;1    ID_DSA_1024_A.PUB;1

Total of 2 files.
$!
$! Now create the IDENTIFICATION. file.  This contains the name of
$! all the keys you wish to use for public-key authentication.
$!
$ set default [.ssh2]
$ copy tt: identification. idkey id_dsa_1024_a
  ^Z
$!
$! Copy the key to the user's [.ssh2] directory on the server system
$!
$ copy id_dsa_1024_a.pub myserv"myname mypass"::[.ssh2]
$!
```

```
$! Now log into the server system and create the AUTHORIZATION file
$!
$ set host myserv

      Welcome to OpenVMS (TM) VAX Operating System, Version V7.3

Username: myname
Password:

      Welcome to OpenVMS VAX V7.3

    Last interactive login on Tuesday,  2-MAR-2022 13:46
    Last non-interactive login on Tuesday,  2-MAR-2022 13:47

$ set default [.ssh2]
$ directory [.ssh2]id*.*

Directory DKA0:[MYNAME.SSH2]

ID_DSA_1024_A.PUB;1

Total of 1 file.
$ copy tt: authorization.key id_dsa_1024_a.pub
  ^Z
$ logout
  MYNAME      logged out at 2-MAR-2004 14:10:26.16 %REM-S-END, control
returned to node MYCLIENT::
```

```
$ ! An example of the procedure of setting up SSH to enable
$ ! RSA-based authentication.
$ ! Using SSH client node to connect to an SSH server node.
$ !
$ ! On the client node
$ !
$ PSCSSH SSHKEYGEN /SSH1
Initializing random number generator...
Generating p:  ..............................++ (distance 662)
Generating q:  ................++ (distance 370)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (DISK$SYS_LOGIN:[MYNAME.ssh]identity.):
Enter passphrase:
Enter the same passphrase again:
Your identification has been saved in DISK$SYS_LOGIN:[MYNAME.ssh]identity..
Your public key is:
1024 33 13428.........29361 MYNAME@long.hair.com
Your public key has been saved in DISK$SYS_LOGIN:[MYNAME.ssh]identity.pub
$ !
$ ! A TCP/IP stack must be loaded on the remote system.
```

```
$ !
$  FTP DAISY /USER=MYNAME/PASSWORD=DEMONSOFSTUPIDITY -
_$ PUT DISK$SYS_LOGIN:[MYNAME.ssh]identity.PUB
_$ DISK$SYS_LOGIN:[MYNAME.ssh]identity.PUB
long.hair.com MultiNet FTP user process V5.4(119)
Connection opened (Assuming 8-bit connections)
<daisy.hair.com MultiNet FTP Server Process V5.4(16) at Thu 4-Mar-2022
3:20PM-EDT
[Attempting to log in as myname]
<User MYNAME logged into DISK$SYS_LOGIN:[MYNAME] at Thu 4-MAR-2022
3:21PMEDT, job 20e00297.
<VMS Store of DISK$SYS_LOGIN:[MYNAME.SSH]IDENTITY.PUB; started.
<Transfer completed.  395 (8) bytes transferred.
<QUIT command received. Goodbye.
$
$ TELNET DAISY
Trying... Connected to DAISY.HAIR.COM.

        Authorized Users Only (TM) VAX Operating System, Version V7.1

Username: MYNAME
Password:

     Welcome to OpenVMS (TM) VAX Operating System, Version V7.1 on node
DAISY
    Last interactive login on Thursday,  4-MAR-2022 08:07
    Last non-interactive login on Thursday,  6-MAR-2022 15:21
         Logged into DAISY at  4-MAR-2022 15:22:43.68 $ !
$ ! For the first entry into the AUTHORIZED_KEYS file copy
$ ! (or rename) the file [.SSH]IDENTITY.PUB to [.SSH]AUTHORIZED_KEYS.
$ !
$ COPY [.SSH]IDENTITY.PUB [.SSH]AUTHORIZED_KEYS.
$
$ ! FOR SUBSEQUENT ENTRIES use the APPEND command
$ !
$ APPEND [.SSH]IDENTITY.PUB [.SSH]AUTHORIZED_KEYS.
$
$ ! A sanity check of the file protections shows
$ !
$ DIRECTORY/PROTECTION [.SSH]*.*

Directory DISK$SYS_LOGIN:[MYNAME.SSH]

AUTHORIZED_KEYS.;1   (RWE,RWED,RE,E)
IDENTITY.;1          (RWD,RWD,,)
IDENTITY.PUB;1       (RWE,RWED,RE,E)
KNOWN_HOSTS.;1       (RWD,RWD,,)
RANDOM_SEED.;1       (RWD,RWD,,)

Total of 5 files.
$ !
$ DIRECTORY/PROTECTION SSH.DIR
```

```
Directory DISK$SYS_LOGIN:[MYNAME]

SSH.DIR;1             (RWD,RWD,,)
Total of 1 file.
```

Client setup: Copy the private key and certificate (`.crt`) into the user's `[.ssh2]` directory, and edit the `[.ssh2]identification` file, adding the entry "`certkey` *`private_key_name`*".

```
$ dir [.ssh2]
Directory DKA0:[JDOE.SSH2]
AUTHORIZATION.;13 IDENTIFICATION.;1 MYCERT.;1 MYCERT1.CRT.;2
Total of 4 files.

$ type [.ssh2]identification.
certkey mycert1
$
```

Server setup:

1. Copy the CA certificate into your `SSH2_DIR:` directory.

2. Add the following entries in `SSH2_DIR:SSHD2_CONFIG`:

```
Pki SSH2_DIR:CAcertname
Mapfile SSH2_DIR:CAcertname.map
```

The `Pki` keyword begins an authority block for a given CA certificate. There might be more than one CA certificate along with its own mapping file.

The `Mapfile` keyword specifies the location of the certificate to username mapping file.

In addition, for testing, you might use `PkiDisableCRLs yes` to disable CRL checking for the given authorization block.

3. Create the mapping file `SSH2_DIR:`*`CAcertname`*`.map`. The mapping file consists of rows of the following format:

```
userid mappingrule mapdata
```

`userid` is the user ID that's allowed to login for the given cert (there might be multiple *`userid`* values for a given certificate).

`mappingrule` is one of `subject`, `email`, `serialandissuer` or `emailregex`.

- `subject` means that the following *`mapdata`* is matched against the subject of the certificate.
- `email` is the e-mail alternative subject extension
- `emailregex` allows the use of regular expressions - e.g., `%`*`subst`*`%` `emailregex ([a-z}|+)@example\.com` would allow any trusted certificate having an e-mail alternative name of *`username`*`@example.com` to login with `userid` *`username`*)
- `SerialAndIssuer` is the serial number and DN of the issuer separated by whitespace.

DNs are used in reverse LDAP order (e.g., `c=US,o=Foobar,cn=Jane Doe`).

Server setup:

1. Create a certificate for the server. The host certificate must contain the fully-qualified domain name (FQDN) as the DNS alternative name.

2. Copy the private key and certificate into the `PSCSSH_SSH2_HOSTKEY_DIR` directory.

3. Add the following entries into the `ssh2_dir:sshd2_config` file:

```
HostKeyFile PSCSSH_ssh_hostkey_dir:hostcert
HostCertificateFile PSCSSH_ssh_hostkey_dir:hostcert.crt
```

Client setup:

1. Copy the CA certificate into the `PSCSSH_SSH2_HOSTKEY_DIR` directory.

2. Add the following entries into the `ssh2_dir:ssh2_config` file:

```
HostCA PSCSSH_ssh_hostkey_dir:CAcert.crt
DefaultDomain client_FQDN
```

> **Note:** For testing purposes, you can use `HostCANoCRLs` instead of `HostCA` to disable CRL checking.

Port Forwarding

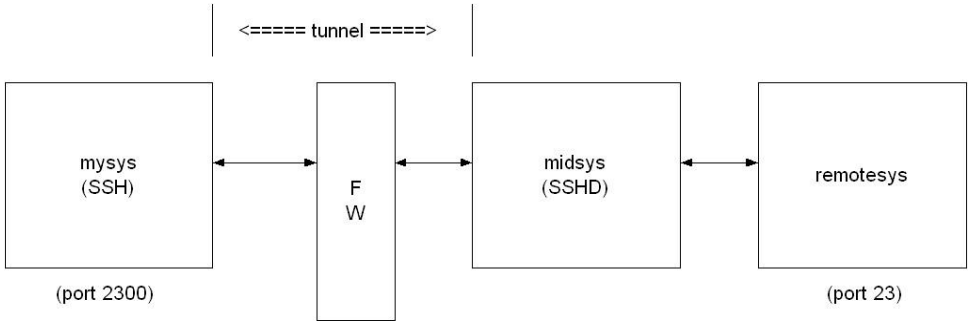
Port forwarding is a mechanism whereby programs that use known TCP/IP ports can have encrypted data forwarded over unsecure connections. This is also known as "tunneling".

If the user is using an authentication agent, the connection to the agent is forwarded automatically to the remote side unless disabled on the command line or in a configuration file. Forwarding of arbitrary TCP/IP connections over the secure channel can be specified either on the command line or in a configuration file.

> **Note:** Forwarded ports (tunnels) exist only as long as the SSH session that established them exists; if the SSH session goes away, so do the forwardings.

`/LOCAL_FORWARD=(`*`localport`*`:`*`remotehost`*`:`*`remoteport`*`)`
This causes *`localport`* on the system the client is running on to be forwarded to *`remotehost`*`:`*`remoteport`*. The system to which SSH2 connects acts as the intermediary between the two endpoint systems.

For example: Use port forwarding to allow a system (`midsys`) to encrypt and forward TELNET sessions between itself (`mysys`) that's outside a corporate firewall to a system (`remotesys`) that is inside a corporate firewall. Note that the use of port 2300 in the examples is arbitrary.



From the DCL prompt on `mysys`:

```
$ SSH midsys /local_forward=(2300:remotesys:23)
```

With the SSH session to `midsys` now active, type in another window on `mysys`:

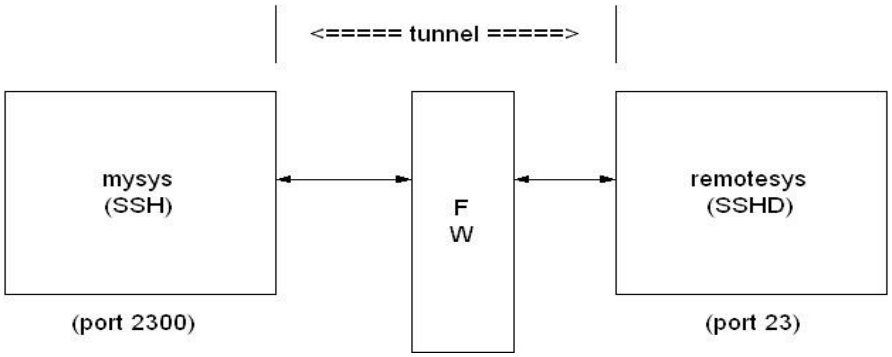
```
$ telnet localhost /port=2300
```

> **Note:** The SSH session must remain active for port forwarding activity.

This causes a connection to `mysys:2300`. The SSH2 client has bound to this port and will see the connection request. SSH sends an "open channel" request to `midsys`, telling it there's a connect request for port 23 on `remotesys`. `Midsys` will connect to `remotesys:23` and send back the port information to `mysys`. `Mysys` completes the connection request, and the TELNET session between `mysys` and `remotesys` is now in place, using the tunnel just created through the firewall between `mysys` and `midsys`.

All traffic between `mysys` and `midsys` (through the firewall) is encrypted/decrypted by SSH on `mysys` and SSHD on `midsys`, and hence, is safe. TELNET does not know this, of course, and does not care.

Note that ports can also be forwarded from a local host to the remote host that's running SSHD, as illustrated in this figure.



In this example, port 2300 on `mysys` is being forwarded to `remotesys:23`. To do this, use SSH on `mysys`:

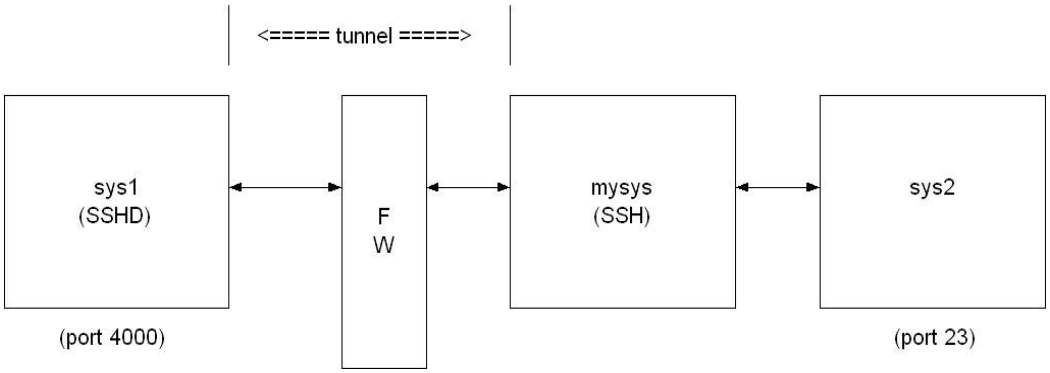
```
$ SSH remotesys /local_forward=(2300:remotesys:23)
```

Then, also on `mysys`, type:

```
$ telnet localhost /port=2300
```

When SSH and SSHD start their dialog, SSHD on `remotesys` connects back to itself, port 23, and the TELNET session is established.

/REMOTE_FORWARD=(*remoteport1*:*remotehost*:*remoteport2*)
This causes *remoteport1* on the system to which SSH connects to be forwarded to *remotehost*:*remoteport2*. In this case, the system on which the client is running becomes the intermediary between the other two systems.



For example, a user wants to use `mysys` to create a tunnel between `sys1:4000` and `sys2:23`, so that TELNET sessions that originate on `sys1:4000` get tunneled to `sys2` through the firewall. On `mysys`:

```
$ SSH sys1 /remote_forward=(4000:sys2:23)
```

Now, on `sys1`, a user could establish a TELNET session to `sys1` by doing:

```
$ telnet localhost /port=4000
```

The mechanism used for making the TELNET connection (setting up the tunnel) is essentially the same as described in the `/LOCAL_FORWARD` example above, except that the roles of SSH and SSHD in the dialog are reversed.

Other Files

The files in the below table are used by SSH. Note that these files generally reside in the `[.SSH2]` subdirectory from the user's `SYS$LOGIN` directory. The `[.SSH2]` subdirectory is created automatically on your local system the first time SSH is executed, and on a remote OpenVMS system the first time an SSH connection is made to that system. File protection for `SYS$LOGIN:SSH2.DIR` should be (`S:RWD,O:RWD,G:,W:`).

File Name	Resides On	Description
`[.SSH2]SSH2_CONFIG.`	Client System	This is the individual configuration file. This file is used by the SSH2 client. It does not contain sensitive information. The recommended file protection is (`S:RWD,O:RWD,G:,W:`).
`[.SSH2]IDENTIFICATION`	Client System	Contains the information about private keys that can be used for public-key authentication when logging in.

[.SSH2]ID_*alg_bits_seq*	Client System	Contains a private key for authentication. • *alg* is either RSA or DSA • *bits* is the length of the key • *seq* is an incrementing alphabetic value Thus, a key named ID_DSA_1024_A. indicates this is a private DSA key 1024 bits long, and it is the first time the key was generated using SSHKEYGEN. A user may have multiple private key files in a directory.
[.SSH2]ID_*alg_bits_seq*.PUB	Client System and Server System	Contains a public key for authentication. • *alg* is either RSA or DSA • *bits* is the length of the key • *seq* is an incrementing alphabetic value Thus, a key named ID_DSA_1024_B.PUB indicates this is a public DSA key 1024 bits long, and it is the second time the key was generated using SSHKEYGEN. A user may have multiple public key files in a directory.
[.SSH2.HOSTKEYS]*xxx*.PUB	Client System	Contains public host keys for all hosts the user has logged into. The files specifications have the format KEY_port_hostname.PUB • *port* is the port over which the connection was made. • *hostname* is the hostname of the key's host. For example, if tulip.example.com was accessed via port 22, the key file would be KEY_22_TULIP_EXAMPLE_COM.PUB.

		If this file changes on the host (for example, the system manager regenerates the host key), SSH2 will note this and ask if you want the new key saved. This helps prevent man-in-the-middle attacks.
`[.SSH2]RANDOM_SEED.`	Client System	Seeds the random number generator. This file contains sensitive data and MUST have a protection of no more than (`S:RWD,O:RWD,G:,W:`), and it must be owned by the user. This file is created the first time the program is run and is updated automatically. The user should never need to read or modify this file. On OpenVMS systems, multiple versions of this file will be created; however, all older versions of the file may be safely purged. Use the DCL command: `SET FILE /VERSION_LIMIT=n RANDOM_SEED` to set a limit on the maximum number of versions of this file that may exist at any given time.
`SSH2_DIR:SSH2_CONFIG`	Client System	This is a system-wide client configuration file. This file provides defaults for those values that are not specified in a user's configuration file, and for users who do not have a configuration file. This file must be world readable.
`PSCSSH_SSH2_KNOWNHOSTS_DIR`	Server System	Contains public host keys for all hosts the system has logged into. The files specifications have the format `KEY_`*`port`*`_`*`hostname`*`.PUB` • *`port`* is the port over which the connection was made. • *`hostname`* is the hostname of the key's host. For example, if tulip.example.com was accessed via port 22, the key file would be: `KEY_22_TULIP_EXAMPLE_`

		`COM.PUB` If this file changes on the host (for example, the system manager regenerates the host key), SSH will note this and ask if you want the new key saved. This helps prevent man-in-the-middle attacks.

SSHKEYGEN

Generates authentication key pairs. The format of the keys is incompatible between SSH1 and SSH2. Therefore, the correct format keys must be generated for each version of the protocol to be supported.

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

Each key may be protected via a passphrase, or it may be left empty. Good passphrases are 10-30 characters long and are not simple sentences or otherwise easily guessable. Note that the passphrase can be changed later, but a lost passphrase cannot be recovered, as a “one-way” encryption algorithm is used to encrypt the passphrase.

```
PSCSSH SSHKEYGEN /SSH1 [/BITS=n] [/IDENTITY_FILE=file]
                         [/PASSPHRASE=passphrase] [/COMMENT=comment]

PSCSSH SSHKEYGEN /SSH1 /CHANGE_PASSPHRASE [/PASSPHRASE=old_passphrase]
[/NEW_PASSPHRASE=new_passphrase]

PSCSSH SSHKEYGEN /SSH1 /CHANGE_COMMENT [/PASSPHRASE=passphrase]
                         [/COMMENT=comment]

PSCSSH SSHKEYGEN /SSH1 /CHANGE_CIPHER [/IDENTITY_FILE=file]
                         [/PASSPHRASE=passphrase]

PSCSSH SSHKEYGEN /SSH1 /HOST [/BITS=n][/COMMENT=comment]
```

Option	Description
/BITS=*nnn*	Specify key strength in bits (default = 1024).
/CHANGE_PASSPHRASE	Change the passphrase of private key file.
/CHANGE_COMMENT	Change the comment for a key.
/CHANGE_CIPHER	Change the cipher to current default (3DES).
/COMMENT=”*comment*”	Provide the comment.
/HOST	Generate the host key.
/IDENTITY_FILE=*file*	Specify the name of the host key file.
/PASSPHRASE=*ppp*	Provide the current passphrase.
/NEW_PASSPHRASE=*ppp*	Provide new passphrase.
/VERSION	Print sshkeygen version number.

```
PSCSSH SSHKEYGEN /SSH2[/BITS=n][/COMMENT=comment][/KEYTYPE=type]
                     [/KEYS=(key1...keyn)]
                     [/PASSPHRASE=ppp|/NOPASSPHRASE][/STIR=file][/QUIET]
PSCSSH SSHKEYGEN /SSH2/HOST
                     [/BITS=n][/COMMENT=comment][/STIR=file][/QUIET]
PSCSSH SSHKEYGEN /SSH2/DERIVE_KEY=file
PSCSSH SSHKEYGEN /SSH2/EDIT=file
PSCSSH SSHKEYGEN /SSH2/FINGERPRINT=file
PSCSSH SSHKEYGEN /SSH2/INFO=file [/BASE=n]
PSCSSH SSHKEYGEN /SSH2/SSH1_CONVERT=file
PSCSSH SSHKEYGEN /SSH2/X509_CONVERT=file
PSCSSH SSHKEYGEN /SSH2/PKCS_CONVERT=file
PSCSSH SSHKEYGEN /SSH2/EXTRACT_CERTS=file
PSCSSH SSHKEYGEN /SSH2/HELP
PSCSSH SSHKEYGEN /SSH2/VERSION
```

Option	Description
/BASE=*nnn*	Number base for displaying key info
/BITS=*nnn*	Specify key strength in bits (default = 1024).
/COMMENTS="*comment*"	Provide the comment.
/PKCS_CONVERT=*file*	Convert a PKCS 12 file to an SSH2 format certificate and private key.
/SSH1_CONVERT=*file*	Convert SSH1 identity to SSH2 format.
/X509_CONVERT=*file*	Convert private key from X.509 format to SSH2 format.
/DERIVE_KEY=*file*	Derive the private key given in *file* to public key.
/EDIT=*file*	Edit the comment/passphrase of the key.
/EXTRACT_CERTS=*file*	Extract certificates from a PKCS 7 file.
/FINGERPRINT=*file*	Dump the fingerprint of file.
/INFO=*file*	Load and display information for *file*.
/HELP	Print help text.
/HOST	Generate the host key.
/KEYS=(*key1,...,keyn*)	Generate the specified key file(s).
/KEYTYPE=(*dsa | rsa*)	Choose the key type: dsa or rsa.
/OPENSSH_CONVERT=*file*	Convert the specified OpenSSH key to SSH2 format
/OUTPUT_FILE=*file*	Write the key to the specified output file

/PASSPHRASE=*ppp*	Provide the current passphrase.
/NOPASSPHRASE	Assume an empty passphrase.
/QUIET	Suppress the progress indicator.
/STIR=file	Stir data from file to random pool.
/VERSION	Print sshkeygen version number.
/[NO]WARN	Enable or disable warnings if the process of generating host keys using /HOST will cause existing host keys to be overwritten. If enabled, the user will be prompted to overwrite them. If disabled, no warnings or prompts are issued if the host keys exist. Default is /WARN.

There is also a comment field in the public key file that is for the 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 *nnn*-bit dsa, *username*@*hostname*, *ddd mm-dd-yyyy hh*:*mm*:*ss* when the key is created unless the /COMMENT qualifier is used, and may be changed later using the /EDIT qualifier.

> **Note:** When the /HOST qualifier is used, the /KEYS=(*key1*,...*keyn*) qualifier is ignored.

> **Note:** The public key file must be world readable.

SSHAGENT

PSCSSH SSHAGENT

SSHAGENT is a program that holds authentication private keys. Both SSH1 and SSH2 keys are supported by SSHAGENT. SSHAGENT may be started in the beginning of a login session by including the commands to start it in, for example, LOGIN.COM. It may also be started interactively at any time during a login session.

To start SSHAGENT, one of the three methods may be used:

1. Start it in a separate window:
```
$ PSCSSH SSHAGENT
```
2. Spawn it as a subprocess:
```
$ SPAWN/NOWAIT PSCSSH SSHAGENT
```
3. Run it in a detached process:
```
$ RUN/DETACHED/OUTPUT=AGENT.OUT/INPUT=NLA0:/PROCESS_NAME="SSH AGENT" -
$_ SSH_EXE:SSH-AGENT2
```

The agent is used for public key authentication when logging to other systems using SSH. A connection to the agent is available to all programs run by all instances of the user on a specific system. The name of the mailbox used for communicating with the agent is stored in the PSCSSH_SSH_AGENT_*username* logical name.

The agent does not have any private keys initially. Keys are added using SSHADD. When executed without arguments, SSHADD adds the user's identity files. If the identity has a passphrase, SSHADD asks for the passphrase. It then sends the identity to the agent. Several identities can be stored in the agent; the agent can use any of these identities automatically.

PSCSSH SSHADD /LIST displays the identities currently held by the agent.

[.SSH]IDENTITY in SYS$LOGIN:	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 is used to encrypt the private part of this file. This file is not used by SSHAGENT, but is added to the agent using SSHADD at login.

SSHADD

Adds identities for the authentication agent.

```
PSCSSH SSHADD [OPTIONS] [FILE[,FILE,FILE]]
```

SSHADD adds identities to SSHAGENT, the authentication agent. When run without arguments, SSHADD adds the file [.SSH]IDENTITY. Alternative file names can be given on the command line. If any file requires a passphrase, SSHADD asks for the passphrase from the user.

The authentication agent must be running and must have been executed by the user for SSHADD to work.

FILE is an identity or certificate file. If no file is specified, the files in the user's [.SSH2] directory are used.

/HELP	Display help text.
/LIST	List all identities currently represented by the agent.
/LOCK	Lock the agent with a password.
/NOSSH1	Agent cannot use SSH1 keys.
/PURGE	Remove all identities from the agent.
/REMOVE	Remove the identity from the agent. In order to remove identities, you must either issue the command from the subdirectory that the identities are located in, or issue the command using the full path name of the identity (as is seen in an SSHADD /LIST command).
/TIMEOUT=*n*	Agent should delete this key after the timeout value (in minutes) expires.
/UNLOCK	Unlock the locked agent.
/URL	Give key to the agent as a URL.

FILES

These files exist in SYS$LOGIN:

[.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 is used to encrypt the private part of this file. This is the default file added by SSHADD when no other files have been specified. If SSHADD needs a passphrase, it reads the passphrase from the current terminal if it was run from a terminal. If SSHADD does not have a terminal

	associated with it but DECW$DISPLAY is set, it opens an X11 window to read the passphrase.
[.SSH]IDENTITY.PUB	Contains the public key for authentication. The contents of this file should be added to [.SSH]AUTHORIZED_KEYS on all systems where you want to log in using RSA authentication. There is no need to keep the contents of this file secret.
[.SSH]RANDOM_SEED	Seeds 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 SSHKEYGEN is run.

CERTTOOL

`pscssh certtool [options] /pk10 /subject=subject /key_usage=flags /extended_key_usage=flags`

`pscssh certview [options] /pk12 /input_files=objects`

The `CERTTOOL` utility is used for different needs concerning X.509 certificates.

`/BITS=n`	Key strength in bits (default 2048)
`/DEBUG=n`	Set debug level to *n*
`/EXTENDED_KEY_USAGE=(flag1…flagn)`	(PKCS#10 only) Extended key usage flags, as a comma-separated list. Valid values are: • `anyExtendedKeyUsage` • `ServerAuth` • `clientAuth` • `codeSigning` • `emailprotection` No extended flags are set by default.
`/HELP[=(PK10,PK12)]`	Display help. More detailed help on manipulating PKCS#10 and PKCS#12 certs is available by adding the `PK10` and `PK12` qualifier, respectively, to the `HELP` switch.
`/INPUT_FILES=(file1…filen)`	(PKCS#12 only) List of files to include in the PFX package.
`/KEY_TYPE=type`	Create a new key of type DSA or RSA.
`/KEY_USAGE=(flag1…flagn)`	(PKCS#10 only) Key usage flags, as a comma-separated list. Valid values are: • `digitalSignature` • `nonRepudiation` • `keyEncipherment` • `dataEncipherment` • `keyAgreement`

	• keyCertSign • CRLSign • encipherOnly • decipherOnly Default values are digitalSignature and keyEncipherment.
/OPTION=(*x*,*y*)	Set certificate option *x* to *y*. The options that can be set are dependent upon the type of certificate (PKCS#10 or PKCS#12) being affected. For PKCS#10: • DNS - set certificate DNS names. • Email - set certificate email addresses. For PKCS#12: • KeyPBE - set the PBE scheme for shrouding keys. default means pbeWithSHAAnd3-KeyTripleDES-CBC. • SafePBE - set the PBE scheme for protecting safes. default means pbeWithSHAAand40BitRC2-CBC.
/OUTPUT_FILE=*prefix*	Use *prefix* as the prefix for all output filenames. Private key filenames will be *prefix*.SSH2 and PKCS#10 fields will be *prefix*.PKCS10.
/PRIVATE_KEY=*keyname*	Use *keyname* as the private key.
/SUBJECT="*subject*"	(PKCS#10 only) Use *subject* as the certificate subject.
/VERSION	Display the version of CERTTOOL.

```
$ PSCSSH CERTTOOL /PK10 /SUBJECT=("cn=john doe,cn=lima,cn=beans" -
$_ /PRIVATE_KEY=DKA0:[JOHENDOE.SSH2]ID_DSA_1024_A

PKCS#10 creation succcesful.
Wrote certificate request to output.pkcs10.
```

CERTVIEW

`pscssh certview [options] certificate [, certificate, …, certificate]`

`CERTVIEW` can be used to view certificates and check their validity. This tool can also be used to output the data in format that is suitable for insertion in the `SSH2_DIR:SSHD2_CONFIG` configuration file.

`/COMMENT`	Prepend information lines with # (comment mark)
`/DEBUG=n`	Set debug level to *n*
`/FORMAT_OUTPUT`	Output data in a format suitable for insertion to *usermap*
`/HELP`	Display help
`/QUIET`	Don't display certificate information
`/VALIDATE=certificate`	Validate using the CA certificate *certificate*
`/VERBOSE`	Increase verbosity (display extensions).
`/VERSION`	Display version information

```
$ PSCSSH CERTVIEW MYCERT_PKCS7.P7B-1_SSH2_CRT
Certificate MYCERT_PKCS7.P7B-1_SSH2_CRT
Certificate issuer ........... : MAILTO=foo@bar.com, C=US, ST=CO,L=Colorado
Springs, CN=FOOCA
Certificate serial number .... : 206680290271582356976177697926629044 21
Certificate subject .......... : MAILTO=foo@bar,com, C=US, ST=CO, L=Colorado
Springs, CN=FOOCA
```

CMPCLIENT

`pscssh cmpclient [options]/ca_access_url="url" /subject="subject" /cert-file [private-key]`

Allows users to enroll certificates. It will connect to a CA (certification authority) and use the CMPv2 protocol for enrolling a certificate. The user may supply an existing private key when creating the certification request or allow a new key to be generated.

`url`	Specifies the URL for the Certification Authority
`subject`	Specifies the subject name for the certificate. For example, `"c-ca,o=acme,ou=development,cn=Bob Jones"`
`Cert-file`	Specifies the file the certification is written to.
`Private-key`	Specifies the private key to be written to.

`/BASE=name`	Specify base prefix for the generated files.
`/BITS=n`	Specify the key length in bits.
`/CA_URL="url"`	Specify the URL of the Certification Authority.
`/DEBUG=n`	Set debug to level *n* (0-60).
`/ENROLLMENT_PROTOCOL=prot`	Use specified enrollment protocol (SCEP or CMP).
`/EXTENSIONS`	Enable extensions in the subject name.
`/GENERATE_KEY`	Generate a new private key.
`/HELP`	Print this help text.
`/PROXY_URL"=url"`	Specify the URL of the HTTP proxy server URL to be used when connecting to the certification authority.
`/REFNUM=refnum:key`	Specify the CMP enrollment reference number and key.
`/SOCKS_SERVER="url"`	Specify the URL of the SOCKS server URL to be used when connecting to the certification authority.
`/SUBJECT="subject"`	Specifies the subject name for the certificate.
`/TYPE=rsa|dsa`	Specify the key type to generate (default: RSA)

/USAGE_BITS=*n*	Specify the key usage bits.
/VERSION	Print the version information for this program.

Examples:

1. Enroll a certificate and generate a DSA private key:

```
$ pscssh cmpclient/type=dsa/generate_key/base=mykey/refnum=1234:abc -
_$ /ca_access_url="http://www.ca-auth.domain:8080/pkix/"-
_$ /subject="c=us,o=foobar,cn=John Doe" ca-certification.crt
```

This will generate a private key called `mykey.prv` and a certificate called `mykey-0.crt`.

2. Enroll a certificate using a supplied private key and provide an e-mail extension:

```
$ pscssh cmpclient/base=mykey/refnum=12345:abcd -
_$ /ca_access_url="http://www.ca-auth.domain:8080/pkix/"-
_$ /subject="c=us,o=foobar,cn=John Doe:email=jdoe@example.com" -
_$ ca-certification.crt my_private_key.prv
```

This will generate and enroll a certificate called `mykey-0.crt`.

> **Note:** SSH stores and uses software certificates in DER encoded binary format. You can use `sshkeygen` to import and convert PKCS#12 packages (`/pkcs_convert=`*`file`*) into private key/certificate pair, X.509 format private key into SSH private key (`/x509_convert=`*`file`*) or PKC#7 into certificate (`/extract_certs=`*`file`*).

Public Key Subsystem

The public key subsystem and assistant that can be used to add, remove and list public keys stored on a remote server. The public key assistant and server are based upon a recent IETF draft, so other implementations of SSH may not yet offer this functionality.

The public key assistant can be started with:

```
$ PSCSSH PUBLICKEY_ASSISTANT [qualifiers] [[user@]host[#port]]
```

Public Key Assistant Commands

ADD *key-file-name*
Transfers the key file name to the remote system. The file name specified is expected to be in the `SSH2_CONFIG` directory from the user's login directory. e.g., `ADD ID_DSA_1024_A.PUB` will transfer the public key in `ID_DSA_1024_A.PUB` to the remote system and updates the `AUTHORIZATION.` file on the remote system to include this key name.

CLOSE
Closes the connection to the remote system.

DEBUG {no | *debug_level*}
Sets debug level (like in SFTP2).

DELETE *key finger-print*
Deletes the key that matches the fingerprint specified. It is necessary to do a `LIST` command before this to get a list of the fingerprints (and for the program to build its internal database mapping fingerprints to keys).

EXIT
Exits the program.

HELP
Displays a summary of the commands available.

LIST
Displays the fingerprint and attributes of keys stored on the remote system. The attributes that are listed will vary with key. Example output:

```
Fingerprint: xozil-bemup-favug-fimid-tohuk-kybic-loz-fukuc-kuril-gezahloxex
key type: ssh-dss
Comment: 1024-bit dsa, user@simple.example.com, Wed Jun 05 2022 21:05:40
```

OPEN [*user*@]*host*[#*port*]
Opens a connection to a remote public key subsystem.

QUIT
Quits the program.

UPLOAD *key-file-name*
Synonym for the `ADD` command.

VERSION [*protocol-version*]
Displays or sets the protocol version to use. The protocol version can only be set before the `OPEN` command is used. The default version is 1.

Public Key Assistant Qualifiers

/BATCHFILE
Provides a file with public key assistant commands to be executed. Starts SSH2 in batch mode. Authentication must not require user interaction.

/CIPHER
Selects encryption algorithm(s).

/COMPRESS
Enables SSH data compression.

/DEBUG
Sets debug level (0-99).

/HELP
Displays a summary of the qualifiers available.

/MAC
Selects MAC algorithm(s). `/MAC=(mac-1,…,mac-n)`

/PORT
Tells SFTP2 which port SSHD2 listens to on the remote machine.

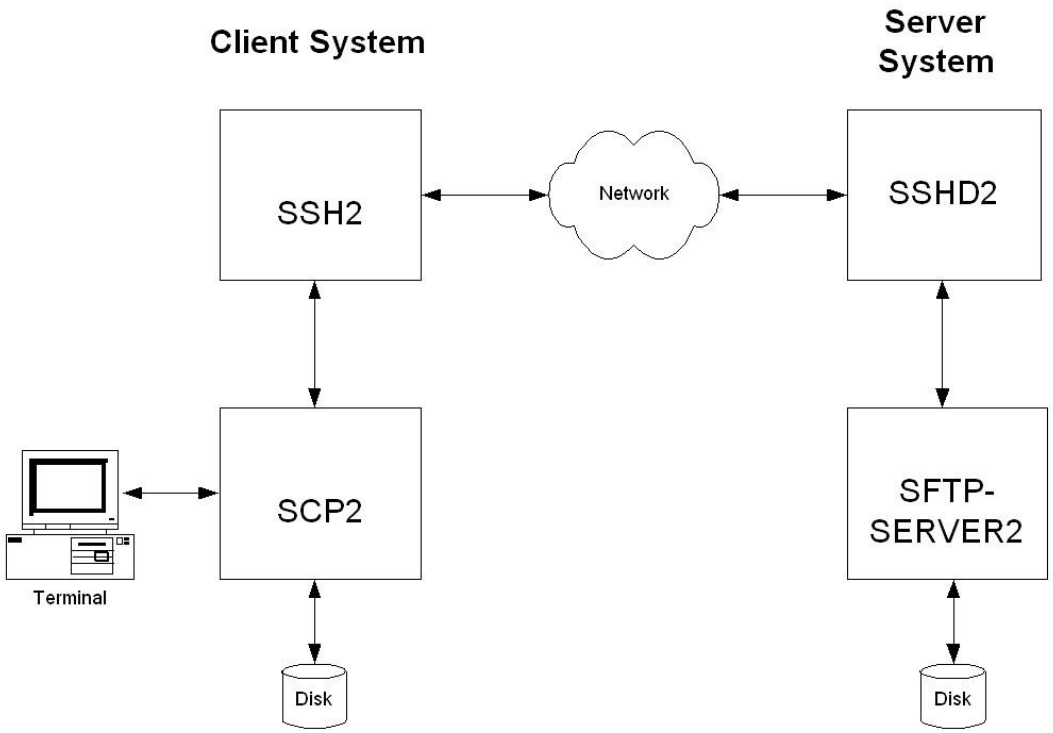
/VERBOSE
Enables verbose mode debugging messages. Equal to `/DEBUG=2`. You can disable verbose mode by using `debug disable`.

/VERSION
Displays version number only.

7. Secure File Transfer

There are three methods to do secure file transfer: SCP2, SFTP2, and FTP over SSH2. SCP2 and SFTP2 communicate with SSH2 for authentication and data transport (which includes encryption) to remote systems. An SCP1 server is provided for compatibility with OpenSSH SCP.

The following diagram illustrates the relationship among the client and server portions of an SCP2 or SFTP2 file transfer:



SCP file transfers are different from FTP file transfers. With FTP a file can be transferred as ASCII, BINARY, RECORD, or in OpenVMS format (if MultiNet or TCPware is in use). In SCP the primary transfer format is BINARY. Also, the defined syntax for a file specification is UNIX syntax. Due to these restrictions, files that are transferred from dissimilar systems may or may not be useful. ASCII transfers are done by searching the transferred data for the specified newline sequence and making the specified substitution. Process Software has used methods available in the protocol to attempt to improve the chances that files will be useful upon transfer.

Process Software has used the defined extensions in the protocol to transfer information about the VMS file header characteristics such that when a file is transferred between two VMS systems running MultiNet v5.4 or higher, TCPware v5.9 or higher, and/or PSCSSH, the file header information will also

be transferred and the file will have the same format on the destination system as it had on the source system. Also, when a text file is transferred to a non-VMS system, a method has been provided to convert those files that can be translated into a format that will be usable on the remote system. Files that are converted from non-VMS systems are stored as stream files on the VMS system, which provides compatibility for text files from those systems. Filenames are SRI encoded when files are stored on ODS-2 disks.

SCP2

Usage

`SCP2 [qualifiers] [[user@]host[#port]::]file [[user@]host[#port]::]file`

> **Note:** The source and destination file specification must be quoted if they contain a user specification or a non-VMS file specification.

Qualifiers

Qualifier	Description
`/ASCII[=convention]`	The newline convention specified is the newline convention to use if a newline convention is not specified by the server. Allowed values: `dos` (\r\n), `mac` (\r) , `unix` (\n), `vms` (\n) , `sftp` (\r\n). Default = `unix`.
`/BATCH`	Starts SSH2 in batch mode. Authentication must be possible without user interaction.
`/BUFFER_SIZE=integer`	Number of bytes of data to transfer in a buffer. Default is 7500. Minimum value is 512.
`/CIPHER=(cipher1,…,cipher-n)`	Selects an encryption algorithm(s).
`/COMPRESS`	Enables SSH data compression.
`/CONCURRENT_REQUEST=integer`	Number of concurrent read requests to post to the source file. Default is 4.
`/DEBUG=level`	Sets a debug level. (0-99)
`/DIRECTORY`	Forces the target to be a directory.
`/HELP`	Displays the help text.
`/IDENTITY_FILE=file`	Identifies the file for public key authentication.
`/PORT=number`	Tells the SCP2 client which port the SSHv2 server listens to on the remote machine.

/PRESERVE	Preserves file attributes and timestamps.
/NOPROGRESS	Does not show progress indicator.
/QUIET	Does not display any warning messages.
/RECURSIVE	Processes the entire directory tree.
/REMOVE	Removes the source files after copying.
/TRANSLATE_VMS= *(ALL, NONE, VARIABLE, FIXED, VFC)*	Selects the VMS text files to be translated (default=ALL). Note that /ASCII performs a similar function and may be supported in other SCP products.
/VERBOSE	Displays verbose debugging messages. Equal to "/debug=2".
/VERSION	Displays the version number only.
/VMS	Negotiates the ability to transfer VMS file information.

> **Note:** /ASCII, /VMS and /TRANSLATE_VMS are mutually exclusive

File Specifications

The source and destination strings are changed to lowercase unless they are enclosed in quotes, in which case they are left the same. File specification must be in UNIX format for remote systems, unless the remote system is running TCPware 5.9 or higher, MultiNet 5.5 or higher, or PSCSSH; and /VMS or /TRANSLATE_VMS (source files only) are used. UNIX format file specifications need to be enclosed in quotes (") if they contain the / character to prevent the DCL parsing routines from interpreting the string as a qualifier.

Qualifiers

/ASCII[=*convention*]

Uses the newline convention specified if the server does not specify a newline convention.

Available conventions are: dos (\r\n), mac (\r) , unix (\n), vms (\n) , sftp (\r\n). Default = unix.

/BATCH
Starts the SSH2 client in BATCH mode. When SSH2 is running in BATCH mode it does not prompt for a password, so user authentication must be performed without user interaction.

/BUFFER_SIZE=*integer*
Number of bytes of data to transfer in a buffer. Default is 7500.

/CIPHER=(*cipher*,…,*cipher-n*)
Lets you select which SSH2 cipher to use.

/COMPRESS
Enables SSH2 data compression. This can be beneficial for large file transfers over slow links. The compression level is set by the client configuration file for SSH2.

/CONCURRENT_REQUEST=*integer*
Number of concurrent read requests to post to the source file. Default is 4.

/DEBUG
Enables debugging messages for SCP2 and SSH2. Higher numbers get more messages. The legal values are between 0 (none) and 99. Debugging for the SFTP2 server is enabled via the `PSCSSH_SSH_SFTP_SERVER_DEBUG` logical.

/DIRECTORY
Informs SCP2 that the target specification should be a directory that the source file(s) will be put in. This qualifier is necessary when using wildcards in the source file specification, or `/RECURSIVE`.

/HELP
Displays command qualifier list and parameter format.

/IDENTITY_FILE=*file*
Specifies the identity file that SSH2 should use for public key authentication.

/PORT=*number*

Specifies the port that SSH2 uses on the remote system. Note that if both the source and destination files are remote, this value is applied to both. If SSH2 is available on different ports on the two systems, then the #port method must be used.

/PRESERVE
Sets the Protection, Owner (UIC), and Modification dates on the target file to match that of the source file. The adjustment of timestamps for time zones is dependent upon the logical `SYS$LOCALTIME` being set correctly. This is defined automatically on OpenVMS versions 7+ and can be defined similarly on earlier versions of VMS. `/PRESERVE` is not very useful when the target machine is a VMS system as VMS does not provide runtime library calls for setting the file attributes (owner, protection) and timestamps. Note that the VMS modification date (not the creation date) is propagated to the remote system. When files are copied between two VMS systems and `/VMS` is used `/PRESERVE` is implied and the process of transferring VMS attributes preserves the information about the protection, dates, and file characteristics.

/NOPROGRESS
SCP2, by default, updates a progress line at regular intervals when it is run interactively to show how much of the file has been transferred. This qualifier disables the progress line.

/QUIET
Disables warning messages. Note that it does not disable warning messages from the SFTP2 server, which return on the error channel.

/RECURSIVE
Copies all of the files in the specified directory tree. Note that the top level directory on the local system is not created on the remote system. Only the most recent version is copied unless in VMS mode and the `PSCSSH_SFTP_VMS_ALL_VERSIONS` logical is defined to be `TRUE`.

/REMOVE
Deletes the source files after they have been copied to the remote system.

/TRANSLATE_VMS
Translates VMS text files in the copying process to byte streams separated by linefeeds because the defined data transfer format for SCP2 is a binary stream of bytes.

`/TRANSLATE_VMS` is only applicable to the source specification. If a remote source file is specified, then that system must be running MultiNet 4.4 or higher, TCPware 5.6 or higher, or PSCSSH. If `/TRANSLATE_VMS` is specified with no value, then `VARIABLE`, `FIXED`, and `VFC` (Variable, Fixed

Control) files are translated to stream linefeed files. If the value is NONE, no files are translated. VARIABLE, FIXED, and VFC can be combined in any manner. The SFTP2 server process uses the value of the logical PSCSSH_SFTP_TRANSLATE_VMS_FILE_TYPES to determine which files should be translated automatically. This is a bit mask with bit 0 (1) = FIXED, bit 1 (2) = VARIABLE, and bit 2 (4) = VFC. These values can be combined into a number between 0 and 7 to control which files are translated.

> **Note:** Due to the structure of the programs, the SCP2 program uses the PSCSSH_SFTP_TRANSLATE_VMS_FILE_TYPES logical if the /TRANSLATE_VMS qualifier has not been specified.

/VERBOSE
Displays debugging messages that allow the user to see what command was used to start up SSH and other basic debugging information. Note that debugging information can interfere with the normal display of the progress line. Equivalent to /DEBUG=2.

/VERSION
Displays the version of the base SCP2 code.

/VMS
Transfers VMS file information similar to that transferred in OVMS mode in FTP such that VMS file structure can be preserved. All of the information transferred in FTP OVMS mode is transferred along with the file creation date and protection. Timestamps are not adjusted for time zone differences in VMS transfers. If the file is a contiguous file, and it is not possible to create the file contiguously, and the logical PSCSSH_SFTP_FALLBACK_TO_CBT has the value of TRUE, the SFTP2 server attempts to create the file Contiguous, Best Try.

The logical name PSCSSH_SCP2_VMS_MODE_BY_DEFAULT can be defined to TRUE to specify that /VMS should be the default unless /NOVMS or /TRANSLATE_VMS are specified. /VMS and /TRANSLATE_VMS cannot be used on the same command line. If /VMS is not specified, but the logical is set to enable it by default, a /TRANSLATE_VMS on the command line will take precedence.

Note that even though SCP2 and the SFTP2 server pass the request for VMS file transfers or to translate a VMS file in a manner that is consistent with the protocol specification, other implementations may not handle this information well. Since there is no error response present at that point in the protocol, the program hangs. To prevent it from hanging forever, the logical PSCSSH_SCP2_CONNECT_TIMEOUT is checked to see how long SCP2 should wait for a response when establishing the connection. The format for this logical is a VMS delta time. The default value is 2 minutes. If SCP2 times out before a

connection is established with the SFTP2 server and /VMS or /TRANSLATE_VMS were specified, a warning message is displayed, and the initialization is tried again without the request for VMS information (or /TRANSLATE_VMS). This retry is also subject to the timeout, and if the timeout happens again, then SCP2 exits. This helps for implementations that ignore the initialization message when information they do not recognize is present; implementations that abort will cause SCP2 to exit immediately.

Logicals

For the following logicals, all that start PSCSSH_SFTP apply to the SCP2 client, SFTP2 client and SFTP2 server.

PSCSSH_SFTP_FALLBACK_TO_CBT

When defined to TRUE and a VMS file transfer is being performed, this logical creates a Contiguous file if that file has Contiguous characteristics. The file will be created as Contiguous Best Try if there is insufficient space to create it as Contiguous.

PSCSSH_SFTP_TRANSLATE_VMS_FILE_TYPES

This is a bit mask that determines which VMS file types should be translated when not operating in VMS mode.

- Bit 0 (1) = FIXED
- Bit 1 (2) = VARIABLE
- Bit 2 (4) = VFC

The values are:

- 0 (zero) = NONE
- 7 = ALL

Note that this logical affects SCP2 as well as the server, as SCP2 has the server built into it for handling local file access. If this logical is not defined, the value 7 will be used.

PSCSSH_SCP2_CONNECT_TIMEOUT

This logical defines a number specifying how long SCP2 should wait for a response to the INITIALIZE command from the server program. This is a VMS delta time number. The default is 2 minutes.

PSCSSH_SCP2_VMS_MODE_BY_DEFAULT

When defined to TRUE, this logical chooses the /VMS qualifier if /TRANSLATE_VMS or /NOVMS has not been specified.

PSCSSH_SFTP_RETURN_ALQ

When defined to TRUE and files are being transferred in VMS mode, this logical includes the Allocation Quantity for the file in the file header information. This is disabled by default because copying a small file from a disk with a large cluster size to a disk with a small cluster size causes the file to be allocated with more space than necessary. You have the option of retaining the allocated size of a file if it was allocated the space for a reason. Some combinations of file characteristics require that the Allocation Quantity be included in the file attributes; this is handled by SCP2 or the SFTP2 server.

PSCSSH_SSH_SCP_SERVER_DEBUG

Enables debugging messages for the SCP server that provides service to SCP commands that use the RCP over SSH2 protocol (OpenSSH). When this is defined, the file SCP-SERVER.LOG is created in the user's login directory. These files are not purged. Larger values yield more debugging information.

PSCSSH_SSH_SFTP_SERVER_DEBUG

Enables debugging messages for the SFTP2 server that provides service to SCP2 commands that use the SFTP protocol. When this is defined, the file SFTP-SERVER.LOG is created in the user's login directory. These files are not purged. Larger values yield more debugging information

PSCSSH_SFTP_MAXIMUM_PROTOCOL_VERSION

This logical can be used to limit the version of the SSH File Transfer Protocol that the SFTP client and server use. This can sometimes provide a work-around for problems encountered with different implementations of the protocol. The default value is 4. Protocol versions 2 and 3 are also used by popular implementations.

PSCSSH_SFTP_VMS_ALL_VERSIONS

This logical controls whether all versions of a file are returned. The value TRUE will cause all versions to be returned, any other value is to only return the name of the file without a version. The default is to return only one filename without the version number.

PSCSSH_SFTP_NEWLINE_STYLE

This logical controls the newline style that SFTP uses, which can be helpful in transferring text files. The values are: UNIX <lf>, VMS <lf>, MAC <cr>. If the logical is not defined, or defined to any other value,

then <cr><lf> will be used for the text line separator as documented in the SSH File Transfer specification.

PSCSSH_SFTP_CASE_INSENSITIVE
This logical causes SFTP to treat filenames in a case insensitive manner when it is defined to TRUE.

PSCSSH_SFTP_ODS2_SRI_ENCODING
This logical controls whether SRI encoding is used for filenames on VMS ODS-2 disks. If the logical is not defined, or is defined to TRUE then SRI encoding is used on ODS-2 disks for filenames that contain uppercase letters and special characters.

PSCSSH_SFTP_FILE_ESTIMATE_THRESHOLD
This logical controls the minimum number of blocks that a text file must be for an estimated transfer size to be returned instead of an exact size. The default is to estimate the transfer size for all text files.

PSCSSH_SFTP_DEFAULT_FILE_TYPE_REGULAR
If this logical is defined to TRUE, then the SFTP server will use a default file type of REGULAR instead of UNKNOWN for OPEN operations. This can correct problems with filenames without a . (dot) in them getting .dir added to them. The filename will appear with a . at the end of the name in directory listings.

PSCSSH_SFTP_*username*_CONTROL
This logical can be defined /SYSTEM to any combination of NOLIST, NOREAD, NOWRITE, NODELETE, NORENAME, NOMKDIR, NORMDIR, to restrict operations for the username in the logical. NOWRITE will disable PUT, DELETE, RENAME, MKDIR, RMDIR; NOREAD will disable GET and LIST.

PSCSSH_SFTP_*username*_ROOT
This logical can be defined /SYSTEM to restrict the user to the directory path specified. Subdirectories below the specified directory are allowed.

SSH_SFTP_LOG_SEVERITY
This logical can be defined /SYSTEM to 20000 to log file transfers or 30000 to log all SFTP operations.

SSH2_SFTP_LOG_FACILITY

This logical must also be defined /SYSTEM to specify the logging class that is used with OPCOM. Values below 5 will use the network class; 5 will use OPER1, 6 will user OPER2, etc. The maximum value that can be specified is 12, which will use OPER8.

PSCSSH_SFTP_SEND_VENDOR_ID

If this logical is defined as FALSE, then the SFTP2 client will not send the extended command containing the vendor ID upon completion of version negotiation with the server.

SFTP2

File Specifications

File specification must be in UNIX format for remote systems, unless `/VMS` transfers are being used.

SFTP2 Command Syntax and Qualifiers

Usage

`SFTP2 [qualifiers] [[user@]host[#port]]`

If the `username@` is included in the remote system specification, the specification must be enclosed in quotes.

Qualifiers

Qualifier	Description
`/BATCHFILE=file_spec`	Provides file with SFTP commands to be executed. Starts SSH2 in batch mode. Authentication must not require user interaction.
`/BUFFER_SIZE=integer`	Number of bytes of data to transfer in a buffer. Default is 7500.
`/CIPHER=(cipher-1,…,cipher-n)`	Selects encryption algorithm(s).
`/COMPRESS`	Enables SSH data compression.
`/CONCURRENT_REQUEST=integer`	Number of concurrent read requests to post to the source file. Default is 4.
`/DEBUG=level`	Sets debug level (0-99).
`/HELP`	Displays help.
`/MAC=(mac-1,…,mac-n)`	Select MAC algorithm(s).
`/NOPROGRESS`	Do not show progress indicator.
`/PORT`	Tells SFTP2 which port the SSHD2 server is listening on.

/VERBOSE	Enables verbose mode debugging messages. Equal to /debug=2. You can disable verbose mode by using debug disable.
/VERSION	Displays version number only.
/[NO]VMS	Negotiates ability to transfer VMS file information. VMS transfer mode will be automatically negotiated if SFTP2 detects that the server is capable of doing VMS transfers unless /NOVMS is specified.

SFTP2 Commands

SFTP2 Command	Description
`ASCII [{-s | remote [local]}]`	With `-s` option, shows current newline convention. *remote* sets remote newline convention. *local* operates on local side, but is not as useful (the correct local newline convention is usually compiled in, so this is mainly for testing). You can set either of these to "ask", which will cause sftp to prompt you for the newline convention when needed. With the exception of the `-s` option, this command sets transfer mode to ascii. Available conventions are `dos`, `unix`, `sftp`, `vms`, or `mac`, using "\r\n", "\n", "\r\n", "\n" and "\r" as newlines, respectively. Note that some implementations of SFTP may check to see if a file can be transferred in ASCII mode before doing so, and return errors for files that cannot be transferred. PSCSSH, MultiNet, and TCPware make this check.
`AUTO`	Sets the transfer mode (ASCII or BINARY) to depend upon the extension of the file specification.
`BINARY`	Sets the transfer mode to be binary. (This is the default.)
`BUFFERSIZE number`	Sets the size of the buffer used for file transfer. A larger buffer size helps speed large transfers. Displays the current buffer size when no parameter is specified.
`CD dirspec`	Changes current directory on remote system. VMS file specifications may be used when operating in VMS mode. A logical name must include the trailing colon so that it can be recognized as such. SFTP from other vendors cannot use VMS specifications due to the way that SFTP works.
`CHMOD [-R] mode file [file…]`	Change the protection on a file or directory to the specified octal mode. (Unix values). `-R` recurses over directories.
`CLOSE`	Closes connection to the remote server.

DEBUG {disable | *debug level}*	Sets the debug level for SFTP2. It does not change the current debug level for SSH2 for an existing connection, but will be used with SSH2 for a new connection. With disable, this disables all debugging current sessions for SFTP2.
DELETE *filespec*	Removes the specified file from the remote system.
DIRECTORY [*file* | *dirspec*]	Displays the contents of the current directory or specified directory in VMS format when the transfer mode is VMS. File names are displayed as they would be with a DIR command from DCL.
EXIT	Exits SFTP client.
GET [-p] *file*1 [*file2* …]	Retrieves the specified file(s) from the remote system and stores it in the current working directory on the local system. File names are case sensitive and in UNIX format. When operating in VMS mode, either UNIX or VMS-style file specifications can be used. Directories are recursively copied with their contents. Multiple files may be specified by separating the names with spaces. If -p is specified, then SFTP attempts to preserve timestamps and access permissions. Note that a target filename cannot be provided.
GETEXT	Displays the list of file extensions to use ASCII transfers when in AUTO mode. The initial value is txt,htm*,pl,php*
HELP	Displays help on commands.
LCD *dirspec*	Changes the current directory on the local system. VMS file specifications may be used when in VMS mode.
LCHMOD [-R] *mode file* [*file*…]	Change the protection on a file or directory on the local connection to the specified octal mode. (Unix values). -R recurses over directories.
LCLOSE	Close the local connection.
LDELETE *file*	Removes the specified file from the local system. VMS file specifications may be used when in VMS mode.
LDIRECTORY [*file* | *dirspec*]	Displays the contents of the current directory for the local system in VMS format when the transfer mode is VMS.

	File names are displayed as they would be with a `DIR` command from DCL.
`LLS [`*`file	dirspec`*`]`	Displays the contents of the current directory or specified directory in UNIX format. Lists the names of files on the local server. For directories, contents are listed. See `LS` for options and more details.
`LLSROOTS`	Like `LSROOTS`, but for the local side.
`LMKDIR` *`dirspec`*	Creates the specified directory on the local system.
`LOCALOPEN {[`*`user@host[#port]`*` | -1}`	Tries to connect the local side to the host *host*. If successful, `LLS` and friends will show the contents of the filesystem on that host. With the `-1` option, connects to the local filesystem (which doesn't require a server). There is an implied `LOCALOPEN -1` when SFTP2 starts up. Note that an implicit `LOCALOPEN` is done when SFTP2 starts, so the only time that a user needs to do a `LOCALOPEN` is when neither directory tree is immediately accessible. `OPEN` is the command that is generally used to establish the connection with the remote system. `LOPEN` is a synonym for `LOCALOPEN`.
`LPWD`	Displays the current working directory on the local system.
`LREADLINK` *`path`*	Provided that *`path`* is a symbolic link, shows where the link is pointing to. This command is not supported for VMS servers.
`LRENAME` *`oldfile newfile`*	Renames a file on the local system.
`LRM` *`filespec`*	Removes the specified file from the local system. VMS file specifications may be used when in VMS mode.
`LRMDIR` *`dirspec`*	Deletes a directory on the local system.
`LS [-R] [-l] [-S] [-r] [`*`file`*` …]`	Displays the contents of the current directory or specified directory in UNIX format. Lists the names of files on the remote server. For directories, contents are listed. When the `-R` option is given, directory trees are listed recursively. (By default, subdirectories of the arguments are not visited.) When the `-l` option is given,

	permissions, owners, sizes, and modification times are also shown. When the `-S` options is specified sorting is based upon file size instead of alphabetically. The `-r` option reverses the sort order. When no arguments are given, it assumes that the contents of the current working directory are being listed. Currently, the options `-R` and `-l` are mutually incompatible. `LS` will fill a screen with output, then wait for the user to decide if they want more or have seen enough.
`LSROOTS`	Displays the virtual roots of the server. This is a VMS-only extension to display the roots (devices) on the VMS system.
`LSYMLINK` *`targetpath linkpath`*	Like `SYMLINK`, but for the "local" side.
`MGET [-p]` *`file1`* `[`*`file2`*`…]`	Retrieves multiple files from the remote system and stores them in the current working directory on the local system. If `-p` is specified, then SFTP attempts to preserve timestamps and access permissions.
`MKDIR` *`dirspec`*	Creates the specified directory on the remote system.
`MPUT [-p]` *`file1`* `[`*`file2`*`…]`	Stores multiple files in the current working directory on the remote system. File names are case-sensitive and in UNIX format. When operating in VMS mode, either UNIX or VMS-style file specifications can be used. Directories are recursively copied with their contents. Multiple files may be specified by separating the names with spaces. If `-p` is specified, then SFTP attempts to preserve timestamps and access permissions.
`OPEN {-1 | [`*`user@`*`]`*`host`*`[#`*`port`*`]}`	Tries to connect to *host*. Or with the `-l` option, connects the remote side to the local filesystem (which doesn't require a server).
`PUT [-p]` *`file1`* `[`*`file2`*`…]`	Stores the specified file in the current working directory on the remote system. File names are case-sensitive and in UNIX format. When operating in VMS mode, either UNIX or VMS-style file specifications can be used. Directories are recursively copied with their contents. Multiple files may be specified by separating the names with spaces.

	If `-p` is specified, then SFTP attempts to preserve timestamps and access permissions. Note that a target filename cannot be provided.
`PWD`	Displays the current working directory on the remote system. Displayed in VMS format when in VMS mode; otherwise displayed in UNIX format.
`QUIT`	Exits the SFTP client.
`READLINK` *`targetpath linkpath`*	Provided that <*path*> is a symbolic link, shows where the link is pointing to. Not valid for VMS systems as VMS does not have symbolic links.
`RECORD`	Enters record transfer mode if the server supports Process Software's record open. The direction in which record transfer mode is possible will be displayed in response to this command. In record transfer mode the source file is opened as binary records and the destination file is opened as binary. This produces the same effect as MultiNet's FTP server `BINARY` transfer when a `BLOCK_SIZE` has not been specified, and can be used to transfer a file that contains VMS records to a system that can only handle "flat" files.
`RENAME` *`oldfile newfile`*	Renames file on the remote system.
`RM` *`filespec`*	Removes the specified file from the remote system.
`RMDIR` *`dirspec`*	Deletes a directory on the remote system.
`SETEXT` *`ext1`* `[`*`ext2`* `…]`	Sets the list of file extensions to use ASCII transfers when in `AUTO` mode. Individual file extensions must be separated by spaces.
`STATUS`	Shows the transfer mode, remote server name, and remote server version. The current newline sequence is displayed if operating in `ASCII` or `AUTO` mode.
`SYMLINK` *`targetpath`* `linkpath`	Creates symbolic link *`linkpath`*, which will point to *`targetpath`*. Not valid for VMS servers as VMS does not have symbolic links.
`VERBOSE`	Enables verbose mode (identical to the `/DEBUG=2` command-line option). You may later disable verbose mode with the command `DEBUG DISABLE`.

VMS	Sets the transfer mode to include VMS file information.

Logicals

The following logicals are specific to SFTP2:

PSCSSH_SFTP_VMS_MODE_BY_DEFAULT
When defined to TRUE, this logical chooses the /VMS qualifier if /NOVMS has not been specified.

Configuration File Parameters

The system wide configuration file (SSH2_DIR:SSH2_CONFIG.) or the user's configuration file (SYS$LOGIN:[.SSH2]SSH2_CONFIG.) can be used to specify the following parameters. The user's configuration file takes precedence over the system configuration file.

FilecopyMaxBuffers	This is equivalent to the /CONCURRENT_REQUEST qualifier on the SFTP2 or SCP2 command line. The command line qualifier will supersede any value in the configuration file.
FilecopyMaxBuffersize	This is equivalent to the SFTP2 BUFFERSIZE command or the SCP2 /BUFFER_SIZE qualifier. The command or qualifier takes precedence.

The system server configuration file (SSH2_DIR:SSHD2_CONFIG.) can include parameters to control which users can perform remove SSH commands (including SSH terminal sessions) as well as SFTP2 access:

Terminal.AllowUsers	Allow users in the specified list to create SSH2 terminals and do interactive commands
Terminal.DenyUsers	Prevent users in the specified list from creating SSH2 terminals and performing interactive commands. The users can still use the SFTP2, SCP1 and public key servers.
Terminal.AllowGroups	Allow groups in the specified list to create SSH2 terminals and do interactive commands

`Terminal.DenyGroups`	Prevent groups in the specified list from creating SSH2 terminals and performing interactive commands. The groups can still use the SFTP2, SCP1 and public key servers.

FTP over SSH

SSH2 can be used to set up port forwarding that can be used for FTP. This allows users to use the richness of the FTP command set to access files on a remote system and have their control and data information encrypted. The command format to set up the SSH port forwarding is:

```
$ ssh remote_host_name -
_$ /local_forward=(" " "ftp/forwarded_port_number:localhost:21" " ")
```

The usual SSH authentication mechanisms come into play, so there may be a request for a password and a terminal session is established to the remote host. As long as this terminal session is alive, other users on the local system can use FTP to access the remote system over an encrypted channel. The location of the quotes is important, as it is necessary to prevent DCL from interpreting the `/` in the local forwarding information as the start of a new qualifier, and SSH2 does not know or expect to find the `( )` around the forwarding information. Note that the `localhost` inside of the forwarding string is important, as it will make the connection to FTP on the remote system come from localhost, which will then allow FTP to open the data port.

When a user desires to use an encrypted FTP connection, the following sequence of commands would be issued:

```
SSH> PORT forward_port_number
SSH> OPEN LOCALHOST
```

Normal FTP authentication takes place and multiple FTP sessions may use a single forwarded port. The FTP protocol filter in SSH2 scans the FTP command stream for the FTP `PORT` and `PASV` commands and their replies, and makes substitutions in these commands and replies to use a secure data stream through the SSH2 session that has been set up. This command will establish an encrypted FTP session with the remote host that the SSH connection is sent to.

To allow a single system to act as a gateway between two networks, add `/ALLOW_REMOTE_CONNECT` to the SSH command that initiates the connection.

8. Monitoring and Controlling SSH

PSCSSH provides utilities for monitoring and controlling the SSH server environment. The following topics describe the utilities, their capabilities, and their use.

Controlling SSH Server Functions

The following control functions are available for the SSH servers:

- Startup
- Shutdown
- Restart
- Set debug level

The SSHCTRL Utility

The `SSHCTRL` utility is used to perform all but the startup function. For the startup function, the `SYS$STARTUP:PSCSSH_STARTUP.COM` file is used. Usage:

```
$ SSHCTRL operation options
```

The below table shows the various operations that can be used with the `SSHCTRL` utility.

Operation	Description
`SET /DEBUG=`*`n`*	Set debug level (0 = no debug)
`SHOW`	Show session information.
`SHOW /ALL`	Show all sessions. This is the default if no switch is used with the `SHOW` keyword.
`SHOW /USER=`*`username`*	Show sessions for *`username`*
`SHOW /HOST=`*`address`*	Show sessions for *`address`*
`SHUTDOWN`	Stop all SSH server sessions.

RESTART	Stop/restart SSH server.
HELP	Display help text.
VERSION	Display version information.

Starting the SSHD Master Process

```
$ @SYS$STARTUP:PSCSSH_STARTUP
Starting PSCSSH...
%RUN-S-PROC_ID, identification of created process is 22C000AD
$
```

Shutting down the SSHD Master Process

This function is used to stop the SSHD Master process on the system, so it won't accept new connections. Note that shutting down the SSHD Master process will also terminate all outstanding SSH server sessions on the system. OPER privilege is required to shut down the SSHD Master process and its servers.

```
$ SSHCTRL SHUTDOWN
Shutting down PSCSSH...
$
```

Restarting the SSHD Master Process

Restarting the SSHD Master process is required after the CNFSSH utility is used to modify the existing configuration. Note that restarting the SSHD Master process will terminate all outstanding SSH server sessions on the system. OPER privilege is required to restart the SSHD Master process.

```
$ SSHCTRL RESTART
Shutting down PSCSSH...
Starting PSCSSH...
%RUN-S-PROC_ID, identification of created process is 22C000B8 $
```

Changing the Server Debug Level

The server debug level is changed using SSHCTRL. The debug level controls the amount of debug information written to the SSH_LOG:SSHD.LOG file for each server instance. This may be a value from 0 (no debug) to 50 (maximum debug). Process Software recommends this value not be set above 5 without instructions from Process Software, as the amount of debug information written to the log at higher levels can severely impact both the SSH server performance and the server host disk resources.

Note that setting the debug level only affects new server processes which are started after setting the level. Currently active servers use the debug level set when they were started. OPER privilege is required to change the debug level.

```
$ SSHCTRL SET/DEBUG=4
SSHCTRL-S-DEBUGSET - old debug level = 2, new debug level = 4
$
```

Displaying SSH Server Utilization

The SSHCTRL SHOW command is used to display the active SSH server sessions on a system. It can display all users (/ALL), users with a specific username (/USER=jdoe), or users with sessions that originate from a specific host (/HOST=192.168.29.248).

Normally, a user may only display the sessions with the same UIC as his own. GROUP privilege is required to display the sessions with UICs in the same group as the user. WORLD privilege is required to display all other servers.

For each session, the display is of the following form:

```
Process "processname" (pid pid) - an <ssh1|ssh2> session
  User = username
  From system address port port
  Started: date/time session was started
  Bytes in: count out: count (from child process PID)
  Child process = "process name" (pid pid) - an type session
  PTD Device = FTAnn:
  Started date/time this child started
```

Note that SSH2 provides the capability for one server to handle multiple child sessions. The child sessions may be a mixture of interactive SSH2 sessions and file transfer (SCP/SFTP) sessions.

In the below example, a display of all users on the system is done. Note that server "SSHD 0003" actually has six active child processes.

```
$ SSHCTRL SHOW /ALL
SSHD Master PID = 22C000B8 (SSHD_MASTER)
```

```
Debug level is set to 4
Process "SSHD 0000" (pid 22C000B9) - an SSH2 session
      User = JDOE
      From system 192.168.29.52 port 49152
      Started: 01/15/2010 03:05:22
      Bytes in: 262  out: 0  (from child process: 15100)
      Child process = "JDOE_@FTA4" (pid 22C000BA) - an SSH2 session
      PTD Device = _FTA4:
         Started: 01/15/2010 03:05:35
Process "SSHD 0003" (pid 22C000BF) - an SSH2 session
      User = ALICE
      From system 192.168.29.50 port 1129
      Started: 01/15/2010 03:07:46
      Bytes in: 0  out: 0  (from child process: 55215)
      Child process = "ALICE_@FTA9" (pid 22C000C0) - an SSH2 session
      PTD Device= _FTA9:
          Started: 01/15/2010 03:07:54
      Child process = "SSHD 0003A SFTP" (pid 22C000C1) - an SFTP-SERVER2
session
      PTD Device = _FTA10:
          Started: 01/15/2010 03:07:55
      Child process = "ALICE_@FTA11" (pid 22C000C2) - an SSH2 session
      PTD Device = _FTA11:
          Started: 01/15/2010 03:07:57
      Child process = "SSHD 0003B SFTP" (pid 22C000C3) - an SFTP-SERVER2
session
      PTD Device = _FTA12:
          Started: 01/15/2010 03:08:00
      Child process = "SSHD 0003C SFTP" (pid 22C000C4) - an SFTP-SERVER2
session
      Device = _FTA13:
          Started: 01/15/2010 03:08:07
      Child process = "ALICE_@FTA14" (pid 22C000C5) - an SSH2 session
      PTD Device = _FTA14:
          Started: 01/15/2010 03:08:09
Process "SSHD 0004" (pid 22C000C6) - an SSH1 session
      User = BOB
      From system 192.168.29.51 port 1023
      Started: 01/15/2010 03:08:29
      Bytes in: 0  out: 537  (from child process: 17)
      Child process = "BOB_@FTA15" (pid 22C000C7) - an SSH1 session
      PTD Device = _FTA15:
          Started: 01/15/2010 03:08:29
```

The below example illustrates showing the sessions that originate from a specific TCP/IP address:

```
$ SSHCTRL SHOW /HOST=192.168.29.51
SSHD Master PID = 22C000B8 (SSHD_MASTER)

Debug level is set to 4
Process "SSHD 0004" (pid 22C000C6) - an SSH1 session
      User = ALICE
```

```
From system 192.168.29.51 port 1023
Started: 01/15/2010 03:08:29
Bytes in: 0  out: 537  (from child process: 17)
Child process = "ALICE_@FTA15" (pid 22C000C7) - an SSH1 session
PTD Device = _FTA15:
  Started: 01/15/2010 03:08:29
```

Appendix A. SRI Encoding of Filenames

SFTP2, SCP2, SFTP2 server, and SCP server use SRI mapping to preserve case and characters that are not valid in filenames on OpenVMS ODS-2 disks.

OpenVMS disk filenames can be 39 characters long (as can file extensions) and include only the following characters: `0` through `9`, `A` through `Z`, dollar sign (`$`), hyphen (`-`), and underscore (`_`).

The below table shows the default SRI International mapping.

ASCII character...	Is mapped to...	With octal value...
Ctrl/A (soh)	$4A	001
Ctrl/B (stx)	$4B	002
Ctrl/C (etx)	$4C	003
Ctrl/D (eot)	$4D	004
Ctrl/E (enq)	$4E	005
Ctrl/F (ack)	$4F	006
Ctrl/G (bel)	$4G	007
Ctrl/H (bs)	$4H	010
Ctrl/I (ht)	$4I	011
Ctrl/J (nl)	$4J	012
Ctrl/K (vt)	$4K	013
Ctrl/L (np)	$4L	014
Ctrl/M (cr)	$4M	015
Ctrl/N (so)	$4N	016
Ctrl/O (si)	$4O	017
Ctrl/P (dle)	$4P	020
Ctrl/Q (dc1)	$4Q	021
Ctrl/R (dc2)	$4R	022
Ctrl/S (dc3)	$4S	023

Ctrl/T (dc4)	$4T	024
Ctrl/U (nak)	$4U	025
Ctrl/V (syn)	$4V	026
Ctrl/W (etb)	$4W	027
Ctrl/X (can)	$4X	030
Ctrl/Y (em)	$4Y	031
Ctrl/Z (sub)	$4Z	032
Ctrl/[(esc)	$6B	033
Ctrl/(fs)	$6C	034
Ctrl/] (gs)	$6D	035
Ctrl/^ (rs)	$6E	036
Ctrl/_ (us)	$6F	037
SPACE (sp)	$7A	040
!	$5A	041
"	$5B	042
#	$5C	043
$	$$ (See Rule 8 in the following table.)	044
%	$5E	045
&	$5F	046
'	$5G	047
(	$5H	050
)	$5I	051
*	$5J	052
+	$5K	053
,	$5L	054
-	same	055
.	. or $5N (See the following table.)	056
/	not mapped (directory delimiter)	057
0 to 9	same	060 to 071

:	$5Z	072
;	$7B	073
<	$7C	074
=	$7D	075
>	$7E	076
?	$7F	077
@	$8A	100
A to Z	same	101 to 132
[	$8B	133
	$8C	134
]	$8D	135
^	$8E	136
_	same	137
`	$9A	140
a to z	same	141 to 172
{	$9B	173
|	$9C	174
}	$9D	175
~	$9E	176
DEL	$9F	177
octal 200 to ¿	$200 to $277	200 to 277 (Multinational)
À to octal 377	$300 to $377	300 to 377 (Multinational)

The SRI mapping filename translation rules in the below table are based on the character mapping scheme from above.

Rule	**What happens to filenames** on **OpenVMS...**
1	Lowercase characters become uppercase (unless Rule 2 applies; see also Rule 3): `foobar.txt` becomes `FOOBAR.TXT;1`
2	Initial uppercase characters or a sequence of case-shifted characters get a `$` prefix:

	`CaseShiftedFile` becomes `$C$ASE$S$HIFTED$F$ILE.;1`
3	An unversioned file gets a version number preceded by a semicolon: `foobar.txt` becomes `FOOBAR.TXT;1`
4	If a filename does not include a file extension dot (`.`), it gets one before the version number semicolon: `foobar` becomes `FOOBAR.;1`
5	The first dot in a filename is preserved, unless the result fails the 39-character extension limit test in Rule 5 (if so, the dot becomes `$5N`). Each successive dot becomes `$5N`, unless the filename exceeds the limits in Rule 5. `more.file.txt` becomes `MORE.FILE$5NTXT;1`
6	If the filename is a directory name, each dot in it becomes `$5N` and the filename gets the `.DIR` extension: `dot.directory.list` becomes `DOT$5NDIRECTORY$5NLIST.DIR;1`
7	Invalid OpenVMS characters become the escape character sequences in the second column of the above table (`$` followed by a digit and a letter): `special#character&file` becomes `SPECIAL$5CCHARACTER$5FFILE.;1` (`#` becomes `$5C` and `&` becomes `$5F`)
8	Any existing `$` becomes `$$` (plus any `$` added due to Rule 2 or 8 above): `dollar$Sign$5cfile` becomes `DOLLAR$$$S$IGN$$5CFILE.;1`