HP TCP/IP Services for OpenVMS Guide to SSH

HP Part Number: BA548-90007
Published: July 2006, Edition 2.0

Legal Notices

Table of Contents

About This Document....7
1 Intended Audience....7
2 New and Changed Information in This Edition....7
3 Typographic Conventions....7
4 Document Organization....8
5 Related Information....8
6 Publishing History....9
7 HP Encourages Your Comments....9

1 Overview....11
1.1 Introduction to SSH....11
1.1.1 The Secure Shell Server....11
1.1.2 The Secure Shell Client....11
1.2 Introduction to Keys....12
1.2.1 Host Keys....12
1.2.2 User Keys....12
1.2.3 Generating Keys....12
1.2.4 Managing User Keys....12
1.3 SSH Authentication....12
1.4 How the SSH Client and Server Communicate....13
1.5 Port Forwarding....14
1.6 Upgrading SSH for OpenVMS....14
1.6.1 Server and Client Configuration Files....14

2 Configuring the Secure Shell Software....15
2.1 Running the TCPIP$CONFIG Configuration Command Procedure....15
2.2 Configuring the SSH Client....16
2.3 Configuring the SSH Server....17

3 Customizing the SSH Run-Time Environment....19
3.1 Setting up the Server Public Keys....19
3.1.1 Automatically Copying Key Files....19
3.1.2 Manually Copying the Key Files....20
3.1.3 Naming Conventions for the Server's Public Host Key....20
3.2 Modifying Client Configuration Parameters....21
3.3 Modifying Server Configuration Parameters....21
3.4 Authentication Methods....21
3.5 Specifying Authentication Methods....22
3.6 Setting Up Password Authentication....23
3.6.1 Setting Up Password Authentication on the Client....23
3.6.2 Setting Up Password Authentication on the Server....23
3.7 Setting Up Host-based Authentication....23
3.7.1 Setting Up Host-based Authentication on the Client....23
3.7.2 Setting Up Host-based Authentication on the Server....23
3.8 Setting Up Public-Key Authentication....24
3.8.1 Setting Up Public-Key Authentication on the Client....24
3.8.1.1 Using SSH_KEYGEN to Rename Public-Key Files....25
3.8.2 Setting Up Public-Key Authentication on the Server....25

4 Managing the SSH Service....27
4.1 Starting and Stopping the SSH Client....27

4.2 Starting and Stopping the SSH Server....27
4.3 Enabling IPv6 Networking....27
4.4 SSH Logical Names....28
4.5 Managing Auditing....28
4.5.1 How the Server Performs Auditing....28
4.5.2 Auditing Options for the Server Configuration File....29
4.5.3 Auditing Options for the Client Configuration File....29
4.6 Managing Account Passwords....29
4.6.1 Secondary Passwords....30

5 Port Forwarding....31
5.1 Standard Port Forwarding....31
5.2 Port Forwarding for FTP....32
5.3 X11 Port Forwarding....33
5.4 Managing Port Forwarding....34

6 Setting Up Kerberos/SSH Connections....35
6.1 Installing Kerberos RTL Images....35
6.2 Setting Up Kerberos....35
6.2.1 Configuring the Kerberos User and Host Principles....35
6.2.2 Configuring the Kerberos Authentication Method....36
6.2.3 SSH Kerberos Authentication Interoperability....36
6.2.4 SSH Client Configuration....36
6.3 Setting Up a Kerberos SSH Connection....36
6.3.1 Forwarding Credentials....36
6.3.2 Managing Kerberos Credential Forwarding....37
6.3.3 Forwarding Kerberos Credentials Example....37
6.4 Kerberos Password Authentication....38
6.5 Solving SSH/Kerberos Problems....38

7 SSH Command Reference....41
7.1 Before You Begin....41
7.2 Copying Files....41
7.2.1 File Types....41
7.2.2 Using the SCP Command....41
7.2.2.1 Command Synopsis....42
7.2.2.2 Parameters....42
7.2.2.3 Options....42
7.2.2.4 Example....42
7.2.3 Using the SFTP Command....43
7.2.3.1 Command Synopsis....43
7.2.3.2 Parameters....43
7.2.3.3 Options....43
7.2.3.4 Example....43
7.3 Remote Login and Command Execution....44
7.3.1 Command Synopsis....44
7.3.2 Parameters....44
7.3.3 Options....44
7.3.4 Example....45
7.4 Using SSH Commands in Batch Jobs....45
7.5 Using the SSH_KEYGEN Utility....46
7.5.1 Command Synopsis....46
7.5.2 Parameters....46
7.5.3 Options....46
7.6 Using the SSH_ADD Utility....46
7.6.1 Command Synopsis....46
7.6.2 Parameters....46

7.6.3 Options....47
7.6.4 Description....47
7.6.5 Return Status....47
7.6.6 Example....47
7.7 Using the SSH_AGENT Utility....47
7.7.1 Command Synopsis....48
7.7.2 Options....48
7.7.3 Examples....48

8 Solving SSH Problems....49

A SSH Directories and Files....51
A.1 Client Directories and Files....51
A.2 Server Directories and Files....52

B SSH Client and Server Configuration Parameters....55
B.1 Client Configuration Parameters....55
B.2 Client Configuration File....56
B.3 Server Configuration Parameters....58
B.4 Server Configuration File....61

Glossary....65

Index....67

About This Document

1 Intended Audience

This guide is intended for Secure Shell users and for system managers who need to configure, customize, manage, and use SSH. The *Guide to SSH* assumes that you are familiar with:

- OpenVMS concepts and operation
- TCP/IP Services installation, configuration, and management
- SSH concepts and utilities

2 New and Changed Information in This Edition

This document supersedes the TCP/IP Services for OpenVMS *Guide to SSH* for Version 5.4 (AA-RVBUA-TE).

This manual is updated with the following features and information:

- Kerberos authentication, documented in *Setting Up Kerberos*.
- Enhancements to SSH for OpenVMS, including:
  - Support for IPv6
  - Command options for port forwarding
  - SSH commands in batch jobs
  - SSH command options
  - Server and client configuration files
  - Account passwords for non-OpenVMS clients
  - Secondary passwords

If you are upgrading from an earlier version of SSH for OpenVMS, see "Upgrading SSH."

3 Typographic Conventions

Book Title	Title of a book. On the web and on the Instant Information DVD, it may be a hot link to the book itself.
`Command`	Command name or qualified command phrase.
`ComputerOut`	Text displayed by the computer.
Emphasis	Text that is emphasized.
Emphasis	Text that is strongly emphasized.
KeyCap	Name of a keyboard key. Note that **Return** and **Enter** both refer to the same key.
Term	Defined use of an important word or phrase.
`UserInput`	Commands and other text that you type.
`Variable`	Name of a variable that you may replace in a command or function or information in a display that represents several possible values.
[ ]	Contents are optional in formats and command descriptions. If the contents are a list separated by |, you must choose one of the items.
{ }	Contents are required in formats and command descriptions. If the contents are a list separated by |, you must choose one of the items.
...	Preceding element may be repeated an arbitrary number of times.
|	Separates items in a list of choices.
`<element>`	An element used in a markup language.
`attribute=`	An attribute used in a markup language.

4 Document Organization

This guide describes how to configure, customize, manage, and use the Secure Shell software. It contains the following chapters and appendixes:

- Chapter 1 introduces definitions and concepts that are important to understanding the Secure Shell (SSH).
- Chapter 2 describes how to run the configuration procedure for SSH, how to configure the SSH server, and how to configure the SSH client.
- Chapter 3 describes how to customize the SSH run-time environment to meet your organization's specific security needs.
- Chapter 4 describes how to manage the SSH client and server.
- Chapter 5 describes port forwarding with SSH.
- Chapter 6 describes how to set up Kerberos security with SSH connections.
- Chapter 7 describes SSH commands and utilities that you can use to invoke SSH, copy files, and manage keys.
- Chapter 8 describes how to solve login problems.
- Appendix A summarizes information about files and directories that the SSH client and server use.
- Appendix B shows the systemwide SSH client and server files that the TCPIP$CONFIG utility generates during configuration.
- Glossary describes some of the terms that are used in this manual.

5 Related Information

The following manuals describe how to install, customize, and use TCP/IP Services:

- *Compaq TCP/IP Services for OpenVMS Concepts and Planning*

  This manual provides conceptual information about TCP/IP networking on OpenVMS systems, including general planning issues to consider before configuring your system to use the TCP/IP Services software.

  This manual also describes the manuals in the TCP/IP Services documentation set and provides a glossary of terms and acronyms for the TCP/IP Services software product.

- *HP TCP/IP Services for OpenVMS Release Notes*

  The release notes provide version-specific information that supersedes the information in the documentation set. The features, restrictions, and corrections in this version of the software are described in the release notes. Always read the release notes before installing the software.

- *HP TCP/IP Services for OpenVMS Installation and Configuration*

  This manual explains how to install and configure TCP/IP Services.

- *HP TCP/IP Services for OpenVMS User's Guide*

  This manual describes how to use the applications available with TCP/IP Services such as remote file operations, e-mail, TELNET, TN3270, and network printing.

- *HP TCP/IP Services for OpenVMS Management*

  This manuals describes how to configure and manage the TCP/IP Services product.

- *HP TCP/IP Services for OpenVMS Management Command Reference*

  This manual describes TCP/IP Services management commands.

- *HP TCP/IP Services for OpenVMS Management Command Quick Reference Card*

  This reference card lists the TCP/IP management commands by component and describes the purpose of each command.

- *HP TCP/IP Services for OpenVMS UNIX Command Equivalents Card*

  This reference card contains information about commonly performed network management tasks and their corresponding TCP/IP management and UNIX command formats.

- *HP TCP/IP Services for OpenVMS ONC RPC Programming*

  This manuals presents an overview of high-level programming using open network computing remote procedure calls (ONC RPCs). This manual also describes the RPC programming interface and how to use the RCPGEN protocol compiler to create applications.

- *HP TCP/IP Services for OpenVMS Guide to SSH*

  This manual describes how to configure, set up, use, and manage the SSH for OpenVMS software.

- *HP TCP/IP Services for OpenVMS Sockets API and System Services Programming*

  This manual describes how to use the Sockets API and OpenVMS system services to develop network applications.

- *HP TCP/IP Services for OpenVMS SNMP Programming and Reference*

  This manual describes the Simple Network Management Protocol (SNMP) and the SNMP application programming environment (eSNMP). It describes the subagents provided with TCP/IP Services, utilities provided for managing subagents, and how to build your own subagents.

- *HP TCP/IP Services for OpenVMS Tuning and Troubleshooting*

  This manual provides information about how to isolate the causes of network problems and how to tune the TCP/IP Services software for the best performance.

- *HP TCP/IP Services for OpenVMS Guide to IPv6*

  This manual describes the IPv6 environment, the roles of systems in this environment, the types and function of the different IPv6 addresses, and how to configure TCP/IP Services to access the IPv6 network.

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

`http://www.hp.com/go/openvms`

For a comprehensive overview of SSH, refer to the book:

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

O'Reilly and Associates. January 2001.

6 Publishing History

Manufacturing Part Number	Supported Operating Systems	Supported Versions	Edition Number	Publication Date
AA–RVBUA–TE	OpenVMS Alpha Versions 7.3-1 and 7.3-2	TCP/IP Services for OpenVMS Version 5.4	1.0	September 2003
BA548–90007	OpenVMS Alpha and I64 Versions 8.2 and 8.3	TCP/IP Services for OpenVMS Version 5.6	2.0	July 2006

7 HP Encourages Your Comments

HP encourages your comments concerning this document. We are truly committed to providing documentation that meets your needs.

Please submit comments to:

http://docs.hp.com/assistance/feedback.html, or email your comments to openvmsdoc@hp.com

Please include the document title, manufacturing part number, and any comment, error found, or suggestion for improvement you have concerning this document.

1 Overview

This chapter describes how SSH works on OpenVMS, including a review of some concepts that are important for understanding the Secure Shell (SSH).

The following topics are covered:

- Introduction to SSH
- Introduction to Keys
- SSH Authentication
- How the SSH Client and Server Communicate
- Port Forwarding
- Upgrading SSH for OpenVMS

1.1 Introduction to SSH

Secure Shell is a combination of client and server software that transparently encrypts and decrypts data flow between hosts on a network. SSH provides a suite of secure network commands that you can use in addition to, or in place of, traditional nonsecure network commands like `TELNET` and `FTP`.

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

- Authentication – Secure Shell servers and clients use an authentication method to reliably determine each other's identity and the user's identity.
- Data encryption – Secure Shell servers and clients exchange encrypted data. Data encryption is transparent to the user.

1.1.1 The Secure Shell Server

A **Secure Shell server** (SSH server) is a system on which the system manager installs and runs the Secure Shell server software.

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

The SSH server provides the following functions:

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

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

1.1.2 The Secure Shell Client

A **Secure Shell client** (**SSH client**) is a system on which the system manager installs the Secure Shell client software.

SSH commands invoke the following SSH utilities:

- The `SCP` and `SFTP` commands copy files to and from an SSH server.
- The `SSH` command logs in to a remote server and performs remote command execution (tunnelling).
- The SSH key management utilities generate public-private key pairs and manipulate keys.

These commands and utilities are described in Chapter 7.

NOTE: SSH for OpenVMS software is based on SSH2 software from SSH Communication Security version 3.2.1. In the OpenVMS implementation, the commands `SSH`, `SCP`, and `SFTP` mean the same as `SSH2`,

SCP2, and SFTP2. You can use either set of commands with SSH for OpenVMS. For more information about these commands, enter the DCL HELP command. For example:

```
$ HELP SSH
```

1.2 Introduction to Keys

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

1.2.1 Host Keys

The SSH host public and private keys are asymmetric keys that distinguish and identify hosts. Specifically:

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

Host keys are created either during TCP/IP configuration by the TCPIP$CONFIG.COM command procedure, or manually by a system manager, using the SSH_KEYGEN utility, as described in Chapter 7.

NOTE: SSH for OpenVMS is configured with a single SSH service listening port (22) and a single host key. All incarnations of the SSH server process use the same host key.

1.2.2 User Keys

Public key authentication requires that a user also have a public-private key pair. The public key is published and distributed, or copied, to all the SSH servers with which the user communicates. The private key is kept on the local SSH client and must not be revealed to anyone except the key's owner. The user creates the public-private key pair by using the SSH_KEYGEN key generation utility (described in Chapter 7). The user's keys are used during public-key authentication. For information about the public-key authentication method, see Chapter 3.

1.2.3 Generating Keys

Key are generated by using the SSH_KEYGEN utility, as described in Chapter 7. SSH_KEYGEN generates both user's keys and host keys. For each key, the SSH_KEYGEN utility generates a pair of files: one with a public key and one with a private key. These files are used by cryptographic algorithms.

1.2.4 Managing User Keys

A user might need several, even hundreds of keys. For example, you might use one key for each remote server to which you connect, or one key for each account on a remote server. The following utilities are available to help manage multiple keys:

- SSH_AGENT helps you manage and use keys.
- SSH_ADD helps you add private keys to the authentication agent.

For more information about these utilities, see Chapter 7.

1.3 SSH Authentication

Every SSH connection involves two types of authentication:

- Server authentication

  For server authentication, the client verifies the identity of the SSH server. The SSH server authentication process uses the server's host public key to ensure that the SSH server is not an impostor.

- User authentication

  For user authentication, the server verifies the identity of the user requesting access. The user authentication process uses the authentication methods specified in the server configuration file to verify the user's identity.

You enable the authentication methods by editing the server and client configuration files, as described in Chapter 3.

If your server also runs Kerberos for OpenVMS, you can enable Kerberos-based authentication, as described in Chapter 6.

1.4 How the SSH Client and Server Communicate

During SSH client and server configuration, two configuration files are installed: a client configuration file, which is read by an SSH client process when the SSH command is invoked; and the server configuration file, which is read by an SSH server process when a connection request arrives from an SSH client. All configuration files are ASCII text files and have either STREAM_LF format (for example, if copied directly from a UNIX system), or variable-length format (if created with the TCPIP$CONFIG.COM command procedure or with a text editor). Appendix B shows the SSH client and server configuration files.

After you install and configure the SSH software on all client and server hosts:

- Specify the authentication methods on the clients and server.
- Create and distribute key files.
- Start the SSH client and server.

When TCP/IP Services is started on an SSH server host, the auxiliary server creates a listening socket for SSH. The SSH server is now ready to accept a remote connection request. When you execute an SSH command on a remote client host, the SSH client is initiated. The client reads the configuration file and initiates a TCP connection to a server host using the specified destination port. On an SSH server host, the auxiliary server creates a copy of the server process, which reads the server's configuration file.

To establish a secure connection:

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

   ```
   Host key not found from the list of known hosts.
   Are you sure you want to continue connecting (yes/no)
   ```

   (Note that the user response is case sensitive. Enter the response in lowercase letters.)
3. If the user response is `yes`, SSH copies the server's public host key to the SSH client host. The client host uses this public host key to authenticate the SSH server on subsequent connections.
4. If, during subsequent connection attempts, the SSH client detects that the SSH server's host key differs from the one stored on the client, the following message is displayed:

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

   The message continues with text that warns of a possible "man-in-the-middle" attack. This message does not necessary mean that data has been compromised. The host key may have been legitimately changed. The user should copy the server's new key or contact the system manager.
5. The SSH client and server negotiate session parameters by exchanging information about supported parameters, including authentication methods, hash functions, and data compression methods.

6. The SSH client and server develop a shared (symmetric) session key to encrypt data using a key exchange method. When both the client and server know the secret data encryption key, a secure connection is established and the client and server can exchange data securely. The session key expires either when the session ends or when the session timer expires and a rekey operation is done.
7. After the SSH client and server authenticate each other, the session is ready to authenticate the user by applying one or more of the authentication methods.
8. The SSH server checks the user's identity. The user must have a valid user account and home directory on the server. If the server fails to authenticate the user, the server refuses the connection.
9. After SSH authenticates the user's identity, the actual secure data transfer between client and server occurs.
10. The SSH server runs in a loop, accepting messages from the client, performing required actions, and returning reply messages to the client.
11. When the user closes the connection, the server process terminates. The auxiliary server continues to listen for new SSH connection requests.

1.5 Port Forwarding

Port forwarding encapsulates the TCP-based communication session between the SSH client and the SSH server programs. Any TCP-based application or service can use port forwarding to take advantage of all the benefits of SSH. Using port forwarding, SSH allows you to establish a "secure tunnel" between two hosts, through which the participating applications operate transparently. For example, when you forward a regular TELNET connection through SSH, all information, including your user name, password, and actual data, are automatically encrypted and checked for integrity.

X11 port forwarding encrypts the X protocol (for X Window Systems). Using SSH, you can invoke X programs on a remote machine and have them appear on your local display. In this case, all X-protocol data is secured. For more information, see Chapter 5.

1.6 Upgrading SSH for OpenVMS

This version of SSH for OpenVMS is based on Version 3.2 of the SSH product offered by SSH Communications, Inc. If you are upgrading SSH for OpenVMS from the version available on TCP/IP Services for OpenVMS Version 5.4 and lower, read the product release notes to understand the changes in setting up and managing this version of SSH.

1.6.1 Server and Client Configuration Files

The server and client configuration files in this version of SSH are different in several ways from those used with earlier versions of SSH for OpenVMS. When you upgrade to the current version, you must recreate both the server and client configuration files.

For this version of SSH, the server configuration file is:

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

The client configuration file is:

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

NOTE: When you extract these files, make sure to specify the final dot in the file name. For example:

```
$ LIBRARY/EXTRACT=sshd2_config/out=sshd2_config. SYS$LIBRARY:TCPIP$TEMPLATES.TLB
```

For more information about recreating the configuration files, refer to the product release notes.

Some of the configuration keywords have changed in this version of SSH. For more information, see Appendix B.

2 Configuring the Secure Shell Software

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

This chapter covers the following topics:

- Running the TCPIP$CONFIG Configuration Procedure
- Configuring the SSH Client
- Configuring the SSH Server

2.1 Running the TCPIP$CONFIG Configuration Command Procedure

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

Once you complete the client and server configuration using TCPIP$CONFIG, you can customize the configuration with parameters to meet the needs of your specific run-time environment. For more information about customizing your run-time environment, see Chapter 3. For general configuration procedures, refer to the HP TCP/IP Services for OpenVMS *Installation and Configuration* manual.

To run the configuration command procedure, follow these steps:

1. Invoke the TCPIP$CONFIG configuration command procedure. The main Configuration menu is displayed:

```
$ @ SYS$STARTUP:TCPIP$CONFIG.COM
 TCP/IP Network Configuration Procedure

        This procedure helps you define the parameters required
        to run HP TCP/IP Services for OpenVMS on this system.


      Checking TCP/IP Services for OpenVMS configuration database files.




        HP TCP/IP Services for OpenVMS Configuration Menu

        Configuration options:

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

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

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

```
Enter configuration option:
```

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

During the configuration procedure, TCPIP$CONFIG creates the systemwide environment necessary to run the SSH client and server, including:

- The SSH server account TCPIP$SSH, and the account's default directory, TCPIP$SSH_DEVICE:[TCPIP$SSH]. Note that the default device of the account is defined by the logical name TCPIP$SSH_DEVICE. This logical name can be assigned by the system manager. If this logical name is not defined, the default name is SYS$SYSDEVICE.
- All subdirectories and files required by the SSH server.

In addition, the configuration procedure copies all the necessary files from the distribution kit into the appropriate directories. Table 2-1 lists the files that are created during the configuration of the SSH server and SSH client.

Table 2-1 Files and Directories Created During SSH Configuration

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

2.2 Configuring the SSH Client

When you choose Client components from the TCPIP$CONFIG Main Menu, the Client Components Configuration Menu is displayed:

```
HP TCP/IP Services for OpenVMS Client Components Configuration Menu
Configuration options:
1  -  DHCP Client      Disabled Stopped
2  -  FTP Client       Enabled  Stopped
3  -  NFS Client       Disabled Stopped
4  -  REXEC and RSH    Disabled Stopped
5  -  RLOGIN           Disabled Stopped
6  -  SMTP             Disabled Stopped
7  -  SSH Client       Disabled Stopped
8  -  TELNET           Enabled  Stopped
9  -  TELNETSYM        Disabled Stopped

A  -  Configure options 1 - 9
[E] -  Exit menu
Enter configuration option: 7
```

Configure the SSH client as described in the following procedure.

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

```
SSH CLIENT Configuration
Service is not defined in the SYSUAF.
Service is not enabled.
Service is stopped.

  SSH CLIENT configuration options:
     1 - Enable service on this node
     2 - Enable & Start service on this node
     [E] - Exit SSH_CLIENT configuration
Enter configuration option:
```

2. Choose the appropriate menu option. For example, choose configuration option 1 to enable the SSH client on this node. The configuration procedure copies the systemwide client configuration file SSH2_CONFIG. from the distribution kit into the directory TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2].

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

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

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

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

If you want to configure the SSH server, type YES. Otherwise, press Enter or type NO.

2.3 Configuring the SSH Server

When you choose Server components from the TCPIP$CONFIG command procedure Main Menu, the Server Components Configuration Menu is displayed:

```
HP TCP/IP Services for OpenVMS Server Components Configuration Menu
Configuration options:
1 - BIND          Disabled Stopped       12 - NTP          Disabled Stopped
2 - BOOTP         Disabled Stopped       13 - PC-NFS       Disabled Stopped

3 - DHCP          Disabled Stopped       14 - POP          Disabled Stopped

4 - FINGER        Disabled Stopped       15 - PORTMAPPER   Disabled Stopped

5 - FTP           Disabled Started       16 - RLOGIN       Disabled Started

6 - IMAP          Disabled Stopped       17 - RMT          Disabled Stopped

7 - LBROKER       Disabled Stopped       18 - SNMP         Disabled Started

8 - LPR/LPD       Disabled Stopped       19 - SSH          Enabled  Started

9 - METRIC        Disabled Stopped       20 - TELNET       Disabled Started
10 - NFS          Disabled Stopped       21 - TFTP         Disabled Stopped
11 - LOCKD/STATD  Disabled Stopped       22 - XDM          Disabled Stopped


A  -  Configure options 1 - 22
```

```
[E]   Exit menu

Enter configuration option: 19
```

Configure the SSH server as described in the following procedure.

1. Enter option `19` (SSH configuration) at the prompt. The SSH Configuration Option menu appears.

```
SSH Configuration
Service is defined in the SYSUAF.
Service is defined in the TCPIP$SERVICE database.
Service is enabled on specific node.
Service is started.

  SSH configuration options:
     1 - Enable service on all nodes
     2 - Disable service on this node

     3 - Stop service on this node
     4 - Disable & Stop service on this node
     [E] - Exit SSH configuration

Enter configuration option:
```

2. Choose the appropriate menu option. For example, choose option 1 to enable SSH on this server. The configuration procedure creates the SSH service entry and server configuration file:

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

3. Respond to the following question:

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

   Type `YES` or press Enter to create new host key pair files, HOSTKEY and HOSTKEY.PUB. The TCPIP$CONFIG command procedure creates the default key pair in the directory TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2].

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

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

```
The SSH CLIENT is not enabled.
```

   If you want to configure the SSH client, type `YES`.

3 Customizing the SSH Run-Time Environment

When the TCP/IP configuration procedure has completed successfully, all of the required systemwide SSH configuration parameters are established. The host is now prepared to become an SSH server by accepting remote connections, and the SSH client is ready to execute SSH commands. Additional security methods can be applied at the following levels:

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

This chapter describes how to customize the SSH run-time environment to meet your specific security requirements, and discusses the following topics:

- Setting up the Server Public Keys
- Modifying Client Configuration Parameters
- Modifying Server Configuration Parameters
- Authentication Methods

3.1 Setting up the Server Public Keys

Any connection request from a client to an SSH server requires that the client obtain the server's public key.

During SSH server configuration, the TCPIP$CONFIG configuration procedure creates the systemwide directory TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.HOSTKEYS]. On a connection request, the SSH client checks this directory for the appropriate server's host key and proceeds with authentication if the key is found.

You can copy into this directory the host keys of all the remote servers to which you will connect from the client host. You can specify the public key file to copy from the client when you make the first connection to the server, or you can copy the public key files manually.

3.1.1 Automatically Copying Key Files

On a connection request, the SSH client checks the user's [*username*.SSH2.HOSTKEYS] directory for the appropriate server's host key and proceeds with authentication if the key is found. If the file is not found in either the systemwide or account-specific [*username*.SSH2.HOSTKEYS] directory, the first time that the user makes a connection from the SSH client to a remote SSH server, the user is prompted to accept a copy of the server's public host key (by default). This behavior can be controlled using the `StrictHostKeyChecking` parameter in the client configuration file. This parameter accepts the following values:

- `yes` -- Causes authentication to fail if the file is not found.
- `no` – Causes the SSH client to create the [*username*.SSH2.HOSTKEYS] subdirectory (if it does not exist), and copies the SSH server's public key file into this subdirectory automatically.
- `ask` -- Causes the SSH server to prompt the user for a copy of the server's public host key. This is the default.

If the `StrictHostKeyChecking` parameter is set to `ask`, and if the server's host key is in neither the server nor the client's HOSTKEYS directory, the following message is displayed:

```
Host key not found from database.
Key fingerprint:
xikan-rokyr-miduc-zofut-nysig-ciryt-pyroc-fegil-zadyb-cokel-loxex
You can get a public key's fingerprint by running
$ ssh_keygen "-F" publickey.pub on the keyfile.
Are you sure you want to continue connecting (yes/no)?
```

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

If you do not specify the `StrictHostKeyChecking` option, the default is `ask`.

SSH uses the [.SSH2] subdirectory to store multiple files needed for SSH to function. This directory is created automatically by the SSH software, or the user can create one in the login directory (as defined by SYS$LOGIN). For example, if SYS$LOGIN is defined as DKA0:[*username*], then the SSH2 subdirectory is DKA0:[*user-name*.SSH2]. Throughout this manual, this directory is referred to as the [*username*.SSH2] directory.

If necessary, you can create an SSH2 subdirectory in your user directory by entering the following commands:

```
$ SET DEFAULT SYS$LOGIN$
$ CREATE/DIR [.SSH2]
```

3.1.2 Manually Copying the Key Files

If your SSH client host does not have keys from remote servers in the systemwide directory, you can copy the keys manually, as follows:

1. Create the subdirectory [.HOSTKEYS] in the user's SSH directory.
2. Copy the server's public key to this directory using the COPY/FTP command. Specify the proper access privileges (for example: `S:RWED,O:RWED,G:RE,W:R`).

   The format for the file name is KEY_*portnumber_hostname*.PUB, where:

   - *portnumber* is the port number to be used to connect to the SSH server (22 by default).
   - *hostname* is the host name used to connect to the SSH server (either *hostname* or *hostname*.COM. Any dots in the host name are converted to underscore characters (`myhost.com` becomes `myhost_com`).

If necessary, the file format is converted to Stream_LF format and the following message is displayed:

```
$ ssh VMSHOST
warning: Converting file ssh2/hostkeys/key_22_vmshost.pub to Stream_LF.
warning: File ssh2/hostkeys/key_22_vmshost.pub converted successfully to Stream_LF.
Authentication successful.
```

If the public key file cannot be converted, the following error message is displayed:

```
Error calling CONV$PASS_FILES for ssh2/hostkeys/key_22_vmshost.pub.
Status = {status text}
```

In this case, you can convert the file to Stream_LF format using the following command:

```
$ convert/fdl=SYS$SYSTEM:TCPIP$CONVERT.FDL keyfilename.PUB keyfilename.PUB
```

Where *keyfilename* is *device*:[*username*.SSH2.HOSTKEYS]KEY_22_*remotename*.PUB.

3.1.3 Naming Conventions for the Server's Public Host Key

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

```
KEY_port_hostname.PUB
```

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

For example, connect to an SSH server using the following command:

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

This command copies the remote SSH server's public key file HOSTKEY.PUB into a local directory as a file named KEY_22_MYSERVER_MYLAB_COM.PUB. Note that underscores replace the dots in the destination file.

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

3.2 Modifying Client Configuration Parameters

During configuration, the SSH2_CONFIG. file is copied to TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]. When the user invokes the SSH command, the SSH client process reads the file and creates the run-time version of the configuration parameters. If you want to specify user-specific parameters, you can create a client configuration file in the [*username*.SSH2] directory. You can copy this file from a UNIX or an OpenVMS system and then edit it, or create a new file from the template supplied by TCP/IP Services. The file can be in either Stream_LF or variable-length format. For more information about the client configuration file, see Appendix B.

3.3 Modifying Server Configuration Parameters

During configuration, the SSHD2_CONFIG. file is copied to the TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2] directory. When the connection is initiated from a remote client, the SSH server reads the file and creates the run-time version of the configuration parameters. The SSH server loads this file and modifies the run-time version of the parameters accordingly. You can copy this file from a UNIX or OpenVMS system and edit it, or you can create a new file. The file can be in either Stream_LF or variable-length format.

3.4 Authentication Methods

Before it makes a connection, the SSH server determines the authentication methods that it will use by looking in the server configuration file. For the SSH client to connect to the SSH server, it must find the same authentication method in the client configuration file. Therefore, each of these methods requires configuration on both the SSH client and server.

After the SSH client makes a connection request to a remote SSH server, the server sends the client its permitted authentication methods. Depending on the agreed upon authentication method, the SSH server may require the client to pass multiple authentication tests before connecting.

To configure the SSH client to use an authentication method, specify the authentication method in the client configuration file in either the systemwide client configuration file (TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]SSH2_CONFIG.) or the user-specific client configuration file ([*username*.SSH2]SSH_CONFIG.). The authentication methods can be specified as arguments to the following configuration parameters in the SSH server configuration file:

- `AllowedAuthentications`
- `AccountingAuthentications`
- `AllowedAuthentications`
- `IntrusionAuthentications`
- `IntrusionIdentMethod`
- `IntrusionIdentSSH`
- `LogfailAuthentications`

The following SSH authentication methods are available:

- **Password authentication**. This method requires that you supply a password to the client, which transmits the password encrypted to the server over the network. Then the SSH server performs authorization, verifying the supplied password using the OpenVMS native password-authentication mechanism.
- **Host-based authentication**. This method allows you to avoid specifying any secret information about the SSH client. Host-based authentication method trusts the relationships between hosts and does not require you to prove your identity.

  The SSH server host authenticates by verifying the following:

  - The identity of the client host using the client's host public key file, which the system manager maintains in the **known hosts database**. The directory [TCPIP$SSH.SSH2.KNOWNHOSTS] contains public keys for all client hosts that use the host-based authentication method to connect to the server.
  - That the client host belongs to the **trusted hosts** list, which the system manager maintains on the server. This list of trusted hosts enables you to log in to the server without proving your identity.
  - Optionally, you can restrict users to only certain user names on the client host.

  If any of these authentication checks fail, the connection is refused. An advantage of this method is that it does not require the client to type a password or passphrases or to generate, distribute, and maintain keys. This is convenient for batch processing. One disadvantage, however, is a reliance on the identification of the host.

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

  - The known hosts database, which contains the public key files of remote hosts.
  - A trusted hosts file, which lists the trusted hosts (and, optionally, the user names).

- **Public-key authentication**. This method uses public-key cryptography to verify the client's identity and requires two pieces of data: your private-public key pair, and, optionally, a passphrase to encode this key for saving it in a file. This method is flexible because it allows additional control over authorization by providing multiple keys and by applying restrictions to each key.

  Public-key authentication requires management actions on both ends of an SSH connection. Both the user on the client host and the system manager on the server host must create and maintain keys on the client, copy public keys from the client to the server hosts, and provide passphrases.

You can also specify Kerberos authentication methods, as described in Chapter 6.

3.5 Specifying Authentication Methods

The authentication methods specified in the client configuration file (SSH2_CONFIG.) are attempted in the order in which they are listed for the `AllowedAuthentications` keyword. If `hostbased` is listed first, the SSH server tries `hostbased`authentication first.

For example, the client configuration file contains the following:

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

In this case, the server first tries to use hostbased, then public key, then password authentication. The first successful authentication is used.

The order of the authentication methods specified in the server configuration file (SSHD2_CONFIG.) is irrelevant. If the `AllowedAuthentications` keyword is missing or has no entries, the server accepts the following authentication methods:

- `hostbased`
- `publickey`
- `password`

3.6 Setting Up Password Authentication

To enable password authentication, set the `AllowedAuthentications` parameter in the SSH client host and server host configuration files to `password`. No additional parameters are required. Password authentication is the default.

3.6.1 Setting Up Password Authentication on the Client

Set the value of the `AllowedAuthentications` keyword to include the word `password`. For example:

```
AllowedAuthentications password
```

3.6.2 Setting Up Password Authentication on the Server

Set the following configuration parameters:

1. Specify the value of the `AllowedAuthentications` keyword as `password`, or omit the line. For example:

   ```
   AllowedAuthentications password
   ```

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

   ```
   PasswordGuesses 4
   ```

   The default is 3 password attempts.

3.7 Setting Up Host-based Authentication

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

3.7.1 Setting Up Host-based Authentication on the Client

Set the following configuration parameters:

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

  ```
  AllowedAuthentications hostbased
  ```

- Specify the value of the `DefaultDomain` keyword to be the fully-qualified domain name for the local host. For example, if the fully-qualified domain name for the local host is `color.art.com`, enter the following:

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

3.7.2 Setting Up Host-based Authentication on the Server

1. Edit the server configuration file as follows:
   - Set the value of the `AllowedAuthentications` parameter to include the word `hostbased`. For example:

     ```
     AllowedAuthentications hostbased
     ```

   - To enable use of the user-specific SHOSTS. files, set the value of the `IgnoreRhosts` parameter to `no`. For example:

     ```
     IgnoreRhosts no
     ```

     Because `no` is the default, the parameter can also be commented out, as follows:

```
#IgnoreRhosts no
```

If the `IgnoreRhosts` parameter is set to `no`, the SSH server looks up the host name in the user-specific SHOSTS. file. If this parameter is set to `yes`, the host name is assumed to be in the systemwide SHOSTS.EQUIV fle. See Appendix B for more information.

2. Edit the systemwide trusted hosts file, TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]SHOSTS.EQUIV, to add the fully qualified name of every SSH client host that will communicate with the server. You can also enter a specific user name to limit access to that user. For example:
   `MYHOST.MYLAB.COM`

   or

   `MYHOST.MYLAB.COM smith`

   If the `IgnoreRhosts` parameter is set to `no`, you can also add the client host and optional user names to the file SYS$LOGIN:SHOSTS. for a specific user.

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

3. In host-based authentication, the client and server hosts authenticate each other. Therefore, the server host must have the client's host public key. Copy the client's host public-key file (`CLIENTHOST::TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]HOSTKEY.PUB`) to the server directory (`SERVERHOST::TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.KNOWNHOSTS]`). Name the key file name using the following format:

   `fully-qualified-hostname_` `ssh-dss.pub.`

   Specify the proper protection for the new file using the /PROTECTION=W=RE qualifier to the COPY command. Without the proper protection, host-based authentication will not work.

   For example, if the host name is `green` and its domain name is `color.art.com`, copy the client's host public-key file as follows:

```
$ COPY SYS$LOGIN:[SSH2.KNOWNHOSTS]green_color_art_com_ssh-dss.pub -
_$ SERVERHOST::TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.KNOWNHOSTS]-
_$ green_color_art_com_ssh-dss.pub/PROTECTION=(W=RE)
```

4. If you want your own version of the host public key files in addition to the systemwide file, copy the file into your [*username*.SSH2.KNOWNHOSTS] directory. If the same file exists in both directories, the SSH server uses the user-specific key-name file in your user directory.

3.8 Setting Up Public-Key Authentication

Public-key authentication requires the following configuration actions on the client and server host sides of the connection.

1. Create public-private key pairs on the client host.
2. Install your public key in your accounts on all server hosts to which you want to connect. Your user account on each server host might have many public keys for accessing it in different ways.

3.8.1 Setting Up Public-Key Authentication on the Client

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

   ```
   AllowedAuthentications publickey, password
   ```

2. From the user account, run the `SSH_KEYGEN` utility, as described in Chapter 7. This action creates the public private key file. The default public key file name is [*username*.SSH2]ID_DSA_2048_A.

   The file contains your private key, which you must protect so that only you can access it. To protect the file, use the DCL command SET FILE/PROTECTION. For example:

   ```
   $ SET FILE/PROTECTION=(S,W,G,O:RW) ID_DSA_2048_A.
   ```

The [*username*.SSH2]ID_DSA_2048_A.PUB file contains your public key, which you can copy to other hosts. Ensure that this file is available for world read access.

3. Create a file named [*username*.SSH2]IDENTIFICATION. The IDENTIFICATION. file identifies your private-key file. For example, add the following line to the IDENTIFICATION. file if the name of your private-key file (as generated by the SSH_KEYGEN utility) is ID_DSA_2048_A:

   ```
   IdKey ID_DSA_2048_A
   ```

   The IDENTIFICATION. file tells the client which private keys are available for use in authenticating the server.

3.8.1.1 Using SSH_KEYGEN to Rename Public-Key Files

If you need multiple keys, use the SSH_KEYGEN utility to rename the public key files to be used with a particular SSH server host. Rename the public key files to file names in the following format: *username-serverhostname*.PUB.

Use the following format for the private key: *username-serverhostname*.

This convention makes it easier to copy designated public key files to the appropriate server hosts. For example, assume that the public and private key files have been either generated as or renamed to the file MEUSER-MYHOST_MYDOMAIN_COM.*. Create a file called [*username*.SSH2]IDENTIFICATION. and add a line that identifies the name of your private key. Add the following line to the IDENTIFICATION. file:

```
IdKey MEUSER-MYHOST_MYDOMAIN_COM
```

For more information about the SSH_KEYGEN utility, see Chapter 7.

3.8.2 Setting Up Public-Key Authentication on the Server

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

   ```
   AllowedAuthentications publickey
   ```

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

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

   For example, if the user's public key file name is MEUSER-MYHOST_MYDOMAIN_COM, add the following line to the AUTHORIZATION. file:

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

5. Copy the public key file to the server in the user's [*username*.SSH2] directory. Make sure the file is protected properly (`/PROTECTION=(S:WRED,O:WRED,G:RE,W:R)`).

4 Managing the SSH Service

This chapter describes how to manage the SSH client and the SSH server, and includes the following topics:

- Starting and Stopping the SSH Client
- Starting and Stopping the SSH Server
- Enabling IPv6 Networking
- SSH Logical Names
- Managing Auditing
- Managing Account Passwords

4.1 Starting and Stopping the SSH Client

To start the SSH client, enter the following command:

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

To stop the SSH client, enter the following command:

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

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

4.2 Starting and Stopping the SSH Server

To start the SSH server, enter the following command:

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

To stop the server, enter the following command:

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

If you enable the SSH service using the TCPIP$CONFIG.COM command procedure, the SSH server starts automatically when TCP/IP Services is started. To enable the SSH server, from the SSH Server Configuration menu, choose the `Enable service` option (1). For example:

```
SSH Configuration Service is not defined in the SYSUAF.
Service is defined in the TCPIP$SERVICE database.
Service is not enabled.
Service is stopped.

  SSH configuration options:
    Enable service on this node
    {E] - Exit SSH configuration
  Enter configuration option: 1
```

4.3 Enabling IPv6 Networking

This version of SSH for OpenVM supports IPv6 networking environments. To work in IPv6 environments, the SSH service must be set to IPv6. To display the setting for SSH, enter the following TCP/IP management command:

```
$ SHOW SERVICE SSH /FULL

Service: SSH
                           State:     Enabled
Port:                 22   Protocol:  TCP            Address:  0.0.0.0
Inactivity:            5   User_name: TCPIP$SSH      Process:  TCPIP$SSH
Limit:             10000   Active:        0          Peak:          1

File:          TCPIP$SYSTEM:TCPIP$SSH_RUN.COM
Flags:         Listen IPv6
```

```
Socket Opts:  Rcheck Scheck
 Receive:            0     Send:               0

Log Opts:     Acpt Actv Dactv Conn Error Exit Logi Logo Mdfy Rjct TimO Addr
 File:        TCPIP$SSH_DEVICE:[TCPIP$SSH]TCPIP$SSH_RUN.LOG

Security
 Reject msg:  TCPIP SSH Connection refused

 Accept host: 0.0.0.0
 Accept netw: 0.0.0.0
```

If the IPv6 flag is not included, enter the following command:

```
TCPIP> SET SERVICE SSH /FLAG=IPV6
```

For more information about these commands, see the TCP/IP Services for OpenVMS *Management Command Reference* guide.

4.4 SSH Logical Names

The logical names described in Table 4-1 can be used to modify the behavior of the SSH service.

Table 4-1 Logical Names

Name	Function
TCPIP$SSH_DEVICE	Defines the device on which the SSH client default directory is located. If you do not define this logical, the default is SYS$SYSDEVICE.
TCPIP$SSH_HOME	Defines the OpenVMS equivalent of the /etc directory on UNIX systems. Used internally by SSH software. Translates to TCPIP$SSH_DEVICE:[TCPIP$SSH].
TCPIP$SSH_AGENT_PORT	Defines the port on which an SSH authentication agent listens for connections from clients. Used internally by SSH software, and defined only in the job table for a process and its subprocesses.
TCPIP$SSH_CLIENT_PORT	Defines the port on which a client of an SSH authentication agent communicates with the agent. Used internally by SSH software, and defined only in the job table for a process and its subprocesses.
TCPIP$SSH_TOLERANT_PROTOCOL_STATUS	Suppresses the warning messages you usually receive when you use the OpenVMS SSH client to access the OpenSSH server.
TCPIP$SSH_SERVER_PARAMS	Specifies alternate operational control over the SSH server. The default is –1, which invokes the auxiliary server. Do not change this logical name.
TCPIP$SSH_SERVER_DEBUG	Specifies the level of debug information. By default, this logical name is set to 99.

4.5 Managing Auditing

The following sections describe the SSH server auditing functions and the configuration parameters that you can use to modify SSH auditing functions. For more information about the configuration parameters, see Appendix B.

4.5.1 How the Server Performs Auditing

When auditing is enabled for the specified authentication method, the SSH server performs the following functions depending on the type of login and whether the login attempt is successful.

When an interactive login is successful:

- The login failure count is set to 0.
- The last interactive login date is updated to the current date and time.
- If the user's password has expired but the user is not forced to change it before logging in, a warning message is displayed and the `pwd_expired` flag is not set in the user's SYSUAF record.
- The user is allowed three failed attempts to log in. If all three attempts fail, the login failure count is incremented by three.
- If the `AccountingAuthentications` keyword includes the current authentication method, the accounting data is updated.

When a remote command execution is successful, no updates are made to the user's SYSUAF record; thus:

- The login failure count is not changed.
- The last noninteractive login date is not updated.

If the user's password has expired but the user is not forced to change it before logging in, a warning message is displayed and the `pwd_expired` flag in the user's SYSUAF record is not set.

When the login or remote command execution fails:

- The login failure count in the user's SYSUAF record is incremented.
- If the `IntrusionAuthentications` keyword includes the current authentication method, the intrusion database is updated with text controlled by the `IntrusionIdentSsh` and `IntrusionIdentMethod` keywords in the server configuration file.
- If the `AccountingAuthentications` keyword includes the current authentication method, the accounting data is updated.

4.5.2 Auditing Options for the Server Configuration File

You can include the following options in the server configuration file (TCPIP$SSHD_CONFIG.) to control auditing functions.

- `AccountingAuthentications`
- `AllowNonvmsLoginWithExpiredPw`
- `IntrusionAuthentications`
- `IntrusionIdentMethod`
- `IntrusionIdentSsh`
- `LogfailAuthentications`
- `PubkeyPassphraseGuesses`
- `UserLoginLimit`

4.5.3 Auditing Options for the Client Configuration File

You can include the following options in the client configuration file (TCPIP$SSH_CONFIG.) to control auditing functions.

- `NumberOfHostkeyCopyPrompts`
- `NumberOfPasswordVerificationPrompts`
- `PubkeyPassphraseGuesses`

The configuration parameters are described in Appendix B.

4.6 Managing Account Passwords

Some SSH client implementations allow users to change expired account passwords on OpenVMS. This behavior is controlled in the server configuration file using the `AllowNonvmsLoginWithExpiredPwd` parameter, as described in Appendix B.

4.6.1 Secondary Passwords

Some SSH client implementations require users to enter a second password. To make the SSH server prompt for the second password, set the `NumberOfPasswordVerificationPrompts` parameter in the client configuration file to at least 2, as described in Appendix B.

5 Port Forwarding

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

X11 port forwarding encrypts the X protocol (for X Window Systems). Using SSH, you can invoke X Window programs on a remote machine and have them appear on your local display. In this case, all X-protocol data is secured.

This chapter describes the following:

- Standard Port Forwarding
- Port Forwarding for FTP
- X11 Port Forwarding
- Managing Port Forwarding

5.1 Standard Port Forwarding

Use local port forwarding when the client application is running on the same system as the SSH client. Use remote port forwarding if the client application is running on remotely. Local and remote port forwarding are specified using the `-"L"` and `-"R"` options.

- From an OpenVMS system to another OpenVMS system
  - Local port forwarding

    On system VMSHOST1, enter the following command:

    ```
    $ SSH -"L" 2001:localhost:23 VMSHOST2
    ```

    From another window on VMSHOST1, enter the following command:

    ```
    $ TELNET localhost 2001
    ```

    Result: The login prompt is displayed for VMSHOST2.
  - Remote port forwarding

    On system VMSHOST1, enter the following command:

    ```
    $ SSH -"R" 2001:localhost:23 VMSHOST2
    ```

    On system VMSHOST2, enter the following command:

    ```
    $ TELNET localhost 2001
    ```

    Result: The login prompt is displayed for VMSHOST1.
- From an SSH for OpenVMS client to a non SSH for OpenVMS server
  - Local port forwarding

    On system VMSHOST1, enter the following command:

    ```
    $ SSH -"L" 2001:localhost:23 NONVMSHOST
    ```

    From another window on VMSHOST1, enter the following command:

    ```
    $ TELNET localhost 2001
    ```

Result: The login prompt is displayed for NONVMSHOST.

- Remote port forwarding
  On system VMSHOST1, enter the following command:
  ```
  $ SSH -"R" 2001:localhost:23 NONVMSHOST
  ```

  On the system NONVMSHOST, enter the following command:
  ```
  # TELNET localhost 2001
  ```

  Result: The login prompt is displayed for VMSHOST1.

- From a non SSH for OpenVMS client to an SSH for OpenVMS server
  - Local port forwarding
    On system NONVMSHOST, enter the following commands:
    ```
    # SSH -L 2001:localhost:23 VMSHOST1

    # TELNET localhost 2001
    ```

    Result: The login prompt is displayed for VMSHOST1.
  - Remote port forwarding
    On system NONVMSHOST, enter the following command:
    ```
    # SSH -R 2001:localhost:23 VMSHOST1
    ```

    On system VMSHOST1, enter the following command:
    ```
    $ TELNET localhost 2001
    ```

    Result: The login prompt is displayed for VMSHOST1.

5.2 Port Forwarding for FTP

When you are connecting to an OpenVMS FTP server, in addition to the `-"L"` or `-"R"` options, you must specify the FTP protocol and set the connection to passive mode, as shown in the following examples.

- From an OpenVMS system to another OpenVMS system
  - Local FTP port forwarding
    On system VMSHOST1, enter the following commands:
    ```
    $ SSH -"L" ftp/2001:localhost:21 VMSHOST2
    ```

    From another window on VMHOST1, enter the following commands:
    ```
    $ FTP localhost 2001

    ftp> set passive on
    ```

    Result: A secure FTP connection is established.
  - Remote FTP port forwarding
    On system VMSHOST1, enter the following command:
    ```
    $ SSH -"R" ftp/2001:localhost:21 VMSHOST2
    ```

    On system VMSHOST2, enter the following command:

```
$ FTP localhost 2001

ftp> set mode passive
```

Result: The connection is made to VMSHOST1.

- SSH for OpenVMS client to a non-SSH for OpenVMS server
  - Local FTP port forwarding

    On system VMSHOST1, enter the following command:

    ```
    $ SSH -"L" ftp/2001:localhost:21 NONVMSHOST
    ```

    From another windows on VMSHOST1, enter the following command:

    ```
    $ FTP localhost 2001
    ```

    Result: The connection is made to NONVMSHOST.
  - Remote FTP port forwarding

    On system VMSHOST1, enter the following command:

    ```
    $ SSH -"R" ftp/2001:localhost:21 NONVMSHOST
    ```

    On system NONVMSHOST, enter the following command:

    ```
    # ftplocalhost 2001

    ftp> set mode passive
    ```

    Result: The connection is made to the VMSHOST1 system.
- Non SSH for OpenVMS client toSSH for OpenVMS server
  - Local FTP port forwarding

    On system NONVMSHOST, enter the following commands:

    ```
    # ssh -L ftp/2001:localhost:21 VMSHOST1
    # ftp localhost 2001

    ftp> set mode passive
    ```

    Result: The connection is made to the VMSHOST1 system.
  - Remote FTP port forwarding

    On system NONVMSHOST, enter the following command:

    ```
    # ssh -R ftp/2001:localhost:21 VMSHOST1
    ```

    On system VMSHOST1, enter the following command:

    ```
    $ FTP localhost 2001
    ```

    Result: The connection is made to the NONVMSHOST system.

5.3 X11 Port Forwarding

To use X11 port forwarding, include the `ForwardX11` in the client configuration file (SSH_CONFIG.), or use one of the following options to the `SSH` command:

- The `+x` option, which enables untrusted X11 connections (default)
- The `+X` option, which enables trusted X11 connections
- The `-x` option, which disables X11 connections

When X11 port forwarding is enabled on both the SSH client and server, you can use SSH to connect to an SSH server and invoke X11 client programs there, while having them appear on your local display. You can also "chain" port forwarding across multiple systems, even if the intermediate systems are not running the X11 server. For example, from SYSTEM1 you can use SSH to connect to SYSTEM2, and then from SYSTEM2 connect to SYSTEM3. An X11 client application running on SYSTEM3 will be displayed securely on SYSTEM1.

X11 access to an OpenVMS X11 server requires enabling access to the X11 client. On HP DECwindows Motif for OpenVMS Systems, this can be done through the Style Manager/security option:

1. Add the appropriate values for node and user name, and the value `tcpip` as the transport. Details of how to enable access on other platforms may differ.
2. To direct output to the forwarded X11 server port, enter the following command:

   ```
   $ SET DISPLAY/CREATE/TRANSPORT=TCPIP/NODE=Xserver
   ```

   For *Xserver*, specify the name of the SSH server. To direct output to the local SSH server, enter the keyword LOCAL. To verify that the settings are correct, enter the following commands:

   ```
   $ SHOW DISPLAY
       Device:    WSA777:  [user]
       Node:      sshclient.myplace.com
       Transport: TCPIP
       Server:    10
       Screen:    0$ SHOW LOGICAL DECW$DISPLAY
   ```

   For more details about the SET DISPLAY command, see the *OpenVMS DCL Dictionary*.
3. To terminate the display, exit the X11 client application, and log out of the SSH session, enter the following command:

   ```
   $ SET DISPLAY /DELETE
   ```

5.4 Managing Port Forwarding

The terminology used in port forwarding is reversed from that used in setting up an SSH session. Specifically:

- Use the SSH command on host VMSHOST1 to set up an X11 port forward to the SSH server on VMSHOST2.
- Run the X11 client on host VMSHOST2 for display by the X11 server on host VMSHOST1.

The following configuration parameters allow you to control port forwarding:

- `ForwardX11`, specified in the client configuration file, controls whether the SSH client performs port forwarding. This is set to `yes` by default.
- `AllowX11Forwarding`, specified in the server configuration file, controls whether the SSH server allows port forwarding. This is set to `yes` by default.
- `xauthpath`, specified in the client configuration file, specifies the location of the Xauthentication executable file.

For more information about these configuration parameters, see Appendix B.

6 Setting Up Kerberos/SSH Connections

In addition to the authentication methods described in Chapter 3, you can use Kerberos, the network authentication protocol from the Massachusetts Institute of Technology. This chapter describes how to set up and use Kerberos on SSH. Before you can use Kerberos authentication, you must set up Kerberos on OpenVMS, as described in the OpenVMS *HP Open Source Security for OpenVMS Volume 3: Kerberos* manual.

NOTE: This version of SSH for OpenVMS supports Kerberos for OpenVMS Version V2.1 and higher.

This chapter describes how to set up Kerberos authentication, including the following topics:

- Installing Kerberos RTL Images
- Setting Up Kerberos
- Setting Up a Kerberos SSH Connection
- Solving Problems with SSH and Kerberos

6.1 Installing Kerberos RTL Images

To use one of the Kerberos-based authentication methods, install the 32-bit Kerberos RTL image SYS$SHARE:KRB$RTL32.EXE. For example, enter the following command:

```
$ INSTALL CREATE SYS$SHARE:KRB$RTL32.EXE/OPEN/HEADER_RESIDENT/SHARED
```

To use `gssapi-with-mic` authentication on the SSH client or server, you must also install the 32 bit GSSAPI RTL image SYS$SHARE:GSS$RTL32.EXE. Enter the following command:

```
$ INSTALL CREATE SYS$SHARE:GSS$RTL32.EXE/OPEN/HEADER_RESIDENT/SHARED
```

In the system startup file, make sure that the Kerberos startup command procedured (KRB$STARTUP.COM) runs only after the TCP/IP Services for OpenVMS startup command procedure (TCPIP$STARTUP.COM) has been run.

6.2 Setting Up Kerberos

To use Kerberos-based SSH, you must:

- Configure the Kerberos user and host principals. You must also create the associated Kerberos `keytab` entries, as described in the *HP Open Source Security for OpenVMS Volume 3: Kerberos* manual.
- Configure the SSH server and client to use the appropriate Kerberos authentication methods.

6.2.1 Configuring the Kerberos User and Host Principles

On each SSH server host to which a user may want to connect using Kerberos-based authentication method or Kerberos password check, define a Kerberos host principal, as described in the OpenVMS *Guide to Security (Kerberos)*. The host principle for the SSH server host must be fully qualified. For example, a host principal for the SSH server host with DNS name `myhost.abcd.org` in the Kerberos realm `ABCD.ORG` would be specified as `host/myhost.abcd.org@ABCD.ORG`

To use the `gssapi-with-mic` authentication method, the server's local host database for itself must be configured so that the first entry in the list is the fully-qualified domain name. Use the TCP/IP management command SHOW HOST/LOCAL on the SSH server to make sure the server is defined by its fully-qualified canonical name. For example, to set up `gssapi-with-mic` authentication on the SSH server host named `myhost.abcd.org`, the local host entry must be defined as displayed by the following command:

```
MYHOST> TCPIP SHOW HOST/LOCAL MYHOST
     LOCAL database
Host address    Host name
10.0.0.1   myhost.abcd.org, myhost, MYHOST,MYHOST.ABCD.ORG
```

6.2.2 Configuring the Kerberos Authentication Method

In addition, the SSH servers must be configured to accept requests for each authentication method that clients will use. The following Kerberos authentication methods can be enabled:

- `gssapi-with-mic`
- `kerberos-2` (`kerberos-2@ssh.com`)
- `kerberos-tgt-2` (`kerberos-tgt-2@ssh.com`)

To enable the SSH server to provide an authentication method to SSH clients, add the Kerberos authentication method name to the `AllowedAuthentications` configuration keyword in the server configuration file. For more information about SSH authentication methods, see Chapter 3.

For example, if you want your SSH server to offer `gssapi-with-mic` and `publickey` authentication, include the following in the server configuration file:

```
AllowedAuthentications: gssapi-with-mic, publickey
```

The Kerberos authentication methods are not part of the default configuration. You must explicitly include them in the configuration file to use Kerberos authentication. For more information about modifying the server and client configuration files, see Appendix B.

6.2.3 SSH Kerberos Authentication Interoperability

The `kerberos-2@ssh.com` and `kerberos-tgt-2@ssh.com` authentication methods are proprietary (not specified by an IETF draft or RFC), and as such are supported only by the SSH implementations based on software from SSH Communications, Inc. HP Tru64 UNIX also supports these authentication methods.

The `gssapi-with-mic` authentication method is based on an IETF draft (GSSAPI Authentication and Key Exchange for the Secure Shell Protocol). As a public-domain specification, it is supported by a broader range of SSH implementations, including those based on OpenSSH.

TCP/IP Services does not implement the key exchange part of the GSSAPI protocol. It implements only the user authentication portion of this specification.

6.2.4 SSH Client Configuration

To enable a Kerberos authentication method on the SSH client, add the appropriate argument to the `AllowedAuthentications` keyword in the SSH client configuration file. Kerberos-based authentication methods are not enabled for `AllowedAuthentications` by default. You must enable them in your SSH client configuration file. For example, to tell your SSH client to use the `gssapi-with-mic` method first, then `publickey`, and then `password` authentication, enter the following:

```
AllowedAuthentications: gssapi-with-mic, publickey, password
```

6.3 Setting Up a Kerberos SSH Connection

To establish an SSH connection using Kerberos authentication, the SSH client user enters the `kinit` command to obtain a ticket-granting ticket (TGT).

6.3.1 Forwarding Credentials

Kerberos allows SSH to forward Kerberos credentials from client host to server host, obviating the need for users to re-enter their Kerberos password each time they use a Kerberized application. For example, with credentials forwarding, a user on HOSTA issues a `kinit` command, connects to SSH from HOSTA to HOSTB, and, once logged into HOSTB, connects to HOSTC, without ever issuing a `kinit` command on HOSTB. After entering the `kinit` command on HOSTA, the credentials follow the user to the session on HOSTB and then to the session on HOSTC.

To forward Kerberos credentials, include the `-f` option on the `kinit` command. This option indicates that a forwardable TGT is to be produced.

In addition to the presence of a forwardable TGT, the Kerberized application being used must support credentials forwarding. You can forward credentials when you are using the `kerberos-tgt-2` and

gssapi-with-mic authentication methods. The kerberos-2 authentication method does not support forwarding of the user's Kerberos credentials to the process on the SSH server host.

6.3.2 Managing Kerberos Credential Forwarding

Credentials are only forwarded from the client to the server if the GssapiDelegateCredentials client configuration parameter is set to yes. If the parameter is set to yes, the SSH client delegates credentials to the server. If it is set to no, the client does not delegate credentials. The default is no. For more information about client configuration parameters, see Appendix B.

6.3.3 Forwarding Kerberos Credentials Example

The following example shows how to create forwardable Kerberos credentials and use them to connect:

```
!!! User issues kinit with -f to get a forwardable TGT.
!!! In this example the Kerberos principal user name is lower case and
!!! the realm is uppercase.
SYSA> kinit -f "smith"
Password for smith@SYSA.XYZ.COM:

!!! Connect to system "sysb" forcing use of kerberos-tgt-2 authentication
!!! method.
SYSA> ssh -o"AllowedAuthentications kerberos-tgt-2@ssh.com" smith@sysb
Authentication successful.

 Welcome to HP OpenVMS Industry Standard 64 Evaluation Release V8.2

!!! We've been allowed in. A klist -f (-f for "full") shows that we have a
!!! TGT without having issued a kinit command on SYSB.
SYSB> klist -f
Ticket cache: FILE:WORK10$:[SMITH.KRB.SYSB.TMP]KRB5CC_1480589921
Default principal: smith@SYSA.XYZ.COM

Valid starting     Expires            Service principal
09/22/05 14:18:53  09/23/05 00:17:16  krbtgt/SYSA.XYZ.COM@SYSA.XYZ.COM
        Flags: FfT


Kerberos 4 ticket cache: krb$user:[tmp]k4_tkt_cache33488912
KRB$KLIST: You have no tickets cached

!!! Now use ssh to connect back to sysa but this time use the simpler
!!! kerberos-2 authentication method.
SYSB> ssh -o"AllowedAuthentications kerberos-2@ssh.com" smith@sysa
Authentication successful.

UNAUTHORIZED ACCESS PROHIBITED OpenVMS AXP (TM) Operating System, Version V8.2

!!! We have been allowed in but have no TGT created for us because we
!!! used kerberos-2:
SYSA> klist -f
KRB$KLIST: No credentials cache found (ticket cache FILE:krb$user:[tmp]krb5cc_33488912)



Kerberos 4 ticket cache: krb$user:[tmp]k4_tkt_cache33488912
KRB$KLIST: You have no tickets cached
```

6.4 Kerberos Password Authentication

In password authentication mode, the SSH server checks the password against Kerberos before checking it against the SYSUAF. If the Kerberos password check passes, the SSH server considers the SSH password authentication successful and the user is allowed in. If not, the password authentication continues on with the SYSUAF check.

When the Kerberos password check succeeds, the SSH server provides to the user process on the server system a forwardable TGT so that the user need not issue a kinit once logged in. Essentially the SSH server has performed a `kinit -f` command on behalf of the user.

By default, Kerberos password authentication is not enabled. To enable Kerberos password check in password authentication mode, set the `TryKerberosPassword` configuration parameter in the SSH server configuration file to `yes`.

The `TryKerberosPassword` configuration parameter tells the SSH server in password authentication mode to validate the user's password against Kerberos before validating against the SYSUAF. A `yes` value tells the SSH server to validate the user's password against Kerberos. A `no` value tells the SSH server not to check Kerberos. The `TryKerberosPassword` configuration field defaults to `no`.

To use Kerberos password authentication, you must have SYS$SHARE:KRB$RTL32.EXE installed, as described in Installing Kerberos RTL Images.

6.5 Solving SSH/Kerberos Problems

Kerberos, while powerful, can be cumbersome to configure and deploy. Often, problems that occur with Kerberos are related to misconfiguration of Kerberos, not a software problem with the applications using it. To help determine the cause of connection problems, you can set the following configuration parameters:

- The `GssapiSendError` parameter tells the SSH server in `gssapi-with-mic` authentication mode to send a GSSAPI error message (SSH_MSG_USERAUTH_GSSAPI_ERROR) to the client when an error occurs. This message provides information about the error and is usually displayed to the client user.

  A `yes` value tells the SSH server to send a GSSAPI error message. A `no` value tells it not to send the message. By default, his parameter is set to `no`.

- The `GssapiSendErrtok` parameter tells the SSH server or client in `gssapi-with-mic` authentication mode to send the `SSH_MSG_USERAUTH_GSSAPI_ERRTOK` message to the peer when an error occurs. This message contains a `gssapi` error token that is decoded on the peer and causes diagnostic information to be displayed.

  A `yes` value tells the SSH client or server to send the `SSH_MSG_USERAUTH_GSSAPI_ERRTOK` message. A `no` value tells it not to send the `SSH_MSG_USERAUTH_GSSAPI_ERRTOK` message. This parameter defaults to `no`.

The following list describes how to respond to some of the common errors related to using SSH with Kerberos:

- Expired Kerberos ticket

  Kerberos tickets have expiration dates. If you attempt to use the SSH client to connect using a Kerberos authentication method and your ticket has expired, the following message is displayed:

  ```
  $ ssh -o"AllowedAuthentications kerberos-2@ssh.com smith@sysb
  Kerberos5 authentication failed: Your ticket has expired. Use kinit
  ```

- Nonforwardable Kerberos ticket with `kerberos-tgt-2`

  When you use the `kerberos-tgt-2` authentication method, your `kinit` command must include the `-f` option to tell Kerberos that you want a forwardable TGT. If you have not specified kinit with `-f` and try to use SSH with `kerberos-tgt-2` authentication, the following message is displayed:

  ```
  $ ssh -o"AllowedAuthentications kerberos-tgt-2@ssh.com" smith@sysb
  Kerberos5 TGT forwarding failed: Ticket not forwardable? Try using kinit
  -f
  ```

- No Kerberos ticket at all

  If you use Kerberos based SSH authentication without having issued a `kinit` command to get a Kerberos TGT, you see this message:

  ```
  SYSA> ssh -o"AllowedAuthentications kerberos-tgt-2@ssh.com" smith@sysb
  Kerberos5 TGT forwarding failed: You have no ticket. Use kinit -f
  ```

- Internal credentials cache error with `gssapi-with-mic`

  You have tried to use the `gssapi-with-mic` authentication method with a forwardable TGT and the `GssapiDelegateCredentials` client parameter is set to `yes` (the default). Authentication fails. The following message is displayed:

  ```
  MYHOST> ssh -o"allowedauthentications gssapi-with-mic" -
  _MYHOST> -o"GssapiDelegateCredentials yes" smith@sysb
  GSSAPI error from server: Miscellaneous failure
                            Internal credentials cache error
  warning: Authentication failed.
  ```

  This type of failure causes the following warning in the SSH server log file:

  ```
  Mon 12 06:43:13 WARNING: Miscellaneous failure
  Internal credentials cache error
  ```

  This message is only displayed to the SSH client user if the `GssapiSendError` server configuration parameter is set to `no`.

  To correct the problem while using forwardable tickets, set the `GssapiDelegateCredentials` client configuration parameter for this session by entering the command as follows:

  ```
  # ssh -o"allowedauthentications gssapi-with-mic" -
  _$ -o"GssapiDelegateCredentials no" smith@sysb
  ```

  To continue without change the configuration options and without using forwardable tickets, enter the `kinit` again, without the `-f` option. This command requesets a nonforwardable ticket.

Many problems are caused by errors in systemwide or Kerberos configuration. Misconfiguration can cause problems like:

- Your SSH server log file shows the following error when trying to authenticate using `gssapi-with-mic`:

  ```
  Mon 12 09:37:37 WARNING: Miscellaneous failure
  Wrong principal in request
  ```

  The problem could be an incorrect local host database entry on the SSH server for the SSH server itself. Make sure the fully-qualified host name is used. Note that this message applies only to `gssapi-with-mic` authentication.

- Your SSH server log file shows the following error when trying to authenticate using `gssapi-with-mic`:

  ```
  Mon 19 09:19:37 WARNING: Miscellaneous failure
  No principal in keytab matches desired name
  ```

  This problem can be caused by a missing or incorrect host principal for the SSH server, or the host principal is correct but the `keytab` entry is missing or incorrect. This is how the problem manifests itself with `gssapi-with-mic` authentication.

- Your SSH client displays something similar to the following:

```
$ ssh -o"AllowedAuthentications kerberos-2@ssh.com" smith@sysb
Kerberos5 authentication failed: krb5_get_credentials(): KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN
warning: Authentication failed.
```

This problem can be caused by a missing or incorrect host principal for the SSH server, or the host principal is correct but the keytab entry is missing or incorrect. This is how the problem manifests itself with the `kerberos-2` or `kerberos-tgt-2` authentication.

- The SSH client or server (in the SSH server log) shows a message simlar to one of these:

```
WARNING: kerberos-2@ssh.com authentication failed since Kerberos wasn't initialized.
WARNING: gssapi-with-mic authentication failed since Kerberos wasn't initialized.
```

This message indicates that Kerberos for OpenVMS has not been installed properly. Refer to the *HP Open Source Security for OpenVMS Volume 3: Kerberos* manual for installation instructions to remedy the situation.

If SSH with Kerberos authentication fails, it is helpful to test another Kerberized application, such as TELNET, which will often fail in the same way as does SSH, pointing to a problem with Kerberos configuration, rather than to an SSH software problem.

The SSH server and client diagnostics features show errors and warnings indicating problems with calls to Kerberos library routines. These types of errors list the Kerberos routine that failed and the failure return status.

When you repeatedly make changes to host principals and `keytab` entries and files, existing tickets in the credentials cache can contain stale information, making it appear that your changes to the Kerberos configuration have not taken effect. Use the `kdestroy` and `kinit` commands to clear the credentials cache before testing your configuration changes.

7 SSH Command Reference

This chapter describes SSH commands that you can use to invoke SSH, copy or transfer files, and manage keys. It covers the following topics:

- Before You Begin
- Copying Files
- Remote Login and Command Execution
- Using SSH Commands in Batch Jobs
- Using the SSH_KEYGEN Utility
- Using the SSH_ADD Utility
- Using the SSH_AGENT Utility

7.1 Before You Begin

To use SSH client utilities at the DCL prompt, run the TCPIP$DEFINE_COMMANDS.COM command procedure. Enter the following command:

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

NOTE: When you specify uppercase options, enclose them in quotation marks: for example, `"-D"`.

7.2 Copying Files

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

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

7.2.1 File Types

You can use SCP and SFTP to copy files of the following types:

- Variable-length
- VFC
- Fixed
- Fortran Carriage Control
- Stream-LF

Files that are indexed or relative cannot be copied directly. To transfer indexed files:

1. Create an encoded copy of the file with sequential organization with any one of the following options:
   - Backup saveset
   - Self-extracting image (such as from SPOOL COMPRESS /METHOD=DCX_AXPEXE)
   - Zip archive
2. Transfer the converted file.
3. Convert back on the receiving end.

7.2.2 Using the SCP Command

The SCP command securely copies files between a Secure Shell client and server. This command is intended as a secure replacement for the `rcp` command. When the user enters the SCP command, the client establishes an SSH session. If authentication succeeds and the user's identity has been accepted by the server, the server

executes the command. All communication is automatically encrypted. The session terminates when the command completes. The SCP command does not require special privileges.

7.2.2.1 Command Synopsis

```
SCP [-D -dqQpuBrav146Vh] debug_level_spec]
[-c cipher] [-S ssh2-path]
[-P ssh-port]-b buf_size]
[-N max_requests] [-o ssh-option]
source-name destination-name
```

7.2.2.2 Parameters

source-name specifies the file to be copied, in the following format:

destination-name specified the location and file name for the copied file.

The general format for the source and destination name is as follows:

user@host#port :[directory]file-name

You can copy files or entire directories.

7.2.2.3 Options

Table 7–1 describes the options in you can use with the SCP command.

Table 7-1 SCP Command Options

Option	Description
-D *debug-level*	Displays debug information to SYS$OUTPUT. The *debug-level* is a number between 0 and 99, where 99 specifies that all debug information should be displayed.
-d	Makes sure that the destination-name parameter is a directory. If not, the SCP command exits with an error message.
-q	Makes SCP quiet (only fatal errors are displayed).
-Q	Suppresses the progress indicator.
-k	Replaces files of the same name at the destination. This option applies to OpenVMS SSH servers only.
-B	Sets batch mode on.
-v	Displays information in verbose mode. This is equal to specifying the `-D 2` option.
-1	Engage scp1 compatibility.
-4	Restricts communications to IPv4 networking. This is the default.
-6	Enables IPv6 networking.
-c *cipher*	Specifies the encryption algorithm to use. See the description of the `ciphers` configuration parameter in the client configuration file described in Appendix B. The `-c` option specifies one cipher; multiple `-c` options are allowed.
-S *ssh-path*	Specifies an alternate location for the SSH server executable file.
-P *ssh-port*	Specifies the port on which the SSH Server should listen for SCP connections.
-b	Defines the maximum buffer size for one request (default is 2048 bytes).
-N	Defines the maximum buffer size of concurrent requests (default is 10).
-o *ssh-option*	Specifies client configuration parameter settings that override the settings specified in the client configuration file. For more information, see Appendix B.
-V	Displays the version of SSH.
-h	Displays information about using the SCP utility.

7.2.2.4 Example

The following example shows how to copy files from a local system FILE.TXT to a remote system (VMSHOST) and into the directory [MYDIR].

```
$ SCP FILE.TXT KATHY@VMSHOST:DSK0:[MYDIR]
```

The following example shows how to copy FILE.TXT from a remote system (VMSHOST) to a local system and renaming it to LOCAL_FILE.TXT:

```
$ SCP KATHY@VMSHOST:DSK0:[MYDIR]FILE.TXT LOCAL_FILE.TXT
```

7.2.3 Using the SFTP Command

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

7.2.3.1 Command Synopsis

```
SFTP [-D debug_level_spec][-B batchfile] [-S path][-h] [-V] [-P ssh-port]
[b buffer_size] [-4] [-6][-o ssh_option]
user@host
```

For more details about SFTP commands, enter the `help` or `help topic` command at the `sftp>` prompt. For example, to find more information about the `open` command, enter the following command:

```
sftp> help open
```

7.2.3.2 Parameters

The *user@host* parameter specifies the user name and host name of the destination for the file transfer.

7.2.3.3 Options

Table 7–2 describes the options you can use with the SFTP command.

Table 7-2 SFTP Command Options

Option	Description
-D *debug-level*	Displays debug information. The *debug-level* value is a number between 0 and 99, where 99 specifies that all debug information should be displayed.
-B *batchfile*	Reads commands from a file instead of from SYS$INPUT. The default batch file is SYS$LOGIN:TCPIP$SFTP_BATCHFILE.TXT. If you specify a different batch file, make sure the batch file is in Stream-LF format.
-S *ssh-path*	Specifies an alternate location for the SSH server executable file.
-h	Displays information about how to use the SFTP utility.
-V	Displays the version of SSH.
-P	Tells SFTP on which port the SSH server is listening.
-b *buffer-size*	Specifies the buffer size.
-4	Restricts communications to IPv4 networking.
-6	Enables IPv6 networking.
-o *SSH-option*	Specifies additional SSH options.

7.2.3.4 Example

The following example shows how to invoke SFTP. Enter SFTP commands at the `sftp>` prompt. For a list of SFTP commands, enter the `help` command at the `sftp>` prompt. For example:

```
$ SFTP
sftp> help
```

7.3 Remote Login and Command Execution

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

On the client, you can use the SSH command to log in remotely and execute remote commands.

7.3.1 Command Synopsis

```
SSH [options][username@]host[#port][command]
```

7.3.2 Parameters

The *username@host#port* parameter specifies the user name, the remote host, and the port on the remote host to which to make a connection.

The *command* parameter specifies one or more commands to be executed on the remote host.

7.3.3 Options

Table 7–3 lists the options you can use with the SSH command.

Table 7-3 SSH Command Options

Options	Description
-l *login_name*	Specifies the user for login to the remote system (same as *login_name@host*).
+x	Enables X11 port forwarding. Treats X11 clients as "untrusted."
+X	Enables X11 port forwarding. Treats X11 clients as "trusted."
-x	Disables X11 port forwarding.
-i *file*	Specifies the identity file for public-key authentication. This option takes the file name as a parameter. It is assumed that the file resides in the user's [.SSH2] directory. This option can also be specified in the configuration file.
-F *file*	Specifies an alternative client host configuration file instead of the default file. The specified file name must include the directory where the file resides (for example,: [.SSH2]MY_SSH2_CONFIG). Information from this file supersedes information from TCPIP$SSH_DEVICE:[TCPIP$SSH]SSH2_CONFIG. and the user's [.SSH2]SSH2_CONFIG. file.
-t	Allocates a terminal device to the process.
-v	Enables verbose mode. Displays verbose debugging messages. Equivalent to the `-d2` option. This option can also be specified in the client's configuration file.
-d *debug-level*	Displays debug information. The *debug-level* value is a number from 0 to 99, where 99 specifies that all debug information or a comma-separated list of assignments should be displayed.
-V	Displays the version of SSH.
-q	Disables warning messages. This option can also be specified in the client's configuration file.
-p *port*	Specifies the port to which to connect on the remote system.
-S	Does not request a session channel. This type of session does not disconnect automatically. To disconnect a session begun with this option, enter the following TCP/IP Management command: `$ TCPIP DISCONNECT DEVICE BG`*nnnn* Where BG*nnnn* is the SSH session's device socket, as displayed by the TCP/IP management command SHOW DEVICE.

Options	Description
-L *[protocol/] port:host:hostport*	Specifies that the given port on the local (client) system is to be forwarded to the specified host and port on the remote system. This allocates a socket to listen to the port on the local system. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the specified host on the specified port from the remote system. Only privileged user accounts can forward privileged ports. The protocol enables the forwarding for the specfied protocol. The protocols implemented are TCP and FTP; the default is no specific processing. Temporary forwardings are created for the FTP data channel, effectively securing the whole FTP session. This option can also be specified in the client configuration file (see Appendix B). FTP data channel forwarding works in passive mode only. Be sure to set passive mode for FTP data channel connections.
-R *[protocol/] port:host:hostport*	Specifies that the given port on the remote (server) system is to be forwarded to the specified host and port on the local system. This allocates a socket to listen to the port on the remote system. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to the specified host and port from the local system. Only privileged user accounts can forward privileged ports on the remote system. The protocol argument enables protocol-specific forwarding. The protocols implemented are TCP and FTP; the default is no specific processing. Temporary forwardings are created for FTP data channel, effectively securing the whole FTP session. This option can also be specified in the client's configuration file (see Appendix B).
-4	Restricts communications to IPv4.
-6	Enables IPv6 networking.
-o *option*	Specifies an option in the format used in the SSH2_CONFIG. configuration file. This is useful for specifying an option for which there is no command-line option. Comment lines are not accepted with this option.
-h	Displays information about using the SSH utility.

To execute remote commands, enter the SSH command in the following format:

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

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

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

7.3.4 Example

The following example shows how to execute the SHOW SYSTEM command on the remote host VMSHOST.

```
$ SSH VMSHOST SHOW SYSTEM
```

7.4 Using SSH Commands in Batch Jobs

With this version of SSH, you can use the SSH, SCP, and SFTP commands in batch jobs. However, because these commands are ported from UNIX® implementations, some common OpenVMS assumptions must be adjusted, including:

- SYS$INPUT, SYS$OUTPUT, and SYS$ERROR may not work as you expect.
- To execute the following commands in batch mode, you may need to use the following options:

  ```
  $ SSH -o "batchmode yes"
  $ SCP "-B"
  $ SFTP "-B" filename
  ```

  The file indicated by *filename* contains a sequence of SFTP command. If SFTP is invoked in batch mode (from within a DCL command procedure (without the "—B" option), the following file name is used by default: SYS$LOGIN:TCPIP$SSH_SFTP_BATCH.TXT.

  For additional restrictions and guidelines, refer to the TCP/IP Services for OpenVMS *Release Notes*.

7.5 Using the SSH_KEYGEN Utility

SSH_KEYGEN is the key-pair generation utility that generates and manages authentication keys for SSH. Users who need to use SSH with public-key authentication can run this utility to create authentication keys. The system manager can also use this utility to generate host keys. To set up public key authentication, see "Setting Up Public Key Authentication."

7.5.1 Command Synopsis

```
SSH_KEYGEN [options] [key1 key2...]
```

7.5.2 Parameters

The [*key1 key2*...] parameter specifies the name of one or more keys to generate.

7.5.3 Options

Table 7–4 describes the options that you can use with the SSH_KEYGEN command.

Table 7-4 SSH_KEYGEN Command Options

Options	Description
-b *key-number*	Specifies the key strength, in bits. The default is 2048.
-t *key-algorithm*	Specifies the algorithm used to generate the keys. Specify either DSA or RSA. The default is DSA.
-c *comment-string*	Specifies the key's comment string.
-p *passphrase*	Specifies the passphrase used to protect the key.
-P	Specifies that the key will be saved with an empty passphrase.
-h | -?	Displays a short summary of SSH_KEYGEN options.
-q	Hides the progress indicator.
-D *file*	Derives the public key from the private key file.
-i *file*	Loads and displays information on a file.
-B *number*	Specifies the number base for displaying key information. The default is 10.
-V	Displays the version string and exits.
-r *file*	Randomizes data from a file to a random pool.
-F *file*	Dumps the fingerprint (a unique identifier) of the public key file.

7.6 Using the SSH_ADD Utility

The SSH_ADD utility adds private keys into the authentication agent. The authentication agent must have been started, usually with the SSH_AGENT utility, and must be running in a subprocess of the current process.

If a private key requires a passphrase, the SSH_ADD utility prompts you to enter it. Passphrases never go over the network.

For SSH_ADD to process a key, both the private and public key files must be present in the same directory. On OpenVMS, a public key file name must have the file extension .PUB (for example, MYKEY.PUB). A private key file name has no file extension (for example, MYKEY.).

7.6.1 Command Synopsis

```
SSH_ADD [-l] [-d] [-D] files...
```

7.6.2 Parameters

The *files...* parameter specifies one or more public or private key files to load. If you do not specify any key files, SSH_ADD reads the client configuration file (SSH2_CONFIG.) and the IDENTIFICATION. file. If these files do not exist, SSH_ADD exits with an error message. SSH_ADD adds the keys listed in the

IDENTIFICATION. file. The utility then adds any private key files it finds in the user's SSH directory. Note that any file names in the SSH directory that begin with the letters "id" and that do not have the file extension .PUB are assumed to be key files. For example, a file named `id_22.txt` causes SSH_ADD to fail.

7.6.3 Options

Table 7–5 describes the options you can use with the SSH_ADD command.

Table 7-5 SSH_ADD Command Options

Options	Description
-l	Lists all identities currently represented by the agent.
-d	Removes the identity from the agent.
-D	Deletes all identities from the agent.

7.6.4 Description

SSH_ADD attempts to load the identities from the specified key files.

7.6.5 Return Status

SSH_ADD returns one of the following exit codes in the case of an error:

`TCPIP$_SSH_ADD2_EXIT_NOAGENT` – No connection could be made to the authentication agent. Presumably there is no authentication agent active in the execution environment of the SSH_ADD utility.
`TCPIP$_SSH_ADD2_EXIT_BADPASS` – The user did not supply a required passphrase.
`TCPIP$_SSH_ADD2_EXIT_NOFILE` – An identity file could not be found, was unreadable, or was in the wrong format.
`TCPIP$_SSH_ADD2_EXIT_NOIDENTITY` – The agent does not have the requested identity.
`TCPIP$_SSH_ADD2_EXIT_ERROR` – An unspecified error has occurred.

7.6.6 Example

- In this example, the SSH_AGENT is not running:

```
$ SSH_ADD

Failed to connect to authentication agent -- agent not running?

%TCPIP-E-SSH_ADD2_EXIT_N, no connection could be made to the authentication agent
```

- In this example, SSH_ADD adds the keys it finds in the IDENTIFICATION. file:

```
$ SSH_ADD

Unable to open ssh2/ssh2_config

Unable to open ssh2/identification

Adding identity: ssh2/id_dsa_1024_a.pub

Need passphrase for "ssh2/id_dsa_1024_a." (1024-bit dsa, kathy@host.computer.com,

Mon Aug 11 2003 15:39:46). Enter passphrase:
```

7.7 Using the SSH_AGENT Utility

The SSH_AGENT utility starts the SSH authentication agent to use an SSH client that is configured to use public-key user authentication. Because the authentication agent holds private keys in memory, the user does

not need to enter a passphrase if one exists for the key being used. As long as the agent is running, all key-related operations are directed to the agent.

On startup, the agent does not hold any private keys. Keys are added by using the SSH_ADD command. Several identities can be stored in the agent, and the agent can use any of these identities automatically. The command SSH_ADD -l displays the identities currently held by the agent.

When the SSH agent starts up, it assigns the logical name TCPIP$SSH_AGENT_PORT in the process job table, which is shared by a process and its subprocesses. Client programs use this value to start communication with the agent, in turn assigning the logical name TCPIP$SSH_CLIENT_PORT. The agent uses the latter logical name to check that the client is a valid user on the same job (that is, in the parent of the agent subprocess).

The agent terminates when the user logs out or stops the agent.

7.7.1 Command Synopsis

```
SSH_AGENT [-d debug_level]
```

7.7.2 Options

Table 7–6 describes the options you can use with the SSH_AGENT command.

Table 7-6 SSH_AGENT Command Options

Options	Description
-d *debug-level*	Displays debug information. The *debug-level* value is a number from 0 to 99, where 99 specifies that all debug information or a comma-separated list of assignments should be displayed.

7.7.3 Examples

The following example shows a normal agent startup and displays the value of the TCPIP$SSH_AGENT_PORT logical name:

```
 $ SSH_AGENT
%DCL-S-SPAWNED, process USER01_67 spawned

$ SHOW USER/FULL
USER01        OpenVMS User Processes at 12-AUG-2003 13:49:36.29
Total number of users = 1,  number of processes = 2
Username Process Name    PID     Terminal
USER01   USER01        00000B53  RTA1:    (SYS01::USER01)
USER01   USER01_67     00000EB8  (subprocess of 00000B53)

$ SHOW LOGICAL TCPIP$SSH_AGENT_PORT
"TCPIP$SSH_AGENT_PORT" = "49198" (LNM$JOB_81425DC0)
```

The following example shows an attempt to start a second agent within the same parent process.

```
 $ SSH_AGENT
%DCL-S-SPAWNED, process USER01_253 spawned
$  Agent already running on port: 49198.  Cannot start agent
%TCPIP-F-SSH_FATAL, non-specific fatal error condition
```

The following example shows the message displayed when a client not in the current user's job attempts to connect to the user's agent subprocess:

```
$   Possible security attack. Actual socket port of client: 49202 did not match

value of logical name TCPIP$SSH_CLIENT_PORT: ""
```

8 Solving SSH Problems

This chapter describes how to analyze and solve problems that prevent you from logging in using SSH.

NOTE: In this discussion, the user is the client user who executes the SSH command, or is the user who is specified with the `-l` option to the SSH command.

Login is not permitted under the following conditions. In these cases, no auditing occurs.

- The user account does not exist.
- The user account has expired.
- The user account has access restrictions for the current day and time.
- The `pwd_expired` flag is set in the user's SYSUAF record.
- The keyword `userloginlimit`has a value of zero in the SSH server configuration file. (This applies to all users.)

If any of the following conditions are true for the user on the SSH server, login is not permitted and auditing occurs:

- The user failed the authentication (for example, invalid or missing keys for the host-based or public-key method, invalid password for the password method, expired password and configured not to allow client in with expired password).
- The user name is in the DenyUsers list, or is not in the AllowUsers list (if it exists) in the server configuration file (SSHD2_CONFIG.).
- The user is in a group that is in the `DenyGroups` list, or is not in the `AllowGroups` list (if it exists) in the server configuration file (SSHD2_CONFIG.). The groups in the `DenyGroups` and `AllowGroups` lists are specified by the decimal representation that is the group portion of the UIC. That is, if a user's UIC is [777,42], the following syntax denies the user and all other users with UIC [777,*]:

  ```
  DenyGroups 511
  ```

- The `disuser` or `autologin` flag is set in the user's SYSUAF record.
- The user does not have OPER privilege and one of the following is true:
  - The number of interactive logins has exceeded the SYSGEN parameter IJOBLIM.
  - The `UserLoginLimit` parameter in the server configuration file is greater than zero and there are already that number of logins for any individual user name.
  - The client has been identified as an intruder.

If the user's password has expired and the connection is from an OpenVMS system to another OpenVMS system, and the `disforce_pwd_expired` flag is not set in the user's SYSUAF, then the user must change the password. The password dictionary, password history, and generated password lists are not used. The number of failed attempts to verify the new password is specified using the `NumberOfPasswordVerificationPrompts` parameter in the client configuration file.

The client user is not forced to change the password before logging in when:

- The connection is from OpenVMS to OpenVMS and the `disforce_pwd_change` flag is set in the user's SYSUAF record.
- The connection is from a different SSH implementation to an OpenVMS system and the `AllowNonvmsLoginWithExpiredPw` parameter is set to `yes` in the client configuration file. In these cases, the `pwd_expired` flag is set in the user's SYSUAF record, so that any future attempts to log in will fail if the password is not changed during the current session.

The client user login is rejected if:

- The connection is from a different SSH implementation to an OpenVMS system and the `AllowVmsLoginWithExpiredPw` parameter is set to `no` in the server configuration file.
- The connection is from an OpenVMS system to a different SSH implementation, and the `AllowNonVmsLoginWithExpiredPw` parameter is set to `no` in the server configuration file.

Examples

- If login is allowed but the password has expired, and the user is forced to change his password, the following message is displayed before the first DCL prompt:

  ```
  WARNING - Your password has expired; update immediately with SET PASSWORD!
  ```

- If the `NumberOfPasswordVerificationPrompts` parameter is set to 2, the following message is displayed:

  ```
  Your password has expired; you must set a new password to log in
  New password:Verification:
  New password verification error; please try again
  Verification:
  ```

  If verification fails a second time, the login attempt fails.

To get detailed tracing information, on the OpenVMS SSH server, enter the following command:

```
$ ASSIGN/SYS "-i -d 6" TCPIP$SSH_SERVER_PARAM
```

Trace information is written to the TCPIP$SSH_HOME:TCPIP$SSH_RUN.LOG file.

A SSH Directories and Files

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

A.1 Client Directories and Files

`TCPIP$SSH_DEVICE:[TCPIP$SSH]`

Function: Default directory of the TCPIP$SSH account

Creation: During SSH configuration

Scope: Systemwide

Use: By running instances of the client and server processes

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

Function: Contains multiple SSH files and subdirectories

Creation: During SSH client configuration

Scope: Systemwide

Use: By running instances of the client processes

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

Function: Client configuration file

Creation: During SSH client configuration, by extracting a template file from the TCP/IP kit. The system manager edits this file as necessary. Users can copy this file and make user-specific modifications.

Scope: Systemwide

Use: Read by a starting client process

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

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

Creation: Empty during SSH client configuration, the system manager copies the files to this directory from all servers before initiating connections. (This step may not be required. If the server's public host key file is not in TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.HOSTKEYS] when the connection initiated, it can be copied automatically to the user specific directory SYS$LOGIN:[SSH2.HOSTKEYS] on the client.) You control this behavior using the `StrictHostKeyChecking` parameter in the server configuration file, as described in Appendix B.

Scope: Systemwide

Use: For host authentication purposes; the client searches files in this directory for the server's key before it searches the user-specific directory.

`SYS$SYSDEVICE:[`*`username`*`.SSH2]`

Function: Contains multiple SSH files and subdirectories

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

Scope: User specific

Use: By running the client.

`SYS$SYSDEVICE:[`*`username`*`.SSH2]SSH2_CONFIG.`

Function: Client configuration file.

Creation: By the user, if necessary

Scope: User specific

Use: By a starting client process (if one exists) in lieu of the systemwide configuration file.

`SYS$SYSDEVICE:[`*`username`*`.SSH2]IDENTIFICATION.`

Function: Contains the identification of a user.

Creation: By the user when using public-key authentication

Scope: User specific

Use: To identify a user for public-key authentication

`SYS$SYSDEVICE:[username.SSH2.HOSTKEYS]`

Function: Contains the public keys of the server hosts to which the client will connect.

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

Scope: User specific

Use: For host authentication purposes. First, the client tries to locate the remote host key in this directory. If it is not found, the systemwide directory is used.

A.2 Server Directories and Files

`TCPIP$SSH_DEVICE:[TCPIP$SSH]`

Function: Default directory of TCPIP$SSH account

Creation: During SSH server configuration

Scope: Systemwide

Use: By running instances of the server and client processes

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

Function: Contains multiple SSH files and subdirectories.

Creation: During SSH server configuration

Scope: Systemwide

Use: By running instances of the server and client processes

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

Function: Server configuration file

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

Scope: Systemwide

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

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

Function: Contains the private part of the host key pair. This file is owned by the system account and has system read access only.

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

Scope: Systemwide

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

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

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

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

Scope: Systemwide

Use: Server host identification. Required on the SSH client in the [*username*.SSH2.KNOWNHOSTS] directory in order to use any authentication method. Also required on the server for host-based authentication.

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

Function: List of trusted hosts.

Creation: An empty directory is created during SSH server configuration. The system manager populates the file.

Scope: Systemwide

Use: As a systemwide list of trusted hosts checked by a server for host-based authentication.

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

Function: Systemwide directory that contains public keys of all remote client hosts that might attempt to connect to the server using host-based authentication.

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

Scope: Systemwide

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

`SYS$LOGIN:SHOSTS.`

Function: List of trusted hosts

Creation: By the user, if necessary

Scope: User specific

Use: As a user-specific list of trusted hosts, checked by the server for host-based authentication. The server checks this list after it checks the systemwide SHOSTS.EQUIV, enabling the user to allow access by hosts that are not in the systemwide list.

`SYS$LOGIN:[SSH2]`

Function: Contains multiple SSH files and subdirectories.

Creation: By the user, if necessary

Scope: User specific

Use: By running the server

`SYS$LOGIN:[SSH2.KNOWNHOSTS]`

Function: A user-specific directory that contains public keys of all remote client hosts that might try to connect to the server using host-based authentication.

Creation: By the user, if necessary. The user populates the directory by copying files from client hosts.

Scope: User specific

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

`SYS$SYSLOGIN:[SSH2]AUTHORIZATION`

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

Creation: By the user, if necessary. The user populates this file by copying files from the client hosts.

Scope: User specific

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

B SSH Client and Server Configuration Parameters

This appendix lists the systemwide SSH client and server parameters that the TCPIP$CONFIG configuration procedure creates during SSH configuration, as described in Chapter 2.

NOTE: The default settings are used for options that do not appear in the configuration file.

The server configuration file is:

```
TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2]SSHD2.CONFIG.
```

The client configuration file is:

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

B.1 Client Configuration Parameters

Some of the client configuration parameters that you can modify are as follows:

- `AllowedAuthentications`

  Allowed values: `hostbased`, `password`, `publickey`, `gssapi-with-mic`, `kerberos-2@ssh.com`, `kerberos-tgt-2@ssh.com`
  Default: `hostbased,password,publickey`
  Description: Specifies the authentication methods the client will attempt, in the order they will be presented to the server.

  The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.

- `DefaultDomain`

  Specifies the fully qualified domain name for the local host.

- `ForwardX11`

  Enables X11 port forwarding (the default). To disable the SSH client from allowing X11 port forwarding, set this parameter to No.

- `GssapiDelegateCredentials`

  Delegates the user's credentials to the SSH server.

- `NumberOfHostkeyCopyPrompts`

  Allowed values: an integer greater than 0
  Default: 3
  Description: Specifies the number of times the client user gets prompted to answer yes or no when asked about continuing to start an SSH session, if there is no host key and the value of StrictHostKeyChecking is *ask*.

- `NumberOfPasswordVerificationPrompts`

  Allowed values: An integer greater than 0
  Default: 3
  Description: Specifies the number of times the client user is allowed to fail verification of the new password when forced to change it on login. Applies to OpenVMS-to-OpenVMS connections only. This number must be at least 2 to support second passwords.

- `port`

  Allowed values: An integer value.
  Default: 2
  Description: Specifies the port number that SSH listens on. If you change the port number, you must explicitly disable and then reenable the SSH server process with the correct port number.

  For example, to change the port number to 2222, enter the following commands:

```
$ tcpip disable service ssh
$ tcpip set noservice ssh
$ tcpip set service ssh /port=2222 /proc=tcpip$ssh/user=tcpip$ssh —
```

```
_$ /file=tcpip$system:tcpip$ssh_run.com /proto=tcp/limit=10000 —
_$ /log=(all,file=tcpip$ssh_device:[tcpip$ssh]tcpip$ssh_run.log)
$ tcpip enable service ssh
```

- `PubkeyPassphraseGuesses`

  Allowed values: An integer greater than 0
  Default: 3
  Description: Specifies the number of guesses the client user is allowed for the passphrase associated with public/private key pair. Used for public-key authentication method only.
  The value of this option affects connections to servers on all platforms, including those on different SSH implementations that may have problems associated with passphrase entry.
  When the value is different on an OpenVMS client and the associated OpenVMS server, the lower value takes precedence.
  Each prompt for passphrase is of the following format: `Passphrase for key "ssh2/KAREN-SELFDBOB_SQA_UCX_ABC_ACME_COM"with comment "1024-bit dsa, karen@dbob.sqa.ucx.abc.acme.com,Wed May 21 2003 12:42:14":`
  If the user enters an incorrect passphrase, the prompt appears the number of times specified for the `PubkeyPassphraseGuesses`option.

- `StrictHostKeyChecking`

  Controls what happens if the server's public host key file is either invalid or not found in either the user's [*username*.SSH2.HOSTKEYS] directory or the systemwide directory TCPIP$SSH_DEVICE:[TCPIP$SSH.SSH2.HOSTKEYS]. The filename is in the format KEY_*portnumber*_*hostname*.PUB, where *portnumber* is the port number used to establish the SSH session (22 by default), and *hostname* is the host name used by the client user to establish the session. This parameter accepts the following values:

  - `yes` -- Causes authentication to fail if the file is not found.
  - `no` -- Causes the SSH client to create the [*username*.SSH2.HOSTKEYS] subdirectory (if it does not exist), and copies the SSH server's public key file into this subdirectory automatically.
  - `ask` -- Causes the SSH server to prompt the user for a copy of the server's public host key. This is the default. The prompt appears as follows:

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

    If you respond with `yes`, and the existing key file is invalid, the user is also prompted as follows:

    ```
    Do you want to change the host key on disk (yes/no)?
    ```

- `Xauthpath`

  Allowed values: OpenVMS file specification
  Default: SYS$SYSTEM:DECW$XAUTH.EXE
  Description: Specifies the path name of the Xauthentication executable file.

B.2 Client Configuration File

The following is an example of a typical SSH client configuration file:

```
# SSH CONFIGURATION FILE FORMAT VERSION 1.1
# REGEX-SYNTAX egrep
# end of metaconfig
# (do not change above lines!)

#
# File name:      SSH2_CONFIG.
# Product:        HP TCP/IP Services for OpenVMS
# Version:        T5.6-3D
#
#  Copyright 1976, 2005 Hewlett-Packard Development Company, L.P.
#

#
```

```
# ssh 3.2 client configuration information
#
# Note: ".*" is used for all hosts, but you can use other hosts as well
#
.*:

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

## General

    AuthenticationSuccessMsg  yes
#   BatchMode    no
#   Compression    no
#   DontReadStdin   no
#   EscapeChar    ~
#   ForcePTTYAllocation   no
#   GoBackground   no
#   PasswordPrompt   "%U@%H's password: "
    PasswordPrompt   "%U's password: "
#   QuietMode    no
#   SetRemoteEnv   foobar=baz
    VerboseMode    no

## Network

    Port    22
    NoDelay    no
    KeepAlive    yes
#   SocksServer    socks://mylogin@socks.ssh.com:1080/203.123.0.0/16,198.74.23.0/24
#   UseSocks5    no

## Crypto

    Ciphers    AnyStdCipher
    MACs    AnyStdMAC
#   RekeyIntervalSeconds  3600
    StrictHostKeyChecking  no


## User public key authentication

    IdentityFile   identification
    RandomSeedFile   random_seed

## Tunneling

#   ForwardAgent   yes
#   ForwardX11    yes
#   GatewayPorts   no
#   TrustX11Applications  no
#   XauthPath    <set by configure by default>

# Tunnels that are set up upon login

#   LocalForward   "110:pop3.company.com:110"
#   LocalForward   "143:imap.company.com:143"
#   LocalForward   "25:smtp.company.com:25"
#   RemoteForward   "3000:localhost:22"

## SSH1 compatibility
```

```
#   Ssh1InternalEmulation  yes
    Ssh1Compatibility   no
    Ssh1AgentCompatibility  none
#   Ssh1AgentCompatibility  traditional
#   Ssh1AgentCompatibility  ssh2
#   Ssh1MaskPasswordLength  yes
#   Ssh1Path    /usr/local/bin/ssh1

## Authentication
## hostbased, publickey, and password are allowed by default
## (least interactive method should be usually attempted first)

#   AllowedAuthentications  publickey, keyboard-interactive, password
    AllowedAuthentications  hostbased, publickey, password

## Authentication, OpenVMS-specific

#   NumberOfHostkeyCopyPrompts          3
#   NumberOfPasswordVerificationPrompts 3
#   PubkeyPassphraseGuesses             3

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

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

## Examples of per host configurations

#alpha.*:
#   Host    alpha.oof.fi
#   User    username_at_alpha
#   PasswordPrompt   "%U:s password at %H: "
#   Ciphers    aes

#foobar:
#   Host    foo.bar
#   User    foo_user
```

B.3 Server Configuration Parameters

Some of the server configuration parameters that you can modify are as follows:

- `AccountingAuthentications`

  Allowed values: `password`, `publickey`, `hostbased`, `all`, `none`. The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.
  Default: `publickey, password, hostbased`
  Description: Specifies the authentication methods for which accounting data is updated.
  The following command displays the contents of the intrusion database: `ACCOUNTING`

- `AllowedAuthentications`

  Specifies the authentication methods the server will allow.

  Allowed values: `password`, `publickey`, `hostbased`, `gssapi-with-mic`, `kerberos-2@ssh.com`, `kerberos-tgt-2@ssh.com`

  Default: `hostbased`, `password`, `publickey`

  Description: Specifies the authentication methods the server will accept.

  The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.

- `AllowGroups`

  The groups in the `AllowGroups` list are specified by the decimal representation that is the group portion of the UIC. That is, if a user's UIC is [777,42], the following syntax allows the user and all other users with UIC [777,*]:

  ```
  AllowGroups 511
  ```

- `AllowNonvmsLoginWithExpiredPw`

  Allowed values: `yes, no`
  Default: `no`
  Description: Controls behavior when a different SSH client implemention attempts to establish an SSH connection to an OpenVMS server account with an expired password. The password change option is implemented for OpenVMS-to-OpenVMS connections only. The value `yes` allows clients to connect with the following warning message and sets the `pwd_expired` flag in the user's SYSUAF record: `WARNING - Your password has expired; update immediately with SET PASSWORD!` The value `no` rejects the login. The SSH client implementation must support the CHANGEREQ mechanism (message type 60) to update passwords.

- `AllowVmsLoginWithExpiredPw`

  Allows OpenVMS users to change expired passwords, if required. If the value is No, the login is rejected.

  For a user to be allowed to make a connection (from either an OpenVMS client or from a different SSH implementation) with an expired password, the OpenVMS account must set the DISFORCE_PWD_CHANGE flag. To set this flag, enter the following command:

  ```
  $ MCR AUTHORIZE MODIFY USERNAME /FLAG=DISFORCE_PWD_CHANGE
  ```

  When you log in to an account with an expired password, the following message is displayed:

  ```
  WARNING - Your password has expired; update immediately with SET PASSWORD!
  ```

- `AllowX11Forwarding`

  Enables X11 port forwarding.

- `DenyGroups`

  The groups in the `DenyGroups` list are specified by the decimal representation that is the group portion of the UIC. That is, if a user's UIC is [777,42], the following syntax denies the user and all other users with UIC [777,*]:

  ```
  DenyGroups 511
  ```

- `IntrusionAuthentications`

  Allowed values: `password`, `publickey`, `hostbased`, `all`, `none`
  Default: `password`
  Description: Specifies the methods for which the server intrusion database is updated for the user in case of login failure.
  The following command displays the contents of the intrusion database: `SHOW INTRUSION`

- `IntrusionIdentLocalUser`

  Allowed values: yes, no
  Default: yes
  Description: Controls whether intrusion identification records are identified by IP address or user name. Set to yes,then the server uses the lcal user name in intrusion records. If this parameter is set to no, uses SSH_*xxxxxxxx*, where *xxxxxxxx* is the intruder's IP address.

- `IntrusionIdentMethod`

  Allowed values: `password`, `publickey`, `hostbased`, `all`, `none`. The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.
  Default: publickey, password, hostbased

Description: For entries in the intrusion database, this option controls whether the authentication method is included in the text of the intrusion Source (as displayed by the SHOW INTRUSION command). The value of this option is ignored if `IntrusionAuthentications` and `IntrusionIdentSsh` are not both active for the specified method.
The following command displays the contents of the intrusion database: `$ SHOW INTRUSION`

- `IntrusionIdentSSH`

  Allowed values: `password`, `publickey`, `hostbased`, `all`, `none`. The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.
  Default: `publickey, password, hostbased`
  Description: For entries in the intrusion database, this option controls whether the string SSH_ is included in the text of the intrusion "Source" (as displayed by the SHOW INTRUSION command). The value of this option is ignored if the `IntrusionAuthentications` is not active for the specified method.
  The following command displays thecontents of intrusion database: `$ SHOW INTRUSION`

- `LogfailAuthentications`

  Allowed values: `password`, `publickey`, `hostbased`, `all`, `none`. The keyword `all` is equivalent to `publickey, password, hostbased`. The keyword `none` explicitly disables all SSH authentication methods.
  Default: `password`
  Description: Specifies the authentication methods for which the SYSUAF login failure count is updated for the user. The following command displays the number of login failures: `MCR AUTHORIZE SHOW` *`username`*.

- `PasswordGuesses`

  Specifies the number of times the user can enter an incorrect password.

- `IntrusionIdentLocalUser`

  Uses the local user name in the intrusion record. If set to No, uses `SSH_`*`xxxxxxxx`* (where *`xxxxxxx`* is the IP address of the remote host, in hexadecimal format). The default is Yes.

- `IgnoreRhosts`

  Specifies that the SHOSTS.EQUIV file be used to allow a user from one system to log in as a different user from another host. If this parameter is set to No, the user-specific SHOSTS. file is used.

- `PubkeyPassphraseGuesses`

  Allowed values: Integers greater than 0
  Default: 3
  Description: Specifies the number of times the client user is allowed to enter the passphrase associated with public/private key pair. Used for public key authentication method only. In the server configuration file, this value affects all clients, including those on OpenVMS systems. When the value is different on an OpenVMS client and the associated OpenVMS server, the lower value takes precedence.
  Each prompt for passphrase is of the following format: `Passphrase for key "ssh2/KAREN-SELFDBOB_SQA_UCX_ABC_ACME_COM"with comment "1024-bit dsa, karen@dbob.sqa.ucx.abc.acme.com,Wed May 21 2003 12:42:14":`

- `UserLoginLimit`

  Allowed values: integers from -1 to 8192
  Default: -1
  Description: Controls the number of times individual users can be logged in. If the value is -1, the system-wide limit on interactive logins (SYSGEN parameter IJOBLIM) applies. If the value is greater than zero, the number specifies the maximum number of times that an individual user can log in.
  `-1` = no limit on specific users
  `0` = disable all users
  `1 - 8192` = number of logins permitted for individual users
  To display details on login processes for USER, enter the following command:`$ SHOW USER /FULL /NODE=`*`serverhost`*

B.4 Server Configuration File

The following is an example of a typical SSH server configuration file.

```
# SSH CONFIGURATION FILE FORMAT VERSION 1.1
# REGEX-SYNTAX egrep
# end of metaconfig
# (do not change above lines!)
#
# File name:      SSHD2_CONFIG.
# Product:        HP TCP/IP Services for OpenVMS
# Version:        T5.6-3D
#
#  Copyright 1976, 2005 Hewlett-Packard Development Company, L.P.
#
#
# ssh 3.2 server configuration file
#
## General
    HostKeyFile    hostkey
    PublicHostKeyFile   hostkey.pub
    RandomSeedFile   random_seed
#   BannerMessageFile   /etc/ssh2/ssh_banner_message
#   BannerMessageFile   /etc/issue.net
#   SftpSyslogFacility   LOCAL7
    SyslogFacility   AUTH
#   SyslogFacility   LOCAL7
#   QuietMode    no
    VerboseMode    no
## Network

    Port    22
    ListenAddress   any
    RequireReverseMapping  no
#   ResolveClientHostName  yes
    MaxBroadcastsPerSecond  0
#   MaxBroadcastsPerSecond  1
#   NoDelay    no
#   KeepAlive    yes
#   MaxConnections   50
#   MaxConnections   0
# 0 == number of connections not limited


## Crypto

    Ciphers    AnyCipher
#   Ciphers    AnyStdCipher
#   Ciphers    3des
# Following includes "none" 'cipher':
#   Ciphers    AnyStd

    MACs    AnyMAC
#   MACs    AnyStdMAC
# Following includes "none" 'mac':
#   MACs    AnyStd

#   RekeyIntervalSeconds  3600


## User

    CheckMail    yes
    PrintMotd    yes
#   StrictModes    yes
```

```
# Specifies 1 hour
# (you can also use 'w' for week, 'd' for day, 'm' for minute, 's' for seconds)
#   IdleTimeOut    1h
# without specifier, the default number is in seconds
#   IdleTimeOut    3600

    UserConfigDirectory   "%Dssh2"
#   UserConfigDirectory   "/etc/ssh2/auth/%U"
    AuthorizationFile   authorization

# This variable is set here, because by default it's empty, and so no
# variables can be set. Because of that, we set a few common ones here.
    SettableEnvironmentVars  LANG,LC_(ALL|COLLATE|CTYPE|MONETARY|NUMERIC|TIME),PATH,TERM,TZ


## Tunneling

    AllowX11Forwarding   yes
    AllowTcpForwarding   yes

#   AllowTcpForwardingForUsers  sjl, cowboyneal@slashdot\.org
#   DenyTcpForwardingForUsers  2[[:digit:]]*4,peelo

#   AllowTcpForwardingForGroups  privileged_tcp_forwarders
#   DenyTcpForwardingForGroups  coming_from_outside

# Local port forwardings to host 10.1.0.25 ports 143 and 25 are
# allowed for all users in group users.
# Note that forwardings using the name of this host will be allowed (if
# it can be resolved from the DNS).

#   ForwardACL allow local .*%users \i10\.1\.0\.25%(143|25)

# Local port forwardings requested exactly to host proxy.company.com
# port 8080 are allowed for users that have 's' as first character
# and belong to the group with group id 10:

#   ForwardACL allow local s.*%10 proxy\.company\.com%8080

# Remote port forwarding is denied for all users to all hosts:
#   ForwardACL    deny remote .* .*


## Authentication
## hostbased, publickey and password are allowed by default

#   AllowedAuthentications  hostbased, publickey, password
#   AllowedAuthentications  publickey
   AllowedAuthentications gssapi-with-mic, kerberos-tgt-2@ssh.com, kerberos-2@ssh.com, publickey, password

#   Requireduthentications  publickey, password
#   LoginGraceTime   600
#   AuthInteractiveFailureTimeout 2

#   HostbasedAuthForceClientHostnameDNSMatch no
#   UserKnownHosts   yes
#
#   AuthPublicKey.MaxSize  0
#   AuthPublicKey.MinSize  0
    AllowAgentForwarding  yes

#   AuthKbdInt.NumOptional  0
#   AuthKbdInt.Optional   pam, password
#   AuthKbdInt.Required   password
#   AuthKbdInt.Retries    3
```

```
#   PermitEmptyPasswords  no
    PasswordGuesses   3


## Host restrictions

    AllowHosts    localhost, *

## Next one matches with, for example, taulu.foobar.com, tuoli.com, but
## not tuoli1.com. Note that you have to input string "\." when you want it
## to match only a literal dot. You also have to escape "," when you
## want to use it in the pattern, because otherwise it is considered a list
## separator.

##  AllowHosts    t..l.\..*

## The following matches any numerical IP-address (yes, it is cumbersome)

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

## Same thing is achieved with using the special prefix "\i" in a
## pattern. This means that the pattern is only used to match
 # IP-addresses.   Using the above example:
#
#   AllowHosts    \i.*
#
# You can probably see the difference between the two.
#
# Also, you can use subnet masks, by using prefix "\m"
#
#   AllowHosts    \m127.0/8
#
#   AllowHosts    \m127.0.0.0/24
#
# would match localhost ("127.0.0.1").
#
#   DenyHosts    evil\.org, aol\.com
#   AllowSHosts    trusted\.host\.org
#   DenySHosts    not\.quite\.trusted\.org
#   IgnoreRhosts   no
#   IgnoreRootRHosts   no
#
# (the above, if not set, is defaulted to the value of IgnoreRHosts)


## User restrictions

#   AllowUsers    sj.*,s[[:digit:]]*,s(jl|amza)
#   DenyUsers    skuuppa,warezdude,31373
#   DenyUsers    don@untrusted\.org
#   AllowGroups    staff,users
#   DenyGroups    guest,anonymous
    PermitRootLogin   yes
#   PermitRootLogin   nopwd


## Chrooted environment

#   ChRootGroups   sftp,guest
#   ChRootUsers    anonymous,ftp,guest


## SSH1 compatibility
```

```
#   Ssh1Compatibility   no
#   Sshd1Path    <set by configure by default>

# This is given as argument to sshd1 with "-f" if sshd2 is invoked
# with "-f", otherwise the default configuration for sshd1 is used.

#   Sshd1ConfigFile   /etc/sshd_config_alternate


## Subsystem definitions

# Subsystems don't have defaults, so this is needed here (uncommented).

    subsystem-sftp   /sys$system/tcpip$ssh_sftp-server2
# Also internal sftp-server subsystem can be used.

#   subsystem-sftp   internal://tcpip$ssh_sftp-server2


## Subconfiguration

# There are no default subconfiguration files. When specified the last
# obtained keyword value will prevail. Note that the host specific files
# are read before the user specific files.

# Following matches (from) any host
#
#   HostSpecificConfig .* /etc/ssh2/subconfig/host_ext.example
#
# Following matches to subnet mask:
#
#   HostSpecificConfig \m192.168.0.0/16 /etc/ssh2/subconfig/host_int.example
#
# Following matches to users from ssh.com that have two character long
# username or is sjl and belong to group wheel or wheel[0-9]

#   UserSpecificConfig (..|sjl)%wheel[[:digit:]]?@ssh\.com /etc/ssh2/subconfig/user.example

# Following matches to the user anonymous from any host

#   UserSpecificConfig anonymous@.* /etc/ssh2/subconfig/anonymous.example


## OpenVMS auditing and access control

#   AllowVmsLoginWithExpiredPw          no
#   AllowNonvmsLoginWithExpiredPw       no
#   UserLoginLimit                      -1
#
#   V5.5-02 Change pubkey to publickey
   AccountingAuthentications           kerberos-tgt-2@ssh.com, kerberos-2@ssh.com, publickey, password ,hostbased
   IntrusionAuthentications            kerberos-tgt-2@ssh.com, kerberos-2@ssh.com, publickey, password ,hostbased
#   IntrusionIdentMethod                publickey,password,hostbased
#   IntrusionIdentSsh                   publickey,password,hostbased
   LogfailAuthentications              kerberos-tgt-2@ssh.com, kerberos-2@ssh.com, publickey, password ,hostbased
#   PubkeyPassphraseGuesses             3
GssapiSendError yes
GssapiSendErrtok yes
```

Glossary

This glossary defines some of the terms, abbreviations, and acronyms found in this document.

asymmetric key The use of two different keys (public and private) to authenticate a user connection.

authentication The process of determining another's identity. For example, the SSH server identifies itself to a connecting client during session setup using the server host key and the public/private keypair. The SSH client uses password, publickey, or hostbased authentication to establish its identity to the server.

authentication agent The SSH_AGENT utility, which allows you to manage keys.

data integrity The state that exists when data has not been changed.

decrypt The process of modifying encrypted data so that it can be read.

DSA Private digital key signature algorithm

encrypt The process of modifying data to make it impossible to be read except by the proper decryption function.

encryption The process of modifying the data stream such that it can only be read by the appropriate decryption technique.

GSSAPI GSSAPI Authentication and Key Exchange for the Secure Shell Protocol (a Kerberos authentication method).

host keys The public-private key pair that identifies the server host.

host-based authentication The authentication method where the client and server hosts authenticate each other.

kerberos The security protocol that provides strong authentication by using secret-key cryptography.

kerberos password authentication The authentication method used by Kerberos—aware applications.

known hosts database The database that contains public keys for all client hosts that use the host-based authentication method to connect to the server.

nonrepudiation The function that identifies data so that a user or entity cannot deny ownership or action related to the data.

password authentication The authentication method in which the client transmits an encrypted password encrypted to the server.

port forwarding The function of encapsulating the TCP-based communication session between the SSH client and the SSH server programs. The result is a secure tunnel.

private key Of the key pair, the key that is known only to the user. When a message is encrypted with a public key, it can only be decrypted using the private key.

public key Of the key pair, the key that is distributed to other systems as part of authentication or another security procedure.

public key cryptography The process of using a pair of mathematically related keys to verify the identity of hosts and users.

public-key authentication The authentication method that uses public-key cryptography to verify the client's identity and requires two pieces of data: your private-public key pair, and, optionally, a passphrase.

public-key cryptography A method of identifying hosts and users using two cryptographically generated keys: a public key and a private key.

public-private key pair The set of keys required to perform cryptographic security.

remote command execution The process of establishing an interactive session on a remote system without connecting to it (also called tunneling).

remote login The process of logging into a system running the SSH server from another system. SSH ensures the data communicated between your client and the SSH server is secure.

RSA	Rivest-Shamir-Adelman. A private key encryption algorithm
Secure Shell	The Internet standard for secure file transfers and remote login and command execution. Also known as SSH.
Secure Shell client	*See* SSH client.
Secure Shell server	*See* SSH server.
secure tunnel	A communication path established for securely transmitting data using applications that are not SSH-aware.
SSH	See Secure Shell. In the context of the TCP/IP Services for OpenVMS documentation, this is also referred to as SSH for OpenVMS.
SSH client	Secure Shell client.
SSH server	Secure Shell server.
stream-LF file	File record format in which data is stored as a stream of bytes.
trusted hosts	Hosts to which you can log in without proving your identity.
tunneling	*See* Remote Command Execution.
variable-length file	Record-oriented file structure inwhich the length of the record varies, and is determined from an explicit field or end-of-record marker.
X Window System	A protocol for displaying server data on a client system.
X11	A protocol for displaying X terminal formatted server data on client systems.
X11 port forwarding	An authentication method that encrypts X protocol, which is used by X Window systems.

Index

A
AllowedAuthentications, 22
authentication, 12
  customizing password, 23
  host-based, 22
  password, 22
  public-key, 22
  setting up host-based, 23
  setting_up public-key, 24
authentication methods, 21
  setting up, 22

C
client
  configuring, 16
  starting and stopping, 27
Client Components Configuration Menu, 16
Client-server communication, 13
comands
  and utilities, 11
configuration files, 13
configuring the SSH client, 16
configuring the SSH server, 17
copying files
  to and from an SSH server, 11
copying the server's public key to the client, 19
Credentials, forwarding, 36
cryptographic algorithms, 12

G
generating keys, 12, 46

H
host based authentication, 13
Host principal, Kerberos, 35
HOSTKEY., 16
HOSTKEY.PUB, 16

K
Kerberos
  authentication methods, 36
  client configuration, 36
  password authentication, 38
Kerberos connections, establishing, 36
Kerberos host principal, configuring, 35
Kerberos/SSH setup, 35
key
  agent, 47
  public host, naming conventions for the Server's, 20
  public, copying the server's, 19
keys
  generating, 12, 46
  host, 12
  managing user's, 12
  overview, 12
  user, 12
know hosts database, 22

L
logical names, 28

M
Main TCPIP$CONFIG.COM Configuration menu, 15
modifying client configuration parameters, 21

P
Password authentication
  Kerberos, 38
password authentication, 13
port forwarding, 14, 31

R
RANDOM_SEED., 16
remote command execution, 44
remote login, 44
RTL images, installing, 35
running the TCPIP$CONFIG configuration command procedure, 15

S
SCP, 11
SCP command, 41
Secure Server (SSH)
  client definition, 11
  server definition, 11
Secure Shell
  client, 11
  overview, 11
  server, 11
secure tunnel, 14
server
  configuring, 17
  starting and stopping, 27
Server Components Configuration Menu, 17
SFTP, 11
SFTP command, 43
SHOSTS.EQUIV, 16
specifying authentication methods, 22
SSH command, 44
SSH communication process, 13
SSH2_CONFIG, 16
SSH_ADD, 46
SSH_ADD utility, 12
SSH_AGENT utility, 12
SSH_KEYGEN, 46
SSH_KEYGEN utility, 12
SSHD2_CONFIG., 16, 21
starting and stopping the SSH client, 27
starting and stopping the SSH server, 27
STREAM_LF format, 13
SYS$SYSDEVICE, 16

T
TCPIP$CONFIG.COM command procedure, 12–13
TCPIP$DEFINE_COMMANDS procedure, 41
TELNET, 31

U
utilities
  and commands, 11
  SSH_ADD, 12
  SSH_AGENT, 12
  SSH_KEYGEN, 12

V
variable-length format, 13

X
X Windows Systems, 14, 31