Managing Secure Shell Access in Oracle® Solaris 11.3

Part No: E54793
October 2019

Managing Secure Shell Access in Oracle Solaris 11.3

Part No: E54793

Contents

Using This Documentation

- **Overview** – Describes how to administer and use Secure Shell on Oracle Solaris systems.
- **Audience** – System administrators who must implement security on the enterprise.
- **Required knowledge** – Familiarity with security concepts and terminology.

Product Documentation Library

Documentation and resources for this product and related products are available at `http://www.oracle.com/pls/topic/lookup?ctx=E53394-01`.

Feedback

Provide feedback about this documentation at `http://www.oracle.com/goto/docfeedback`.

♦ ♦ ♦ C H A P T E R 1

Using Secure Shell

The Secure Shell feature of Oracle Solaris provides secure access to a remote host over an unsecured network. The shell provides commands for remote login, remote window display, and remote file transfer. This chapter covers the following topics:

For reference information, see Chapter 2, "Secure Shell Reference".

What's New in Secure Shell in Oracle Solaris 11.3

This section highlights information for existing customers about important new Secure Shell features in this release.

- Oracle Solaris offers an `openssh` implementation of Secure Shell. The implementation is built on the latest OpenSSH project, plus modifications for the Oracle Solaris environment.

  The `sunssh` implementation (SunSSH) is still the default. The `pkg mediator` command switches between the two implementations. For more information, see "OpenSSH Implementation of Secure Shell" on page 13 and "How to Use the OpenSSH Implementation of Secure Shell" on page 26.
- The packaging for Secure Shell is changed to handle the SunSSH and OpenSSH implementations. See "Secure Shell Packages and Configuration Files" on page 12.

- Recent releases of Oracle Solaris contain modifications to the SunSSH implementation. For more information, see "SunSSH Implementation of Secure Shell" on page 18.
- Recent releases of Oracle Solaris enable different versions of Secure Shell on networked systems to use identical Secure Shell configuration files. For more information, see "Sharing `.ssh/config` Files Between Multiple Oracle Solaris Releases" on page 22.
- Oracle Solaris supports GSS-API credentials for authentication. See "GSS-API Authentication in Secure Shell" on page 24.

Note - Because the man pages differ between the SunSSH and OpenSSH implementations, you should open a terminal and use the `man` command to view the Secure Shell man pages that document the Secure Shell implementation on your system.

About Secure Shell

Secure Shell is the active remote access protocol on a newly installed Oracle Solaris system. The default implementation of Secure Shell is the `sunssh` implementation (SunSSH). The `openssh` implementation (OpenSSH) is also available.

SunSSH uses low-level cryptography APIs from the OpenSSL `libcrypto` library. The OpenSSL toolkit implements the Secure Sockets Layer and Transport Layer Security. To display the version of OpenSSL, run the `openssl version` command in a terminal window.

Note - OpenSSL can implement FIPS 140-2, a U.S. government computer security standard for cryptography modules. See "OpenSSH and FIPS 140-2" on page 18 and "SunSSH and FIPS 140-2" on page 19.

In Secure Shell, authentication is provided by the use of passwords, public keys, or both. All network traffic is encrypted. Thus, Secure Shell prevents a would-be intruder from being able to read an intercepted communication. Secure Shell also prevents an adversary from spoofing the system.

Secure Shell can also be used as an on-demand virtual private network (VPN). A VPN can forward X Window system traffic or can connect individual port numbers between the local systems and remote systems over an encrypted network link.

With Secure Shell, you can perform these actions:

- Log in to another host securely over an unsecured network.
- Copy files securely between the two hosts.

- Run commands securely on the remote system.

On the server side, Secure Shell supports version 2 (v2) of the Secure Shell protocol. On the client side, in addition to v2, the client supports version 1 (v1).

Secure Shell Authentication

Secure Shell provides public key and password methods for authenticating the connection to the remote system. Public key authentication is a stronger authentication mechanism because the private key never travels over the network.

The authentication methods are tried in the following order. When the configuration does not satisfy an authentication method, the next method is tried.

- **GSS-API authentication –** Uses credentials for GSS-API mechanisms such as `mech_krb5` (Kerberos V) to authenticate Secure Shell clients and servers. For more information about GSS-API authentication, see "Introduction to GSS-API" in *Developer's Guide to Oracle Solaris 11.3 Security*.
- **Host-based authentication –** Uses host keys and `rhosts` files. Uses the Secure Shell client's RSA or DSA public/private host keys to authenticate the client. Uses the files. Uses the client's RSA or DSA public/private host keys to authenticate the client. Uses the `rhosts` files to authorize clients to users.
- **Public key authentication –** Authenticates users with their RSA or DSA public/private keys.
- **Keyboard-interactive authentication –** Uses PAM to authenticate users. Keyboard authentication method in v2 allows for arbitrary prompting by PAM. For more information, see the `SECURITY` section in the `sshd`(1M) man page.

The following table shows the requirements for authenticating a user who is trying to log into a remote system. The user is on the local system, the client system. The remote system, the Secure Shell server, is running the `sshd` daemon. The table shows the Secure Shell authentication methods and the system requirements.

TABLE 1 Authentication Methods for Secure Shell

Secure Shell Authentication Method	Local Host (Client) Requirements	Remote Host (Server) Requirements
GSS-API	Initiator credentials for the GSS mechanism.	Acceptor credentials for the GSS mechanism. For more information, see "Acquiring GSS Credentials in Secure Shell" on page 48.
Host-based	User account	User account

Secure Shell Authentication Method	Local Host (Client) Requirements	Remote Host (Server) Requirements

	Local host private key in `/etc/ssh/ssh_host_rsa_key` or `/etc/ssh/ssh_host_dsa_key`
	
	`HostbasedAuthentication yes` in `/etc/ssh/ssh_config`
		Local host public key in `/etc/ssh/known_hosts` or `~/.ssh/known_hosts`
		
		`HostbasedAuthentication yes` in `/etc/ssh/sshd_config`
		
		`IgnoreRhosts no` in `/etc/ssh/sshd_config`
		
		Local host entry in `/etc/ssh/shosts.equiv`, `/etc/hosts.equiv`, `~/.rhosts`, or `~/.shosts`

Password-based	User account	User account Supports PAM.
RSA or DSA public key	User account Private key in `~/.ssh/id_rsa` or `~/.ssh/id_dsa` User's public key in `~/.ssh/id_rsa.pub` or `~/.ssh/id_dsa.pub`	User account User's public key in `~/.ssh/authorized_keys`
X.509 public key certificates	User certificates `HostKey` pointer to certificate URI Policy in `/etc/ssh/policy.xml` PIN file in `etc/ssh/pinfile`	KMF Policy `HostKey` pointer to certificate URI Policy in `/etc/ssh/policy.xml` Certificates in `/etc/ssh/cert`, OpenSSL keystore, or PKCS #11 keystore

Secure Shell Packages and Configuration Files

With the introduction of the OpenSSH implementation of Secure Shell, Oracle Solaris provides four Secure Shell packages:

`network/ssh`	Components for the `sunssh` implementation
`network/openssh`	Components for the `openssh` implementation
`network/ssh/ssh-utilities`	Secure Shell utilities that the two implementations share
`network/ssh-common`	SMF service and Secure Shell system configuration files that the two implementations share

SunSSH and OpenSSH share the following Secure Shell system configuration files:

- `/etc/ssh/ssh_config`
- `/etc/ssh/sshd_config`
- `/etc/ssh/moduli`

Implementation-specific man pages are installed with each package.

OpenSSH Implementation of Secure Shell

OpenSSH in Oracle Solaris is built on the latest version of the OpenSSH project, plus additions that are particular to the Oracle Solaris environment. To see which version of the OpenSSH project is the basis for OpenSSH on your system, run the `pkg info openssh` command.

Although SunSSH is the default, you can switch to the new OpenSSH implementation. You can use only one implementation at a time.

Caution - Beginning with the Oracle Solaris 11.3 SRU 5, OpenSSH has been upgraded to a version which requires active attention from system administrators. Review the changes in "Oracle Solaris Modifications to OpenSSH" on page 13 and "Oracle Solaris Additions to OpenSSH" on page 17.

Oracle Solaris Modifications to OpenSSH

Note - If your system is running SunSSH, these OpenSSH modifications do not affect the system's behavior.

Oracle Solaris makes the following changes to OpenSSH:

- "SunSSH Keywords Are Deprecated in OpenSSH" on page 14
- "Secure Shell Protocol 1 Support Is Removed" on page 14
- "Unsafe Algorithms Are Removed From OpenSSH" on page 14
- "`diffie-hellman-group1-sha1` Is Disabled by Default" on page 15
- "`ssh-dss` Keys Are Disabled by Default" on page 15
- "Default Value of `UseDNS` Is `No` in OpenSSH" on page 16
- "TCP Wrappers Are Not Supported in OpenSSH" on page 16

SunSSH Keywords Are Deprecated in OpenSSH

To ease the transition from SunSSH to OpenSSH, SunSSH-only options are available but deprecated in OpenSSH. OpenSSH ignores a deprecated option.

Deprecated server options include the following:

```
GSSAPIStoreDelegatedCredentials
GSSStoreDelegCreds
KMFPolicyDatabase
KMFPolicyName
LookupClientHostnames
MaxAuthTriesLog
PreUserauthHook
TrustedAnchorKeystore
UseFIPS140
UseOpenSSLEngine
UseUnsupportedSshv1
```

Deprecated client options include the following:

```
KMFPolicyDatabase
KMFPolicyName
TrustedAnchorKeystore
UseFIPS140
UseOpenSSLEngine
```

Secure Shell Protocol 1 Support Is Removed

Secure Shell Protocol 1 (v1) support is absent from OpenSSH on both the server side and the client side. If you have network entities that support only v1, such as old network routers, you can no longer connect to such devices by using OpenSSH. However, Oracle Solaris 10 users can still use SunSSH to access systems that use v1. And, Oracle Solaris 11.3 users can still use SunSSH to access systems that use v1.

Unsafe Algorithms Are Removed From OpenSSH

The default set of ciphers and MACs has been altered to remove unsafe algorithms. You can use the following commands to list all supported ciphers and MACs:

```
$ ssh -Q cipher
$ ssh -Q mac
```

For more information, see the `sshd_config`(4) and `ssh_config`(4) man pages.

`diffie-hellman-group1-sha1` Is Disabled by Default

Because the `diffie-hellman-group1-sha1` key exchange is no longer considered secure, it is disabled on both the Secure Shell client and server sides.

If your servers support only `diffie-hellman-group1-sha1`, you should upgrade them to support `diffie-hellman-group-exchange-sha256`. Or, as a second choice, upgrade Oracle Solaris to a version which supports `diffie-hellman-group14-sha1`.

If upgrading the peer is not an option, users connecting to systems that do not support `diffie-hellman-group-exchange-sha256`, `diffie-hellman-group14-sha1`, or `diffie-hellman-group-exchange-sha1` can enable `diffie-hellman-group1-sha1` as follows:

```
# ssh -oKexAlgorithms=+diffie-hellman-group1-sha1 user@system1
```

For the OpenSSH implementation of Secure Shell, the server administrator can allow logins from systems that do not support secure key exchange methods by explicitly enabling insecure key exchange methods. Add this line to the `/etc/ssh/sshd_config` file.

```
KexAlgorithms
diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group-exchange-
sha1,diffie-hellman-group-exchange-sha256
```

Then, restart the `ssh` SMF service on the server.

```
ssh-server# svcadm restart ssh
```

For additional information, see Using OpenSSH with Legacy SSH Implementations.

`ssh-dss` Keys Are Disabled by Default

Because the `ssh-dss` and `ssh-dss-cert-*` host and user key types are inherently weak, they are disabled by default at run time.

If you have been using `ssh-dss` keys for public key authentication, you should create new `ssh-rsa` keys and remove existing `ssh-dss` keys from all `authorized_keys` files. For instructions about creating new keys, see "How to Generate a Public/Private Key Pair for Use With Secure Shell" on page 35.

If `ssh-rsa` and `ssh-dss` host keys are not already present, `svc:/network/ssh:default` creates both. So, Oracle Solaris servers usually have `ssh-dss` host keys and `ssh-rsa` keys. In the rare cases where servers were provisioned with only an `ssh-dss` host key, you should add a `ssh_rsa` host key. If you cannot create the key, then the user needs to enable the `ssh-dss` key type on the command line to connect to the server by running the following command:

```
# ssh -oHostKeyAlgorithms=+ssh-dss user@somehost
```

For additional information, see Using OpenSSH with Legacy SSH Implementations.

Default Value of `UseDNS` Is `No` in OpenSSH

If no `UseDNS` value is specified in the `sshd_config` file, the default value of `UseDNS` is `No`. The former default value provided no security benefit.

A `UseDNS` value of `No` means that you cannot use host names when configuring an `ssh` service. You have two options:

- You can explicitly specify `UseDNS yes` in the `sshd_config` file.
- You can use IP addresses instead of host names in the `sshd_config` file as shown in the following examples.
  - In the `Match` block section of the `sshd_config` file, use an `Address` criterion instead of a `Host` criterion.

    For example, you would replace `Match Host somehost.domain` with `Match Address 192.0.2.11`.
  - In the `sshd_config` entries for `AllowUsers`, `AllowGroups`, `DenyUsers`, and `DenyGroups`, use an IP address instead of the host name.

    For example, you would replace `AllowUsers jsmith@somehost.domain` with `AllowUsers jsmith@192.0.2.11`.
  - In `/etc/ssh/shosts.equiv` or `~/.shosts` entries, use an IP address instead of a host name.

    For example, you would replace `somehost.domain` with `192.0.2.11`.
  - In the `~/.ssh/authorized_keys` entry, use an IP address instead of a host name when specifying the `from` option.

    For example, you would replace `from="somehost.domain" ssh-rsa AAAAB3...Q== jsmith@work` with `from="192.0.2.11" ssh-rsa AAAAB3...Q== jsmith@work`.

TCP Wrappers Are Not Supported in OpenSSH

OpenSSH does not support TCP wrappers. You will need to modify the `sshd_config` file or use a firewall to preserve a configuration that was previously enforced by TCP wrappers.

Note - The openssh implementation of Secure Shell continues to use TCP connections. Only the TCP wrapper function, libwrap, is no longer supported.

If you have been using TCP wrappers, you have been using /etc/hosts.allow or /etc/hosts.deny to allow or deny logins. Use the Match block in the sshd_config file to set up an equivalent configuration.

For example, to allow logins only from the 192.0.2.0/16 subnet, you might have set up TCP wrappers as follows:

```
root@jsmith-cz:~# cat /etc/hosts.allow
    sshd : 192.0.
    root@jsmith-cz:~# cat /etc/hosts.deny
    ALL : ALL
```

The following entry in the sshd_config file sets an equivalent restriction:

```
Match Address *,!192.0.2.0/16
            MaxAuthTries 0
```

Another option is to configure a firewall for access control. Settings similar to these examples can be enforced by a firewall. Firewall access control occurs before the network connection is established in the kernel.

Oracle Solaris Additions to OpenSSH

OpenSSH is an optional package, openssh, that you can install on your system. Although features have been added to the openssh implementation, it remains compatible with the OpenSSH project.

The following Oracle Solaris features have been added to the openssh implementation:

- DisableBanner keyword – disables the display of a banner message from the Secure Shell client. For more information, see the ssh_config(4) man page.
- PAM support – PAMServiceName and PAMServicePrefix keywords are available in the openSSH implementation.
- Auditing support – OpenSSH generates Oracle Solaris audit records for login and logout.
- Xforwarding – Is functional with non-writable home directories.
- Proxy commands – For SOCKS5 and HTTP protocols work in SunSSH and in OpenSSH. For an example, see "How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall" on page 44. The netcat utility provides similar functionality.

- GSSAPI-Authenticated Diffie-Hellman Key Exchange – Is implemented as specified in RFC 4462.
- Role login – Is implemented for host-based authentication if properly configured in PAM and `sshd`.

Delegated credentials in OpenSSH are stored differently on Oracle Solaris than the credentials are stored on other platforms.

- Unlike the OpenSSH project, which stores delegated credentials in a non-default credential cache such as `/tmp/krb5cc_101_WO4082`, the `openssh` implementation of Secure Shell uses a default credential cache such as `/tmp/krb5cc_101`.
- Credentials in a default credential cache can be used to access NFS file systems that are protected by Kerberos.

  For more information, see *Managing Kerberos and Other Authentication Services in Oracle Solaris 11.3*.
- OpenSSH can run in FIPS 140-2 mode.

  For more information, see "OpenSSH and FIPS 140-2" on page 18.

OpenSSH and FIPS 140-2

Unlike SunSSH, OpenSSH enables FIPS 140-2 mode dynamically. If the OpenSSL library that OpenSSH links with is FIPS 140-2 capable, OpenSSH runs in FIPS 140-2 mode.

See "Example of Running in FIPS 140-2 Mode on an Oracle Solaris 11.3 SRU 5.6 System" in *Using a FIPS 140-2 Enabled System in Oracle Solaris 11.3*.

SunSSH Implementation of Secure Shell

The SunSSH implementation of Secure Shell is a fork of the OpenSSH (`http://www.openssh.com`) project.

Security fixes for vulnerabilities in later versions of OpenSSH have been integrated into the `sunssh` implementation of Secure Shell (SunSSH), as are individual bug fixes and features.

The following features have been implemented in the current release of SunSSH:

- `ForceCommand` keyword – Forces the execution of the specified command regardless of what the user types on the command line. This keyword is very useful inside a `Match` block.

- `AES-128` passphrase protection – Private keys that are generated by the `ssh-keygen` command are protected with the `AES-128` algorithm. This algorithm protects newly generated keys and re-encrypted keys, such as when a passphrase is changed.
- `-u` option to `sftp-server` command – Enables user to set an explicit `umask` on files and directories. This option overrides the user's default `umask`. For an example, see the description of `Subsystem` on the `sshd_config(4)` man page.
- Additional keywords for `Match` blocks – `AuthorizedKeysFile`, `ForceCommand`, and `HostbasedUsesNameFromPacketOnly` are supported inside `Match` blocks. By default, the value of `AuthorizedKeysFile` is `$HOME/.ssh/authorized_keys` and `HostbasedUsesNameFromPacketOnly` is `no`. To use `Match` blocks, see "How to Create User and Host Exceptions to Secure Shell Defaults" on page 31.
- `UseOpenSSLEngine` – On x86 systems and on T4-Series and later SPARC systems, this keyword is disabled by default. The platform-specific instructions are already embedded in the OpenSSL cryptographic implementation.

SunSSH differs from OpenSSH in the following ways:

- SunSSH-only keywords – SunSSH-only keywords are deprecated in OpenSSH. For a list, see "SunSSH Keywords Are Deprecated in OpenSSH" on page 14.
- PAM – SunSSH and OpenSSH always use PAM. Changes to the `UsePAM` keyword are ignored.
- Privilege separation – SunSSH privilege separation code is always on and cannot be switched off. No keyword is associated with privilege separation. The SunSSH implementation separates the processing of auditing, record keeping and re-keying from the processing of the session protocols.

  On OpenSSH, configure the `UsePrivilegeSeparation` keyword.
- Locale – SunSSH fully supports language negotiation as specified in Secure Shell Transfer Protocol, RFC 4253. After the user logs in, the user's login shell profile can override the Secure Shell negotiated locale settings.

  OpenSSH does not support language negotiation.
- Auditing – SunSSH is fully integrated into the Oracle Solaris audit service. For information about the audit service, see *Managing Auditing in Oracle Solaris 11.3*.

SunSSH and FIPS 140-2

Like OpenSSH, SunSSH is a consumer of the OpenSSL FIPS 140-2 module. Unlike OpenSSH, SunSSH must be configured to link to the OpenSSL FIPS 140-2 module. To comply with FIPS 140-2 requirements, administrators should configure and use the FIPS 140-2 options for SunSSH.

SunSSH in FIPS 140-2 mode is not the default. As the administrator, you must explicitly enable SunSSH to run in FIPS 140-2 mode with the following command:

```
$ ssh -o "UseFIPS140 yes" remote-host
```

You can also set a keyword in the configuration files.

Briefly, the implementation consists of the following:

- The following FIPS 140-2 approved ciphers are available on the SunSSH server and client side: `aes128-cbc`, `aes192-cbc`, and `aes256-cbc`.

  `3des-cbc` is available by default on the client side, but it is not in the SunSSH server-side cipher list because of potential security risks.
- The following FIPS 140-2 approved Message Authentication Codes (MAC) are available:
  - `hmac-sha1`, `hmac-sha1-96`
  - `hmac-sha2-256`, `hmac-sha2-256-96`
  - `hmac-sha2-512`, `hmac-sha2-512-96`
- Four SunSSH server-client configurations are supported:
  - No FIPS 140-2 mode on either the client or the server
  - FIPS 140-2 mode on both the client and the server
  - FIPS 140-2 mode on the server but no FIPS 140-2 mode on the client
  - No FIPS 140-2 mode on the server but FIPS 140-2 mode on the client
- The `ssh-keygen` command has an option to generate the user's private key in the PKCS #8 format that SunSSH clients in FIPS 140-2 mode require. For more information, see the `ssh-keygen`(1) man page.

For more information about FIPS 140-2 and SunSSH, see *Using a FIPS 140-2 Enabled System in Oracle Solaris 11.3*, "Creating a Boot Environment With FIPS 140-2 Enabled" in *Managing Encryption and Certificates in Oracle Solaris 11.3*, and the `sshd`(1M) man pages.

When you use a Sun Crypto Accelerator 6000 card for Secure Shell operations, SunSSH runs with FIPS 140-2 support at Level 3. Level 3 hardware is certified to resist physical tampering, use identity-based authentication, and isolate the interfaces that handle critical security parameters from the hardware's other interfaces.

New Keywords to Control Key Types in SunSSH

In SunSSH, new keywords were added to enable you to control accepted public key types to disable weak key types. The default is to accept all key types.

The following new keywords have been added to SunSSH.

- The following keywords have been added to the server configuration:

  ```
  HostKeyAlgorithms
  HostbasedAcceptedKeyTypes
  PubkeyAcceptedKeyTypes
  Kexalgorithms
  ```

  For more information, see the `sshd_config`(4) man page.
- The following keywords have been added to the SunSSH client configuration:

  ```
  HostbasedAcceptedKeyTypes
  PubkeyAcceptedKeyTypes
  Kexalgorithms
  ```

For more information, see the `ssh_config`(4) man page.

Using X.509 Certificates With SunSSH

X.509 certificates are a good choice for SunSSH authentication. They are the safest option for remote logins where interaction with the user is not permitted, such as when running remote scripts. Also, the user is not prompted to accept the host identity and the user public keys do not need to be present on the remote server.

When a user (a SunSSH client) tries to connect to the SunSSH server, the server passes the host certificate to the client. By using the public key of the Certificate Authority (CA) in the CA certificate, the client verifies the host certificate on the server against the digital signature associated with the CA.

X.509 certificate configuration requires the following steps:

1. The administrator generates an X.509 certificate for the server on the server that users will remotely log in to.
2. Users who plan to remotely log in to the server generate an X.509 certificate for themselves.
3. The administrator sends the public part of the server's root certificate to the administrator who configures users.
4. Every user sends the public part of their root certificate (referred to in SunSSH configuration files as a "Trusted Anchor" or TA) to the administrator of the remote server.
5. The server administrator stores the users' TA certificates where the `ssh` daemon can read them.
6. The user administrator stores the server's TA certificate where the `ssh` daemon can read it.

7. Then, users can use SunSSH to log in to the remote server.

You can also allow users to generate a self-signed trusted anchor (TA) certificate and sign it. Self-signed certificates are less secure. Users who self-sign certificates must be familiar with the technical and security issues around certificates.

For the procedure, see How to Add a CA Cert to Oracle Solaris (`https://blogs.oracle.com/solaris/how-to-add-a-ca-cert-to-solaris-v2`).

Sharing `.ssh/config` Files Between Multiple Oracle Solaris Releases

If your home directory is on a network, you can share the `~/.ssh/config` file among multiple systems, even if those systems are running different Oracle Solaris releases or different Secure Shell implementations. However, Secure Shell implementations might not recognize all the configuration options from different Secure Shell implementations. In some cases, Secure Shell implementations might not recognize configuration options from different versions of the same Secure Shell implementation.

For Oracle Solaris 10 Update 11 and later releases, when the Secure Shell configuration options cannot be recognized by the different systems on the network, you can modify the `ssh_config` file so that it will ignore options that are unrecognized, thus enabling use of the shared `~/.ssh/config` file among multiple systems.

Secure Shell Implementations and `Ignore` Keywords

Two keywords, `IgnoreIfUnknown` and `IgnoreUnknown`, can be used to ignore Secure Shell configuration keywords that are unrecognized among multiple systems. The `IgnoreIfUnknown` keyword is available in SunSSH and the `IgnoreUnknown` keyword is available in OpenSSH.

Both `IgnoreIfUnknown` and `IgnoreUnknown` specify a comma-separated list of `ssh_config` keywords, which, if unknown to the `ssh` program, are ignored by Secure Shell. However, while `IgnoreIfUnknown` applies to the entire configuration file, `IgnoreUnknown` applies only to unknown keywords that follow it in the configuration file.

The following table identifies the Secure Shell implementations in each Oracle Solaris release and the `Ignore` keywords that are available in each implementation.

TABLE 2 `Ignore` Keywords in Secure Shell

Release	Secure Shell Implementation	Supported `Ignore` Keyword
Oracle Solaris 11.3	SunSSH	`IgnoreIfUnknown` and `IgnoreUnknown`
Oracle Solaris 11.3	OpenSSH	`IgnoreUnknown`
Oracle Solaris 11 releases prior to Oracle Solaris 11.3	SunSSH	`IgnoreIfUnknown`
Oracle Solaris 10 Update 11	SunSSH	`IgnoreIfUnknown`

The following releases do not support the `Ignore` keywords and cannot be included as part of a shared Secure Shell configuration over a network.

- Oracle Solaris 9
- Oracle Solaris 10 prior to Update 11
- OpenSSH 6.2 and older OpenSSH versions

Ignoring Secure Shell Keywords to Enable Interoperability

If your `~/.ssh/config` file is on a network whose systems run different implementations of Secure Shell, you can enable the Secure Shell configuration keywords to work by adding the `IgnoreUnknown` and `IgnoreIfUnknown` keywords to the file.

Note - All systems must be able to use at least one of the `Ignore` keywords as listed in Table 2, "`Ignore` Keywords in Secure Shell," on page 23.

EXAMPLE 1 Sharing a Secure Shell Configuration Across Releases That Support Different Keywords

This example shows how to use the `HostBasedKeyTypes` keyword that was introduced in OpenSSH 6.8. The users are on a network where some systems are running a release of Secure Shell that does not support this keyword.

Add the following entries to the `ssh_config` file:

```
---
IgnoreUnknown HostBasedKeyTypes,IgnoreIfUnknown
IgnoreIfUnknown HostBasedKeyTypes,IgnoreUnknown

HostBasedKeyTypes ssh-rsa-cert-v01@openssh.com, ssh-rsa
```

```
---
```

You add both the `Ignore` keywords to enable all Secure Shell implementations to communicate. For more information, see the `ssh_config`(4) man page.

GSS-API Authentication in Secure Shell

To use GSS-API authentication in Secure Shell, the server must have GSS-API acceptor credentials and the client must have GSS-API initiator credentials. Support is available for `mech_krb5`.

For `mech_krb5`, the server has GSS-API acceptor credentials when the host principal that corresponds to the server has a valid entry in `/etc/krb5/krb5.keytab`.

The client has initiator credentials for `mech_krb5` if one of the following has been done:

- The `kinit` command has been run.
- The `pam_krb5` module is used in the `pam.conf` file.

See also "Per-Session GSS-API Credentials" on page 24. For more information about mechanisms, see the `mech`(4) man page.

Per-Session GSS-API Credentials

Oracle Solaris supports per-session GSS-API credentials. The `sshd_config` file contains the option, `GSSAPIUseDefaultCCache`. By default, this option is set to `yes`.

The default behavior is the following:

- Delegated credentials are stored in the default credential cache
- `KRB5CCNAME` is not set
- The credentials are not deleted when the session ends

When you change this value to `no`, you create per-session credentials:

- Delegated credentials are stored in a per-session credential cache in `/tmp/krb5cc_uid_`*XXXXXX*

  The file path is created using a random pattern.
- `KRB5CCNAME` points to the cache file path.
- If the `GSSAPICleanupCredentials` variable is set to the default value of `yes`, the credential cache is destroyed upon logout. This variable is in the `sshd_config` file.

For more information, use the `man` command to view the `sshd_config`(4) man page.

Configuring Secure Shell

Secure Shell is configured at installation with the `sunssh` implementation set as the default implementation. Changing the defaults for Secure Shell requires administrative intervention. The following tasks demonstrate how to configure Secure Shell at your site.

Configuring Secure Shell Task Map

The following task map points to administrative procedures for configuring Secure Shell. Procedures that regular users can perform are in "Using Secure Shell" on page 34.

Task	Description	For Instructions
Run OpenSSH.	Switches from the default SunSSH to the latest OpenSSH implementation of Secure Shell.	"How to Use the OpenSSH Implementation of Secure Shell" on page 26
Run Secure Shell in FIPS 140-2 mode.	Enables your Secure Shell implementation to use FIPS 140-2 ciphers from OpenSSL.	"Example of Running in FIPS 140-2 Mode on an Oracle Solaris 11.3 SRU 5.6 System" in *Using a FIPS 140-2 Enabled System in Oracle Solaris 11.3* and "Creating a Boot Environment With FIPS 140-2 Enabled" in *Managing Encryption and Certificates in Oracle Solaris 11.3*
Configure X.509 certificates for the SunSSH client and server.	Enables SunSSH to use certificate-based authentication.	How to add a CA cert to Oracle Solaris (`https://blogs.oracle.com/solaris/how-to-add-a-ca-cert-to-solaris-v2`).
Configure host-based authentication.	Configures host-based authentication on the client and server.	"How to Set Up Host-Based Authentication for Secure Shell" on page 27
Increase buffer size to handle connection latency.	Raises the value of the TCP property `recv_buf` for high bandwidth, high latency networks.	"Changing the TCP Receive Buffer Size" in *Administering TCP/IP Networks, IPMP, and IP Tunnels in Oracle Solaris 11.3*
Configure port forwarding.	Enables users to use port forwarding.	"How to Configure Port Forwarding in Secure Shell" on page 30
Configure exceptions to Secure Shell defaults.	For users, hosts, groups, and addresses, specifies Secure Shell values that are different from the defaults.	"How to Create User and Host Exceptions to Secure Shell Defaults" on page 31
Isolate a `root` environment for `sftp` transfers.	Provides a protected directory for file transfers.	"How to Create an Isolated Directory for `sftp` Files" on page 32

▼ How to Use the OpenSSH Implementation of Secure Shell

Before You Begin You must be assigned the Software Installation rights profile to add packages to the system. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

Caution - Beginning with the Oracle Solaris 11.3 SRU 5 release, the OpenSSH implementation of Secure Shell has been upgraded to a version that requires active attention from system administrators. See "OpenSSH Implementation of Secure Shell" on page 13.

1. **Find out whether the `openssh` package is installed.**

```
# pkg list openssh
pkg list: no packages matching the following patterns are installed:
  openssh
```

2. **If the `openssh` package is not listed, install the package.**

```
# pkg install network/openssh
```

3. **View all implementations of Secure Shell on the system.**

```
# pkg mediator -a ssh
MEDIATOR     VER. SRC. VERSION IMPL. SRC. IMPLEMENTATION
ssh          vendor            vendor     sunssh
ssh          system            system     openssh
```

In the output, `vendor` indicates the default implementation, which is `sunssh` in this release.

4. **Switch to the `openssh` implementation.**

```
# pkg set-mediator -I openssh ssh
            Packages to change:   3
           Mediators to change:   1
            Services to change:   1
       Create boot environment:  No
Create backup boot environment: Yes
PHASE                                            ITEMS
Removing old actions                             34/34
Updating modified actions                        25/25
Updating package state database                   Done
Updating package cache                             0/0
Updating image state                              Done
Creating fast lookup database                     Done
```

```
Updating package cache                           1/1
```

Note - The changes will include all the appropriate man pages for the implementation that you select.

This command restarts the Secure Shell server. The existing Secure Shell connections continue to work. Users currently on the server can continue to use the prior implementation, or they can log out and log in to use the new implementation.

5. **(Optional) Display the implementation of Secure Shell that is in effect.**

```
$ pkg mediator ssh
MEDIATOR   VER. SRC.   VERSION   IMPL. SRC.   IMPLEMENTATION
ssh        system                local        openssh
```

In this example, the `openssh` implementation is enabled.

For further information about using the `pkg mediator` command, see "Changing the Preferred Application" in *Adding and Updating Software in Oracle Solaris 11.3* and the `pkg(1)` man page.

6. **(Optional) Revert to SunSSH.**

```
# pkg set-mediator -I sunssh ssh
```

This command restarts the Secure Shell server. The existing Secure Shell connections continue to work. Users currently on the server can continue to use the prior implementation, or they can log out and log in to use the new implementation.

▼ How to Set Up Host-Based Authentication for Secure Shell

The following procedure sets up a public key system where the client's public key is used for authentication on the Secure Shell server. The user must also create a public/private key pair.

In the procedure, the terms *client* and *local host* refer to the system where a user types the `ssh` command. The terms *server* and *remote host* refer to the system that the client is trying to reach.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the client, enable host-based authentication.**

   In the client configuration file, `/etc/ssh/ssh_config`, add the following entry:

```
HostbasedAuthentication yes
```

For the syntax of the file, see the `ssh_config`(4) man page.

2. **On the Secure Shell server, enable host-based authentication.**

   In the server configuration file, `/etc/ssh/sshd_config`, add the same entry:

   ```
   HostbasedAuthentication yes
   ```

3. **On the server, either you or the user should configure a file that enables the client to be recognized as a trusted host.**

   For more information, see the `FILES` section of the `sshd`(1M) man page.

   - **If you are doing the configuration, add the client as an entry to the server's `/etc/ssh/shosts.equiv` file.**

     client-host

   - **If your users are doing the configuration, they should add an entry for the client to their `~/.shosts` file on the server.**

     client-host

4. **On the server, ensure that the `sshd` daemon can access the list of trusted hosts.**

   Set `IgnoreRhosts` to `no` in the `/etc/ssh/sshd_config` file.

   ```
   ## sshd_config
   IgnoreRhosts no
   ```

5. **Ensure that users of Secure Shell at your site have accounts on both hosts.**

6. **Put the client's public key on the server using one of the following methods:**

   - **Modify the `sshd_config` file on the server, then instruct your users to add the client's public host keys to their `~/.ssh/known_hosts` file.**

     ```
     ## sshd_config
     IgnoreUserKnownHosts no
     ```

     For user instructions, see "How to Generate a Public/Private Key Pair for Use With Secure Shell" on page 35.

   - **Copy the client's public key to the server.**

The host keys are stored in the `/etc/ssh` directory. The keys are typically generated by the `sshd` daemon on first boot.

a. **Add the key to the `/etc/ssh/ssh_known_hosts` file on the server.**

   On the client, type the following command on one line with no backslash.

```
# cat /etc/ssh/ssh_host_rsa_key.pub | ssh RemoteSys \
'cat >> /etc/ssh/ssh_known_hosts && echo "Host key copied"'
```

Note - If host keys are missing from the server, using Secure Shell generates an error message similar to the following:

```
Client and server could not agree on a key exchange algorithm:
client "diffie-hellman-group-exchange-sha256,diffie-hellman-group-
exchange-sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1",
server "gss-group1-sha1-toWM5Slw5Ew8Mqkay+al2g==". Make sure host keys
are present and accessible by the server process. For more information, see
the description of the HostKey keyword in ssd_config(4).
```

---

b. **When you are prompted, supply your login password.**

   When the file is copied, the message "Host key copied" is displayed.

c. **Prepend *RemoteHost* to the copied entry in the `ssh_known_hosts` file.**

   Each line in the `/etc/ssh/ssh_known_hosts` file consists of fields that are separated by spaces:

   hostnames algorithm-name publickey comment

   Place *RemoteHost* in the *hostnames* field.

```
## /etc/ssh/ssh_known_hosts File
RemoteHost  <copied entry>
```

Example 2 Setting Up Host-based Authentication

In the following example, each host is configured as a server and as a client. A user on either host can initiate an `ssh` connection to the other host. The following configuration makes each host a server and a client:

- On each host, the Secure Shell configuration files contain the following entries:

```
## /etc/ssh/ssh_config
HostBasedAuthentication yes
```

```
#
## /etc/ssh/sshd_config
HostBasedAuthentication yes
IgnoreRhosts no
```

- On each host, the `shosts.equiv` file contains an entry for the other host:

  ```
  ## /etc/ssh/shosts.equiv on system2
  system1

  ## /etc/ssh/shosts.equiv on system1
  system2
  ```

- The public key for each host is in the `/etc/ssh/ssh_known_hosts` file on the other host:

  ```
  ## /etc/ssh/ssh_known_hosts on system2
  ... system1

  ## /etc/ssh/ssh_known_hosts on system1
  ... system2
  ```

- Users have an account on both hosts. For example, the following information would appear for user Jane Doe:

  ```
  ## /etc/passwd on system1
  jdoe:x:3111:10:J Doe:/home/jdoe:/bin/sh

  ## /etc/passwd on system2
  jdoe:x:3111:10:J Doe:/home/jdoe:/bin/sh
  ```

▼ How to Configure Port Forwarding in Secure Shell

Port forwarding enables a local port be forwarded to a remote system. Effectively, a socket is allocated to listen to the port on the local side. Similarly, a port can be specified on the remote side.

Note - Secure Shell port forwarding must use TCP connections. Secure Shell does not support UDP connections for port forwarding.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Configure a Secure Shell setting on the remote server to allow port forwarding.**

   Change the value of `AllowTcpForwarding` to `yes` in the `/etc/ssh/sshd_config` file.

```
# Port forwarding
AllowTcpForwarding yes
```

2. **Restart the Secure Shell service.**

   ```
   remoteHost# svcadm restart network/ssh:default
   ```

   For information about managing persistent services, see Chapter 1, "Introduction to the Service Management Facility" in *Managing System Services in Oracle Solaris 11.3* and the `svcadm`(1M) man page.

3. **Verify that port forwarding can be used.**

   ```
   remoteHost# /usr/bin/pgrep -lf sshd
   1296 ssh -L 2001:remoteHost:23 remoteHost
   ```

▼ How to Create User and Host Exceptions to Secure Shell Defaults

This procedure adds a conditional `Match` block after the global section of the `/etc/ssh/sshd_config` file. Keyword-value pairs that follow the `Match` block specify exceptions for the user, group, host, or address that is specified as the match.

Before You Begin You must become an administrator who is assigned the `solaris.admin.edit/etc/ssh/sshd_config` authorization. By default, the `root` role has this authorization. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Open the `/etc/ssh/sshd_config` file for editing.**

   ```
   # pfedit /etc/ssh/sshd_config
   ```

2. **Configure a user, group, host, or address to use different Secure Shell settings from the default settings.**

   Place the `Match` blocks after the global settings.

   Note - The global section of the file might not always list the default settings. For the defaults, see the `sshd_config`(4) man page.

For example, you might have users who should not be allowed to use TCP forwarding. In this configuration, any user in the group `public` and any user whose name begins with `test` cannot use TCP forwarding:

```
## sshd_config file
## Global settings

# Example (default SunSSH values):
#
# Host *
#   ForwardAgent no
#   ForwardX11 no
#   PubkeyAuthentication yes
#   PasswordAuthentication yes
#   FallBackToRsh no
#   UseRsh no
#   BatchMode no
#   CheckHostIP yes
#   StrictHostKeyChecking ask
#   EscapeChar ~
Match Group public
AllowTcpForwarding no
Match User test*
AllowTcpForwarding no
```

For information about the syntax of the `Match` keyword, see the `sshd_config`(4) man page.

▼ How to Create an Isolated Directory for `sftp` Files

This procedure configures an `sftponly` directory that you create specifically for `sftp` transfers. Users cannot see any files or directories outside this directory.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the Secure Shell server, create the isolated directory as a `chroot` environment.**

```
# groupadd sftp
# useradd -m -G sftp -s /bin/false sftponly
# chown root:root /export/home/sftponly
# mkdir /export/home/sftponly/WWW
# chown sftponly:staff /export/home/sftponly/WWW
```

In this configuration, `/export/home/sftonly` is the `chroot` directory that only the `root` account has access to. Users have write permission to the `sftponly/WWW` subdirectory.

2. **Still on the server, configure a `Match` block for the `sftp` group.**

   In the `/etc/ssh/sshd_config` file, locate the `sftp subsystem` entry and modify the file as follows:

```
# pfedit /etc/ssh/sshd_config
...
## sftp subsystem
##Subsystem       sftp    /usr/lib/ssh/sftp-server
Subsystem       sftp    internal-sftp
...
## Match Group for Subsystem
## At end of file, to follow all global keywords
Match Group sftp
ChrootDirectory %h
ForceCommand internal-sftp
AllowTcpForwarding no
```

   You can use the following variables to specify the `chroot` path:

   - `%h` – Specifies the home directory.
   - `%u` – Specifies the username of the authenticated user.
   - `%%` – Escapes the `%` sign.

3. **On the client, verify that the configuration works correctly.**

   The files in your `chroot` environment might be different.

```
root@client:~# ssh sftponly@server
This service allows sftp connections only.
Connection to server closed.       No shell access, sftp is enforced.
root@client:~# sftp sftponly@server
sftp> pwd       sftp access granted
Remote working directory: /       chroot directory looks like root directory
sftp> ls
WWW             local.cshrc     local.login     local.profile
sftp> get local.cshrc
Fetching /local.cshrc to local.cshrc
/local.cshrc    100%  166     0.2KB/s   00:00      user can read contents
sftp> put /etc/motd
Uploading /etc/motd to /motd
Couldn't get handle: Permission denied      user cannot write to / directory
sftp> cd WWW
sftp> put /etc/motd
```

```
Uploading /etc/motd to /WWW/motd
/etc/motd     100%  118     0.1KB/s   00:00     user can write to WWW directory
sftp> ls -l
-rw-r--r--    1 101  10    118 Jul 20 09:07 motd     successful transfer
sftp>
```

Using Secure Shell

This section provides procedures to familiarize users with Secure Shell.

Using Secure Shell Task Map

The following task map points to user procedures for using Secure Shell.

Task	Description	For Instructions
Create a public/private key pair.	Enables access to Secure Shell for sites that require public-key authentication.	"How to Generate a Public/Private Key Pair for Use With Secure Shell" on page 35
Change your passphrase.	Changes the phrase that authenticates your private key.	"How to Change the Passphrase for a Secure Shell Private Key" on page 37
Log in with Secure Shell.	Provides encrypted Secure Shell communication when logging in remotely.	"How to Log In to a Remote Host With Secure Shell" on page 37
Log in to Secure Shell without being prompted for a password.	Enables login by using an agent which provides your password to Secure Shell.	"How to Reduce Password Prompts in Secure Shell" on page 39
Log in to Secure Shell as `root`.	Enables login as `root` for ZFS `send` and `receive` commands.	"How to Remotely Administer ZFS With Secure Shell" on page 40
Use port forwarding in Secure Shell.	Specifies a local port or a remote port to be used in a Secure Shell connection over TCP.	"How to Use Port Forwarding in Secure Shell" on page 42
Copy files with Secure Shell.	Securely copies files between hosts.	"How to Copy Files With Secure Shell" on page 43
Securely connect from a host inside a firewall to a host outside the firewall.	Uses Secure Shell commands that are compatible with HTTP or SOCKS5 to connect hosts that are separated by a firewall.	"How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall" on page 44

▼ How to Generate a Public/Private Key Pair for Use With Secure Shell

Users must generate a public/private key pair when their site implements host-based authentication or user public-key authentication. For additional options, see the `ssh-keygen`(1) man page.

Before You Begin Ask your system administrator whether host-based authentication is configured.

1. **Start the key generation program.**

   ```
   mySystem$ ssh-keygen -t rsa
   Generating public/private rsa key pair.
   ...
   ```

   where `-t` is the type of algorithm, either `rsa`, `dsa`, or `rsa1`.

2. **Specify the path to the file that will hold the key.**

   By default, the file name `id_rsa`, which represents an RSA v2 key, appears in parentheses. You can select this file by pressing the Return key or provide an alternative file name.

   ```
   Enter file in which to save the key (/home/username/.ssh/id_rsa):     <Press Return>
   ```

   The file name of the public key is created automatically by appending the string `.pub` to the name of the private key file.

3. **Type a passphrase for using your key.**

   This passphrase is used for encrypting your private key. A null entry is *strongly discouraged*. Note that the passphrase is not displayed when you type it in.

   ```
   Enter passphrase (empty for no passphrase): passphrase
   ```

4. **Retype the passphrase to confirm it.**

   ```
   Enter same passphrase again: passphrase
   Your identification has been saved in /home/username/.ssh/id_rsa
   Your public key has been saved in /home/username/.ssh/id_rsa.pub
   The key fingerprint is:
   0e:fb:3d:57:71:73:bf:58:b8:eb:f3:a3:aa:df:e0:d1 username@mySystem
   ```

5. **Check that the path to the key file is correct.**

   ```
   $ ls ~/.ssh
   ```

```
id_rsa
id_rsa.pub
```

At this point, you have created a public/private key pair.

6. **Log in to the remote host.**

   Choose one of the following login steps based on your site's authentication method.

   - **For host-based authentication, copy the local host's public key to the remote host.**

     a. **Type the following command on one line with no backslash.**

     ```
     $ cat /etc/ssh/ssh_host_rsa_key.pub | ssh RemoteSys \
     'cat >> ~/.ssh/known_hosts && echo "Host key copied"'
     ```

     b. **When you are prompted, supply your login password.**

     ```
     Enter password: password
     Host key copied
     $
     ```

     You can now log in to the remote host. For details, see "How to Log In to a Remote Host With Secure Shell" on page 37.

   - **For user authentication with public keys, populate your `authorized_keys` file on the remote host.**

     a. **Copy your public key to the remote host.**

     Type the following command on one line with no backslash.

     ```
     mySystem$ cat ~/.ssh/id_rsa.pub | ssh myRemoteSys \
     'cat >> ~/.ssh/authorized_keys && echo "Key copied"'
     ```

     b. **When you are prompted, supply your login password.**

     ```
     Enter password: password
     Key copied
     mySystem$
     ```

7. **(Optional) Avoid future prompting for passphrases.**

   See "How to Reduce Password Prompts in Secure Shell" on page 39. For more information, see the `ssh-agent`(1) and `ssh-add`(1) man pages.

▼ How to Change the Passphrase for a Secure Shell Private Key

The following command changes the authentication mechanism for the private key, the passphrase, and not the actual private key. For more information, see the `ssh-keygen`(1) man page.

- **Change your passphrase.**

  Type the `ssh-keygen` command with the `-p` option, and answer the prompts.

  ```
  mySystem$ ssh-keygen -p
  Enter file which contains the private key (/home/username/.ssh/id_rsa):     <Press Return>
  Enter passphrase (empty for no passphrase): passphrase
  Enter same passphrase again: passphrase
  ```

  where `-p` requests changing the passphrase of a private key file.

▼ How to Log In to a Remote Host With Secure Shell

1. **Start a Secure Shell session.**

   Type the `ssh` command, and specify the name of the remote host and your login.

   ```
   mySystem$ ssh myRemoteSys -l username
   ```

2. **If prompted, verify the authenticity of the remote host key.**

   A prompt might appear that question the authenticity of the remote host:

   ```
   The authenticity of host 'myRemoteHost' can't be established....Are you sure you want to
    continue connecting(yes/no)?
   ```

   This prompt is normal for initial connections to remote hosts.

   - **If you cannot confirm the authenticity of the remote host, type `no` and contact your system administrator.**

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

     The administrator is responsible for updating the global `/etc/ssh/ssh_known_hosts` file. An updated `ssh_known_hosts` file prevents this prompt from appearing.

- **If you confirm the authenticity of the remote host, answer the prompt and continue to the next step.**

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

3. **Authenticate yourself to Secure Shell.**

   a. **When prompted, type your passphrase.**

   ```
   Enter passphrase for key '/home/username/.ssh/id_rsa': passphrase
   ```

   b. **When prompted, type your account password.**

   ```
   username@myRemoteSys's password: password
   Last login: Wed Sep  7 09:07:49 2016 from mySystem
   Oracle Corporation      SunOS 5.11  11.3     September 2016
   myRemoteSys$
   ```

4. **Conduct transactions on the remote host.**

   The commands that you send are encrypted. Any responses that you receive are encrypted.

5. **Close the Secure Shell connection.**

   When you are finished, type **`exit`** or use your usual method for exiting your shell.

```
myRemoteSys$ exit
myRemoteSys$ logout
Connection to myRemoteSys closed
mySystem$
```

Example 3 Displaying a Remote GUI in Secure Shell

In this example, `jdoe` is the initial user on both systems and is assigned the Software Installation rights profile. The default value of the `X11Forwarding` keyword is still `yes`, and the `xauth` package is installed on the remote system.

```
$ ssh -l jdoe -X myRemoteSys
jdoe@myRemoteSys's password: password
Last login: Wed Sep  7 09:07:49 2016 from myLocalHost
Oracle Corporation      SunOS 5.11 11.3       September 2016
myRemoteSys$ useful-app-with-GUI &
```

▼ How to Reduce Password Prompts in Secure Shell

If you do not want to type your passphrase and your password to use Secure Shell, you can use the agent daemon. If you have different accounts on different hosts, add the keys that you need for the session.

You can start the agent daemon manually when needed, as described in the following procedure.

1. **Start the agent daemon.**

```
mySystem$ eval `ssh-agent`
Agent pid 9892
```

2. **Verify that the agent daemon has been started.**

```
mySystem$ pgrep ssh-agent
9892
```

3. **Add your private key to the agent daemon.**

```
mySystem$ ssh-add
Enter passphrase for /home/username/.ssh/id_rsa: passphrase
Identity added: /home/username/.ssh/id_rsa(/home/username/.ssh/id_rsa)
mySystem$
```

4. **Start a Secure Shell session.**

```
mySystem$ ssh myRemoteSys -l username
```

You are not prompted for a passphrase.

Example 4 Using `ssh-add` Options

In this example, `jdoe` adds two keys to the agent daemon. At the end of the session, `jdoe` removes all the keys from the agent daemon.

```
mySystem$ ssh-agent
mySystem$ ssh-add
Enter passphrase for /home/jdoe/.ssh/id_rsa: passphrase
Identity added: /home/jdoe/.ssh/id_rsa(/home/jdoe/.ssh/id_rsa)
mySystem$ ssh-add /home/jdoe/.ssh/id_dsa
Enter passphrase for /home/jdoe/.ssh/id_dsa: passphrase
Identity added:
/home/jdoe/.ssh/id_dsa(/home/jdoe/.ssh/id_dsa)
```

```
mySystem$ ssh-add -l
SHA256:OX5V4xxoVozwqdZfAbykwawMuwVM+sfc+ThMeai8r9
/home/jdoe/.ssh/id_rsa(RSA)
SHA256:OX5V4xxoVozwqdZfAbykwawMuwVM+sfc+ThMeai8r9
/home/jdoe/.ssh/id_dsa(DSA)
```

User conducts Secure Shell transactions

```
mySystem$ ssh-add -D
Identity removed:
/home/jdoe/.ssh/id_rsa(/home/jdoe/.ssh/id_rsa.pub)
/home/jdoe/.ssh/id_dsa(DSA)
```

▼ How to Remotely Administer ZFS With Secure Shell

By default, the `root` role cannot log in remotely with Secure Shell. Historically, `root` has used Secure Shell for important tasks, such as sending ZFS pool data to storage on a remote system. In this procedure, the `root` role creates a user who can act as a remote ZFS administrator.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create the user on both systems.**

   For example, create the `zfsroot` user and provide a password.

   ```
   source # useradd -c "Remote ZFS Administrator" -u 1201 -d /home/zfsroot zfsroot
   source # passwd zfsroot
   New Password: password
   Re-enter new password: password
   passwd: password successfully changed for zfsroot
   #

   dest # useradd -c "Remote ZFS Administrator" -u 1201 -d /home/zfsroot zfsroot
   dest # passwd zfsroot
   ...
   ```

   The `zfsroot` user must be identically defined on both systems.

2. **On both systems, assign the ZFS File Management rights profile to `zfsroot`.**

   ```
   source # usermod -P +'ZFS File System Management' -S files zfsroot
   dest # usermod -P +'ZFS File System Management' -S files zfsroot
   ```

3. **Verify that `zfsroot` on the destination system is assigned the rights profile.**

   ```
   dest # profiles zfsroot
   zfsroot:
   ZFS File System Management
   Basic Solaris User
   All
   ```

4. **Create the user's key pair for Secure Shell authentication.**

   The key pair is created on the source system. Then, the public key is copied to the `zfsroot` user on the destination system.

   a. **Generate the key pair and put it in the file `id_migrate`.**

   ```
   # ssh-keygen -t rsa -P "" -f ~/id_migrate
   Generating public/private rsa key pair.
   Your identification has been saved in /root/id_migrate.
   Your public key has been saved in /root/id_migrate.pub.
   The key fingerprint is:
   SHA256:BLNj0v9...izsQ cpltester@Local
   The key's randomart image is:
   +---[RSA 2048]----+
   |      o       .=B|
   ...
   ```

   b. **Send the public part of the key pair to the destination system.**

   ```
   # scp ~/id_migrate.pub zfsroot@dest:
   The authenticity of host 'dest (192.0.2.126)' can't be established.
   RSA key fingerprint is 44:37:ab:4e:b7:2f:2f:b8:5f:98:9d:e9:ed:6d:46:80.
   Are you sure you want to continue connecting (yes/no)? yes
   Warning: Permanently added 'dest,192.0.2.126' (RSA) to the list of known hosts.
   Password:
   id_migrate.pub 100% |*****************************| 399 00:00
   ```

5. **On the destination system, move the public part of the key pair to the private `/home/zfsroot/.ssh` directory.**

   ```
   root@dest # su - zfsroot
   Oracle Corporation      SunOS 5.11      11.1    May 2012
   zfsroot@dest $ mkdir -m 700 .ssh
   zfsroot@dest $ cat id_migrate.pub >> .ssh/authorized_keys
   ```

6. **Verify that the configuration works.**

   ```
   root@source# ssh -l zfsroot -i ~/id_migrate dest \
   pfexec /usr/sbin/zfs snapshot zones@test
   ```

```
root@source# ssh -l zfsroot -i ~/id_migrate dest \
pfexec /usr/sbin/zfs destroy zones@test
```

7. **(Optional) Verify that you can create a snapshot and replicate the data.**

```
root@source# zfs snapshot -r rpool/zones@migrate-all
root@source#  zfs send -rc rpool/zones@migrate-all | \
ssh -l zfsroot -i ~/id_migrate dest pfexec /usr/sbin/zfs recv -F zones
```

8. **(Optional) Remove the ability to use the `zfsroot` account for ZFS administration.**

```
root@dest# usermod -P -'ZFS File System Management' zfsroot
root@dest#  su - zfsroot
zfsroot@dest#  cp .ssh/authorized_keys .ssh/authorized_keys.bak
zfsroot@dest#  grep -v root@source .ssh/authorized_keys.bak> .ssh/authorized_keys
```

▼ How to Use Port Forwarding in Secure Shell

You can specify that a local port be forwarded to a remote host. Effectively, a socket is allocated to listen to the port on the local side. The connection from this port is made over a secure channel to the remote host. For example, you might specify port `143` to obtain email remotely with IMAP4. Similarly, a port can be specified on the remote side.

Before You Begin To use port forwarding, the administrator must have enabled port forwarding on the remote Secure Shell server. For details, see "How to Configure Port Forwarding in Secure Shell" on page 30.

- **Set secure port forwarding either from a remote port to a local port or from a local port to a remote port.**

  - **To set a local port to receive secure communication from a remote port, specify both ports.**

    Specify the local port that listens for remote communication. Also, specify the remote host and the remote port that forward the communication.

    ```
    mySystem$ ssh -L localPort:remoteHost:remotePort
    ```

  - **To set a remote port to receive a secure connection from a local port, specify both ports.**

    Specify the remote port that listens for remote communication. Also, specify the local host and the local port that forward the communication.

    ```
    mySystem$ ssh -R remotePort:localhost:localPort
    ```

Example 5 Using Local Port Forwarding to Receive Mail

The following example demonstrates how you can use local port forwarding to receive mail securely from a remote server.

```
mySystem$ ssh -L 9143:myRemoteSys:143 myRemoteSys
```

This command forwards connections from port `9143` on `myLocalHost` to port `143`. Port `143` is the IMAP v2 server port on `myRemoteSys`. When the user launches a mail application, the user specifies the local port number for the IMAP server, as in `localhost:9143`.

Example 6 Using Remote Port Forwarding to Communicate Outside of a Firewall

This example demonstrates how a user in an enterprise environment can forward connections from a host on an external network to a host inside a corporate firewall.

```
mySystem$ ssh -R 9022:mySystem:22myOutsideHost
```

This command forwards connections from port `9022` on `myOutsideHost` to the port where the `sshd` daemon listens on the local host. Typically, the listening port is port 22.

```
myOutsideHost$ ssh -p 9022 localhost
mySystem$
```

▼ How to Copy Files With Secure Shell

The following procedure shows how to use the `scp` command to copy encrypted files between hosts. You can copy encrypted files either between a local host and a remote host, or between two remote hosts. The `scp` command prompts for authentication. For more information, see "Remote Copying With the scp Command" in *Managing Remote Systems in Oracle Solaris 11.3* and the `scp`(1) man page.

You can also use the `sftp` secure file transfer program. For more information, see the `sftp`(1) man page. For an example, see Example 7, "Specifying a Port When Using the `sftp` Command," on page 44 and "Logging In to a Remote System to Copy a File (sftp)" in *Managing Remote Systems in Oracle Solaris 11.3*.

Note - The audit service can audit SunSSH `sftp` transactions by using the `ft` audit class. For `scp` transactions, the audit service can audit access and exit for the `ssh` session. For more information, see "How to Audit FTP and SFTP File Transfers" in *Managing Auditing in Oracle Solaris 11.3*.

1. **Start the secure copy program.**

   Specify the source file, the user name at the remote destination, and the destination directory.

   ```
   mySystem$ scp myfile.1 username@myRemoteSys:~
   ```

2. **Supply your passphrase when prompted.**

   ```
   Enter passphrase for key '/home/username/.ssh/id_rsa': passphrase
   myfile.1       25% |*******                      |    640 KB  0:20 ETA
   myfile.1
   ```

   After you type the passphrase, a progress meter is displayed, as shown in the second line in the output. The progress meter displays:

   - The file name
   - The percentage of the file that has been transferred
   - A series of asterisks that indicate the percentage of the file that has been transferred
   - The quantity of data transferred
   - The estimated time of arrival, or ETA, of the complete file (that is, the remaining amount of time)

Example 7 Specifying a Port When Using the `sftp` Command

In this example, the user wants the `sftp` command to use a specific port. The user uses the `-o` option to specify the port.

```
$ sftp -o port=2222 guest@RemoteFileServer
```

▼ How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall

You can use Secure Shell to make a connection from a host inside a firewall to a host outside the firewall. This task is done by specifying a proxy command for `ssh` either in a configuration file or as an option on the command line. For the command-line option, see Example 8, "Connecting to Hosts Outside a Firewall From the Secure Shell Command Line," on page 46.

You can customize your `ssh` interactions through your own personal configuration file, `~/.ssh/config`, or you can use the settings in the administrative configuration file, `/etc/ssh/ssh_config`.

The files can be customized with two types of proxy commands. One proxy command is for HTTP connections. The other proxy command is for SOCKS5 connections. For more information, see the `ssh_config`(4) man page.

1. **Specify the proxy commands and hosts in a configuration file.**

   Use the following syntax to add as many lines as you need:

   ```
   [Host outside-host]
   ProxyCommand proxy-command [-h proxy-server] \
   [-p proxy-port] outside-host|%h outside-port|%p
   ```

   `Host` *outside-host*

   Limits the proxy command specification to occasions when a remote host name is specified on the command line. If you use a wildcard for *outside-host*, you apply the proxy command specification to a set of hosts.

   proxy-command

   Specifies the proxy command.

   The command can be either of the following:

   - `/usr/lib/ssh/ssh-http-proxy-connect` for HTTP connections
   - `/usr/lib/ssh/ssh-socks5-proxy-connect` for SOCKS5 connections

   `-h` *proxy-server* and `-p` *proxy-port*

   These options specify a proxy server and a proxy port, respectively. If present, the proxies override any environment variables that specify proxy servers and proxy ports, such as `HTTPPROXY`, `HTTPPROXYPORT`, `SOCKS5_PORT`, `SOCKS5_SERVER`, and `http_proxy`. The `http_proxy` variable specifies a URL. If the options are not used, then the relevant environment variables must be set. For more information, see the `ssh-socks5-proxy-connect`(1) and `ssh-http-proxy-connect`(1) man pages.

   outside-host

   Designates a specific host to connect to. Use the `%h` substitution argument to specify the host on the command line.

   outside-port

   Designates a specific port to connect to. Use the `%p` substitution argument to specify the port on the command line. By specifying `%h` and `%p` without using the `Host` *outside-host* option, the proxy command is applied to the host argument whenever the `ssh` command is invoked.

2. **Run Secure Shell, specifying the outside host.**

For example:

```
mySystem$ ssh myOutsideHost
```

This command looks for a proxy command specification for `myOutsideHost` in your personal configuration file. If the specification is not found, then the command looks in the system-wide configuration file, `/etc/ssh/ssh_config`. The proxy command is substituted for the `ssh` command.

Example 8 Connecting to Hosts Outside a Firewall From the Secure Shell Command Line

"How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall" on page 44 explains how to specify a proxy command in a configuration file. In this example, a proxy command is specified on the `ssh` command line.

```
$ ssh -o'Proxycommand=/usr/lib/ssh/ssh-http-proxy-connect \
-h myProxyServer -p 8080 myOutsideHost 22' myOutsideHost
```

The `-o` option to the `ssh` command provides a command-line method of specifying a proxy command. This example command does the following:

- Substitutes the HTTP proxy command for `ssh`
- Uses port `8080` and `myProxyServer` as the proxy server
- Connects to port `22` on `myOutsideHost`

♦ ♦ ♦ CHAPTER 2

Secure Shell Reference

This chapter describes the configuration options in the Secure Shell feature of Oracle Solaris, and covers the following topics:

- "Typical Secure Shell Sessions" on page 47
- "Secure Shell Configuration" on page 49
- "Secure Shell Files" on page 52
- "Secure Shell Commands" on page 53

For procedures to configure Secure Shell, see Chapter 1, "Using Secure Shell".

Typical Secure Shell Sessions

The Secure Shell daemon (`sshd`) is normally started at boot time when network services are started. The daemon listens for connections from clients. A Secure Shell session begins when the user runs an `ssh`, `scp`, or `sftp` command. A new `sshd` daemon is forked for each incoming connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange with the client. These session characteristics are determined by client-side configuration files and server-side configuration files. Command-line arguments can override the settings in the configuration files.

The client and server must authenticate themselves to each other. After successful authentication, the user can execute commands remotely and copy data between systems.

Session Characteristics in Secure Shell

The Secure Shell server-side behavior of the `sshd` daemon is controlled by keyword settings in the `/etc/ssh/sshd_config` file. For example, the `sshd_config` file controls which types of authentication are permitted for accessing the server. The server-side behavior can also be controlled by the command-line options when the `sshd` daemon is started.

The behavior on the client side is controlled by Secure Shell keywords in this order of precedence:

- Command-line options
- User's configuration file, `~/.ssh/config`
- System-wide configuration file, `/etc/ssh/ssh_config`

For example, a user can override a system-wide configuration `Ciphers` setting that prefers `aes128-ctr` by specifying `-c aes256-ctr,aes128-ctr` on the command line. The first cipher, `aes256-ctr`, is now preferred.

```
$ ssh -c aes256-ctr,aes128-ctr,arcfour
```

The first cipher, `aes256-ctr`, is now preferred.

Authentication and Key Exchange in Secure Shell

The Secure Shell protocol supports client user/host authentication and server host authentication. Cryptographic keys are exchanged for the protection of Secure Shell sessions. Secure Shell provides various methods for authentication and key exchange. Some methods are optional. Client authentication mechanisms are listed in Table 1, “Authentication Methods for Secure Shell,” on page 11. Servers are authenticated by using known host public keys.

For authentication, Secure Shell supports user authentication and generic interactive authentication, which usually involves passwords. Secure Shell also supports authentication with user public keys and with trusted-host public keys. The keys can be RSA or DSA. Additionally, Secure Shell can use GSS credentials for authentication.

Acquiring GSS Credentials in Secure Shell

To use GSS-API authentication in Secure Shell, the server must have GSS-API acceptor credentials and the client must have GSS-API initiator credentials. Support is available for `mech_krb5`.

For `mech_krb5`, the server has GSS-API acceptor credentials when the host principal that corresponds to the server has a valid entry in `/etc/krb5/krb5.keytab`.

The client has initiator credentials for `mech_krb5` if one of the following has been done:

- The `kinit` command has been run.
- The `pam_krb5` module is used in the `pam.conf` file.

For information about GSS-API and Kerberos, see "How to Set Up a Secure NFS Environment With Multiple Kerberos Security Modes" in *Managing Kerberos and Other Authentication Services in Oracle Solaris 11.3*. For more information about mechanisms, see the `mech`(4) and `mech_spnego`(5) man pages.

Command Execution and Data Forwarding in Secure Shell

After authentication is complete, the user can use Secure Shell, generally by requesting a shell or executing a command. Through the `ssh` command options, the user can make requests. Requests can include allocating a pseudo-TTY, forwarding X11 connections or TCP/IP connections, or enabling an `ssh-agent` authentication program over a secure connection.

The basic components of a user session are as follows:

1. The user requests a shell or the execution of a command, which begins the session mode.

   In this mode, data is sent or received through the terminal on the client side. On the server side, data is sent through the shell or a command.
2. When data transfer is complete, the user program terminates.
3. All X11 forwarding and TCP/IP forwarding is stopped, except for those connections that already exist. Existing X11 connections and TCP/IP connections remain open.
4. The server sends an exit status message to the client. When all connections are closed, such as forwarded ports that had remained open, the client closes the connection to the server. Then, the client exits.

Secure Shell Configuration

The characteristics of a Secure Shell session are controlled by configuration files. Some keyword values in the configuration files can be overridden by options on the command line.

Client and Server Configuration in Secure Shell

Client configuration	In most cases, the client-side characteristics of a Secure Shell session are governed by the system-wide configuration file, `/etc/ssh/ssh_config`. The settings in the `ssh_config` file can be overridden by the user's

	configuration file, `~/.ssh/config`. In addition, the user can override both configuration files on the command line. The settings in the server's `/etc/ssh/sshd_config` file determine which client requests are permitted by the server. For a list of server configuration settings, see the `sshd_config`(4) man page. The keywords in the client configuration file are described in detail in the `ssh`(1), `scp`(1), `sftp`(1), and `ssh_config`(4) man pages.
Host-specific client configuration	Sometimes, having different Secure Shell characteristics for different local hosts is useful. The administrator can define separate sets of parameters in the `/etc/ssh/ssh_config` file to be applied according to host or regular expression by grouping entries in the file by `Host` keyword. If the `Host` keyword is not used, the entries in the client configuration file apply to the local system.
Server configuration	The server-side characteristics of a Secure Shell session are governed by the `/etc/ssh/sshd_config` file. For a full description of the keywords, see the `sshd_config`(4) man page.

Protecting the Secure Shell Configuration

Each host that needs to communicate securely with another host must have the server's public key stored in the local host's `/etc/ssh/ssh_known_hosts` file. Although a script could be used to update the `/etc/ssh/ssh_known_hosts` files, such a practice is heavily discouraged because a script opens a major security vulnerability.

The `/etc/ssh/ssh_known_hosts` file should be distributed only by a secure mechanism as follows:

- Over a secure connection, such as Secure Shell, IPsec, or Kerberized `ftp` from a known and trusted system
- At system install time

To avoid the possibility of an intruder gaining access by inserting bogus public keys into a `known_hosts` file, you should use a known and trusted source of the `ssh_known_hosts` file. The `ssh_known_hosts` file can be distributed during installation. Later, scripts that use the `scp` command can be used to copy the latest version.

Login Environment Variables and Secure Shell Implementations

SunSSH supports login environment variables from the `/etc/default/login` entries for unset keyword values and uses the environment variables that users set in their login shell. OpenSSH can use only the `PATH` variable from a user's login shell.

In SunSSH, when the following keywords are not set in the `sshd_config` file, they obtain their value from equivalent entries in the `/etc/default/login` file. In the `sshd_config` file, you can modify values for the keywords pertaining to logins in the following table.

Note - OpenSSH does not use the values from this file.

Entry in `/etc/default/login`	Keyword and Value in `sshd_config`
`CONSOLE=*`	`PermitRootLogin=without-password`
`#CONSOLE=*`	`PermitRootLogin=yes`
`PASSREQ=YES`	`PermitEmptyPasswords=no`
`PASSREQ=NO`	`PermitEmptyPasswords=yes`
`#PASSREQ`	`PermitEmptyPasswords=no`
`TIMEOUT=`*seconds*	`LoginGraceTime=`*seconds*
`#TIMEOUT`	`LoginGraceTime=120`
`RETRIES` and `SYSLOG_FAILED_LOGINS`	Apply only to `password` and `keyboard-interactive` authentication methods

To see a full list of optional values for these keywords, see the `sshd_config(4)` man page.

When the following variables are set by the initialization scripts from the user's login shell, the `sshd` daemon uses those values. When the variables are not set, the daemon uses the default value.

`TIMEZONE`	Controls the setting of the `TZ` environment variable. When not set, the `sshd` daemon uses value of `TZ` when the daemon was started.
`ALTSHELL`	Controls the setting of the `SHELL` environment variable. The default is `ALTSHELL=YES`, where the `sshd` daemon uses the value of the user's shell. When `ALTSHELL=NO`, the `SHELL` value is not set.
`PATH`	Controls the setting of the `PATH` environment variable. When the value is not set, the default path is `/usr/bin`.

Note - The OpenSSH `sshd` daemon uses this variable. It does not support other initialization variables from a user's login shell.

`SUPATH`	Controls the setting of the `PATH` environment variable for `root`. When the value is not set, the default path is `/usr/sbin:/usr/bin`.

For more information, see the `login`(1) and `sshd`(1M) man pages.

Secure Shell Files

The following table shows the main Secure Shell files and the suggested file permissions.

TABLE 3 Secure Shell Files

Secure Shell File Name	Description	Suggested Permissions and Owner
`~/.rhosts`	Contains the host-user name pairs that specify the hosts to which the user can log in without a password. This file is also used by the `rlogind` and `rshd` daemons.	`-rw-r--r--` *username*
`~/.shosts`	Contains the host-user name pairs that specify the host systems to which the user can log in without a password. This file is not used by other utilities. For more information, see the `sshd`(1M) man page in the `FILES` section.	`-rw-r--r--` *username*
`~/.ssh/authorized_keys`	Holds the public keys of the user who is allowed to log in to the user account.	`-rw-r--r--` *username*
`~/.ssh/config`	Configures user settings which override system settings.	`-rw-r--r--` *username*
`~/.ssh/environment`	Contains initial assignments at login. By default, this file is not read. The `PermitUserEnvironment` keyword in the `sshd_config` file must be set to `yes` for this file to be read.	`-rw-r--r--` *username*
`/etc/hosts.equiv`	Contains the hosts that are used in `.rhosts` authentication. This file is also used by the `rlogind` and `rshd`daemons.	`-rw-r--r-- root`
`~/.ssh/known_hosts`	Contains the host public keys for all hosts with which the client can communicate securely. The file is maintained automatically. Whenever the user connects with an unknown host, the remote host key is added to the file.	`-rw-r--r--` *username*
`/etc/default/login`	Provides defaults for the `sshd` daemon when corresponding `sshd_config` parameters are not set.	`-r--r--r-- root`
`/etc/nologin`	If this file exists, the `sshd` daemon permits only `root` to log in. The contents of this file are displayed to users who are attempting to log in.	`-rw-r--r-- root`

Secure Shell File Name	Description	Suggested Permissions and Owner
`~/.ssh/rc`	Contains initialization routines that are run before the user shell starts. For a sample initialization routine, see the `sshd`(1M) man page.	`-rw-r--r--` *username*
`/etc/ssh/shosts.equiv`	Contains the hosts that are used in host-based authentication. This file is not used by other utilities.	`-rw-r--r-- root`
`/etc/ssh/ssh_config`	Configures system settings on the client system.	`-rw-r--r-- root`
`/etc/ssh/ssh_host_dsa_key` or `/etc/ssh/ssh_host_rsa_key`	Contains the host private key.	`-rw------- root`
`/etc/ssh_host_key.pub` or `/etc/ssh/ssh_host_dsa_key.pub` or `/etc/ssh/ssh_host_rsa_key.pub`	Contains the host public key, for example, `/etc/ssh/ssh_host_rsa_key.pub`. Used to copy the host key to the local `known_hosts` file.	`-rw-r--r-- root`
`/etc/ssh/ssh_known_hosts`	Contains the host public keys for all hosts with which the client can communicate securely. The file is populated by the administrator.	`-rw-r--r-- root`
`/etc/ssh/sshd_config`	Contains configuration data for `sshd`, the Secure Shell daemon.	`-rw-r--r-- root`
`/system/volatile/sshd.pid`	Contains the process ID of the Secure Shell daemon, `sshd`. If multiple daemons are running, the file contains the last daemon that was started.	`-rw-r--r-- root`
`/etc/ssh/sshrc`	Contains host-specific initialization routines that are specified by an administrator.	`-rw-r--r-- root`

Note - The `sshd_config` file can be overridden by a file from a site-customized package. For more information, see the definition of the `overlay` file attribute in the `pkg(5)` man page.

Secure Shell Commands

The following table summarizes the main Secure Shell commands.

Note - Because the man pages differ between the SunSSH and OpenSSH implementations, you should open a terminal and use the `man` command to view the Secure Shell man pages that document the Secure Shell implementation on your system.

TABLE 4 Commands in Secure Shell

Man Page for Secure Shell Command	Description
`ssh`(1)	Logs a user in to a remote system and securely executes commands on a remote system. The `ssh` command enables secure encrypted communications between two untrusted host systems over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.
`sshd`(1M)	The daemon for Secure Shell. The daemon listens for connections from clients and enables secure encrypted communications between two untrusted hosts over an insecure network.
`ssh-add`(1)	Adds RSA or DSA identities to the authentication agent, `ssh-agent`. Identities are also called *keys*.
`ssh-agent`(1)	Holds private keys that are used for public key authentication. The `ssh-agent` program is started at the beginning of an X-session or a login session. All other windows and other programs are started as clients of the `ssh-agent` program. Through the use of environment variables, the agent can be located and used for authentication when users use the `ssh` command to log in to other systems.
`ssh-keygen`(1)	Generates and manages authentication keys for Secure Shell.
`ssh-keyscan`(1)	Gathers the public keys of a number of Secure Shell hosts. Aids in building and verifying `ssh_known_hosts` files.
`ssh-keysign`(1M)	Used by the `ssh` command to access the host keys on the local host. Generates the digital signature that is required during host-based authentication with Secure Shell v2. The command is invoked by the `ssh` command, not by the user.
`scp`(1)	Securely copies files between hosts on a network over an encrypted `ssh` transport. Unlike the `rcp` command, the `scp` command prompts for passwords or passphrases if password information is needed for authentication.
`sftp`(1)	An interactive file transfer program that is similar to the `ftp` command. Unlike the `ftp` command, the `sftp` command performs all operations over an encrypted `ssh` transport. The command connects, logs in to the specified host name and then enters interactive command mode.

Index