Secure Shell Version 2

Feature History for Secure Shell Version 2

This table provides release and platform support information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature Name and Description	Supported Platform
Cisco IOS XE 17.18.1	SSH Version 2: SSH Version 2 provides secure remote access, command execution, and file transfer (SCP) through strong encryption and robust authentication over untrusted networks.	Cisco C9350 Series Smart Switches Cisco C9610 Series Smart Switches

What is Secure Shell Version 2?

Secure Shell Version 2 (SSH) is a network protocol that enables secure remote access and command execution over an untrusted network. It provides strong authentication and encryption capabilities over a reliable transport layer, such as TCP. SSH also includes the Secure Copy Protocol (SCP) feature for secure file transfer.

SSHv2 Enhancements

- **Virtual Routing and Forwarding (VRF)-Aware SSH**: The **ssh** command was extended to add VRF awareness to the SSH client-side functionality. This allows the VRF instance name in the client to provide the IP address for looking up the correct routing table and establishing a connection.

Note VRF-Aware SSH support depends on your software release.

- **Diffie-Hellman (DH) Group Exchange Support**: Cisco SSH implementations traditionally used a 768-bit modulus. With the increasing need for higher key sizes to accommodate DH Group 14 (2048 bits) and Group 16 (4096 bits) cryptographic applications, a message exchange between the client and server is necessary to establish the favored DH group. Use the **ip ssh dh min size** command to configure the minimum modulus size on the SSH server.
- **Debug Enhancements**: The **debug ip ssh** command was enhanced to simplify debugging. Previously, it printed all SSH-related debug messages. Now, you can configure the command with a keyword to limit messages to specific information.

RSA Key Authentication

User Authentication: RSA-based user authentication uses a private/public key pair associated with each user. The user must generate this key pair on the client and configure the public key on the Cisco SSH server. Cisco Confidential During authentication, the SSH user provides an encrypted signature using their private key. This signature and the user’s public key are sent to the SSH server. The server computes a hash over the provided public key to determine if a matching entry exists. If a match is found, an RSA-based message verification is performed using the public key, authenticating or denying the user based on the encrypted signature.

Server Authentication: The Cisco SSH client authenticates the SSH server during session establishment using the server's host keys exchanged during the key exchange phase. These keys, created when SSH is enabled, are used to identify the SSH server and must be configured on the client for authentication. When the client attempts to establish an SSH session, it receives the server's signature as part of the key exchange message. If the strict host key checking flag is enabled on the client, the client verifies if it has a corresponding host key entry for the server. If a match is found, the client attempts to validate the signature using the server's host key. Successful server authentication allows session establishment to continue; otherwise, the session terminates, and a "Server Authentication Failed" message displays.

Note

- Storing public keys on a server consumes memory. Therefore, the number of public keys configurable on an SSH server is limited to ten users, with a maximum of two public keys per user.
- The Cisco server supports RSA-based user authentication. However, Cisco clients cannot propose public key as an authentication method. If the Cisco server receives an RSA-based authentication request from an OpenSSH client, the server accepts the request.
- For proper server authentication, manually configure the server's RSA public key and configure the **ip ssh stricthostkeycheck** command on the Cisco SSH client.

SSH Ed25519 Encryption

Ed25519 is a public-key signature system that offers features including:

- Fast single-signature verification
- Fast batch verification
- Fast signing
- Fast key generation
- High security level
- Foolproof session keys
- Small signatures (64 bytes)
- Small keys (32 bytes)

SSH and Switch Access Secure

Shell (SSH) provides secure, remote connections to devices, offering stronger encryption and authentication compared to Telnet. This software release supports SSH Version 2 (SSHv2). SSH functionality is consistent across IPv4 and IPv6, enabling secure, encrypted connections with remote IPv6 nodes over an IPv6 transport.

SNMP Trap Generation Simple Network Management Protocol (SNMP) traps can be automatically generated upon SSH session termination, depending on your software release. To enable these traps:

- Ensure SNMP traps are enabled.
- Enable SNMP debugging using the debug snmp packet command to display the traps.
- When configuring the **snmp-server host** command, specify the IP address of the PC hosting the SSH (or Telnet) client, ensuring it has IP connectivity to the SSH server. Trap information includes details such as the number of bytes sent and the protocol used for the SSH session.

SSH Keyboard Interactive Authentication

The SSH Keyboard Interactive Authentication feature, also known as Generic Message Authentication for SSH, allows the implementation of various authentication mechanisms that require only user input. This feature is automatically enabled. Supported authentication methods include:

- Password
- SecurID and hardware tokens (responding to a server challenge with a number or string)
- Pluggable Authentication Module (PAM)
- S/KEY (and other One-Time-Pads)

Prerequisites for Secure Shell Version 2

- Ensure that the required image is loaded on your device. The SSH server requires a k9 (Triple Data Encryption Standard [3DES]) software image, depending on the software release.
- Use an SSH remote device that supports SSH Version 2 to connect to a Cisco device.
- Configure Authentication, Authorization, and Accounting (AAA) on the device. SCP relies on AAA to function correctly, so configuring AAA enables SCP on the SSH server.

Note SSH Version 2 servers and clients are supported on specific Cisco software releases. The SSH client supports both SSH Version 1 and SSH Version 2 protocols and is available in k9 images, depending on the software release.

Restrictions for Secure Shell Version 2

- SSH servers and clients require Triple Data Encryption Standard (3DES) software images.
- Only Execution Shell, remote command execution, and Secure Copy Protocol (SCP) applications are supported.
- Rivest, Shamir, and Adleman (RSA) key generation is required for SSH servers. SSH clients do not need to generate RSA keys.
- The RSA key pair size must be 768 bits or greater.
- The following features are unsupported:
  - Port forwarding
  - Compression

Configuration Guidelines

To configure SSH Version 2, use the **ip ssh version** command. If this command is not configured, SSH operates in compatibility mode by default, honoring both SSH Version 1 and SSH Version 2 connections.

Note SSH Version 1 is not a standardized protocol. To prevent your device from reverting to this undefined protocol, specify Version 2 using the **ip ssh version** command.

The **ip ssh rsa keypair-name** command enables an SSH connection using configured Rivest, Shamir, and Adleman (RSA) keys. This command allows SSH to be enabled without the previous requirement of configuring a hostname and domain name. SSH is enabled if the specified key pair exists or is generated later.

Note The login banner is supported in SSH Version 2, but not in SSH Version 1.

How to Configure Secure Shell Version 2

Configuring a Device for SSH Version 2 Using a Hostname and Domain Name

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 configure terminal

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 hostname *name*

Example:

```
Device(config)# hostname catalyst9k
```

Configures a hostname for your device.

Step 4 ip domain name *name*

Example:

```
catalyst9k(config)# ip domain name example.com
```

Configures a domain name for your device.

Step 5 crypto key generate rsa

Example:

```
catalyst9k(config)# crypto key generate rsa
```

Enables the SSH server for local and remote authentication.

Step 6 ip ssh [time-out *seconds* | authentication-retries *integer*]

Example:

```
catalyst9k(config)# ip ssh time-out 120
```

(Optional) Configures SSH control variables on your device.

Step 7 ip ssh version [1 | 2]

Example:

```
catalyst9k(config)# exit
```

(Optional) Specifies the version of SSH to be run on your device.

Step 8 exit

Example:

```
catalyst9k(config)# ip ssh version 1
```

Exits global configuration mode and enters privileged EXEC mode.

- Use no hostname command to return to the default host.

Configuring a Device for SSH Version 2 Using RSA Key Pairs

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 configure terminal

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 ip ssh rsa keypair-name *keypair-name*

Example:

```
Device(config)# ip ssh rsa keypair-name sshkeys
```

Specifies the RSA key pair to be used for SSH.

Note
A Cisco device can have many RSA key pairs.

Step 4 crypto key generate rsa usage-keys label *key-label* modulus *modulus-size*

Example:

```
Device(config)# crypto key generate rsa usage-keys label sshkeys modulus 768
```

Enables the SSH server for local and remote authentication on the device.

- For SSH Version 2, the modulus size must be at least 768 bits.

Note
To delete the RSA key pair, use the crypto key zeroize rsa command. When you delete the RSA key pair, you automatically disable the SSH server.

Step 5 ip ssh [time-out *seconds* | authentication-retries *integer*]

Example:

```
Device(config)# ip ssh time-out 12
```

Configures SSH control variables on your device.

Step 6 ip ssh version 2

Example:

```
Device(config)# ip ssh version 2
```

Specifies the version of SSH to be run on the device.

Step 7 exit

Example:

```
catalyst9k(config)# ip ssh version 1
```

Exits global configuration mode and enters privileged EXEC mode.

Configuring the Cisco SSH Server to Perform User Authentication

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 configure terminal

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 hostname *name*

Example:

```
Device(config)# hostname host1
```

Specifies the hostname.

Step 4 ip domain name *name*

Example:

```
host1(config)# ip domain name name1
```

Defines a default domain name that the Cisco software uses to complete unqualified hostnames.

Step 5 crypto key generate rsa

Example:

```
host1(config)# crypto key generate rsa
```

Generates RSA key pairs.

Step 6 ip ssh pubkey-chain

Example:

```
host1(config)# ip ssh pubkey-chain
```

Configures public keys for user and server authentication on the SSH server and enters public-key configuration mode.

- The user authentication is successful if the public key stored on the server is verified with the public or the private key pair stored on the client.

Step 7 username *username*

Example:

```
host1(conf-ssh-pubkey)# username user1
```

Configures the SSH username and enters public-key user configuration mode.

Step 8 key-string

Example:

```
host1(conf-ssh-pubkey-user)# key-string
```

Specifies the public key of the remote peer and enters public-key data configuration mode.

Note
You can obtain the public key value from an open SSH client; that is, from the .ssh/id_rsa.pub file.

Step 9 exit

Example:

```
host1(conf-ssh-pubkey-data)# exit
```

Exits public-key data configuration mode and enters public-key server configuration mode.

Step 10 key-hash *key-type key-name*

Example:

```
host1(conf-ssh-pubkey-data)# key-hash ssh-rsa key1
or
host1(conf-ssh-pubkey-data)# key-hash ssh-ed25519 DFEAF10390E560AEA745CCBA53E044ED
```

(Optional) Specifies the SSH key type and version.

- The key type can be ssh-rsa or ssh-ed25519 for the configuration of private public key pairs.
- This step is optional only if the key-string command is configured.
- You must configure either the key-string command or the key-hash command.

Note
You can use a hashing software to compute the hash of the public key string, or you can also copy the hash value from another Cisco device. Entering the public key data using the key-string command is the preferred way to enter the public key data for the first time.

Step 11 end

Example:

```
host1(conf-ssh-pubkey-server)# end
```

Exits public-key server configuration mode and returns to privileged EXEC mode.

Step 12 configure terminal

Example:

```
host1# configure terminal
```

Enters global configuration mode.

Step 13 ip ssh stricthostkeycheck

Example:

```
host1(config)# ip ssh stricthostkeycheck
```

Ensures that server authentication takes place.

- The connection is terminated in case of a failure.
- Use no hostname command to return to the default host.

Step 14 end

Example:

```
host1(config)# end
```

Exits global configuration mode and returns to privileged EXEC mode.

Configuring the Cisco IOS SSH Client to Perform Server Authentication

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 configure terminal

Example:

```
Device# configure terminal
```

Enters global configuration mode.

Step 3 hostname *name*

Example:

```
Device(config)# hostname host1
```

Specifies the hostname.

Step 4 ip domain name *name*

Example:

```
host1(config)# ip domain name name1
```

Defines a default domain name that the Cisco software uses to complete unqualified hostnames.

Step 5 crypto key generate rsa

Example:

```
host1(config)# crypto key generate rsa
```

Generates RSA key pairs.

Step 6 ip ssh pubkey-chain

Example:

```
host1(config)# ip ssh pubkey-chain
```

Configures public keys for user and server authentication on the SSH server and enters public-key configuration mode.

Step 7 server *server-name*

Example:

```
host1(conf-ssh-pubkey)# server server1
```

Enables the SSH server for public-key authentication on the device and enters public-key server configuration mode.

Step 8 key-string

Example:

```
host1(conf-ssh-pubkey-server)# key-string
```

Specifies the public key of the remote peer and enters public-key data configuration mode.

Note
You can obtain the public key value from an open SSH client; that is, from the .ssh/id_rsa.pub file.

Step 9 exit

Example:

```
host1(conf-ssh-pubkey-data)# exit
```

Exits public-key data configuration mode and enters public-key server configuration mode.

Step 10 key-hash *key-type key-name*

Example:

```
host1(conf-ssh-pubkey-data)# key-hash ssh-rsa key1
or
host1(conf-ssh-pubkey-data)# key-hash ssh-ed25519 DFEAF10390E560AEA745CCBA53E044ED
```

(Optional) Specifies the SSH key type and version.

- The key type can be ssh-rsa or ssh-ed25519 for the configuration of private public key pairs.
- This step is optional only if the key-string command is configured.
- You must configure either the key-string command or the key-hash command.

Note
You can use a hashing software to compute the hash of the public key string, or you can also copy the hash value from another Cisco device. Entering the public key data using the key-string command is the preferred way to enter the public key data for the first time.

Step 11 end

Example:

```
host1(conf-ssh-pubkey-server)# end
```

Exits public-key server configuration mode and returns to privileged EXEC mode.

Step 12 configure terminal

Example:

```
host1# configure terminal
```

Enters global configuration mode.

Step 13 ip ssh stricthostkeycheck

Example:

```
host1(config)# ip ssh stricthostkeycheck
```

Ensures that server authentication takes place.

- The connection is terminated in case of a failure.
- Use no hostname command to return to the default host.

Step 14 end

Example:

```
host1(config)# end
```

Exits global configuration mode and returns to privileged EXEC mode.

Starting an Encrypted Session with a Remote Device

Note The device you want to connect to must support an SSH server with an encryption algorithm supported by Cisco software. You do not need to explicitly enable the SSH service on your device; SSH can be configured even if it is not actively running.

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 ssh [-v {1 | 2 } | -c {aes128-ctr | aes192-ctr | aes256-ctr | aes128-cbc | 3des | aes192-cbc | aes256-cbc } | -l *user-id* | -l *user-id* : *vrf-name number ip-address ip-address* | -l *user-id* : *rotary number ip-address* | -m {hmac-md5-128 | hmac-md5-96 | hmac-sha1-160 | hmac-sha1-96 } | -o numberofpasswordprompts *n* | -p *port-num*] {*ip-addr* | *hostname* } [command | -vrf]

Example:

```
Device# ssh -v 2 -c aes256-ctr -m hmac-sha1-96 -l user2 10.76.82.24
```

Starts an encrypted session with a remote networking device.

Verifying the Status of the Secure Shell Connection

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 show ssh

Example:

```
Device# show ssh
```

Displays the status of SSH server connections.

Step 3 exit

Example:

```
Device# exit
```

Exits privileged EXEC mode and returns to user EXEC mode.

Example

The following sample output from the show ssh command displays status of various SSH Version 1 and Version 2 connections for Version 1 and Version 2 connections:

```
Device# show ssh
Connection Version Encryption     State          Username
      0      1.5      3DES     Session started    lab
Connection Version Mode Encryption Hmac State
Username
1    2.0 IN aes128-cbc hmac-md5 Session started   lab
1    2.0 OUT aes128-cbc hmac-md5 Session started  lab
```

The following sample output from the show ssh command displays status of various SSH Version 1 and Version 2 connections for a Version 2 connection with no Version 1 connection:

```
Device# show ssh
Connection Version Mode Encryption Hmac State
Username
1 2.0 IN aes128-cbc hmac-md5 Session started lab
1 2.0 OUT aes128-cbc hmac-md5 Session started lab
%No SSHv1 server connections running.
```

The following sample output from the show ssh command displays status of various SSH Version 1 and Version 2 connections for a Version 1 connection with no Version 2 connection:

```
Device# show ssh
Connection Version Encryption   State        Username
   0        1.5       3DES Session started lab
%No SSHv2 server connections running.
```

Verifying the Secure Shell Version 2 Status

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 show ip ssh

Example:

```
Device# show ip ssh
```

Displays the version and configuration data for SSH.

Step 3 exit

Example:

```
Device# exit
```

Exits privileged EXEC mode and returns to user EXEC mode.\

Examples

The following sample output from the show ip ssh command displays the version of SSH that is enabled, the authentication timeout values, and the number of authentication retries for Version 1 and Version 2 connections:

```
Device# show ip ssh
SSH Enabled - version 1.99
Authentication timeout: 120 secs; Authentication retries: 3
```

The following sample output from the show ip ssh command displays the version of SSH that is enabled, the authentication timeout values, and the number of authentication retries for a Version 2 connection with no Version 1 connection:

```
Device# show ip ssh
SSH Enabled - version 2.0
Authentication timeout: 120 secs; Authentication retries: 3
```

The following sample output from the show ip ssh command displays the version of SSH that is enabled, the authentication timeout values, and the number of authentication retries for a Version 1 connection with no Version 2 connection:

```
Device# show ip ssh
3d06h: %SYS-5-CONFIG_I: Configured from console by console
SSH Enabled - version 1.5
Authentication timeout: 120 secs; Authentication retries: 3
```

Monitoring and Maintaining Secure Shell Version 2

Procedure

Step 1 enable

Example:

```
Device> enable
```

Enables privileged EXEC mode. Enter your password, if prompted.

Step 2 debug ip ssh

Example:

```
Device# debug ip ssh
```

Enables debugging of SSH.

Step 3 debug snmp packet

Example:

```
Device# debug snmp packet
```

Enables debugging of every SNMP packet sent or received by the device.

Example

What to do next

-

Example

The following sample output from the debug ip ssh command shows the connection is an SSH Version 2 connection:

```
Device# debug ip ssh
00:33:55: SSH1: starting SSH control process
00:33:55: SSH1: sent protocol version id SSH-1.99-Cisco-1.25
00:33:55: SSH1: protocol version id is - SSH-2.0-OpenSSH_2.5.2p2
00:33:55: SSH2 1: send: len 280 (includes padlen 4)
00:33:55: SSH2 1: SSH2_MSG_KEXINIT sent
00:33:55: SSH2 1: ssh_receive: 536 bytes received
00:33:55: SSH2 1: input: packet len 632
00:33:55: SSH2 1: partial packet 8, need 624, maclen 0
00:33:55: SSH2 1: ssh_receive: 96 bytes received
00:33:55: SSH2 1: partial packet 8, need 624, maclen 0
00:33:55: SSH2 1: input: padlen 11
00:33:55: SSH2 1: received packet type 20
00:33:55: SSH2 1: SSH2_MSG_KEXINIT received
00:33:55: SSH2: kex: client->server aes128-cbc hmac-md5 none
00:33:55: SSH2: kex: server->client aes128-cbc hmac-md5 none
00:33:55: SSH2 1: expecting SSH2_MSG_KEXDH_INIT
00:33:55: SSH2 1: ssh_receive: 144 bytes received
00:33:55: SSH2 1: input: packet len 144
00:33:55: SSH2 1: partial packet 8, need 136, maclen 0
00:33:55: SSH2 1: input: padlen 5
00:33:55: SSH2 1: received packet type 30
00:33:55: SSH2 1: SSH2_MSG_KEXDH_INIT received
00:33:55: SSH2 1: signature length 111
00:33:55: SSH2 1: send: len 384 (includes padlen 7)
00:33:55: SSH2: kex_derive_keys complete
00:33:55: SSH2 1: send: len 16 (includes padlen 10)
00:33:55: SSH2 1: newkeys: mode 1
00:33:55: SSH2 1: SSH2_MSG_NEWKEYS sent
00:33:55: SSH2 1: waiting for SSH2_MSG_NEWKEYS
00:33:55: SSH2 1: ssh_receive: 16 bytes received
00:33:55: SSH2 1: input: packet len 16
00:33:55: SSH2 1: partial packet 8, need 8, maclen 0
00:33:55: SSH2 1: input: padlen 10
00:33:55: SSH2 1: newkeys: mode 0
00:33:55: SSH2 1: received packet type 2100:33:55: SSH2 1: SSH2_MSG_NEWKEYS received
00:33:56: SSH2 1: ssh_receive: 48 bytes received
00:33:56: SSH2 1: input: packet len 32
00:33:56: SSH2 1: partial packet 16, need 16, maclen 16
00:33:56: SSH2 1: MAC #3 ok
00:33:56: SSH2 1: input: padlen 10
00:33:56: SSH2 1: received packet type 5
00:33:56: SSH2 1: send: len 32 (includes padlen 10)
00:33:56: SSH2 1: done calc MAC out #3
00:33:56: SSH2 1: ssh_receive: 64 bytes received
```

```
00:33:56: SSH2 1: input: packet len 48
00:33:56: SSH2 1: partial packet 16, need 32, maclen 16
00:33:56: SSH2 1: MAC #4 ok
00:33:56: SSH2 1: input: padlen 9
00:33:56: SSH2 1: received packet type 50
00:33:56: SSH2 1: send: len 32 (includes padlen 13)
00:33:56: SSH2 1: done calc MAC out #4
00:34:04: SSH2 1: ssh_receive: 160 bytes received
00:34:04: SSH2 1: input: packet len 64
00:34:04: SSH2 1: partial packet 16, need 48, maclen 16
00:34:04: SSH2 1: MAC #5 ok
00:34:04: SSH2 1: input: padlen 13
00:34:04: SSH2 1: received packet type 50
00:34:04: SSH2 1: send: len 16 (includes padlen 10)
00:34:04: SSH2 1: done calc MAC out #5
00:34:04: SSH2 1: authentication successful for lab
00:34:04: SSH2 1: input: packet len 64
00:34:04: SSH2 1: partial packet 16, need 48, maclen 16
00:34:04: SSH2 1: MAC #6 ok
00:34:04: SSH2 1: input: padlen 6
00:34:04: SSH2 1: received packet type 2
00:34:04: SSH2 1: ssh_receive: 64 bytes received
00:34:04: SSH2 1: input: packet len 48
00:34:04: SSH2 1: partial packet 16, need 32, maclen 16
00:34:04: SSH2 1: MAC #7 ok
00:34:04: SSH2 1: input: padlen 19
00:34:04: SSH2 1: received packet type 90
00:34:04: SSH2 1: channel open request
00:34:04: SSH2 1: send: len 32 (includes padlen 10)
00:34:04: SSH2 1: done calc MAC out #6
00:34:04: SSH2 1: ssh_receive: 192 bytes received
00:34:04: SSH2 1: input: packet len 64
00:34:04: SSH2 1: partial packet 16, need 48, maclen 16
00:34:04: SSH2 1: MAC #8 ok
00:34:04: SSH2 1: input: padlen 13
00:34:04: SSH2 1: received packet type 98
00:34:04: SSH2 1: pty-req request
00:34:04: SSH2 1: setting TTY - requested: height 24, width 80; set: height 24,
width 80
00:34:04: SSH2 1: input: packet len 96
00:34:04: SSH2 1: partial packet 16, need 80, maclen 16
00:34:04: SSH2 1: MAC #9 ok
00:34:04: SSH2 1: input: padlen 11
00:34:04: SSH2 1: received packet type 98
00:34:04: SSH2 1: x11-req request
00:34:04: SSH2 1: ssh_receive: 48 bytes received
00:34:04: SSH2 1: input: packet len 32
00:34:04: SSH2 1: partial packet 16, need 16, maclen 16
00:34:04: SSH2 1: MAC #10 ok
00:34:04: SSH2 1: input: padlen 12
00:34:04: SSH2 1: received packet type 98
00:34:04: SSH2 1: shell request
00:34:04: SSH2 1: shell message received
00:34:04: SSH2 1: starting shell for vty
00:34:04: SSH2 1: send: len 48 (includes padlen 18)
00:34:04: SSH2 1: done calc MAC out #7
00:34:07: SSH2 1: ssh_receive: 48 bytes received
00:34:07: SSH2 1: input: packet len 32
00:34:07: SSH2 1: partial packet 16, need 16, maclen 16
00:34:07: SSH2 1: MAC #11 ok
00:34:07: SSH2 1: input: padlen 17
00:34:07: SSH2 1: received packet type 94
00:34:07: SSH2 1: send: len 32 (includes padlen 17)
00:34:07: SSH2 1: done calc MAC out #8
```

```
00:34:07: SSH2 1: ssh_receive: 48 bytes received
00:34:07: SSH2 1: input: packet len 32
00:34:07: SSH2 1: partial packet 16, need 16, maclen 16
00:34:07: SSH2 1: MAC #12 ok
00:34:07: SSH2 1: input: padlen 17
00:34:07: SSH2 1: received packet type 94
00:34:07: SSH2 1: send: len 32 (includes padlen 17)
00:34:07: SSH2 1: done calc MAC out #9
00:34:07: SSH2 1: ssh_receive: 48 bytes received
00:34:07: SSH2 1: input: packet len 32
00:34:07: SSH2 1: partial packet 16, need 16, maclen 16
00:34:07: SSH2 1: MAC #13 ok
00:34:07: SSH2 1: input: padlen 17
00:34:07: SSH2 1: received packet type 94
00:34:07: SSH2 1: send: len 32 (includes padlen 17)
00:34:07: SSH2 1: done calc MAC out #10
00:34:08: SSH2 1: ssh_receive: 48 bytes received
00:34:08: SSH2 1: input: packet len 32
00:34:08: SSH2 1: partial packet 16, need 16, maclen 16
00:34:08: SSH2 1: MAC #14 ok
00:34:08: SSH2 1: input: padlen 17
00:34:08: SSH2 1: received packet type 94
00:34:08: SSH2 1: send: len 32 (includes padlen 17)
00:34:08: SSH2 1: done calc MAC out #11
00:34:08: SSH2 1: ssh_receive: 48 bytes received
00:34:08: SSH2 1: input: packet len 32
00:34:08: SSH2 1: partial packet 16, need 16, maclen 16
00:34:08: SSH2 1: MAC #15 ok
00:34:08: SSH2 1: input: padlen 17
00:34:08: SSH2 1: received packet type 94
00:34:08: SSH2 1: send: len 32 (includes padlen 16)
00:34:08: SSH2 1: done calc MAC out #12
00:34:08: SSH2 1: send: len 48 (includes padlen 18)
00:34:08: SSH2 1: done calc MAC out #13
00:34:08: SSH2 1: send: len 16 (includes padlen 6)
00:34:08: SSH2 1: done calc MAC out #14
00:34:08: SSH2 1: send: len 16 (includes padlen 6)
00:34:08: SSH2 1: done calc MAC out #15
00:34:08: SSH1: Session terminated normally
```

Configuration Examples for Secure Shell Version 2

Example: Configuring Secure Shell Version 2

```
Device> enable
Device# configure terminal
Device(config)# ip ssh version 2
Device(config)# end
```

Example: Configuring Secure Shell Versions 1 and 2

```
Device> enable
Device# configure terminal
Device(config)# no ip ssh version
Device(config)# end
```

Example: Starting an Encrypted Session with a Remote Device

```
Device> enable
Device# ssh -v 2 -c aes256-cbc -m hmac-sha1-160 -l shaship 10.76.82.24
Device# exit
```

Example: Setting an SNMP Trap

The following example shows how to set an SNMP trap is set. The trap notification is generated automatically when the SSH session terminates. In the example, 10.1.1.1 is the IP address of the SSH client.

```
Device> enable
Device# configure terminal
Device(config)# snmp-server trap link switchover
Device(config)# snmp-server host 10.1.1.1 public tty
Device(config)# end
```

Example: SNMP Debugging

The following is sample output from the **debug snmp packet** command. The output provides SNMP trap information for an SSH session.

```
Device1# debug snmp packet

SNMP packet debugging is on
Device1# ssh -l lab 10.0.0.2
Password:

Device2# exit

[Connection to 10.0.0.2 closed by foreign host]
Device1#
*Jul 18 10:18:42.619: SNMP: Queuing packet to 10.0.0.2
*Jul 18 10:18:42.619: SNMP: V1 Trap, ent cisco, addr 10.0.0.1, gentrap 6, spectrap 1
local.9.3.1.1.2.1 = 6
tcpConnEntry.1.10.0.0.1.22.10.0.0.2.55246 = 4
ltcpConnEntry.5.10.0.0.1.22.10.0.0.2.55246 = 1015
ltcpConnEntry.1.10.0.0.1.22.10.0.0.2.55246 = 1056
ltcpConnEntry.2.10.0.0.1.22.10.0.0.2.55246 = 1392
local.9.2.1.18.2 = lab
*Jul 18 10:18:42.879: SNMP: Packet sent via UDP to 10.0.0.2

Device1#
```

Examples: SSH Debugging Enhancements

The following is sample output from the **debug ip ssh detail** command. The output provides debugging information about the SSH protocol and channel requests.

```
Device# debug ip ssh detail
Cisco Confidential
00:04:22: SSH0: starting SSH control process
00:04:22: SSH0: sent protocol version id SSH-1.99-Cisco-1.25
00:04:22: SSH0: protocol version id is - SSH-1.99-Cisco-1.25
00:04:22: SSH2 0: SSH2_MSG_KEXINIT sent
00:04:22: SSH2 0: SSH2_MSG_KEXINIT received
00:04:22: SSH2:kex: client->server enc:aes128-cbc mac:hmac-sha1
00:04:22: SSH2:kex: server->client enc:aes128-cbc mac:hmac-sha1
00:04:22: SSH2 0: expecting SSH2_MSG_KEXDH_INIT
```

```
00:04:22: SSH2 0: SSH2_MSG_KEXDH_INIT received
00:04:22: SSH2: kex_derive_keys complete
00:04:22: SSH2 0: SSH2_MSG_NEWKEYS sent
00:04:22: SSH2 0: waiting for SSH2_MSG_NEWKEYS
00:04:22: SSH2 0: SSH2_MSG_NEWKEYS received
00:04:24: SSH2 0: authentication successful for lab
00:04:24: SSH2 0: channel open request
00:04:24: SSH2 0: pty-req request
00:04:24: SSH2 0: setting TTY - requested: height 24, width 80; set: height 24, width 80
00:04:24: SSH2 0: shell request
00:04:24: SSH2 0: shell message received
00:04:24: SSH2 0: starting shell for vty
00:04:38: SSH0: Session terminated normally
```

The following is sample output from the **debug ip ssh packet** command. The output provides debugging information about the SSH packet.

```
Device# debug ip ssh packet

00:05:43: SSH2 0: send:packet of length 280 (length also includes padlen of 4)
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: input: total packet length of 280 bytes
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 272 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 272 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 272 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 272 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 24 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 272 bytes, maclen 0
00:05:43: SSH2 0: input: padlength 4 bytes
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: input: total packet length of 144 bytes
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 136 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 64 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 136 bytes, maclen 0
00:05:43: SSH2 0: ssh_receive: 16 bytes received
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 136 bytes, maclen 0
00:05:43: SSH2 0: input: padlength 6 bytes
00:05:43: SSH2 0: signature length 143
00:05:43: SSH2 0: send:packet of length 448 (length also includes padlen of 7)
00:05:43: SSH2 0: send:packet of length 16 (length also includes padlen of 10)
00:05:43: SSH2 0: newkeys: mode 1
00:05:43: SSH2 0: ssh_receive: 16 bytes received
00:05:43: SSH2 0: input: total packet length of 16 bytes
00:05:43: SSH2 0: partial packet length(block size)8 bytes,needed 8 bytes, maclen 0
00:05:43: SSH2 0: input: padlength 10 bytes
00:05:43: SSH2 0: newkeys: mode 0
00:05:43: SSH2 0: ssh_receive: 52 bytes received
00:05:43: SSH2 0: input: total packet length of 32 bytes
00:05:43: SSH2 0: partial packet length(block size)16 bytes,needed 16 bytes, maclen 20
00:05:43: SSH2 0: MAC compared for #3 :ok
```