



Implementing Secure Shell

Secure Shell (SSH) is an application and a protocol that provides a secure replacement to the Berkeley r-tools. The protocol secures sessions using standard cryptographic mechanisms, and the application can be used similarly to the Berkeley **rexec** and **rsh** tools.

Two versions of the SSH server are available: SSH Version 1 (SSHv1) and SSH Version 2 (SSHv2). SSHv1 uses Rivest, Shamir, and Adelman (RSA) keys and SSHv2 uses either Digital Signature Algorithm (DSA) keys or Rivest, Shamir, and Adelman (RSA) keys, or Elliptic Curve Digital Signature Algorithm (ECDSA) keys. Cisco software supports both SSHv1 and SSHv2.

This module describes how to implement Secure Shell.

Feature History for Implementing Secure Shell

Release	Modification
Release 6.0	This feature was introduced.
Release 7.0.1	Support was added for these features: <ul style="list-style-type: none">• SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm• Automatic Generation of SSH Host-Key Pairs• SSH and SFTP in Baseline Cisco IOS XR Software Image

- [Prerequisites for Implementing Secure Shell, on page 2](#)
- [SSH and SFTP in Baseline Cisco IOS XR Software Image, on page 2](#)
- [Restrictions for Implementing Secure Shell, on page 2](#)
- [Configure SSH, on page 3](#)
- [Automatic Generation of SSH Host-Key Pairs, on page 7](#)
- [Configure SSH Client, on page 9](#)
- [Order of SSH Client Authentication Methods, on page 11](#)
- [SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm, on page 12](#)
- [SSH Port Forwarding, on page 16](#)
- [Non-Default SSH Port, on page 19](#)
- [Information About Implementing Secure Shell, on page 23](#)

Prerequisites for Implementing Secure Shell

The following prerequisites are required to implement Secure Shell:

- Download the required image on your router. The SSH server and SSH client require you to have a crypto package (data encryption standard [DES], 3DES and AES) from Cisco downloaded on your router.



Note From Cisco IOS XR Software Release 7.0.1 and later, the SSH and SFTP components are available in the baseline Cisco IOS XR software image itself. For details, see, [SSH and SFTP in Baseline Cisco IOS XR Software Image, on page 2](#).

- Configure user authentication for local or remote access. You can configure authentication with or without authentication, authorization, and accounting (AAA).
- AAA authentication and authorization must be configured correctly for Secure Shell File Transfer Protocol (SFTP) to work.

SSH and SFTP in Baseline Cisco IOS XR Software Image

From Cisco IOS XR Software Release 7.0.1 and later, the management plane and control plane components that were part of the Cisco IOS XR security package (k9sec package) are moved to the base Cisco IOS XR software image. These include SSH, SCP, SFTP and IPsec control plane. However, *802.1X protocol (Port-Based Network Access Control)* and data plane components like MACsec remain as a part of the security package as per the export compliance regulations. This segregation of package components makes the software more modular. It also gives you the flexibility of including or excluding the security package as per your requirements.

The base package and the security package allow FIPS, so that the control plane can negotiate FIPS-approved algorithms.

Restrictions for Implementing Secure Shell

The following are some basic SSH restrictions and limitations of the SFTP feature:

- In order for an outside client to connect to the router, the router needs to have an RSA (for SSHv1 or SSHv2) or DSA (for SSHv2) or ECDSA (for SSHv2) key pair configured. ECDSA, DSA and RSA keys are not required if you are initiating an SSH client connection from the router to an outside routing device. The same is true for SFTP: ECDSA, DSA and RSA keys are not required because SFTP operates only in client mode.
- In order for SFTP to work properly, the remote SSH server must enable the SFTP server functionality. For example, the SSHv2 server is configured to handle the SFTP subsystem with a line such as `/etc/ssh2/sshd2_config`:
- `subsystem-sftp /usr/local/sbin/sftp-server`

- The SFTP server is usually included as part of SSH packages from public domain and is turned on by default configuration.
- SFTP is compatible with sftp server version OpenSSH_2.9.9p2 or higher.
- RSA-based user authentication is supported in the SSH and SFTP servers. The support however, is not extended to the SSH client.
- Execution shell and SFTP are the only applications supported.
- The SFTP client does not support remote filenames containing wildcards (* ?, []). The user must issue the **sftp** command multiple times or list all of the source files from the remote host to download them on to the router. For uploading, the router SFTP client can support multiple files specified using a wildcard provided that the issues mentioned in the first through third bullets in this section are resolved.
- The cipher preference for the SSH server follows the order AES128, AES192, AES256, and, finally, 3DES. The server rejects any requests by the client for an unsupported cipher, and the SSH session does not proceed.
- Use of a terminal type other than vt100 is not supported, and the software generates a warning message in this case.
- Password messages of “none” are unsupported on the SSH client.
- Files created on the local device lose the original permission information because the router infrastructure does not provide support for UNIX-like file permissions. For files created on the remote file system, the file permission adheres to the umask on the destination host and the modification and last access times are the time of the copy.

Configure SSH

Perform this task to configure SSH.



Note For SSHv1 configuration, Step 1 to Step 4 are required. For SSHv2 configuration, Step to Step 4 are optional.



Note From Cisco IOS XR Software Release 7.0.1 and later, the SSH host-key pairs are auto-generated at the time of router boot up. Hence you need not perform steps 5 to 7 to generate the host keys explicitly. See, [Automatic Generation of SSH Host-Key Pairs, on page 7](#) for details.

SUMMARY STEPS

1. **configure**
2. **hostname** *hostname*
3. **domain name** *domain-name*
4. Use the **commit** or **end** command.
5. **crypto key generate rsa** [**usage keys** | **general-keys**] [*keypair-label*]
6. **crypto key generate dsa**

7. **crypto key generate ecdsa** [**nistp256** | **nistp384** | **nistp521**]
8. **configure**
9. **ssh timeout** *seconds*
10. Do one of the following:
 - **ssh server** [**vrf** *vrf-name*]
 - **ssh server v2**
11. Use the **commit** or **end** command.
12. **show ssh**
13. **show ssh session details**
14. **show ssh history**
15. **show ssh history details**
16. **show tech-support ssh**

DETAILED STEPS

Step 1 **configure**

Example:

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

Step 2 **hostname** *hostname*

Example:

```
RP/0/RP0/CPU0:router(config)# hostname router1
```

Configures a hostname for your router.

Step 3 **domain name** *domain-name*

Example:

```
RP/0/RP0/CPU0:router(config)# domain name cisco.com
```

Defines a default domain name that the software uses to complete unqualified host names.

Step 4 Use the **commit** or **end** command.

commit —Saves the configuration changes and remains within the configuration session.

end —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

Step 5 **crypto key generate rsa** [**usage keys** | **general-keys**] [*keypair-label*]

Example:

```
RP/0/RP0/CPU0:router# crypto key generate rsa general-keys
```

Generates an RSA key pair. The RSA key modulus can be in the range of 512 to 4096 bits.

- To delete the RSA key pair, use the **crypto key zeroize rsa** command.
- This command is used for SSHv1 only.

Step 6 `crypto key generate dsa`

Example:

```
RP/0/RP0/CPU0:router# crypto key generate dsa
```

Enables the SSH server for local and remote authentication on the router. The supported key sizes are: 512, 768 and 1024 bits.

- The recommended minimum modulus size is 1024 bits.
- Generates a DSA key pair.
To delete the DSA key pair, use the **crypto key zeroize dsa** command.
- This command is used only for SSHv2.

Step 7 `crypto key generate ecdsa [nistp256 | nistp384 | nistp521]`

Example:

```
RP/0/RP0/CPU0:router# crypto key generate ecdsa nistp256
```

Generates an ECDSA key pair. The supported ECDSA curve types are: Nistp256, Nistp384 and Nistp521.

- To delete the ECDSA key pair, use the **crypto key zeroize ecdsa [nistp256 | nistp384 | nistp521]** command.
- This command is used for SSHv2 only.

Step 8 `configure`

Example:

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

Step 9 `ssh timeout seconds`

Example:

```
RP/0/RP0/CPU0:router(config)# ssh timeout 60
```

(Optional) Configures the timeout value for user authentication to AAA.

- If the user fails to authenticate itself to AAA within the configured time, the connection is terminated.
- If no value is configured, the default value of 30 seconds is used. The range is from 5 to 120.

Step 10 Do one of the following:

- **ssh server [vrf vrf-name]**

- `ssh server v2`

Example:

```
RP/0/RP0/CPU0:router(config)# ssh server v2
```

- (Optional) Brings up an SSH server using a specified VRF of up to 32 characters. If no VRF is specified, the default VRF is used.

To stop the SSH server from receiving any further connections for the specified VRF, use the **no** form of this command. If no VRF is specified, the default is assumed.

Note The SSH server can be configured for multiple VRF usage.

- (Optional) Forces the SSH server to accept only SSHv2 clients if you configure the SSHv2 option by using the **ssh server v2** command. If you choose the **ssh server v2** command, only the SSH v2 client connections are accepted.

Step 11 Use the **commit** or **end** command.

commit —Saves the configuration changes and remains within the configuration session.

end —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

Step 12 `show ssh`

Example:

```
RP/0/RP0/CPU0:router# show ssh
```

(Optional) Displays all of the incoming and outgoing SSHv1 and SSHv2 connections to the router.

Step 13 `show ssh session details`

Example:

```
RP/0/RP0/CPU0:router# show ssh session details
```

(Optional) Displays a detailed report of the SSHv2 connections to and from the router.

Step 14 `show ssh history`

Example:

```
RP/0/RP0/CPU0:router# show ssh history
```

(Optional) Displays the last hundred SSH connections that were terminated.

Step 15 `show ssh history details`

Example:

```
RP/0/RP0/CPU0:router# show ssh history details
```

(Optional) Displays the last hundred SSH connections that were terminated with additional details. This command is similar to **show ssh session details** command but also mentions the start and end time of the session.

Step 16 **show tech-support ssh**

Example:

```
RP/0/RP0/CPU0:router# show tech-support ssh
```

(Optional) Automatically runs the `show` commands that display system information.



Note The order of priority while doing negotiation for a SSH connection is as follows:

1. `ecdsa-nistp-521`
2. `ecdsa-nistp-384`
3. `ecdsa-nistp-256`
4. `rsa`
5. `dsa`

Automatic Generation of SSH Host-Key Pairs

This feature brings in the functionality of automatically generating the SSH host-key pairs for the DSA, ECDSA (such as **ecdsa-nistp256**, **ecdsa-nistp384**, and **ecdsa-nistp521**) and RSA algorithms. This in turn eliminates the need for explicitly generating each SSH host-key pair after the router boots up. Because the keys are already present in the system, the SSH client can establish connection with the SSH server soon after the router boots up with the basic SSH configuration. This is useful especially during zero touch provisioning (ZTP) and Golden ISO boot up scenarios.

Before this automation, you had to execute the **crypto key generate** command to generate the required host-key pairs.

Although the host-key pairs are auto-generated with the introduction of this feature, you still have the flexibility to select only the required algorithms on the SSH server. You can use the **ssh server algorithms host-key** command in XR Config mode to achieve the same. Alternatively, you can also use the existing **crypto key zeroize** command in XR EXEC mode to remove the algorithms that are not required.

Prior to the introduction of this feature, you had to execute the **crypto key generate** command in XR EXEC mode to generate the required host-key pairs.



Note In a system upgrade scenario from version 1 to version 2, the system does not generate the SSH host-key pairs automatically if they were already generated in version 1. The host-key pairs are generated automatically only if they were not generated in version 1.

Configure the Allowed SSH Host-Key Pair Algorithms

When the SSH client attempts a connection with the SSH server, it sends a list of SSH host-key pair algorithms (in the order of preference) internally in the connection request. The SSH server, in turn, picks the first matching algorithm from this request list. The server establishes a connection only if that host-key pair is already generated in the system, and if it is configured (using the `ssh server algorithms host-key` command) as the allowed algorithm.



Note If this configuration of allowed host-key pairs is not present in the SSH server, then you can consider that the SSH server allows all host-key pairs. In that case, the SSH client can connect with any one of the host-key pairs. Not having this configuration also ensures backward compatibility in system upgrade scenarios.

Configuration Example

You may perform this (optional) task to specify the allowed SSH host-key pair algorithm (in this example, `ecdsa`) from the list of auto-generated host-key pairs on the SSH server:

```
/* Example to select the ecdsa algorithm */
Router(config)#ssh server algorithms host-key ecdsa-nistp521
```

Similarly, you may configure other algorithms.

Running Configuration

```
ssh server algorithms host-key ecdsa-nistp521
!
```

Verify the SSH Host-Key Pair Algorithms



Note With the introduction of the automatic generation of SSH host-key pairs, the output of the `show crypto key mypubkey` command displays key information of all the keys that are auto-generated. Before its introduction, the output of this show command displayed key information of only those keys that you explicitly generated using the `crypto key generate` command.

```
Router#show crypto key mypubkey ecdsa
Mon Nov 19 12:22:51.762 UTC
Key label: the_default
Type      : ECDSA General Curve Nistp256
Degree   : 256
Created  : 10:59:08 UTC Mon Nov 19 2018
Data     :
04AC7533 3ABE7874 43F024C1 9C24CC66 490E83BE 76CEF4E2 51BBEF11 170CDB26
14289D03 6625FC4F 3E7F8F45 0DA730C3 31E960FE CF511A05 2B0AA63E 9C022482
6E

Key label: the_default
Type      : ECDSA General Curve Nistp384
Degree   : 384
Created  : 10:59:08 UTC Mon Nov 19 2018
Data     :
```

```
04B70BAF C096E2CA D848EE72 6562F3CC 9F12FA40 BE09BFE6 AF0CA179 F29F6407
FEE24A43 84C5A5DE D7912208 CB67EE41 58CB9640 05E9421F 2DCDC41C EED31288
6CACC8DD 861DC887 98E535C4 893CB19F 5ED3F6BC 2C90C39B 10EAED57 87E96F78
B6
```

```
Key label: the_default
Type      : ECDSA General Curve Nistp521
Degree   : 521
Created  : 10:59:09 UTC Mon Nov 19 2018
Data     :
0400BA39 E3B35E13 810D8AE5 260B8047 84E8087B 5137319A C2865629 8455928F
D3D9CE39 00E097FF 6CA369C3 EE63BA57 A4C49C02 B408F682 C2153B7F AAE53EF8
A2926001 EF113896 5F1DA056 2D62F292 B860FDFB 0314CE72 F87AA2C9 D5DD29F4
DA85AE4D 1CA453AC 412E911A 419E9B43 0A13DAD3 7B7E88E4 7D96794B 369D6247
E3DA7B8A 5E
```

The following example shows the output for **ed25519**:

```
Router#show crypto key mypubkey ed25519
Wed Dec 16 16:12:21.464 IST
Key label: the_default
Type      : ED25519
Size     : 256
Created  : 15:08:28 IST Tue Oct 13 2020
Data     :
 649CC355 40F85479 AE9BE26F B5B59153 78D171B6 F40AA53D B2E48382 BA30E5A9

Router#
```

Related Topics

[Automatic Generation of SSH Host-Key Pairs, on page 7](#)

Associated Commands

- `ssh server algorithms host-key`
- `show crypto key mypubkey`

Configure SSH Client

Perform this task to configure an SSH client.

SUMMARY STEPS

1. `configure`
2. `ssh client knownhost device : /filename`
3. Use the `commit` or `end` command.
4. `ssh {ipv4-address | ipv6-address | hostname} [username user- cipher | source-interface type instance]`

DETAILED STEPS

Step 1 **configure**

Example:

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

Step 2 **ssh client knownhost device : /filename**

Example:

```
RP/0/RP0/CPU0:router(config)# ssh client knownhost slot1:/server_pubkey
```

(Optional) Enables the feature to authenticate and check the server public key (pubkey) at the client end.

Note The complete path of the filename is required. The colon (:) and slash mark (/) are also required.

Step 3 Use the **commit** or **end** command.

commit —Saves the configuration changes and remains within the configuration session.

end —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

Step 4 **ssh {ipv4-address | ipv6-address | hostname} [username user- cipher | source-interface type instance]**

Enables an outbound SSH connection.

- To run an SSHv2 server, you must have a VRF. This may be the default or a specific VRF. VRF changes are applicable only to the SSH v2 server.
- The SSH client tries to make an SSHv2 connection to the remote peer. If the remote peer supports only the SSHv1 server, the peer internally spawns an SSHv1 connection to the remote server.
- The **cipher des** option can be used only with an SSHv1 client.
- The SSHv1 client supports only the 3DES encryption algorithm option, which is still available by default for those SSH clients only.
- If the *hostname* argument is used and the host has both IPv4 and IPv6 addresses, the IPv6 address is used.

-
- If you are using SSHv1 and your SSH connection is being rejected, the reason could be that the RSA key pair might have been zeroed out. Another reason could be that the SSH server to which the user is connecting to using SSHv1 client does not accept SSHv1 connections. Make sure that you have specified a hostname and domain. Then use the **crypto key generate rsa** command to generate an RSA host-key pair, and then enable the SSH server.

- If you are using SSHv2 and your SSH connection is being rejected, the reason could be that the DSA, RSA host-key pair might have been zeroed out. Make sure you follow similar steps as mentioned above to generate the required host-key pairs, and then enable the SSH server.
- When configuring the ECDSA, RSA or DSA key pair, you might encounter the following error messages:

- No hostname specified

You must configure a hostname for the router using the **hostname** command.

- No domain specified

You must configure a host domain for the router using the **domain-name** command.

- The number of allowable SSH connections is limited to the maximum number of virtual terminal lines configured for the router. Each SSH connection uses a vty resource.
- SSH uses either local security or the security protocol that is configured through AAA on your router for user authentication. When configuring AAA, you must ensure that the console is not running under AAA by applying a keyword in the global configuration mode to disable AAA on the console.



Note If you are using Putty version 0.63 or higher to connect to the SSH client, set the 'Chokes on PuTTYs SSH2 winadj request' option under SSH > Bugs in your Putty configuration to 'On.' This helps avoid a possible breakdown of the session whenever some long output is sent from IOS XR to the Putty client.

Configuring Secure Shell

The following example shows how to configure SSHv2 by creating a hostname, defining a domain name, enabling the SSH server for local and remote authentication on the router by generating a DSA key pair, bringing up the SSH server, and saving the configuration commands to the running configuration file.

After SSH has been configured, the SFTP feature is available on the router.

From Cisco IOS XR Software Release 7.0.1 and later, the crypto keys are auto-generated at the time of router boot up. Hence, you need to explicitly generate the host-key pair only if it is not present in the router under some scenarios.

```
configure
hostname router1
domain name cisco.com
exit
crypto key generate rsa/dsa
configure
ssh server
end
```

Order of SSH Client Authentication Methods

The default order of authentication methods for SSH clients on Cisco IOS XR routers is as follows:

- On routers running Cisco IOS XR SSH:
 - **public-key**, **password** and **keyboard-interactive**
- On routers running CiscoSSH (open source-based SSH):
 - **public-key**, **keyboard-interactive** and **password**

How to Set the Order of Authentication Methods for SSH Clients

To set the preferred order of authentication methods for SSH clients on Cisco IOS XR routers, use the **ssh client auth-method** command in the XR Config mode. This command is available from Cisco IOS XR Software Release 7.9.2/Release 7.10.1 and later.

Configuration Example

In this example, we set the order of SSH client authentication methods in such a way that public key authentication is negotiated first, followed by keyboard-interactive, and then password-based authentication.

```
Router#configure
Router(config)#ssh client auth-method public-key keyboard-interactive password
Router(config-ssh)#commit
```

Running Configuration

```
Router#show run ssh client auth-methods
Tue Nov 21 17:55:44.688 IST
ssh client auth-methods public-key keyboard-interactive password
Router#
```

SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm

The Cisco IOS XR software provides a new configuration option to control the key algorithms to be negotiated with the peer while establishing an SSH connection with the router. With this feature, you can enable the insecure SSH algorithms on the SSH server, which are otherwise disabled by default. A new configuration option is also available to restrict the SSH client from choosing the HMAC, or hash-based message authentication codes algorithm while trying to connect to the SSH server on the router.

You can also configure a list of ciphers as the default cipher list, thereby having the flexibility to enable or disable any particular cipher.



Caution Use caution in enabling the insecure SSH algorithms to avoid any possible security attack.

To disable the HMAC algorithm, use the **ssh client disable hmac** command or the **ssh server disable hmac** command in XR Config mode.

To enable the required cipher, use the **ssh client enable cipher** command or the **ssh server enable cipher** command in XR Config mode.

The supported encryption algorithms (in the order of preference) are:

1. aes128-ctr
2. aes192-ctr
3. aes256-ctr
4. aes128-gcm@openssh.com
5. aes256-gcm@openssh.com
6. aes128-cbc
7. aes192-cbc
8. aes256-cbc
9. 3des-cbc

In SSH, the CBC-based ciphers are disabled by default. To enable these, you can use the **ssh client enable cipher** command or the **ssh server enable cipher** command with the respective CBC options (aes-cbc or 3des-cbc). All CTR-based and GCM-based ciphers are enabled by default.

Disable HMAC Algorithm

Configuration Example to Disable HMAC Algorithm

```
Router(config)# ssh server disable hmac hmac-shal
Router(config)#commit
```

```
Router(config)# ssh client disable hmac hmac-shal
Router(config)#commit
```

Running Configuration

```
ssh server disable hmac hmac-shal
!
```

```
ssh client disable hmac hmac-shal
!
```

Related Topics

[SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm, on page 12](#)

Associated Commands

- **ssh client disable hmac**

- ssh server disable hmac

Enable Cipher Public Key

Configuration Example to Enable Cipher Public Key

To enable all ciphers on the client and the server:

Router 1:

```
Router(config)# ssh client algorithms cipher aes256-cbc aes256-ctr aes192-ctr aes192-cbc
aes128-ctr aes128-cbc aes128-gcm@openssh.com aes256-gcm@openssh.com 3des-cbc
```

Router 2:

```
Router(config)# ssh server algorithms cipher aes256-cbc aes256-ctr aes192-ctr aes192-cbc
aes128-ctr aes128-cbc aes128-gcm@openssh.com aes256-gcm@openssh.com 3des-cbc
```

To enable the CTR cipher on the client and the CBC cipher on the server:

Router 1:

```
Router(config)# ssh client algorithms cipher aes128-ctr aes192-ctr aes256-ctr
```

Router 2:

```
Router(config)# ssh server algorithms cipher aes128-cbc aes256-cbc aes192-cbc 3des-cbc
```

Without any cipher on the client and the server:

Router 1:

```
Router(config)# no ssh client algorithms cipher
```

Router 2:

```
Router(config)# no ssh server algorithms cipher
```

Enable only the deprecated algorithms on the client and the server:

Router 1:

```
Router(config)# ssh client algorithms cipher aes128-cbc aes192-cbc aes256-cbc 3des-cbc
```

Router 2:

```
Router(config)# ssh server algorithms cipher aes128-cbc aes192-cbc aes256-cbc 3des-cbc
```

Enable the deprecated algorithm (using **enable cipher** command) and enable the CTR cipher (using **algorithms cipher** command) on the client and the server:

Router 1:

```
Router(config)# ssh client enable cipher aes-cbc 3des-cbc
Router(config)# ssh client algorithms cipher aes128-ctr aes192-ctr aes256-ctr
```

Router 2:

```
Router(config)# ssh server enable cipher aes-cbc 3des-cbc
Router(config)# ssh server algorithms cipher aes128-ctr aes192-ctr aes256-ctr
```

Running Configuration

All ciphers enabled on the client and the server:

Router 1:

```
ssh client algorithms cipher aes256-cbc aes256-ctr aes192-ctr aes192-cbc aes128-ctr aes128-cbc
aes128-gcm@openssh.com aes256-gcm@openssh.com 3des-cbc
!
```

Router 2:

```
ssh client algorithms cipher aes256-cbc aes256-ctr aes192-ctr aes192-cbc aes128-ctr aes128-cbc
aes128-gcm@openssh.com aes256-gcm@openssh.com 3des-cbc
!
```

Related Topics

[SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm, on page 12](#)

Associated Commands

- **ssh client enable cipher**
- **ssh server enable cipher**
- **ssh client algorithms cipher**
- **ssh server algorithms cipher**

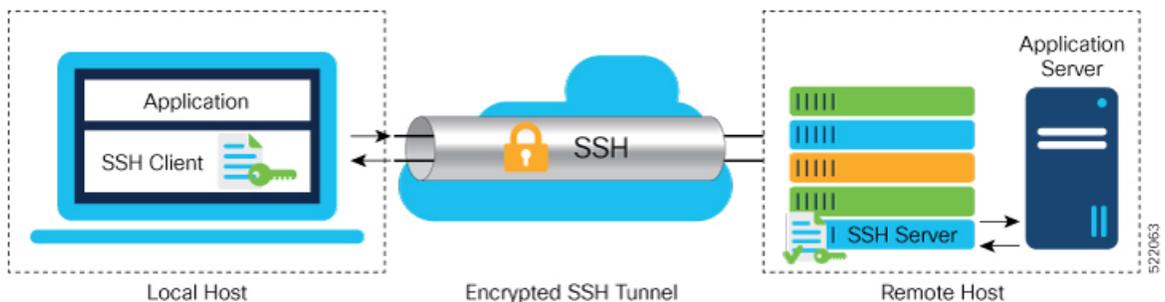
SSH Port Forwarding

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
SSH Port Forwarding	Release 7.3.2	<p>With this feature enabled, the SSH client on a local host forwards the traffic coming on a given port to the specified host and port on a remote server, through an encrypted SSH channel. Legacy applications that do not otherwise support data encryption can leverage this functionality to ensure network security and confidentiality to the traffic that is sent to remote application servers.</p> <p>This feature introduces the <code>ssh server port-forwarding local</code> command.</p>

SSH port forwarding is a method of forwarding the otherwise insecure TCP/IP connections from the SSH client to server through a secure SSH channel. Since the traffic is directed to flow through an encrypted SSH connection, it is tough to snoop or intercept this traffic while in transit. This SSH tunneling provides network security and confidentiality to the data traffic, and hence legacy applications that do not otherwise support encryption can mainly benefit out of this feature. You can also use this feature to implement VPN and to access intranet services across firewalls.

Figure 1: SSH Port Forwarding



Consider an application on the SSH client residing on a local host, trying to connect to an application server residing on a remote host. With tunneling enabled, the application on the SSH client connects to a port on the local host that the SSH client listens to. The SSH client then forwards the data traffic of the application to the SSH server over an encrypted tunnel. The SSH server then connects to the actual application server that is either residing on the same router or on the same data center as the SSH server. The entire communication of the application is thus secured, without having to modify the application or the work flow of the end user.

The SSH port forwarding feature is disabled, by default. You can enable the feature by using the `ssh server port-forwarding local` command in the XR Config mode.

How Does SSH Port Forwarding Work?

Figure 2: Sample Topology for SSH Port Forwarding



Consider a scenario where port forwarding is enabled on the SSH server running on Router-1, in this topology. An SSH client running on a local host tries to create a secure tunnel to the SSH server, for a local application to eventually reach the remote application server running on Router-2.

The client tries to establish an SSH connection to Router-1 using the following command:

```
ssh -L local-port:remote-server-hostname:remote-port username@sshserver-hostname
```

where,

local-port is the local port number of the host where the SSH client and the application reside. Port 5678, in this example.

remote-server-hostname:remote-port is the TCP/IP host name and port number of the remote application server where the recipient (SSH server) must connect the channel from the SSH client to. 192.168.0.2 and 23, in this example.

sshserver-hostname is the domain name or IP address of the SSH server which is the recipient of the SSH client request. 192.168.0.1, in this example.

For example,

```
ssh -L 5678:192.168.0.2:23 admin@192.168.0.1
```

When the SSH server receives a TCP/IP packet from the SSH client, it accepts the packet and opens a socket to the remote server and port specified in that packet. Once the connection between SSH client and server is established, the SSH server connects that communication channel to the newly created socket. From then onwards, SSH server forwards all the incoming data from the client on that channel to that socket. This type of connection is known as port-forwarded local connection. When the client closes the connection, the SSH server closes the socket and the forwarded channel.

How to Enable SSH Port Forwarding

Guidelines for Enabling SSH Port Forwarding Feature

- The Cisco IOS XR software supports SSH port forwarding only on SSH server; not on SSH client. Hence, to utilize this feature, the SSH client running at the end host must already have the support for SSH port forwarding or tunneling.
- The remote host must be reachable on the same VRF where the current SSH connection between the server and the client is established.
- Port numbers need not match for SSH port forwarding to work. You can map any port on the SSH server to any port on the client.

- If the SSH client tries to do port forwarding without the feature being enabled on the SSH server, the port forwarding fails, and displays an error message on the console. Similarly the port-forwarded channel closes in case there is any connectivity issue or if the server receives an SSH packet from the client in an improper format.

Configuration Example

```
Router#configure
Router(config)#ssh server port-forwarding local
Router(config)#commit
```

Running Configuration

```
Router#show running-configuration

ssh server port-forwarding local
!
```

Verification

Use the **show ssh** command to see the details of the SSH sessions. The **connection type** field shows as **tcp-forwarded-local** for the port-forwarded session.

```
Router#show ssh

Wed Oct 14 11:22:05.575 UTC
SSH version : Cisco-2.0

id chan pty location state userid host ver authentication connection
type
-----
Incoming sessions
15 1 XXX 0/RP0/CPU0 SESSION_OPEN admin 192.168.122.1 v2 password
port-forwarded-local

Outgoing sessions

Router#
```

Use the **show ssh server** command to see the details of the SSH server. The **Port Forwarding** column shows as **local** for the port-forwarded session. Whereas, for a regular SSH session, the field displays as **disabled**.

```
Router#show ssh server
```

Syslogs for SSH Port Forwarding Feature

The router console displays the following syslogs at various SSH session establishment events.

- When SSH port forwarding session is successfully established:

```
RP/0/RP0/CPU0:Aug 24 13:10:15.933 IST: SSHD_[66632]:
%SECURITY-SSHD-6-PORT_FWD_INFO_GENERAL : Port Forwarding, Target:=10.105.236.155,
Port:=22, Originator:=127.0.0.1,Port:=41590, Vrf:=0x60000000, Connection forwarded
```

- If SSH client tries to establish a port forwarding session without SSH port forwarding feature being enabled on the SSH server:

```
RP/0/RP0/CPU0:Aug 24 13:20:31.572 IST: SSHD_[65883]: %SECURITY-SSHD-3-PORT_FWD_ERR_GENERAL
: Port Forwarding, Port forwarding is not enabled
```

Associated Command

- `ssh server port-forwarding local`

Non-Default SSH Port

Table 2: Feature History Table

Feature Name	Release Information	Feature Description
Non-Default SSH Port	Release 7.7.1	<p>We have enhanced the system security to minimize the automated attacks that may target the default Secure Socket Shell (SSH) port on your router. You can now specify a non-default port number for the SSH server on your router. The SSH, Secure Copy Protocol (SCP), and Secure File Transfer Protocol (SFTP) client services can then access your router only through this non-default port. The new port option also enables the SSH, SCP, and SFTP clients on your router to connect to SSH servers on the network that use a wide range of non-default port numbers. In earlier releases, these SSH, SCP, and SFTP connections were established through the default SSH port, 22. The non-default SSH port is supported only on SSH version 2.</p> <p>The feature introduces the <code>ssh server port</code> command.</p> <p>The feature modifies these commands to include the <code>port</code> option:</p> <ul style="list-style-type: none"> • <code>ssh</code> • <code>sftp</code> • <code>scp</code>

The SSH, SCP, and SFTP services on the Cisco IOS XR routers used the default SSH port number, 22, to establish connections between the server and the client. From Cisco IOS XR Software Release 7.7.1 and later, you can specify a non-default SSH port number within a specific range for these services on Cisco IOS XR 64-bit routers. This non-default port option is available for routers that are functioning as servers, or as clients for the SSH, SCP and SFTP services. This feature helps to restrict insecure client services from accessing the router through the default SSH server port. Similarly, for Cisco IOS XR routers that are running as SSH clients, the non-default port number option enables them to connect to other SSH servers on the network that listens on a wide range of non-default SSH port numbers.

The non-default SSH port number ranges from 5520 to 5529 for the SSH server, and from 1025 to 65535 for the SSH client.

The SSH server on the router does not listen on both the default and non-default ports at the same time. If you have configured a non-default SSH server port, then the server listens only on that non-default port for the client connections. The SSH clients can then establish sessions through this non-default SSH port. The SCP and SFTP services also use the same SSH port for their connections, and hence they establish the client sessions through the newly configured port.

If a session was already established through the default port, then that session remains intact even if you change the ssh server port to a non-default port. The further client sessions are attempted through the newly configured non-default port.

Restrictions for Non-Default SSH Port

These restrictions apply to the non-default SSH port option:

- Available only on 64-bit Cisco IOS XR routers; not on 32-bit routers
- Available only on version 2 of SSH (SSHv2); not on version 1 (SSHv1)

How to Configure Non-Default SSH Port



Note To establish SSH connections on the non-default port, ensure that the non-default port that you select for the SSH server is not used by any other application on the router.

Configuration Example

SSH Server:

To configure the non-default SSH port for the SSH server on the router, use the **ssh server port** command in the XR Config mode.

```
Router#configure
Router(config)#ssh server port 5520
Router(config)#commit
```

SSH Client:

Similarly, the **port** option is available for the SSH client also, to initiate a connection to another SSH server that listens on a non-default SSH port number.

This example shows how to connect to an SSH server, with IP address 198.51.100.1, that is listening on non-default SSH port 5525.

```
Router#ssh 198.51.100.1 port 5525 username user1
```

Running Configuration

This is a sample running configuration of the SSH server.

```
Router#show running-configuration
!
ssh server v2
ssh server port 5520
ssh server vrf default
!
```

Verification

Use the following **show** commands to verify the SSH server configuration and LPTS entries for SSH connections.

In this example, the **SSH port** field displays the port number, '5520', that you have configured for the SSH server.

```
Router#show ssh server
Fri May 20 07:22:57.579 UTC
-----
SSH Server Parameters
-----

Current supported versions := v2
                        SSH port := 5520
                        SSH vrfs := vrfname:=default (v4-acl:=, v6-acl:=)
                        Netconf Port := 830
                        Netconf Vrfs :=

Algorithms
-----
Hostkey Algorithms :=
x509-sha256-cert-v01@openssh.com,ecdsa-sha2-nistp521,ecdsa-sha2-nistp384,ecdsa-sha2-nistp256,rsa-sha2-512,rsa-sha2-256,ssh-rsa,ssh-dss,ssh-ed25519

Key-Exchange Algorithms :=
ecdsa-sha2-nistp521,ecdsa-sha2-nistp384,ecdsa-sha2-nistp256,diffie-hellman-grip4-sha1,curve25519-sha256,diffie-hellman-grip4-sha256,diffie-hellman-grip6-sha512,curve25519-sha256@libssh.org

Encryption Algorithms :=
aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com

Mac Algorithms := hmac-sha2-512,hmac-sha2-256,hmac-sha1

Authentication Method Supported
-----
PublicKey := Yes
Password := Yes
Keyboard-Interactive := Yes
Certificate Based := Yes

Others
-----
DSCP := 16
Ratelimit := 60
```

```

        Sessionlimit := 64
        Rekeytime := 60
        Server rekeyvolume := 1024
    TCP window scale factor := 1
        Backup Server := Disabled
        Host Trustpoint :=
        User Trustpoint :=
        Port Forwarding := Disabled
Max Authentication Limit := 20
    Certificate username := Common name (CN)
    OpenSSH Host Trustpoint :=
    OpenSSH User Trustpoint :=

```

In the following example, the **Port** field in the **Local-Address,Port** column for the **TCP** entry for SSH displays the port number as '5520'. This is the port on which the SSH server listens for client connections.

```

Router#show lpts bindings brief
Fri May 20 07:23:21.416 UTC

```

@ - Indirect binding; Sc - Scope

Location	Clnt	Sc	L3	L4	VRF-ID	Interface	Local-Address,Port	Remote-Address,Port
0/RP0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, ECHO	any
0/RP0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, TSTAMP	any
0/RP0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, MASKREQ	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, ECHOREQ	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDRTRSLCT	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDRTRADV	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDNBRSLCT	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDNBRADV	any
0/RP0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDREDIRECT	any
0/RP0/CPU0	BFD	LO	IPV4	UDP	*	any	any	any
0/0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, ECHO	any
0/0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, TSTAMP	any
0/0/CPU0	IPV4	LO	IPV4	ICMP	*	any	any, MASKREQ	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, ECHOREQ	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDRTRSLCT	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDRTRADV	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDNBRSLCT	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDNBRADV	any
0/0/CPU0	IPV6	LO	IPV6	ICMP6	*	any	any, NDREDIRECT	any
0/0/CPU0	BFD	LR	IPV4	UDP	*	any	any 128.64.0.0/16	
0/RP0/CPU0	TCP	LR	IPV6	TCP	default	any	any, 5520	any
0/RP0/CPU0	TCP	LR	IPV4	TCP	default	any	any, 5520	any
0/RP0/CPU0	UDP	LR	IPV6	UDP	default	any	any, 33433	any
0/RP0/CPU0	UDP	LR	IPV4	UDP	default	any	any, 33433	any
0/RP0/CPU0	RAW	LR	IPV4	IGMP	default	any	any	any
0/RP0/CPU0	RAW	LR	IPV4	L2TPV3	default	any	any	any
0/RP0/CPU0	RAW	LR	IPV6	ICMP6	default	any	any, MLDLQUERY	any
0/RP0/CPU0	RAW	LR	IPV6	ICMP6	default	any	any, LSTNRREPORT	any
0/RP0/CPU0	RAW	LR	IPV6	ICMP6	default	any	any, MLDLSTNRDN	any
0/RP0/CPU0	RAW	LR	IPV6	ICMP6	default	any	any, LSTNRREPORT	any

```
Router#
```

If the non-default port was not configured, then the SSH server listens on the default SSH port 22, and the above **Port** field displays '22'.

If a session was already established through the default port, and if you change the ssh server port to a non-default port, then the output still displays an entry for that session on the default port, 22. Another entry

shows that the SSH server is listening on the newly configured non-default port. New connections establish through the non-default port, 5520, in this example.

```

Location      Clnt Sc L3   L4   VRF-ID      Interface Local-Address,Port Remote-Address,Port
-----
.
.
.
0/RP0/CPU0 TCP  LR IPV4 TCP default  any      192.0.2.1,5520    198.51.100.1,37764
0/RP0/CPU0 TCP  LR IPV4 TCP default  any      any,5520 any
0/RP0/CPU0 TCP  LR IPV6 TCP default  any      any,5520 any
0/RP0/CPU0 TCP  LR IPV4 TCP default  any      192.0.2.1,22     198.51.100.1,45722
.
.
.

```

Information About Implementing Secure Shell

To implement SSH, you should understand the following concepts:

SSH Server

The SSH server feature enables an SSH client to make a secure, encrypted connection to a Cisco router. This connection provides functionality that is similar to that of an inbound Telnet connection. Before SSH, security was limited to Telnet security. SSH allows a strong encryption to be used with the Cisco software authentication. The SSH server in Cisco software works with publicly and commercially available SSH clients.

SSH Client

The SSH client feature is an application running over the SSH protocol to provide device authentication and encryption. The SSH client enables a Cisco router to make a secure, encrypted connection to another Cisco router or to any other device running the SSH server. This connection provides functionality that is similar to that of an outbound Telnet connection except that the connection is encrypted. With authentication and encryption, the SSH client allows for a secure communication over an insecure network.

The SSH client works with publicly and commercially available SSH servers. The SSH client supports the ciphers of AES, 3DES, message digest algorithm 5 (MD5), SHA1, and password authentication. User authentication is performed in the Telnet session to the router. The user authentication mechanisms supported for SSH are RADIUS, TACACS+, and the use of locally stored usernames and passwords.

The SSH client supports setting DSCP value in the outgoing packets.

```
ssh client dscp <value from 0 - 63>
```

If not configured, the default DSCP value set in packets is 16 (for both client and server).

The SSH client supports the following options:

- DSCP—DSCP value for SSH client sessions.

```

RP/0/5/CPU0:router#configure
RP/0/5/CPU0:router(config)#ssh ?
  client  Provide SSH client service
  server  Provide SSH server service

```

```

    timeout Set timeout value for SSH
RP/0/5/CPU0:router(config)#ssh client ?

```

- **Knownhost**—Enable the host pubkey check by local database.
- **Source-interface**—Source interface for SSH client sessions.

```

RP/0/5/CPU0:router(config)#ssh client source-interface ?
ATM                ATM Network Interface(s)
BVI                 Bridge-Group Virtual Interface
Bundle-Ether       Aggregated Ethernet interface(s)
CEM                 Circuit Emulation interface(s)
GigabitEthernet    GigabitEthernet/IEEE 802.3 interface(s)
IMA                ATM Network Interface(s)
IMtestmain         IM Test Interface
Loopback           Loopback interface(s)
MgmtEth            Ethernet/IEEE 802.3 interface(s)
Multilink          Multilink network interface(s)
Null               Null interface
PFItestmain        PFI Test Interface
PFItestnothw       PFI Test Not-HW Interface
PW-Ether          PWHE Ethernet Interface
PW-IW              PWHE VC11 IP Interworking Interface
Serial             Serial network interface(s)
VASIleft           VASI Left interface(s)
VASIRight          VASI Right interface(s)
test-bundle-channel Aggregated Test Bundle interface(s)
tunnel-ipsec       IPSec Tunnel interface(s)
tunnel-mte         MPLS Traffic Engineering P2MP Tunnel interface(s)
tunnel-te          MPLS Traffic Engineering Tunnel interface(s)
tunnel-tp          MPLS Transport Protocol Tunnel interface
RP/0/5/CPU0:router(config)#ssh client source-interface
RP/0/5/CPU0:router(config)#

```

SSH also supports remote command execution as follows:

```

RP/0/5/CPU0:router#ssh ?
A.B.C.D IPv4 (A.B.C.D) address
WORD    Hostname of the remote node
X:X::X  IPv6 (A:B:C:D...:D) address
vrf     vrf table for the route lookup
RP/0/5/CPU0:router#ssh 10.1.1.1 ?
cipher      Accept cipher type
command     Specify remote command (non-interactive)
source-interface Specify source interface
username    Accept userid for authentication
<cr>
RP/0/5/CPU0:router#ssh 192.68.46.6 username admin command "show redundancy sum"
Password:

Wed Jan  9 07:05:27.997 PST
Active Node      Standby Node
-----
0/4/CPU0        0/5/CPU0 (Node Ready, NSR: Not Configured)

RP/0/5/CPU0:router#

```

SFTP Feature Overview

SSH includes support for standard file transfer protocol (SFTP), a new standard file transfer protocol introduced in SSHv2. This feature provides a secure and authenticated method for copying router configuration or router image files.

The SFTP client functionality is provided as part of the SSH component and is always enabled on the router. Therefore, a user with the appropriate level can copy files to and from the router. Like the **copy** command, the **sftp** command can be used only in XR EXEC mode.

The SFTP client is VRF-aware, and you may configure the secure FTP client to use the VRF associated with a particular source interface during connections attempts. The SFTP client also supports interactive mode, where the user can log on to the server to perform specific tasks via the Unix server.

The SFTP Server is a sub-system of the SSH server. In other words, when an SSH server receives an SFTP server request, the SFTP API creates the SFTP server as a child process to the SSH server. A new SFTP server instance is created with each new request.

The SFTP requests for a new SFTP server in the following steps:

- The user runs the **sftp** command with the required arguments
- The SFTP API internally creates a child session that interacts with the SSH server
- The SSH server creates the SFTP server child process
- The SFTP server and client interact with each other in an encrypted format
- The SFTP transfer is subject to LPTS policer "SSH-Known". Low policer values will affect SFTP transfer speeds



Note In IOS-XR SW release 4.3.1 onwards the default policer value for SSH-Known has been reset from 2500pps to 300pps. Slower transfers are expected due to this change. You can adjust the lpts policer value for this punt cause to higher values that will allow faster transfers

When the SSH server establishes a new connection with the SSH client, the server daemon creates a new SSH server child process. The child server process builds a secure communications channel between the SSH client and server via key exchange and user authentication processes. If the SSH server receives a request for the sub-system to be an SFTP server, the SSH server daemon creates the SFTP server child process. For each incoming SFTP server subsystem request, a new SSH server child and a SFTP server instance is created. The SFTP server authenticates the user session and initiates a connection. It sets the environment for the client and the default directory for the user.

Once the initialization occurs, the SFTP server waits for the SSH_FXP_INIT message from the client, which is essential to start the file communication session. This message may then be followed by any message based on the client request. Here, the protocol adopts a 'request-response' model, where the client sends a request to the server; the server processes this request and sends a response.

The SFTP server displays the following responses:

- Status Response
- Handle Response
- Data Response
- Name Response



Note The server must be running in order to accept incoming SFTP connections.

RSA Based Host Authentication

Verifying the authenticity of a server is the first step to a secure SSH connection. This process is called the host authentication, and is conducted to ensure that a client connects to a valid server.

The host authentication is performed using the public key of a server. The server, during the key-exchange phase, provides its public key to the client. The client checks its database for known hosts of this server and the corresponding public-key. If the client fails to find the server's IP address, it displays a warning message to the user, offering an option to either save the public key or discard it. If the server's IP address is found, but the public-key does not match, the client closes the connection. If the public key is valid, the server is verified and a secure SSH connection is established.

The IOS XR SSH server and client had support for DSA based host authentication. But for compatibility with other products, like IOS, RSA based host authentication support is also added.

RSA Based User Authentication

One of the method for authenticating the user in SSH protocol is RSA public-key based user authentication. The possession of a private key serves as the authentication of the user. This method works by sending a signature created with a private key of the user. Each user has a RSA keypair on the client machine. The private key of the RSA keypair remains on the client machine.

The user generates an RSA public-private key pair on a unix client using a standard key generation mechanism such as ssh-keygen. The max length of the keys supported is 4096 bits, and the minimum length is 512 bits. The following example displays a typical key generation activity:

```
bash-2.05b$ ssh-keygen -b 1024 -t rsa
Generating RSA private key, 1024 bit long modulus
```

The public key must be in base64 encoded (binary) formats for it to be imported correctly into the router.



Note You can use third party tools available on the Internet to convert the key to the binary format.

Once the public key is imported to the router, the SSH client can choose to use the public key authentication method by specifying the request using the “-o” option in the SSH client. For example:

```
client$ ssh -o PreferredAuthentications=publickey 1.2.3.4
```

If a public key is not imported to a router using the RSA method, the SSH server initiates the password based authentication. If a public key is imported, the server proposes the use of both the methods. The SSH client then chooses to use either method to establish the connection. The system allows only 10 outgoing SSH client connections.

Currently, only SSH version 2 and SFTP server support the RSA based authentication.



Note The preferred method of authentication would be as stated in the SSH RFC. The RSA based authentication support is only for local authentication, and not for TACACS/RADIUS servers.

Authentication, Authorization, and Accounting (AAA) is a suite of network security services that provide the primary framework through which access control can be set up on your Cisco router or access server.

SSHv2 Client Keyboard-Interactive Authentication

An authentication method in which the authentication information is entered using a keyboard is known as keyboard-interactive authentication. This method is an interactive authentication method in the SSH protocol. This type of authentication allows the SSH client to support different methods of authentication without having to be aware of their underlying mechanisms.

Currently, the SSHv2 client supports the keyboard-interactive authentication. This type of authentication works only for interactive applications.



Note The password authentication is the default authentication method. The keyboard-interactive authentication method is selected if the server is configured to support only the keyboard-interactive authentication.
