

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 Ed25519 keys. Cisco software supports both SSHv1 and SSHv2.

This module describes how to implement Secure Shell.

Note Any reference to CiscoSSH in this chapter implies OpenSSH-based implementation of SSH that is available on some of the Cisco IOS XR router variants from Cisco IOS XR Software Release 7.3.2 and later. CiscoSSH replaces Cisco IOS XR SSH, which is the older SSH implementation that existed prior to this release.

For more information, see the *CiscoSSH* section later in this chapter.

- Implementing Secure Shell, on page 1
- Selective Authentication Methods for SSH Server, on page 49
- SSH Port Forwarding, on page 51
- Non-Default SSH Port, on page 54
- Public Key-Based Authentication of SSH Clients, on page 59
- Public key-based Authentication to SSH Server on Routers, on page 63
- Multi-Factor Authentication for SSH, on page 68
- DSCP Marking for SSH Packets, on page 73
- SSH server timeouts, on page 75

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 Ed25519 keys. Cisco software supports both SSHv1 and SSHv2.

This module describes how to implement Secure Shell.

Note Any reference to CiscoSSH in this chapter implies OpenSSH-based implementation of SSH that is available on some of the Cisco IOS XR router variants from Cisco IOS XR Software Release 7.3.2 and later. CiscoSSH replaces Cisco IOS XR SSH, which is the older SSH implementation that existed prior to this release.

For more information, see the *CiscoSSH* section later in this chapter.

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
```

To correctly import the public key into the chassis, the public key must be in one of the following formats:

- SSH RSA Public key format
- Base64 encoded (binary) format
- PEM PKCS1 format
- PEM PKCS8 format

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.

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 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 6.

- 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 and SFTP. However, 802.1X protocol (Port-Based Network Access Control) and data plane components 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.

Note This feature is not supported on the following variants of Cisco NCS 540 Series Routers:

- N540-28Z4C-SYS-A
- N540-28Z4C-SYS-D
- N540X-16Z4G8Q2C-A
- N540X-16Z4G8Q2C-D
- N540-12Z20G-SYS-A
- N540-12Z20G-SYS-D
- N540X-12Z16G-SYS-A
- N540X-12Z16G-SYS-D

CiscoSSH

CiscoSSH is a secure shell implementation that

- leverages the key features of openSSH such as strong authentication, cryptography, encryption, port forwarding, and so on to provide secured management access to the router
- provides additional securityenhancements including FIPS compliance and X.509 digital certificate support, and
- ensures backward compatibility for legacy Cisco IOS XR SSH configuration and management.

CiscoSSH is based on OpenSSH, which is a stable, widely deployed open-source SSH protocol implementation. It uses the Linux TCP/IP stack for transmitting SSH packets over management Ethernet and router line card interfaces. CiscoSSH offers interoperability with existing SSH implementations, supports management plane packet path features like MPP, ACLs, and VRF, and retains the majority of commands and functional behavior from prior Cisco IOS XR SSH implementations.

For more information on SSH in general, see Information About Implementing Secure Shell, on page 2 and Configure SSH, on page 20.

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Post-Quantum Cryptography key exchange for CiscoSSH	Release 26.2.1	CiscoSSH mitigates quantum threats to ensure long term confidentiality and integrity of remote access and data transfer by introducing PQC (Post-Quantum Cryptography) hybrid key exchange algorithms. Supported algorithms include ML-KEM and NTRU Prime hybrids: • `mlkem768x25519-sha256` • `sntrup761x25519-sha512` • `sntrup761x25519-sha512@openssh.com`
CiscoSSH	Release 7.3.2	This release introduces CiscoSSH, a newer implementation of SSH on NCS 540 Small Density, NCS 540 Large Density, NCS 540 Medium Density (except N540-ACC-SYS, NC540X-ACC-SYS, and N540-24Z8Q2C-SYS). CiscoSSH leverages OpenSSH implementation, by using the Linux TCP/IP stack to transmit and receive SSH packets over the management Ethernet interface and line card interfaces on the router. CiscoSSH provides additional security features like FIPS compliance and X.509 digital certification. It supports packet path features like MPP, ACL and VRF support, and ensures interoperability with various existing SSH implementations. **Note** Cisco IOS XR SSH, the SSH implementation that existed prior to this release, is now deprecated on all variants of NCS 540 except for N540-ACC-SYS, NC540X-ACC-SYS, and N540-24Z8Q2C-SYS.

Note CiscoSSH is *not* supported on the following Cisco NCS 540 router variants:

- N540-ACC-SYS
- N540X-ACC-SYS (Premium)
- N540-24Z8Q2C-SYS

Note Any reference to CiscoSSH in this chapter implies OpenSSH-based implementation of SSH that is available on this platform from Cisco IOS XR Software Release 7.3.2 and later. CiscoSSH replaces Cisco IOS XR SSH, which is the older SSH implementation that existed prior to this release.

Post-Quantum Cryptography key exchange for CiscoSSH

Cisco IOS XR Release 26.2.1 adds PQC (Post-Quantum Cryptography) in key exchange (KEX) and introduces PQC algorithms, which combine a post-quantum algorithm and a classical algorithm for hybrid security that addresses the long-term quantum risk to current public-key cryptography.

PQC key exchange is enabled by default. You can configure PQC key exchange on both SSH server and SSH client using these existing key exchange commands:

- **ssh server algorithms key-exchange** <*algorithm list*>
- **ssh client algorithms key-exchange** <*algorithm list*>

Supported PQC algorithms

This table lists the supported PQC algorithms.

Table 2: PQC algorithms

PQC algorithm	Description
`mlkem768x25519-sha256`	A combination ML-KEM (Module-Lattice-based Key Encapsulation Mechanism), a NIST-standardized post-quantum algorithm, with the classical X25519 elliptic curve Diffie-Hellman key exchange.
`sntrup761x25519-sha512`	A combination of Streamlined NTRU Prime, a lattice-based post-quantum algorithm, with the classical X25519 elliptic curve Diffie-Hellman key exchange.
`sntrup761x25519-sha512@openssh.com`	An older name for sntrup761x25519-sha512. Used for compatibility with peers expecting the old OpenSSH-style identifier.

Most SSH commands and configurations remain unchanged between CiscoSSH and Cisco IOS XR SSH, though some behavioral differences exist.

Restrictions for CiscoSSH

The following restrictions apply to CiscoSSH:

Note Ensure you do not attempt any of the following, as they are unsupported and may cause issues.

- Do not use SSH version 1.
- Do not configure backup SSH servers; backup SSH servers are not supported.
- Do not use secondary IPv4 addresses; these addresses are not synchronized to Linux; they are not currently synchronized to Linux prior to Cisco IOS XR Software Release 7.7.1.
- Do not use BVI interfaces as source or destination for the SSH connections prior to Cisco IOS XR Software Release 7.7.1.
- Do not use the following cipher algorithms: support these algorithms:
  - *aes128-cbc*, *aes192-cbc*, *aes256-cbc*, and *3des-cbc*
- Do not use the key-exchange algorithm, *diffie-hellman-group1-sha1*.
- Do not attempt to use these commands, as they are unsupported:
  - **show ssh history**
  - **show ssh history details**
  - **clear ssh stale sessions**
- If ingress ACLs are configured only under the management interface and not under the **ssh server** configuration mode, those ACLs don not impact SSH or Netconf traffic.

  This restriction applies only to ingress ACLs attached to management interfaces.

Guidelines for Using CiscoSSH

The following section lists certain functionality aspects and guidelines for using CiscoSSH.

- **Netconf Request**: You must follow a specific syntax when you send Netconf request over CLI. Add the subsystem (*netconf* or *sftp*) name as the last argument while issuing an SSH command.

  For example,

  ```
  ssh username@ipaddress -p 830 -s netconf ---> Correct usage
  ssh username@ipaddress netconf -p 830 -s ---> Incorrect usage
  ```

- **Configuring unsupported algorithms**: Configuring CiscoSSH server only with unsupported algorithms (*3des-cbc* or *diffie-hellman-group1-sha1*) results in commit failure. Hence, you must remove such configurations on your router as a part of the pre-upgrade procedure.

  For example,

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

```
!!% Operation not permitted: 3des-cbc is not supported in ciscossh, SSH cannot work
with this option only
```

Similarly, if you configure CiscoSSH server with both supported and unsupported algorithms, then the router issues the following warning and removes the unsupported algorithm:

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

ssh_conf_proxy[1193]: %SECURITY-SSHD_CONF_PRX-3-ERR_GENERAL : 3des-cbc is not supported,
 will be removed
```

- **SSH session keep alive:** By default, the SSH session keep alive functionality is enabled in CiscoSSH, to detect and terminate unresponsive sessions. The default keep alive time is 60 seconds, with a maximum of three attempts allowed, so that the detection time for unresponsive sessions is 180 seconds. These keep alive parameters are not configurable.
- **TCP window scale**: Although the router accepts the configuration to change the TCP window scale parameter, the configuration does not have any effect with CiscoSSH. This is because, CiscoSSH uses Linux TCP/IP stack that has dynamic window scaling, and hence it does not require applications to specify the window scale.
- **SSH session limit and rate limit**: Although the configuration for SSH session limit and rate limit applies to all VRFs where SSH is enabled, the router enforces the limit for each VRF. However, the maximum number of virtual teletype (VTY) sessions across all VRFs still remains as 200. This in turn limits the total number of SSH sessions that require a VTY interface, across all VRFs. As a result, when upgrading from a release version having Cisco IOS XR SSH to a version having CiscoSSH, the system applies the session limit and rate limit configurations to all VRFs where SSH is enabled. Hence, as part of the post-upgrade procedure, you must reconfigure these limits to achieve the same limit as that of Cisco IOS XR SSH.
- **SSH session limit enforcement**: Information on the number of active SSH sessions on the router is not persistent across SSH server process restarts. Hence, SSH session limit enforcement does not consider the existing sessions after an SSH server restart.
- **SSH with ACL or MPP configuration**: With SSH ACL or MPP configured on the router, the attempt for client connection that is not allowed as per that configuration times out. The router does not send TCP reset for such blocked SSH connections. This implementation is to enhance security.
- **Ingress ACL:** To filter out the ingress SSH and Netconf traffic, we recommend to configure the ingress ACL under the **ssh server** configuration mode instead of configuring under the management interface.

  For SSH:

  ssh server vrf *vrf-name* ipv4 access-list *ipv4-access-list-name* ipv6 access-list *ipv6-access-list-name*

  For Netconf:

  ssh server netconf vrf *vrf-name* ipv4 access-list *ipv4-access-list-name* ipv6 access-list *ipv6-access-list-name*
- **Default VRFs**: Configuring the default SSH VRF using the **ssh vrf default** command enables only version 2 of CiscoSSH, because version 1 is not supported.
- **Non-default VRFs**: If SSH service is enabled on any of the non-default VRFs that is configured on the router, and if you restart the *ssh_conf_proxy* process, there might be a delay in allowing incoming SSH sessions on that non-default VRF. The session establishment might even timeout in such a scenario. This behavior is due to the delay in programming the LPTS entries for those sessions.
- **Public key-based authentication**: In CiscoSSH, the router negotiates public key-based authentication even if there is no public key imported on to the router. So, the authentication attempt from the client

using public key fails in such scenarios. The router displays a syslog on the console for this authentication failure. However, the client and server proceed with subsequent authentication methods like keyboard-interactive and password methods. If the router does not have a public key imported, you may choose to disable public key-based authentication from the client side. For details on public key-based authentication, see the *Implementing Certification Authority Interoperability* chapter in this guide.

- **Modifying SSH configuration**: Any change to the SSH configuration results in process restart of SSH server process. However, it does not impact the existing SSH, SCP, SFTP, or Netconf sessions.
- **Clearing SSH sessions**: The **clear ssh all** command clears all incoming sessions.
- **Line-feed option**: Adding a line-feed option for Gossh-based clients results in SSH session establishment failure. This is because, the SSH client checks for non-zero window size for session establishment. Whereas CiscoSSH sends window size as 0. The workaround for this issue is to use the option to ignore the window size while initiating an SSH connection from such clients.
- **Virtual IP addresses**: After a process restart of *xlncd* or *ip_smiap*, there might be a delay in restoring the virtual IP addresses.
- **More-specific Routes**: Routes that are more specific than a connected route will not be available through Linux.

  For example:

```
XR routing table:

10.0.0.0/24   via 10.0.0.2 (connected route)
10.0.0.192/28 via 20.0.0.1 (static route)
```

  The expected behavior is as follows:

Table 3: Expected Behavior of More-specific Routes with CiscoSSH

Destination IP Range	Cisco IOS XR OS Sends to:	Linux Sends to:	Match (Yes/No)
10.0.0.1 - 10.0.0.191	10.0.0.2	10.0.0.2	Yes
10.0.0.193 - 10.0.0.206	20.0.0.1	10.0.0.2	No
10.0.0.207 - 10.0.0.255	10.0.0.2	10.0.0.2	Yes

- **Verification commands**: During stress test on the router, certain show commands like **show ssh**, **show ssh session details**, and **show ssh rekey** might time out. The console displays the following error message in such cases:

```
"Error: Timed out to obtain information about one or more incoming/outgoing session.
please retry."
```

- **Process restart**:
  - You cannot restart the CiscoSSH server process using the **process restart ssh_server** command, because it is a Linux process. Use the **kill** command on the Linux shell to restart the process.

- CiscoSSH has *ssh_conf_proxy* and *ssh_syslog_proxy* processes that are responsible for processing the SSH configuration and logging syslog messages respectively. You can restart these processes using the **process restart** command.
- A restart of *XR-TCP* process does not have any impact on CiscoSSH functionality, because CiscoSSH uses Linux TCP.

- **Debuggability**:
  - You can enable 3 levels of debugs for CiscoSSH using the **debug ssh server l1/l2/l3** command. Similarly, you can use the **debug ssh client l1/l2/l3** command for CiscoSSH client.
  - The SSH server process restarts every time you enable or disable the debugs, because enabling the debugs results in updating the LOGLEVEL in the internal *sshd_config* file.

Syslog message for CiscoSSH and Cisco IOS XR SSH session events

CiscoSSH introduces new syslogs for key SSH session events. The following table compares session event syslog messages between CiscoSSH and Cisco IOS XR SSH:

Table 4: Syslogs for CiscoSSH and Cisco IOS XR SSH

Session Event	Syslogs on CiscoSSH	Syslogs on Cisco IOS XR SSH
Session login	RP/0/RP0/CPU0:Sep 22 11:06:33.467 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[32504]: Accepted authentication/pam for admin from 203.0.113.1 port 62015 ssh2 RP/0/RP0/CPU0:Sep 22 11:06:33.472 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[32504]: User child is on pid 32564 RP/0/RP0/CPU0:Sep 22 11:06:33.519 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[32564]: Starting session: shell on pts/1 for admin from 203.0.113.1 port 62015 id 0	RP/0/RP0/CPU0:Sep 22 11:46:13.475 IST: SSHD_[67274]: %SECURITY-SSHD-6-INFO_SUCCESS : Successfully authenticated user 'root' from '192.0.2.1' on 'vty0'(cipher 'aes128-ctr', mac 'hmac-sha2-256')

Session Event	Syslogs on CiscoSSH	Syslogs on Cisco IOS XR SSH
Session logout	RP/0/RP0/CPU0:Sep 22 11:11:27.394 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[32564]: Received disconnect from 203.0.113.1 port 62015:11: disconnected by user RP/0/RP0/CPU0:Sep 22 11:11:27.394 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[32564]: Disconnected from user admin 203.0.113.1 port 62015	RP/0/RP0/CPU0:Sep 22 11:46:48.439 IST: SSHD_[67274]: %SECURITY-SSHD-6-INFO_USER_LOGOUT : User 'root' from '192.0.2.1' logged out on 'vty0'
Session login failure	RP/0/RP0/CPU0:Sep 22 19:47:06.211 IST: ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[31103]: Failed authentication/pam for admin from 203.0.113.1 port 60189 ssh2	RP/0/RP0/CPU0:Sep 22 11:47:55.909 IST: SSHD_[67369]: %SECURITY-SSHD-4-INFO_FAILURE : Failed authentication attempt by user 'root' from '192.0.2.1' on 'vty0'
Session rekey	ssh_syslog_proxy[1204]: %SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[24919]: Server initiated time rekey for session=21, session_rekey_count =1	RP/0/RP0/CPU0:Sep 22 19:07:45.435 IST: SSHD_[65640]: %SECURITY-SSHD-6-INFO_REKEY : Server initiated time rekey for session 4 , session_rekey_count = 1

FIDO2 authentication for SSH

A FIDO2 authentication for SSH is an open authentication standard that

- enables secure, passwordless SSH logins and multi-factor authentication on Cisco IOS XR devices,
- uses hardware security keys to store private keys, and
- requires user presence such as touching the key to complete the login process.

Unlike typical SSH key pair generation, where both keys are stored on the ssh client, with FIDO2 the private key remains only on the hardware device. The public key algorithms commonly used are `ed25519-sk` and `ecdsa-sk`.

In SSH, public key authentication works by storing the public key on the host device. The SSH client sends the public key during authentication, and the server verifies if the key exists in the host database for that user. Passwordless authentication is only permitted if the keys are registered.

Table 5: Feature History Table

Feature Name	Release Information	Feature Description
FIDO2 authentication for SSH	Release 26.2.1	FIDO2 support for SSH enables secure, passwordless logins by using hardware security keys to store private keys. This feature requires physical user presence to authenticate sessions and protects against phishing by verifying a signature from the security device during the login process.

Restrictions for configuring FIDO2 authentication for SSH

- Remote authentication using TACACS+ or RADIUS is not supported for FIDO2 keys.
- FIDO2 public keys cannot be stored in cepki sysdb.
- XRSSH doesn't support FIDO2 authentication.
- Only ed25519-sk and ecdsa-sk key types are supported
- Automation of full authentication flow is limited due to mandatory user presence.
- FIDO2 keys cannot be imported using `crypto key import authentication`.

Guidelines for configuring FIDO support for SSH

- FIDO2 authentication is supported only for local SSH public key authentication.
- Public keys must be stored in the authorized_keys file.
- Use the existing ssh-server authorized-keys CLI to import FIDO2 public keys.
- The private key never leaves the FIDO2 device.
- User presence (touch/biometric) is mandatory during SSH authentication.
- Multiple FIDO2 public keys can be imported for a single user.

Configure FIDO2 authentication for SSH

Set up FIDO2-based SSH authentication on Cisco IOS XR routers, enabling secure logins using a hardware security key such as a YubiKey.

FIDO2 authentication strengthens device access security by requiring both the user’s credentials and physical presence of a hardware-backed authenticator. This protects against phishing attacks and credential theft. Use this method when you need to enhance SSH security for administrators or automation systems accessing Cisco IOS XR routers, or when organizational policy mandates strong, multi-factor authentication.

Before you begin

- Ensure you have a supported client - macOS, Windows, or any OS running OpenSSH 8.2+.
- Obtain a FIDO2-compatible device such as a YubiKey.
- Access privileges on the target Cisco IOS XR router.

Procedure

Step 1 Connect your FIDO2 device to the client machine.

Plug in your YubiKey to your client (macOS, Windows, or supported system).

Step 2 Generate FIDO2 key pairs on your client.

Example:

```
ssh-keygen -t ecdsa-sk -f id_ecdsa_sk
```

a) In your terminal, run **ssh-keygen -t ecdsa-sk -f id_ecdsa_sk** command.
b) When prompted, touch your authenticator of YubiKey to authorize key generation.
c) Enter a passphrase for the key if desired, then confirm the passphrase.
d) A public key (`id_ecdsa_sk.pub`) and private key will be generated.

Step 3 Import your public key to the IOS XR router.

a) Copy the contents of `id_ecdsa_sk.pub` from your client.
b) On the router, enter EXEC mode and run the **ssh-server authorized-keys** command.
c) If you do not provide a username, the key is imported for the current logged-in user.
d) Paste the public key when prompted.

Step 4 Verify the imported SSH key on the IOS XR router.

Example:

```
Router#show ssh server authorized-keys user <username>
```

a) Confirm your key appears in the authorized keys list.

Step 5 Test SSH access using the FIDO2 key.

Example:

```
Router#show ssh
SSH version : Cisco-2.0
id       chan pty     location        state           userid    host                  ver
authentication connection type
-----------------------------------------------------------------------------------------
Incoming sessions
974       0   vty1    0/RP0/CPU0      SESSION_OPEN    cafyauto  10.189.219.24         v2
sk-ecdsa-sha2-nistp256@openssh Command-Line-Interface
```

a) On your client, use the command:

Example:

```
ssh -i id_ecdsa_sk <username>@<router_ip>
```

b) When prompted, touch your YubiKey to confirm presence.
c) Upon successful authentication, you will see a login confirmation.

Step 6 (Optional) Import multiple public keys for a user.

You can import additional keys using step 3 for a specific username; the behavior remains as with previous (non-FIDO) public keys.

You have successfully configured FIDO-based SSH authentication for enhanced security. SSH connections now require your FIDO device for login.

Netconf access controls

A Netconf access control is a security mechanism that

- blocks Netconf requests on the SSH port (default 22) and allows only on the designated Netconf port (default 830)
- restricts Netconf access based on specified IP addresses using Access Control Lists (ACLs), and
- allows other SSH services such as Secure Copy Protocol (SCP) and Secure File Transfer Protocol (SFTP) to continue functioning.

Table 6: Feature history table

Cisco IOS XR router uses Netconf over SSH. By default, SSH uses port 22 and Netconf uses port 830. Netconf requests are permitted on the SSH port by default.

Without this feature enabled, a NETCONF connection can be established over the SSH port even from addresses that are not permitted by the ACLs.

Benefits of Netconf access control

This feature provides these benefits:

- You can block Netconf for specific IP addresses while still allowing SSH access for those addresses.
- You can prevent unauthorized Netconf access on the SSH port.
- You retain access to other SSH services such as SCP and SFTP.
- You enforce Netconf access restrictions without disrupting SSH access.

Best practice for Netconf access control

- Use ACLs to specify IP addresses that you block for Netconf.
- Apply the **ssh server netconf disable ssh-port** to disable Netconf on the SSH port.

Restrictions for Netconf access control

- During the initial TCP handshake and authentication, you cannot identify the service from the SSH protocol. At the channel request stage, the server checks for a Netconf request and then rejects your connection request.
- Because the SSH server does not identify the request type until the channel request phase, it cannot reject a Netconf connection on the SSH port by IP address during the initial connection. If an ACL blocks an IP address on the Netconf port, the connection attempt fails during the TCP handshake, and you cannot connect.
- Netconf access control blocks Netconf only on the SSH port for your device, but it does not block SCP or SFTP for the same IP address.

How Netconf access control works

Summary

Use Netconf access control to configure access to Netconf services through the SSH server and secure your device.

The components involved in the process are:

- **SSH server**: Manages incoming SSH connections and enforces access control.
- **SSH port**: Network port used for SSH and Netconf services.
- **Netconf service**: Provides configuration and management operations using the Netconf protocol.
- **Access Control Lists (ACLs)**: These restrict access to Netconf based on IP addresses.
- **Users and clients**: They attempt to access device configuration using Netconf.

Workflow

The SSH server enforces Netconf restrictions on the SSH port using this process:

1. The administrator configures the SSH server to disable Netconf access on the SSH port using the **ssh server netconf disable ssh-port** command.
2. When the server receives a NETCONF request, it verifies that the request is received on a port configured for SSH.
3. After the SSH session is established, the server checks the requested service type.
4. If NETCONF is requested on an SSH port where NETCONF has been disabled through the CLI, the server rejects the request and terminates the session.
5. SSH continues to allow other services for the same IP address, including SCP and SFTP.
6. This behavior runs in XR SSH and CiscoSSH components.

Result

Your configuration restricts Netconf access and allows permitted SSH functions.

Configure Netconf access control

Restrict Netconf access over SSH to enhance device security and prevent unauthorized Netconf sessions.

Netconf is a network management protocol that operates over SSH. You may need to disable Netconf on the SSH port to harden device security or meet specific compliance requirements.

Before you begin

- Ensure you have administrative access to the device.
- Back up the current configuration before you make changes.
- Confirm that disabling Netconf over SSH will not interrupt management tasks if you use Netconf.

Follow these steps to configure Netconf access control:

Procedure

Step 1 Enter configuration mode.

Example:

```
Router#config
```

Step 2 Disable Netconf on the SSH port:

```
Router#ssh server netconf disable ssh-port
```

Note
Use the **ssh server netconf disable ssh-port** configuration for any configured SSH port (default port 22 or any custom port).

Step 3 Commit the configuration.

Example:

```
Router#commit
```

Syslogs for Netconf access control

You can observe these SYSLOGS on the router console:

```
RP/0/RP0/CPU0:Nov 20 07:55:09.476 UTC: ssh_syslog_proxy[1221]:
%SECURITY-SSHD_SYSLOG_PRX-3-ERR_GENERAL : sshd[33048]: On ssh port invalid netconf channel
 request received

RP/0/RP0/CPU0:Nov 20 07:55:09.476 UTC: ssh_syslog_proxy[1221]:
%SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[33048]: subsystem request for netconf by
user cafyauto failed, subsystem not found
```

You have disabled Netconf access over SSH port. The device does not accept Netconf management sessions on the SSH port.

Guidelines and Restrictions for Implementing Secure Shell

The following are some basic SSH guidelines, 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) key pair configured. 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: DSA and RSA keys are not required because SFTP operates only in client mode.
- If you delete all the default crypto keys (the keys with **the_default** label) on the router, the SSH clients cannot establish sessions with the router. Hence, for clients to successfully establish SSH sessions with the router, ensure that at least one default crypto key is always present on the router. In FIPS mode, it is mandatory to have at least one default crypto key of type RSA or ECDSA.
- For SSH sessions, the router supports key-exchange algorithms (**diffie-hellman-group1-sha1** and **curve25519**) and cipher algorithms (**3des-cbc** and **chacha20-poly1305@openssh.com**) only in non-FIPS mode. For routers supporting open source-based CiscoSSH, the SSH session fails to connect if any of

these algorithms is pre-configured prior to enabling FIPS mode. Whereas for routers supporting Cisco IOS XR SSH, the SSH session continues to connect in such scenarios.

Starting Cisco IOS XR Software Release 24.2.2, 24.3.2, and 24.4.1 and later, the SSH session fails to connect in such scenarios where these algorithms are pre-configured even for routers supporting Cisco IOS XR SSH.

- 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 explicilty. See, #unique_296 for details.

Procedure

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 configure

Example:

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

Enters mode.

Step 8 **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 9 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 10 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 11 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 12 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.

SSH client strict host key checks

A SSH client strict host key checking is a feature that

- enforces validation of server host keys using a system-wide `known_hosts` file,
- restricts SSH client connections to hosts with trusted or previously accepted host keys, and
- supports configurable policies for new or changed keys, such as off, accept-new, ask, or yes.

The knownhosts file is located at `/mnt/rdsfs/ciscossh/known_hosts` and is shared across all users, providing persistent, centralized host key management for SSH client operations.

Table 7: Feature History Table

Feature Name	Release Information	Feature Description
SSH client strict host key checking	Release 26.3.1	You enhance SSH security by enforcing strict host key checking, allowing you to control how the SSH client handles new or changed server keys - accept, reject, or prompt for approval. Trusted host keys are stored system-wide and persist across reloads, ensuring consistent validation for your outbound SSH connections.

Restrictions for SSH client strict host key check

Provides reference information about the modes of SSH client strict host key checking, their behaviors, and their effect on host key storage, automation, and system security.

- All SSH client users share the default known hosts file at `/mnt/rdsfs/ciscossh/known_hosts`. Adding, deleting, or accepting a host key in this file affects all SSH client sessions.
- RDSFS must be available to retain host keys across device reloads and switchovers.
- When you use the `ssh-client knownhosts import <file-path>` command, the imported file replaces the contents of the default known-hosts file. The system does not append the imported entries.
- Host keys learned through SSH, SCP, SFTP, or other CLI operations that initiate SSH, SCP, or SFTP sessions are stored only in the default system-wide knownhosts file.
- The following strict host key checking modes are available:
  - `strict-check off`: Automatically accepts new or changed host keys. This mode weakens protection against man-in-the-middle attacks and should be used only for temporary, low-risk troubleshooting.
  - `strict-check accept-new`: Automatically accepts and stores new host keys, but rejects changed host keys.
  - `strict-check ask`: Prompts for approval only when an unknown host key is encountered. After the key is accepted and stored, any later change to that host key is rejected.
  - `strict-check yes`: Rejects both unknown and changed host keys. To avoid connection failures, pre-populate the trusted host list before enabling this policy.



Note If the legacy `ssh client knownhost` command is present in the global configuration, you cannot configure `strict-hostkey-check`. The exec-mode `ssh-client knownhosts` commands are also unavailable until you remove the legacy configuration.

Configure SSH client strict host key checking

Configure SSH client strict host key checking to control how routers verify SSH server host keys for outbound SSH connections and prevent unauthorized access.

- Enforce trust of only verified SSH servers to meet security policies.
- Minimize operational risk by ensuring host trust decisions are policy-driven.



Note The global strict host key checking configuration can be overridden for an individual SSH, SCP, or SFTP session using the `strict-hostkey-check` option.

SSH client strict host key checking validates remote server keys before starting an SSH session. All SSH client operations reference a single, system-wide `knownhosts` file at `/mnt/rdsfs/ciscossh/known_hosts`, regardless of the user.

This mechanism ensures that clients only connect to trusted hosts and enables administrators to choose the level of enforcement (**off**, **accept-new**, **ask**, or **yes**) for their environment.

- No per-user host key management; changes are global for all SSH client users.
- The `strict-check` mode controls the degree of host verification and is set according to deployment security requirements.

Before you begin

- Administrative access to the Cisco router.
- Persistent storage is available and RDSFS is mounted at `/mnt/rdsfs`.
- Your organization's SSH security policies for trusted host validation are known.

Follow these steps to configure SSH client strict host key checking.

Procedure

Step 1 Set the SSH client strict host key checking mode.

Example:

```
Router# configure
Router(config)# ssh client strict-hostkey-check {off | accept-new | ask | yes}
Router(config)# commit
```

- `off`: Automatically accepts new and changed host keys.
- `accept-new`: Accepts and stores new host keys, but rejects changed host keys.
- `ask`: Prompts for an unknown host key; after it is accepted and stored, rejects any later key change.
- `yes`: Rejects both unknown and changed host keys.

Step 2 Exit configuration mode.

Example:

```
Router(config)# end
```

Step 3 For an individual node, use exec-mode commands to manage trusted-host entries.

The following commands operate on the current node. To align trusted-host entries across nodes, copy and import the same baseline known-hosts file on each node.

a) Import a baseline knownhosts file.

Example:

```
Router# ssh-client knownhosts import <file-path>
```

b) Export the system-wide known-hosts database.

Example:

```
Router# ssh-client knownhosts export <file-path>
```

c) Add, delete, or clear trusted-host entries if required.

Example:

```
Router# ssh-client knownhosts add
Router# ssh-client knownhosts delete hostname
Router# ssh-client knownhosts delete line
Router# ssh-client knownhosts clear
```

To manage the trusted host entries, see SSH client knownhosts commands, on page 26.

Step 4 Verify the trusted-host list.

Example:

```
Router# show ssh client knownhosts
```

SSH client strict host key checking is configured. The router handles new and changed SSH server keys according to the selected enforcement mode.

What to do next

Review the router's outbound SSH connection logs to confirm that SSH client connections are handled according to the selected strict host-key checking mode and your organization's SSH security policies.

SSH client knownhosts commands

The trusted-host database is stored in `/mnt/rdsfs/ciscossh/known_hosts` and is shared by all SSH client users.

Use these commands to manage entries in the trusted-host database.

Table 8: SSH client known hosts commands

Command	Description
ssh-client knownhosts add	Adds a trusted-host entry.
ssh-client knownhosts import	Imports a knownhosts file.
ssh-client knownhosts export	Exports the trusted-host database.
ssh-client knownhosts delete hostname	Deletes an entry by hostname.
ssh-client knownhosts delete line	Deletes a numbered entry.
ssh-client knownhosts clear	Removes all entries.
show ssh client knownhosts	Displays the trusted-host entries.

Run these commands in exec mode.

Host keys learned during SSH, SCP, SFTP, or another CLI session are stored only in the system-wide database.

Automatic generation of SSH host-key pairs

An automatic generation of SSH host-key pairs is a security feature that

- creates SSH host-key pairs for supported algorithms (DSA, ECDSA, and RSA) automatically when the router boots,
- eliminates the need for explicit manual key generation after initial setup, and
- ensures SSH clients can connect to the SSH server immediately after bootup with a basic configuration.

Table 9: Feature History Table

Feature Name	Release info	Description
SSH key strength: 3072-bit by default	Release 26.1.1	This update enhances device security by automatically generating RSA 3072-bit SSH host keys during system boot, instead of RSA 2048-bit keys. 3072-bit aligns with industry best practices and provides improved cryptographic protection, ensuring secure SSH access and compliance with the latest security requirements.

The automatic generation feature simplifies device provisioning, especially in zero touch provisioning (ZTP) and Golden ISO boot scenarios. Since SSH host-key pairs are present immediately after boot, administrators do not need to perform manual configurations to enable secure connections.

Starting Cisco IOS XR Software Release 26.1.1, the system automatically generates RSA 3072-bit SSH host keys during boot, replacing the previous default of RSA 2048-bit keys. This enhancement strengthens device security by providing improved cryptographic protection and aligns with the current industry best practices. The default RSA 3072-bit SSH host keys are generated automatically requiring no manual intervention from you.

You can choose specific algorithms to use for SSH host-key pairs by configuring the **ssh server algorithms host-key** command in Global Configuration mode. If certain key pairs are not required, use the **crypto key zeroize** command in EXEC mode to remove them. When upgrading from an earlier software version, the system automatically generates host-key pairs only if they are missing, avoiding duplicate key generation. If SSH host-key pairs are not present after bootup, you can manually generate them using the **crypto key generate** command in EXEC mode.



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.

- On routers with automatic generation enabled, SSH host-key pairs for all supported algorithms are created during initial boot, enabling immediate SSH access by clients.
- During ZTP, there are no additional steps required to configure SSH host-key pairs.

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

#unique_296

Associated Commands

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

Ed25519 Public-Key Signature Algorithm Support for SSH

Table 10: Feature History Table

Feature Name	Release Information	Feature Description
Ed25519 Public-Key Signature Algorithm Support for SSH	Release 7.3.1	This algorithm is now supported on Cisco IOS XR 64-bit platforms when establishing SSH sessions. It is a modern and secure public-key signature algorithm that provides several benefits, particularly resistance against several side-channel attacks. Prior to this release, DSA, ECDSA, and RSA public-key algorithms were supported. This command is modified for this feature: ssh server algorithms host-key

This feature introduces the support for Ed25519 public-key algorithm, when establishing SSH sessions, on Cisco IOS XR 64-bit platforms. This algorithm offers better security with faster performance when compared to DSA or ECDSA signature algorithms.

The order of priority of public-key algorithms during SSH negotiation between the client and the server is:

- ecdsa-sha2-nistp256
- ecdsa-sha2-nistp384
- ecdsa-sha2-nistp521
- ssh-ed25519
- ssh-rsa
- ssh-dsa

Restrictions for ED25519 Public Key for SSH

The Ed25519 public key algorithm is not FIPS-certified. That is, if FIPS mode is enabled on the router, the list of public-key algorithms sent during the SSH key negotiation phase does not contain the Ed25519 key. This behavior is applicable only for new SSH connections. Any existing SSH session that has already negotiated Ed25519 public-key algorithm remains intact and continues to execute until the session is disconnected.

Further, if you have configured the router to negotiate only the Ed25519 public-key algorithm (using the **ssh server algorithms host-key** command), and if FIPS mode is also enabled, then the SSH connection to the router fails.

How to Generate Ed25519 Public Key for SSH

To generate Ed25519 public key for SSH, see .

You must also specify Ed25519 as the permitted SSH host-key pair algorithm from the list of auto-generated host-key pairs on the SSH server. For details, see .

To remove the Ed25519 key from the router, use the **crypto key zeroize ed25519** command in XR EXEC mode.

Configure SSH Client

Perform this task to configure an SSH client.

Procedure

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 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** (prior to Cisco IOS XR Software Release 24.1.1)
  - **public-key**, **keyboard-interactive** and **password** (from Cisco IOS XR Software Release 24.1.1 and later)
- 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 7.10.1and 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#
```

Configuring CBC Mode Ciphers

In Cisco IOS XR Release 7.0.1, you can enable CBC mode ciphers 3DES-CBC and AES-CBC for SSHv2 server and client connections. The ciphers are disabled by default.

Procedure

Step 1 **configure**

Example:

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

Enters mode.

Step 2 **ssh server enable cipher aes-cbc 3des-cbc**

Example:

```
Router(config)# ssh server enable cipher aes-cbc 3des-cbc
```

Step 3 **ssh client enable cipher aes-cbc 3des-cbc**

Example:

```
Router(config)# ssh client enable cipher aes-cbc 3des-cbc
```

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 **show ssh session details**

Example:

```
Router# show ssh session details
```

Configuring CBC Mode Ciphers

```
/*Enable CBC mode ciphers 3DES-CBC and AES-CBC */
Router# configure
Router(config)# ssh server enable cipher aes-cbc 3des-cbc
Router(config)# ssh client enable cipher aes-cbc 3des-cbc
Router(config)# commit
```

Verify CBC Mode Cipher Configuration.

```
Router# show ssh session details

Thu Sep  6 10:16:26.346 UTC
SSH version : Cisco-2.0

id  key-exchange        pubkey    incipher    outcipher   inmac         outmac
------------------------------------------------------------------------------
Incoming Session
```

```
2   ecdh-sha2-nistp256  ssh-rsa   aes128-cbc  aes128-cbc  hmac-sha2-256 hmac-sha2-256
```

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.

Note This feature is not supported on the following variants of Cisco NCS 540 Series Routers:

- N540-28Z4C-SYS-A
- N540-28Z4C-SYS-D
- N540X-16Z4G8Q2C-A
- N540X-16Z4G8Q2C-D
- N540-12Z20G-SYS-A
- N540-12Z20G-SYS-D
- N540X-12Z16G-SYS-A
- N540X-12Z16G-SYS-D

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-sha1
Router(config)#commit
```

```
Router(config)# ssh client disable hmac hmac-sha1
Router(config)#commit
```

Running Configuration

```
ssh server disable hmac hmac-sha1
!
```

```
ssh client disable hmac hmac-sha1
!
```

Related Topics

SSH Configuration Option to Restrict Cipher Public Key and HMAC Algorithm, on page 35

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 35

Associated Commands

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

User Configurable Maximum Authentication Attempts for SSH

Table 11: Feature History Table

Feature Name	Release Information	Feature Description
User Configurable Maximum Authentication Attempts for SSH	Release 7.3.1	This feature allows you to set a limit on the number of user authentication attempts allowed for SSH connection, using the three authentication methods that are supported by Cisco IOS XR. The limit that you set is an overall limit that covers all the authentication methods together. If the user fails to enter the correct login credentials within the configured number of attempts, the connection is denied and the session is terminated. This command is introduced for this feature: ssh server max-auth-limit

The three SSH authentication methods that are supported by Cisco IOS XR are public-key (which includes certificate-based authentication), keyboard-interactive, and password authentication. The limit count that you set as part of this feature comes into effect whichever combination of authentication methods you use. The limit ranges from 3 to 20; default being 20 (prior to Cisco IOS XR Software Release 7.3.2, the limit range was from 4 to 20).

Restrictions for Configuring Maximum Authentication Attempts for SSH

These restrictions apply to configuring maximum authentication attempts for SSH:

- This feature is available only for Cisco IOS XR routers functioning as SSH server; not for the ones functioning as SSH clients.
- This configuration is not user-specific; the limit remains same for all the users.
- Due to security reasons, the SSH server limits the number of authentication attempts that explicitly uses the password authentication method to a maximum of 3. You cannot change this particular limit of 3 by configuring the maximum authentication attempts limit for SSH.

  For example, even if you configure the maximum authentication attempts limit as 5, the number of authentication attempts allowed using the password authentication method still remain as 3.

Configure Maximum Authentication Attempts for SSH

You can use the **ssh server max-auth-limit** command to specify the maximum number of authentication attempts allowed for SSH connection.

Configuration Example

```
Router#configure
Router(config)#ssh server max-auth-limit 5
Router(config)#commit
```

Running Configuration

```
Router#show running-configuration ssh
ssh server max-auth-limit 5
ssh server v2
!
```

Verification

The system displays the following SYSLOG on the router console when maximum authentication attempts is reached:

```
RP/0/RP0/CPU0:Oct 6 10:03:58.029 UTC: SSHD_[68125]: %SECURITY-SSHD-3-ERR_GENERAL : Max
authentication tries reached-exiting
```

Associated Commands

- **ssh server max-auth-limit**

X.509v3 Certificate-based Authentication for SSH

Table 12: Feature History Table

Feature Name	Release Information	Feature Description
X.509v3 Certificate-based Authentication for SSH	Release 7.3.1	This feature adds new public-key algorithms that use X.509v3 digital certificates for SSH authentication. These certificates use a chain of signatures by a trusted certification authority to bind a public key to the digital identity of the user who is authenticating with the SSH server. These certificates are difficult to falsify and therefore used for identity management and access control across many applications and networks. Commands introduced for this feature are: ssh server certificate ssh server trustpoint This command is modified for this feature: ssh server algorithms host-key

This feature adds new public-key algorithms that use X.509v3 digital certificates for SSH authentication. This feature support is available for the SSH server for server and user authentication.

The X.509v3 certificate-based authentication for SSH feature supports the following public-key algorithms:

- **x509v3-ssh-dss**
- **x509v3-ssh-rsa**
- **x509v3-ecdsa-sha2-nistp256**
- **x509v3-ecdsa-sha2-nistp384**
- **x509v3-ecdsa-sha2-nistp521**

Note While user authentication by using X.509v3 certificate-based authentication for the SSH server is supported using all algorithms listed above, server authentication is supported only with the **x509v3-ssh-rsa** algorithm.

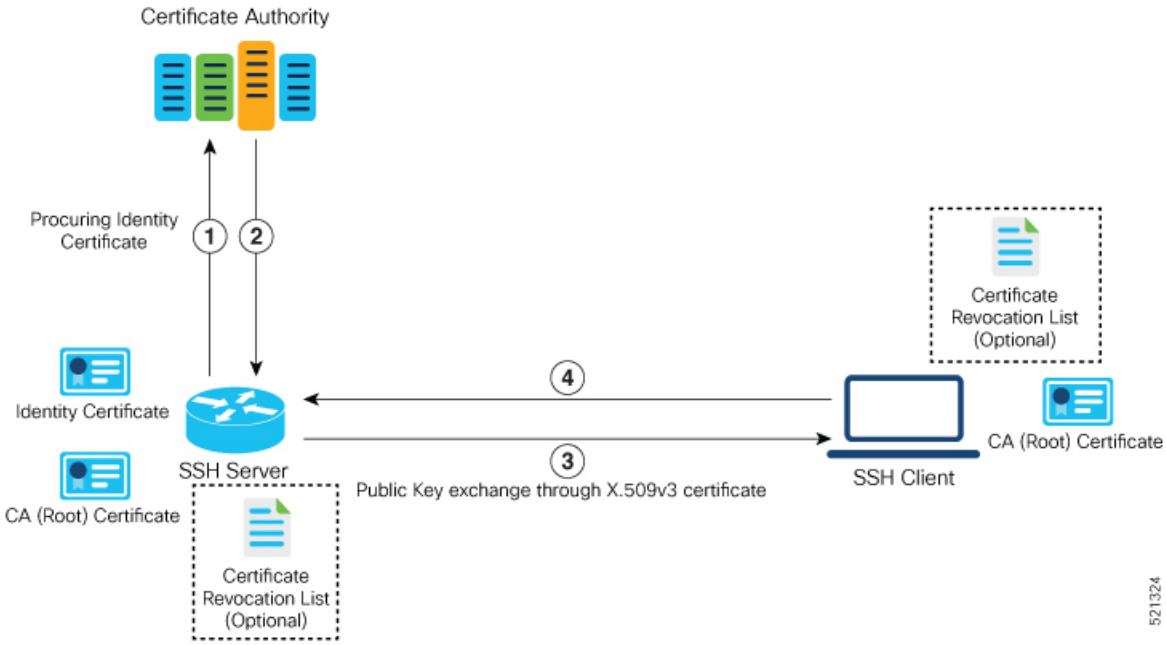
There are two SSH protocols that use public-key cryptography for authentication:

- Transport Layer Protocol (TLP) described in RFC4253—this protocol mandates that you use a digital signature algorithm (called the public-key algorithm) to authenticate the server to the client.

- User Authentication Protocol (UAP) described in RFC4252—this protocol allows the use of a digital signature to authenticate the client to the server (public-key authentication).

For TLP, the Cisco IOS XR SSH server provides its server certificate to the client, and the client verifies the certificate. Similarly, for UAP, the client provides an X.509 certificate to the server. The peer checks the validity and revocation status of the certificate. Based on the result, access is allowed or denied.

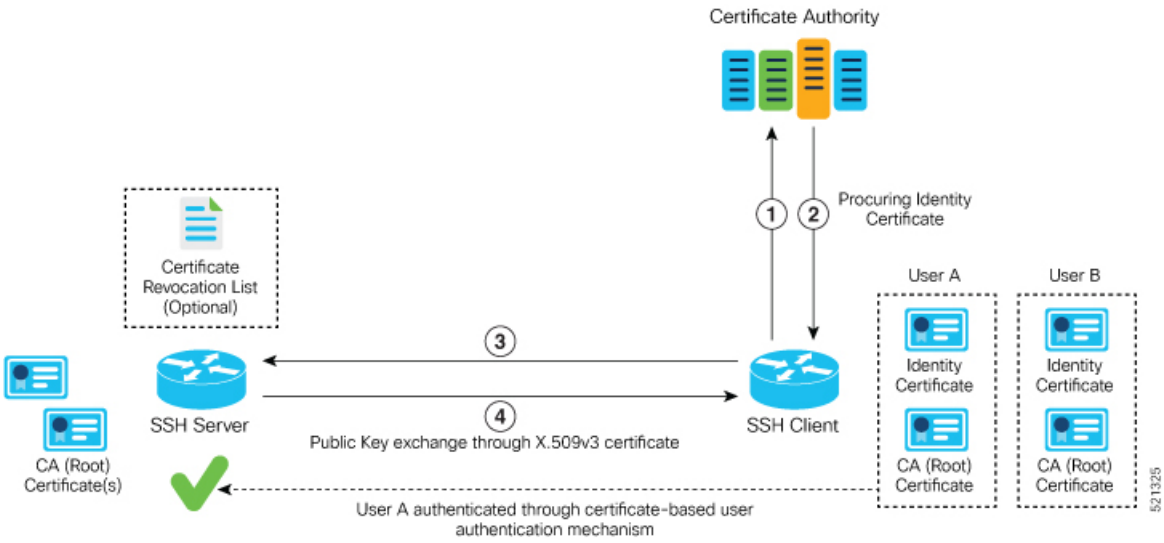
Server Authentication using X.509v3 Certificate



The server authentication process involves these steps:

1. The SSH server procures a valid identity certificate from a well-known certificate authority. This certificate can be obtained manually (through cut-and-paste mechanism) or through protocol implementations such as Simple Certificate Enrollment Protocol (SCEP).
2. The certificate authority provides valid identity certificates and associated root certificates. The requesting device stores these certificates locally.
3. The SSH server presents the certificate to the SSH client for verification.
4. The SSH client validates the certificate and starts the next phase of the SSH connection.

User Authentication using X.509v3 Certificate



The user authentication phase starts after the SSH transport layer is established. At the beginning of this phase, the client sends the user authentication request to the SSH server with required parameters. The user authentication process involves these steps:

1. The SSH client requests a valid identity certificate from a well-known certificate authority.
2. The certificate authority provides valid identity certificates and associated root certificates. The requesting device stores these certificates locally.
3. The SSH client presents the certificate to the SSH server for verification.
4. The SSH server validates the certificate and starts the next phase of the SSH connection.

The certificate-based authentication uses public key as the authentication method. The certificate validation process by the SSH server involves these steps:

- The SSH server retrieves the user authentication parameters, verifies the certificate, and also checks for the certificate revocation list (CRL).
- The SSH server extracts the *username* from the certificate attributes, such as *subject name* or *subject alternate name* (SAN) and presents them to the AAA server for authorization.
- The SSH server then takes the extracted *username* and validates it against the incoming *username* string present in the SSH connection parameter list.

Restrictions for X.509v3 Certificate-based Authentication for SSH

These restrictions apply to the X.509v3 certificate-based authentication feature for SSH:

- Supported only for Cisco IOS XR devices acting as the SSH server; not for the Cisco IOS XR devices acting as the SSH client.
- Supported only for local users because TACACS and RADIUS server do not support public-key authentication. As a result, you must include the **local** option for AAA authentication configuration.

Note Although this feature supports only local authentication, you can enforce remote authorization and accounting using the TACACS server.

- Certificate verification using the Online Certificate Status Protocol (OCSP) is currently not supported. The revocation status of certificates is checked using a certificate revocation list (CRL).
- To avoid user authentication failure, the chain length of the user certificate must not exceed the maximum limit of 9.

Configure X.509v3 Certificate-based Authentication for SSH

To enable X.509v3 certificate-based authentication for SSH, these tasks for server and user authentication:

Server Authentication:

- Configure the list of host key algorithms—With this configuration, the SSH server decides the list of host keys to be offered to the client. In the absence of this configuration, the SSH server sends all available algorithms to the user as host key algorithms. The SSH server sends these algorithms based on the availability of the key or the certificate.
- Configure the SSH trust point for server authentication—With this configuration, the SSH server uses the given trust point certificate for server authentication. In the absence of this configuration, the SSH server does not send **x509v3-ssh-rsa** as a method for server verification. This configuration is not VRF-specific; it is applicable to SSH running in all VRFs.

  The above two tasks are for server authentication and the following ones are for user authentication.

User Authentication:

- Configure the trust points for user authentication—With this configuration, the SSH server uses the given trust point for user authentication. This configuration is not user-specific; the configured trust points are used for all users. In the absence of this configuration, the SSH server does not authenticate using certificates. This configuration is not specific to a VRF; it is applicable to SSH running in all VRFs.

  You can configure up to ten user trust points.
- Specify the *username* to be picked up from the certificate—This configuration specifies which field in the certificate is to be considered as the *username*. The **common-name** from the **subject name** or the **user-principle-name(othername)** from the **subject alternate name**, or both can be configured.
- Specify the maximum number of authentication attempts allowed by the SSH server—The value ranges from 4 to 20. The default value is 20. The server closes the connection if the number of user attempts exceed the configured value.
- AAA authentication configuration—The AAA configuration for public key is the same as that for the regular or keyboard-interactive authentication, except that it mandates local method in the authentication method list.

Configuration Example

In this example, the **x509v3-ssh-rsa** is specified as the allowed host key algorithm to be sent to the client. Similarly, you can configure other algorithms, such as **ecdsa-sha2-nistp521, ecdsa-sha2-nistp384, ecdsa-sha2-nistp256, ssh-rsa**, and **ssh-dsa**.

```
/* Configure the lits of host key algorithms */
Router#configure
Router(config)#ssh server algorithms host-key x509v3-ssh-rsa
Router(config)#commit


/* Configure the SSH trustpoint for server authentication */
Router#configure
Router(config)#ssh server certificate trustpoint host tp1
Router(config)#commit


/* Configure the trustpoints to be used for user authentication */
Router#configure
Router(config)#ssh server trustpoint user tp1
Router(config)#ssh server trustpoint user tp2
Router(config)#commit


/* Specifies the username to be picked up from the certificate.
In this example, it specifies the user common name to be picked up from the subject name
field */
Router#configure
Router(config)#ssh server certificate username common-name
Router(config)#commit


/* Specifies the maximum authentication limit for the SSH server */
Router#configure
Router(config)#ssh server max-auth-limit 5
Router(config)#commit


/* AAA configuration for local authentication with certificate and
remote authorization with TACACS+ or RADIUS */
Router#configure
Router(config)#aaa authentication login default group tacacs+ local
Router(config)#aaa authorization exec default group radius group tacacs+
Router(config)#commit
```

Running Configuration

```
ssh server algorithms host-key x509v3-ssh-rsa
!
ssh server certificate trustpoint host tp1
!
ssh server trustpoint user tp1
ssh server trustpoint user tp2
!
ssh server certificate username common-name
!
ssh server max-auth-limit 5
!
```

Verification of Certificate-based Authentication for SSH

You can use the **show ssh server** command to see various parameters of the SSH server. For certificate-based authentication for SSH, the **Certificate Based** field displays *Yes*. Also, the two new fields, **Host Trustpoint** and **User Trustpoints**, display the respective trust point names.

```
Router#show ssh server
Wed Feb 19 15:23:38.752 IST
---------------------
SSH Server Parameters
---------------------

Current supported versions := v2
                  SSH port := 22
                  SSH vrfs := vrfname:=default(v4-acl:=, v6-acl:=)
              Netconf Port := 830
              Netconf Vrfs := vrfname:=default(v4-acl:=, v6-acl:=)

 Algorithms
---------------
        Hostkey Algorithms := x509v3-ssh-rsa,
ecdsa-sha2-nistp521,ecdsa-sha2-nistp384,ecdsa-sha2-nistp256,ssh-rsa,ssh-dsa
   Key-Exchange Algorithms :=
ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group14-sha1
     Encryption Algorithms :=
aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
            Mac Algorithms := hmac-sha2-512,hmac-sha2-256,hmac-sha1

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

 Others
------------
                     DSCP  := 16
                Ratelimit  := 60
             Sessionlimit  := 100
                Rekeytime  := 60
       Server rekeyvolume  := 1024
  TCP window scale factor  := 1
            Backup Server  := Enabled, vrf:=default, port:=11000
Host Trustpoint            := tp1
User Trustpoints           := tp1 tp2
```

You can use the **show ssh session details** command to see the chosen algorithm for an SSH session:

```
Router#show ssh session details
Wed Feb 19 15:33:00.405 IST
SSH version : Cisco-2.0

id      key-exchange           pubkey               incipher    outcipher   inmac
outmac
-----------------------------------------------------------------------------------------------

Incoming Sessions
1       ecdh-sha2-nistp256     x509v3-ssh-rsa       aes128-ctr  aes128-ctr  hmac-sha2-256
hmac-sha2-256
```

Similarly, you can use the **show ssh** command to verify the authentication method used. In this example, it shows as *x509-rsa-pubkey*:

```
Router#show ssh
Sun Sep 20 18:14:04.122 UTC
SSH version : Cisco-2.0

id  chan pty  location   state        userid    host          ver authentication connection
 type
-----------------------------------------------------------------------------------------------
Incoming sessions
4   1    vty0 0/RP0/CPU0 SESSION_OPEN  9chainuser 10.105.230.198 v2  x509-rsa-pubkey
Command-Line-Interface

Outgoing sessions
```

SYSLOGS

You can observe relevant SYSLOGS on the router console in various scenarios listed here:

- On successful verification of peer certificate:

```
RP/0/RP0/CPU0:Aug 10 15:01:34.793 UTC: locald_DLRSC[133]: %SECURITY-PKI-6-LOG_INFO :
Peer certificate verified successfully
```

- When user certificate CA is not found in the trust point:

```
RP/0/RP0/CPU0:Aug  9 22:06:43.714 UTC: locald_DLRSC[260]: %SECURITY-PKI-3-ERR_GENERAL
: issuer not found in trustpoints configured
RP/0/RP0/CPU0:Aug  9 22:06:43.714 UTC: locald_DLRSC[260]: %SECURITY-PKI-3-ERR_ERRNO :
Error:='Crypto Engine' detected the 'warning' condition 'Invalid trustpoint or trustpoint
 not exist'(0x4214c000), cert verificationn failed
```

- When there is no CA certificate or host certificate in the trust point:

```
RP/0/RP1/CPU0:Aug 10 00:23:28.053 IST: SSHD_[69552]: %SECURITY-SSHD-4-WARNING_X509 :
could not get the host cert chain, 'sysdb' detected the 'warning' condition 'A SysDB
client tried to access a nonexistent item or list an empty directory', x509 host auth
will not be used
RP/0/RP1/CPU0:Aug 10 00:23:30.442 IST: locald_DLRSC[326]: %SECURITY-PKI-3-ERR_ERRNO :
Error:='Crypto Engine' detected the 'warning' condition 'Invalid trustpoint or trustpoint
 not exist'(0x4214c000), Failed to get trustpoint name from
```

How to Disable X.509v3 Certificate-based Authentication for SSH

- Server Authentication — You can disable X.509v3 certificate-based server authentication for SSH by using the **ssh server algorithms host-key** command. From the list of auto-generated host-key pairs algorithms on the SSH server, this command configures allowed SSH host-key pair algorithms. Hence, if you have this configuration without specifying the **x509-ssh-rsa** option in the preceding command, it is equivalent to disabling the X.509v3 certificate-based server authentication for the SSH server.
- User Authentication — You can remove the user trust point configuration (**ssh server trustpoint user**) so that the SSH server does not allow the X.509v3 certificate-based authentication.

Failure Modes for X.509v3 Certificate-based Authentication for SSH

If the **ssh server certificate trustpoint host** configuration is missing, or if the configuration is present, but the router certificate is not present under the trust point, then the SSH server does not add **x509-ssh-rsa** to the list of supported host key methods during key exchange.

Also, the user authentication fails with an error message if:

- User certificate is in an incorrect format.
- The chain length of the user certificate is more than the maximum limit of 9.
- Certificate verification fails due to any reason.

Related Topics

- X.509v3 Certificate-based Authentication for SSH, on page 40

Associated Commands

- **ssh server algorithms hostkey**
- **ssh server certificate username**
- **ssh server max-auth-limit**
- **ssh server trustpoint host**
- **ssh server trustpoint user**
- **show ssh server**
- **show ssh session details**

Importing Public Key

Public key can be imported in EXEC or any configuration modes. To correctly import a public key into the chassis, the public key must be in one of the following formats:

- SSH RSA Public key format—Supported in configuration mode and in EXEC mode as text and file.
- Base64 encoded (binary) format—Supported as EXEC mode as file.



Note Base64 format is not supported if you press enter and copy paste the key.

- PEM PKCS8 format—Supported in EXEC mode as text and file.
- PEM PKCS1 format—Supported in EXEC mode as text and file.

Only the users belonging to user groups with admin and crypto task ID can import public key for other users.

Only the users belonging to groups root-lr, netadmin, maintenance, sysadmin can import public key for other users.

Import Public Key in EXEC Mode

Use the following command to import public key for currently logged in user:

```
Crypto key import authentication rsa file_path
Crypto key import authentication rsa  <press enter here>
```

Use the following command to import public key for other users:

```
Crypto key import authentication rsa username username file_path
Crypto key authentication rsa username <press enter here>
```

Remove Public Key Imported in EXEC Mode

Public Keys imported in EXEC mode can be zeroized and removed by using the following commands:

- For currently logged in user:

  ```
  crypto key zeroize authentication rsa
  ```

- For other users:

  ```
  crypto key zeroize authentication rsa username username
  ```

- For all users:

  ```
  crypto key zeroize authentication rsa all
  ```

Import Public Key in Configuration Mode

Note User must be present in the database to which the keys are imported.

Note Only SSH RSA Public key format is supported in configuration mode.

Use the following command to import public key in configuration mode:

```
ssh server username username keystring ssh-rsa pubkey
```

Remove Public Key Imported in Configuration Mode

Use the following command to remove public key in configuration mode:

```
no ssh server username username
```

Verify the Import of Public Key

Use the following commands to display public keys imported from both configuration and EXEC modes.

- For currently logged in user:

  ```
  show crypto key authentication rsa
  ```

- For other users:

  ```
  show crypto key authentication rsa username username
  ```

- For all users:

```
show crypto key authentication rsa all
```

Selective Authentication Methods for SSH Server

Table 13: Feature History Table

Feature Name	Release Information	Feature Description
Selective Authentication Methods for SSH Server	Release 7.8.1	You now have the flexibility to choose the preferred SSH server authentication methods on the router. These methods include password authentication, keyboard-interactive authentication, and public-key authentication. This feature allows you to selectively disable these authentication methods. By allowing the SSH clients to connect to the server only through these permitted authentication methods, this functionality provides additional security for router access through SSH. Before this release, by default, the SSH server allowed all these authentication methods for establishing SSH connections. The feature introduces these changes: • **CLI**: New **disable auth-methods** command • **YANG Data Model**: New XPaths for `Cisco-IOS-XR-crypto-ssh-cfg.yang` Cisco native model (see GitHub)

By default, the SSH server on the Cisco IOS XR routers allowed various authentication methods such as password authentication, keyboard-interactive authentication, and public-key authentication (including certificate-based authentication) for the SSH connections on the router. The SSH clients could use any of these authentication methods while attempting a connection to the SSH server on the router. From Cisco IOS XR Software Release 7.8.1, you can selectively disable these authentication methods, and allow connection attempts from the SSH client only through the remaining authentication methods. If the SSH client tries to establish a connection to the server using nonpermitted authentication methods (the ones that are disabled), then the login attempt fails.

Disable SSH Server Authentication Methods

Use the **disable auth-methods** command in ssh server configuration mode to disable the specific authentication method for the SSH server.

Public-key authentication includes certificate-based authentication as well. Hence, disabling public-key authentication automatically disables the certificate-based authentication.

Configuration Example

This example shows how to disable the keyboard-interactive authentication method for the SSH server on the router using CLI. Similarly, you can disable other authentication methods.

```
Router#configure
Router(config)# ssh server
Router(config-ssh)# disable auth-methods keyboard-interactive
Router(config-ssh)# commit
```

Running Configuration

```
!
ssh server
 disable auth-methods keyboard-interactive
!
```

Verification

Use the **show ssh server** command to see the list of authentication methods that the SSH server on the router supports. In this example, the keyboard-interactive method is disabled and the SSH server allows all other authentication methods.

```
Router#show ssh server

Wed Feb 23 10:38:37.716 UTC
RP/0/RP0/CPU0:ios(config)
Authentication Method Supported
-----------------------------------
                 PublicKey := Yes
                  Password := Yes
      Keyboard-Interactive := No
         Certificate Based := Yes
```

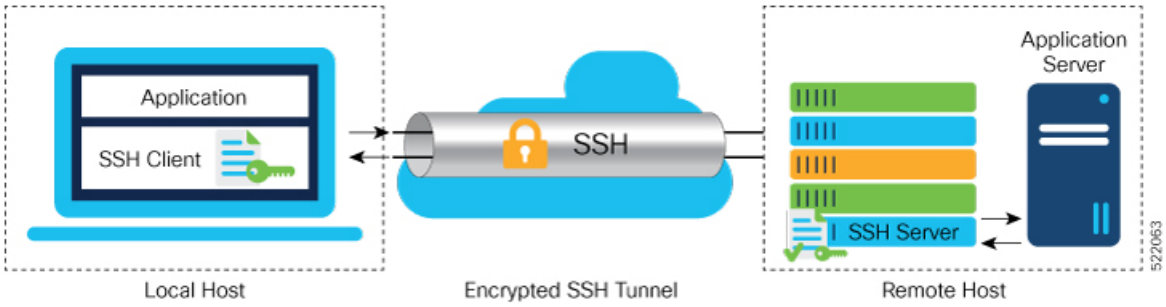
SSH Port Forwarding

Table 14: Feature History Table

Feature Name	Release Information	Feature Description
SSH Port Forwarding	Release 7.3.2	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. This feature introduces the ssh server port-forwarding local command. **Note** For the following Cisco NCS 540 router variants, this feature is supported with Cisco IOS XR SSH: • N540-ACC-SYS • N540X-ACC-SYS (Premium) • N540-24Z8Q2C-SYS For all other Cisco NCS 540 router variants, this feature is supported with CiscoSSH, an OpenSSH-based implementation of SSH.

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

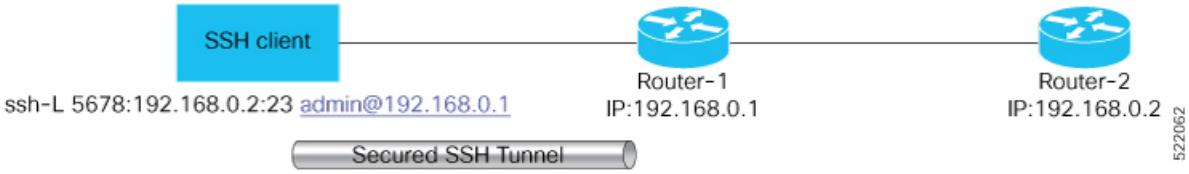


Consider an application on the SSH client residing on a local host, trying to connect to an application server residing on a remote host. With tunneling enabled, the application on the SSH client connects to a port on the local host that the SSH client listens to. The SSH client then forwards the data 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 need 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 15: Feature History Table

Feature Name	Release Information	Feature Description
Support for Non-Default SSH Ports on Cisco NCS 540 series routers that run on Cisco IOS XR7	Release 7.11.1	Non-default SSH port is now supported on all variants of Cisco NCS 540 series routers, include the variants that run on Cisco IOS XR7.

Feature Name	Release Information	Feature Description
Non-Default SSH Port	Release 7.7.1	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. The feature introduces the command. The feature modifies these commands to include the **port** option: • **ssh** • **sftp** • **scp**

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 :=
x509v3-ssh-rsa,ssh-rsa-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 :=
ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group14-sha1,curve25519-sha256,diffie-hellman-group14-sha256,diffie-hellman-group16-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
.
.
.
```

Public Key-Based Authentication of SSH Clients

Table 16: Feature History Table

Feature Name	Release Information	Feature Description

Public Key-Based Authentication of SSH Clients on Cisco IOS XR Routers	Release 7.10.1	You can now avail cryptographic strength and automated password-less log in while establishing SSH connections with the server. Along with password and keyboard-interactive authentication, Cisco IOS XR routers configured as SSH clients now support public key-based authentication. In this authentication method, passwords need not be sent over the network and hence, it provides an additional layer of security as well as aids in automation processes. This feature is available only for users locally configured on the router, not those configured on remote servers.
		
		Previous releases supported SSH public key-based authentication only for Cisco IOS XR routers configured as SSH servers.
		
		The feature introduces these changes:
		
		• **CLI**:
		  • **crypto key generate authentication-ssh rsa**
		  • **crypto key zeroize authentication-ssh rsa**
		  • **show crypto key mypubkey authentication-ssh rsa**
		
		• **Yang Data Models**:
		  New Xpaths for:
		  • `Cisco-IOS-XR-crypto-act.yang`
		  • `Cisco-IOS-XR-crypto-oepki-new-oper.yang`
		
		  (see GitHub, YANG Data Models Navigator)

Cisco IOS XR routers configured as SSH clients supported only password authentication and keyboard-interactive authentication for establishing SSH connection with the SSH server. Whereas those IOS XR routers that are configured as SSH servers supported public key-based user authentication as well. From

Cisco IOS XR Software Release 7.10.1 and later, you can use public-key based user authentication for Cisco IOS XR routers configured as SSH clients as well. This feature thereby allows you to use password-less authentication for secure file transfer and copy operations using SFTP and SCP protocols.

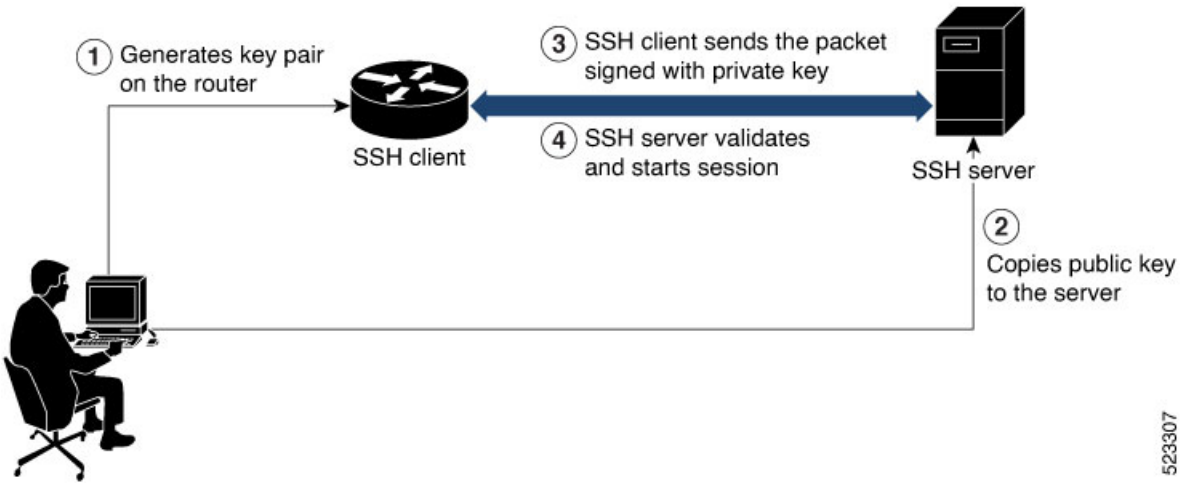
Remote AAA servers such as RADIUS and TACACS+ servers do not support public-key based authentication. Hence this functionality is available only for users who are configured locally on the router and not for users who are configured remotely.

How Does it Work

Public key encryption algorithm works with two keys—a public key and a private key. These keys form a key pair that is specific to a user. They are cryptographically related. The public key is used to encrypt the data and the private key is used to decrypt the data. The data encrypted by the SSH server that holds the public key can then only be read by the entity who holds the corresponding private key.

This image shows the work flow of public key-based authentication of SSH clients.

Figure 3: Public Key-Based Authentication of SSH Clients: Work Flow



You can generate the key pair on the router that is configured as the SSH client. Once it is generated, copy the public key to the SSH server that the user wants to connect to. When the user tries to log in to the server, the SSH client sends a connection request to the SSH server. The SSH server allows access only to users who can confirm that they have the corresponding private key. For this. the SSH server uses the public key of the user to issue a challenge that can be rightly answered by the SSH client using the corresponding private key. The SSH client thus automatically authenticates the user who is logging in to the server using the unique copy of the private key. This process thereby establishes a secure SSH connection to the server in a way that does not require the user to enter the password each time.

Enable Public Key-Based Authentication of SSH Client

Guidelines

These guidelines apply to enabling public key-based SSH authentication on Cisco IOS XR routers that are configured as SSH clients.

- Supports only RSA key.

- Remote AAA servers such as RADIUS and TACACS+ servers do not support public key-based authentication. Hence this functionality is available only for users who are configured locally on the router and not for users who are configured remotely.
- A user with root privileges has permission to create and delete keys for other users.
- If authentication keys are not created, then the SSH client does not proceed with public key-based authentication.
- If user adds the incorrect public key in the SSH server, then the user authentication fails.

Configuration Example

Establishing SSH connection using public key-based authentication on SSH client involves these high-level tasks:

1. Generate RSA key pair on the router that is configured as the SSH client.

   Use the **cyrpto key generate authentication-ssh rsa** command to generate the RSA key pair:

```
Router#crypto key generate authentication-ssh rsa
Wed Dec 21 10:02:57.684 UTC
The name for the keys will be: cisco
  Choose the size of the key modulus in the range of 512 to 4096. Choosing a key modulus
 greater than 512 may take a few minutes.

How many bits in the modulus [2048]:
Generating RSA keys ...
Done w/ crypto generate keypair
[OK]

Router#
```

2. View the details of the generated key.

   Use the **show crypto key mypubkey authentication-ssh rsa** command to view the details of the RSA key. The key value starts with *ssh-rsa* in this output.

```
Router#show crypto key mypubkey authentication-ssh rsa
Wed Dec 21 10:24:34.226 UTC
Key label: cisco
Type     : RSA Authentication
Size     : 2048
Created  : 10:02:59 UTC Wed Dec 21 2022
Data     :
 30820122 300D0609 2A864886 F70D0101 01050003 82010F00 3082010A 02820101
 00A292B0 E45ACBB9 47B9EDA8 47E4664E 58FC3EA5 CE0F6B7A 3C6B7A73 537E6CEB
 .
 .
 .
 FF6BAF95 D9617CF6 65C058CC 7C6C22A9 9E48CC43 FDFF0EB7 ABADEB77 55A274DB
 15020301 0001

OpenSSH Format:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCikrDkWsu5R7ntqEfkZk5Y/.../2uvldlhfPZlwFjMfGwiqZ5IzEP9/w63q63rd1WidNsV

Router#
```

3. Copy the RSA public key from the SSH client to the SSH server.

You can do this either by logging in to the remote SSH server with your established user credentials, or have a system administrator on the remote system add the key on the SSH server.

If the SSH server is a Cisco IOS XR router, then you can use the **crypto key import authentication rsa** command on the router prompt of the server to import the key from the SSH client. You will then be prompted to enter the public key.

If the SSH server is a Linux server, then you must add the public key to the `~/.ssh/authorized_keys` file of the respective user account in that server. This file contains a list of all authorized public keys on that server.

4. The user configured on the SSH client can now log in to the remote SSH server (*209.165.200.225* in this example) without providing the user account password.

```
Router#ssh user1@209.165.200.225
```

This process establishes a successful SSH connection between the client and the server using public key-based authentication.

How to Delete the SSH Public Keys

Use the **crypto key zeroize authentication-ssh rsa username** command to delete the RSA keys.

```
Router#crypto key zeroize authentication-ssh rsa username user1
```

Public key-based Authentication to SSH Server on Routers

Table 17: Feature History Table

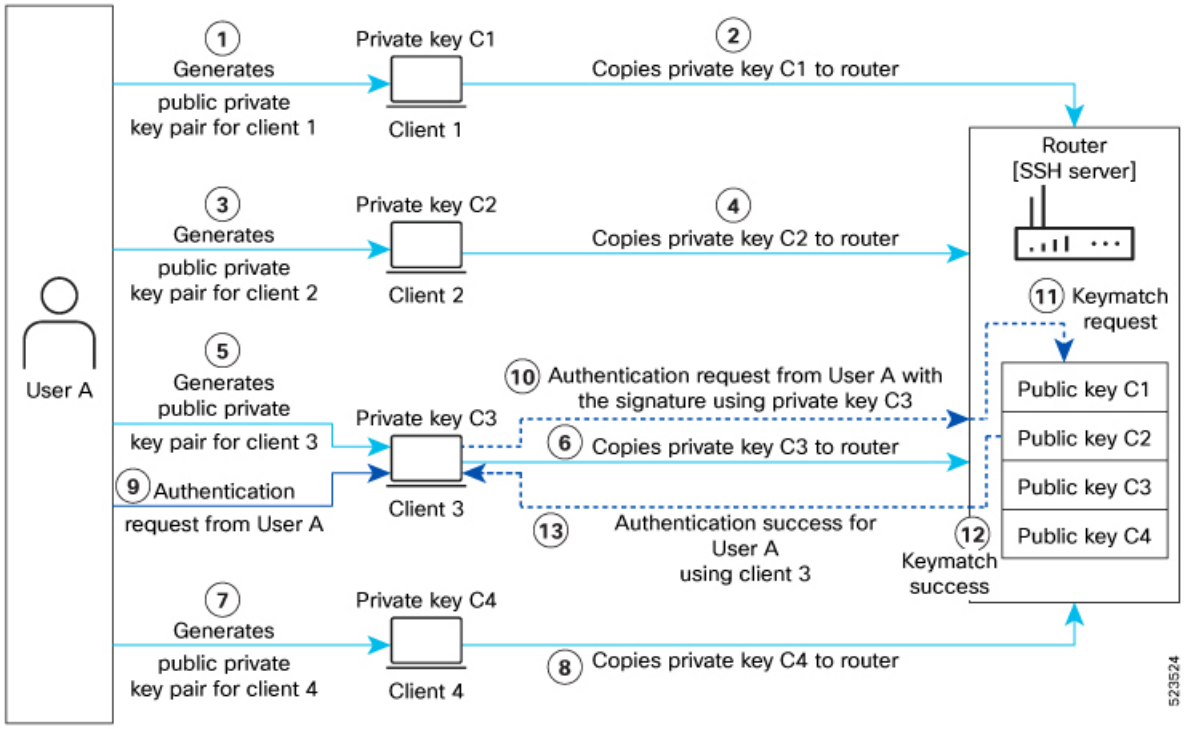
Feature Name	Release Information	Feature Description
Multiple Public Keys per User for Public Key-based Authentication	Release 7.11.1	We provide greater flexibility to access secure routers by allowing four public keys to be used for authentication. With the ability to associate multiple public keys with your user account on the router, we've also simplified the authentication process by eliminating the need to create unique users for each SSH client device. The feature introduces these changes: **CLI:** • The **second**, **third**, and **fourth** keywords are introduced in the **crypto key import authentication rsa** command. • The **second**, **third**, and **fourth** keywords are introduced in the **crypto key zeroize authentication rsa** command. • The **second**, **third**, and **fourth** keywords are introduced in the **keystring** command.`  **YANG Data Models:**  • `Cisco-IOS-XR-crypto-act`  • `Cisco-IOS-XR-um-ssh-cfg` (See GitHub, YANG Data Models Navigator)

Public key-based authentication provides password-less authentication to the routers. In this method, the user authentication relies on a cryptographic key pair: a public key and a private key. The user generates a key pair in the client device using utilities such as ssh-keygen. The public key is imported and stored in the router(SSH server), while the private key is in the user device(SSH client). While attempting public key-based authentication from the client, the user presents a signature created using the private key to the router. The router verifies the authenticity of that signature using the public key associated with that user in its database.

The authentication is successful when the signature matches the public key and user access is permitted. Otherwise, the authentication fails, and the router denies the user access. With public key-based authentication, the routers offer a more secure authentication method than traditional password-based authentication because it is less vulnerable to brute force attacks and password theft.

From Cisco IOS XR Software Release 7.11.1, the routers support up to four public keys per user for public key-based authentication to the routers. Previously, the users could have a single key pair. This constraint restricts an individual user in the router from having multiple systems to access the routers. Also, it necessitates creating a unique user in the router for each device to be an authorized SSH client to the router. However, the multiple public keys per user for public key-based authentication feature overcome these restrictions by allowing up to four public keys per user. As a result, the users can employ any corresponding private key to access the router.

Figure 4: Multiple Public Keys per User for Public Key-based Authentication



You can generate the key pair on each of the SSH clients. You must then copy the public keys to the router. When the user tries to log in to the router, the SSH client sends a connection request with a signature created using the private key. The router then checks the authenticity of the request by matching it against the public keys associated with that user in its database. Suppose one of the multiple public keys associated with that user matches the signature; the router authenticates such user, confirming that the user has the corresponding private key. This process thereby establishes a secure SSH connection to the router.

Guidelines and Restrictions for Public key-based authentication to Routers

- You can add public keys by importing the public key file or directly adding the public keystring to the router.
- The maximum number of public keys supported per user is four.

- The router supports importing or adding only one public key at a time. Even though the router supports up to four keys per user, you can only import or add them to the router one after the other and not simultaneously.
- To import the public key files to the router, use the **crypto key import authentication rsa** command.
- The router supports importing public keys in the following formats:
  - RSA
  - Base 64
  - PEM PKCS1
  - PEM PKCS8
- To delete the public key files in the router, use the **crypto key zeroize authentication rsa** command.
- You can import the public keys using the **crypto key import authentication rsa** command in the XR Config mode and XR EXEC mode. However, use the same operation mode to import and delete the public keys. That is, if you import the public keys in the XR Config mode, delete such keys in XR Config mode only. Similarly, if you import the public keys in the XR EXEC mode, delete such keys in XR EXEC mode only.
- You can use SSH configurations to add or delete a public key in the router.
- The router supports only the RSA key format while using SSH configurations to add a public key for public key-based authentication to the router.

Configure Public key-based Authentication to Routers

This section details different methods of enabling flexible public key-based authentication and importing public keys to the router:

Configurations

Using public-key import:

1. [Router] Create a user in the router:

```
Router# config
Router(config)# username testuser1
Router(config)# commit
```

2. [Client] Generate RSA key pairs on the SSH clients.
3. [Router] Copy the public keys from the clients to the router.

Note You can skip step 3 while using the tftp filepath in step 4. For more details, **crypto key import authentication rsa** command.

4. [Router] Import public keys to the router:

```
Router# configure
Router(config)# crypto key import authentication rsa username testuser1
```

```
disk0:/id_rsa_key1.pub
Router(config)# crypto key import authentication rsa username testuser1 second
disk0:/id_rsa_key2.pub
Router(config)# crypto key import authentication rsa username testuser1 third
disk0:/id_rsa_key3.pub
Router(config)# crypto key import authentication rsa username testuser1 fourth
disk0:/id_rsa_key4.pub
Router(config)# commit
```

You can now access the router from any of the four SSH clients using the same user.

5. [Client] Access the router in the client:

```
[root@userclient test]# ssh testuser1@192.0.2.2
```

Using SSH configurations:

1. [Router] Create a user in the router:

```
Router# config
Router(config)# ssh server username testuser2
Router((config-user-key))# commit
```

2. [Client] Generate RSA key pair on the SSH clients.
3. [Router] Add public keys from the SSH clients for a user to the router:

```
Router# configure
Router(config)# ssh server username testuser2
Router(config-user-key)# keystring ssh-rsa
<redacted>
Router(config-user-key)# keystring ssh-rsa second
<redacted>
Router(config-user-key)# keystring ssh-rsa third
<redacted>
Router(config-user-key)# keystring ssh-rsa fourth
<redacted>
Router(config)# commit
```

You can now access the router from any of the four SSH clients using the same user.

4. [Client] Access the router in the client:

```
[root@userclient test]# ssh testuser2@192.0.2.2
```

Verification

Public-key import:

```
Router# show crypto key authentication rsa testuser1 all
Wed Sep 20 16:28:09.114 IST
Key label: testuser1firstkey
Type     : RSA Signature
Size     : 768
Created  : 16:27:54 IST Wed Sep 20 2023
Data     :
 307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00BDD9A2 B8D61FA3
 AED1B6EC FB975512 32BFE99E 65FDCC01 FA14956C 7B06C2A5 CEE9E637 56FE38F6
 878ED2F4 CD1C1F28 3F535F23 9F5F8763 19BA0269 DA7B2507 0160A28B 7CD1A66D
 75DF194B C217402E 7E74D466 4E39177B 81051774 25A71A0A 0F020301 0001
```

```
Key label: testuser1secondkey
Type     : RSA Encryption
Size     : 768
Created  : 16:27:54 IST Wed Sep 20 2023
Data     :
 307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00B87C2F 9B4972AC
 47B40FB2 B5C10DBB 1205AD30 7E146698 2A6179AD 8F1B030D 5146C097 3A2FB3E2
 19820DA5 2132E7C7 1B7281C4 8427DF76 60E39E3A 70126DAD 108B7805 34B45915
 853956AA 301CCF4B 78F06D75 D7D90320 BE667F1D 1A479713 FD020301 0001

Key label: testuser1thirdkey
Type     : RSA General purpose
Size     : 768
Created  : 16:27:57 IST Wed Sep 20 2023
Data     :
 307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00E0DDF9 53C81AE1
 35CE15E1 C7A9916F 4AED7887 65AC1E4E 48F420E4 2A56079E FD38D069 C97FC0F7
 B6D8663D C7D6FC46 1CD27EA6 AC71D36C 40E35349 0A78DA64 465B7C8B B63E8627
 BF074AF4 EC37AC0C 200AFAF3 C67E8E9B AE931964 8DF86CD9 E5020301 0001

Key label: testuser1fourthkey
Type     : RSA General purpose
Size     : 768
Created  : 16:27:57 IST Wed Sep 20 2023
Data     :
 307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00E0DDF9 53C81AE1
 35CE15E1 C7A9916F 4AED7887 65AC1E4E 48F420E4 2A56079E FD38D069 C97FC0F7
 B6D8663D C7D6FC46 1CD27EA6 AC71D36C 40E35349 0A78DA64 465B7C8B B63E8627
 BF074AF4 EC37AC0C 200AFAF3 C67E8E9B AE931964 8DF86CD9 E5020301 0001
```

SSH configurations:

```
Router# show ssh
SSH version : Cisco-2.0

id       chan pty     location        state           userid           host        ver
authentication     connection type
-----------------------------------------------------------------------------------------
Incoming sessions
26       1    vty1    0/RP0/CPU0      SESSION_OPEN    testuser1     192.0.2.1       v2
 rsa-pubkey     Command-Line-Interface
27       1    vty2    0/RP0/CPU0      SESSION_OPEN    testuser1     192.0.2.2       v2
 rsa-pubkey     Command-Line-Interface
28       1    vty3    0/RP0/CPU0      SESSION_OPEN    testuser1     192.0.2.3       v2
 rsa-pubkey     Command-Line-Interface
29       1    vty4    0/RP0/CPU0      SESSION_OPEN    testuser1     192.0.2.4       v2
 rsa-pubkey     Command-Line-Interface

Outgoing sessions
1                    0/RP0/CPU0      SESSION_OPEN    testuser3  192.0.2.6          v2
  password      Command-Line-Interface
```

Delete Public Keys in the Routers

This section details different methods to delete public keys in the router:

```
Router# configure
Router(config)# crypto key zeroize authentication rsa all
Thu Sep 21 21:45:23.260 IST
Do you really want to remove all these keys ?? [yes/no]: yes
Router# commit
/* Deleting public keys for the user logged in to the router */
```

```
Router# configure
Router(config)# crypto key zeroize authentication rsa username testuser all
Thu Sep 21 21:45:23.260 IST
Do you really want to remove all these keys ?? [yes/no]: yes
Router# commit
/* Deleting public keys for any user in the router */

Router# configure
Router(config)#  no ssh server username testuser
Router# commit
/* Deleting all SSH configurations for a user in the router */

Router# configure
Router(config)# no ssh server username testuser keystring third
Router# commit
/* Deleting a specific public-key for a user using SSH configurations in the router */
```

Multi-Factor Authentication for SSH

Table 18: Feature History Table

Feature Name	Release Information	Feature Description
Multi-Factor Authentication for SSH	Release 24.1.1	You can now deploy robust authentication mechanisms for SSH connections to your routers and reduce security risks due to compromised or weak passwords. We now support multi-factor authentication (MFA)—a secure access management solution that verifies the identity of a user using multiple verification factors—for SSH login on Cisco IOS XR routers. These verification factors include a combination of login credentials such as username and password and a token, a cryptographic device, or a mobile phone with MFA application installed. No new commands or data models were introduced or modified as part of this feature.

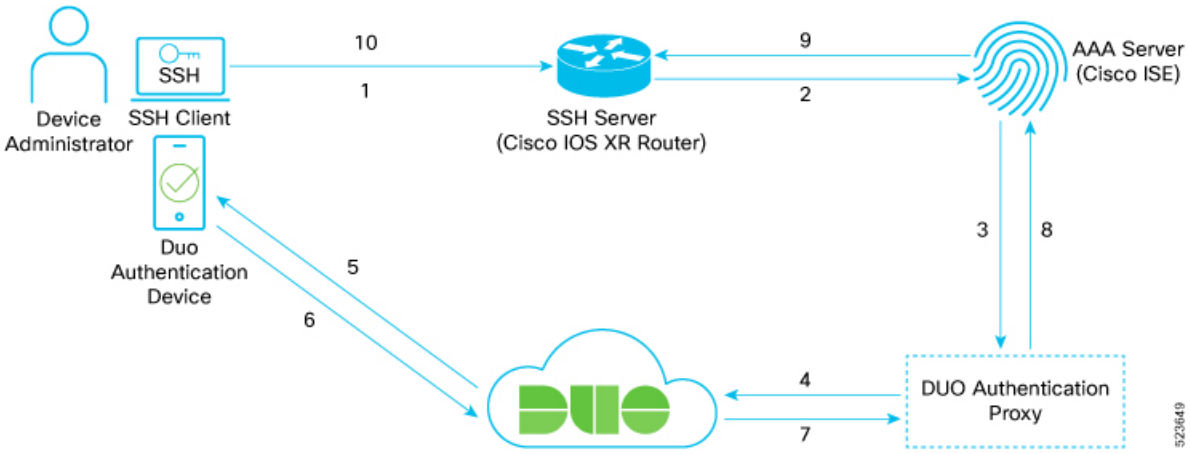
Multi-factor authentication is a multi-step authentication process that requires users to enter two or more verification factors to gain access to a system. These verification factors include something you know—such as a username and a password, and something you have—such as a token, a cryptographic authentication device, or a mobile phone with MFA application installed. MFA thereby enables stronger authentication mechanism and reduces security risk to the network devices arising due to compromised or weak passwords.

To achieve MFA for SSH, the SSH server as well as the client must support keyboard-interactive authentication method. The default order of SSH client authentication methods to support MFA in Cisco IOS XR routers is public-key, keyboard-interactive, and password-based authentication. You can change this default order as per your requirement using the **ssh client auth-method** command.

Multi-Factor Authentication Workflow

This is a sample topology to demonstrate the MFA workflow to establish SSH connection on a Cisco IOS XR router. In this example we have considered Cisco IOS XR router as the SSH server, Cisco ISE as the AAA server, and Cisco DUO authentication proxy and cloud services for MFA.

Figure 5: Multi-Factor Authentication Set-up for SSH Connection: Sample Topology



Key Components

The key components in this sample Duo MFA topology for SSH include:

- SSH client—from where the admin user initiates SSH connection to the SSH server.
- SSH server—which is the network device or router to which SSH connection is to be established.
- Cisco identity services engine (ISE)—that acts as the RADIUS or TACACS+ Server for AAA.
- DUO authentication proxy—is an on-premises software service that receives authentication requests from your local devices and applications through RADIUS or LDAP, optionally performs primary authentication against your existing LDAP directory or RADIUS authentication server, and then contacts Duo to perform secondary authentication.
- DUO cloud service—Cisco cloud-based security platform that provides secure access to any device or application.
- DUO authentication device—such as a mobile phone which has the Duo application installed.

The detailed workflow of Duo MFA for SSH is as follows:

1. The admin user initiates an SSH connection to the SSH server (Cisco IOS XR router, in this case) using the login credentials of the users that are already configured on ISE.
2. The router forwards the request to the TACACS+ AAA server (Cisco ISE, in this case).
3. The Cisco ISE sends the authentication request to Duo authentication proxy. The proxy forwards the request back to ISE for the 1st factor authentication. ISE informs the authentication proxy if the local authentication was successful.
4. Upon successful ISE authentication, the authentication proxy sends an authentication request to Duo cloud for 2nd factor authentication.

5. Duo cloud sends a *PUSH* notification to the DUO authentication device of the admin user.
6. The admin user approves the *PUSH* notification.
7. The Duo cloud informs the authentication proxy of the successful *PUSH* notification.
8. The authentication proxy informs ISE of a successful authentication.
9. The ISE authorizes the admin user.
10. The admin user successfully establishes an SSH connection with the router.

Set Up Multi-Factor Authentication for SSH

This section describes how to set up a sample topology for establishing SSH connection with Cisco IOS XR router using Duo MFA.

Prerequisites

- The Cisco IOS XR router installed with Cisco IOS XR Software Release 24.1.1 or later, that acts as the server to the SSH client, and as the client to the ISE server. The router must be already configured for AAA with ISE.
- Cisco identity services engine (ISE) server that acts as the RADIUS or TACACS+ AAA server.
- Duo MFA proxy application must be installed on either Windows or on Linux machine. For details, see https://duo.com/docs/authproxy-reference.
- DUO application must be installed on the DUO authentication device.

The procedure to set up MFA for SSH involves these high-level tasks:

- Configure Duo System
- Configure Duo Authentication Proxy
- Configure ISE
- Configure RADIUS Server Attributes on the Router
- Verify Duo MFA Set-up

Configure Duo System for MFA

Configuring Duo system for MFA involves these key steps:

1. Create a Duo account in https://duo.com/
2. Perform these Duo system configurations (for details, see the *First Steps* listed in https://duo.com/docs/radius):
   - Login to your Duo account and click on **Applications**.
   - Search for **Cisco ISE server** and click on **Protect This Application**.
   - In a notepad copy and paste your **Integration Key**, **Secret Key**, and **API Hostname**.

3. Add Duo mobile device:

   Select **Dashboard** > **Users** > ***username*** > **Add Phone**

4. Activate Duo mobile:

   Select **Dashboard** > **2FA Devices** > ***phone-number*** > **Activate Duo Mobile**

Configure Duo Authentication Proxy for MFA

Configuring Duo authentication proxy for MFA involves these key steps (For more details, see https://duo.com/docs/authproxy-reference)

1. Download and install the latest Duo authentication proxy on your Windows or Linux machine.

   In this example, we have installed the primary authentication proxy on a Windows 2016 machine and the secondary proxy on an Ubuntu server.

2. Configure the proxy for your primary authenticator.

   Edit the Duo authentication proxy configuration file, authproxy.cfg, located in the conf subdirectory of the proxy installation path in the server using a text editor. You can add multiple ISE servers as RADIUS clients and multiple router subnets/IP addresses as part of the router.

3. Start the proxy server(s) and check the proxy logs for any configuration or connectivity error.

Note For installation on Windows, ensure sure that the Windows firewall is configured to allow connections for the authentication proxy.

Configure ISE for MFA

Configuring ISE for MFA involves these key steps (for more details, see Configure Duo Two Factor Authentication for ISE Management Access)

1. Integrate ISE with Duo authentication proxy:

   a. Add a new RADIUS token server:

      Administration > **Identity Management** > **External Identity Sources** > **RADIUS Token**, and click **Add**

      Ensure that the **Shared Secret** matches the one that you already defined in the *Configure Duo Authentication Proxy* task.

      For details, see step1 listed under ISE Configuration.

   b. Set the authentication method for the identity source:

      Navigate to **Administration** > **System** > **Admin Access** > **Admin Access** > **Authentication Method**, and select previously configured RADIUS token server (for example, **RADIUS:DUO**) as the **Identity Source**.

      For details, see Step 2 listed under ISE Configuration.

2. Create device admin policies:

a. Create a policy set:

   Navigate to **Work Centers** > **Device Administration** > **Device Admin Policy Sets**.

   In this example, we created a policy set that matches on both protocols (RADIUS and TACACS+) with the **Allowed Protocols** set to **Default Device Admin**.

b. Set the following policies inside the policy set:

   - **Authentication Policy**: In this example, we have set a default rule to check the Identity Source Sequence that we defined in the steps above which contains the RADIUS Token Servers (Duo Authentication Proxies) and Active Directory.
   - **Authorization Policy**: In this example, we have set a rule that checks if the authenticated user belongs either to the **Domain Users** or **NS-ISE-IOS-Admins** groups that we have configured in active directory (AD). If the user belongs to one of these groups, then the system returns the pre-configured **Command Sets** and **Shell Profile**.

3. Add and onboard users in Duo:

   You can configure Duo to automatically sync with your AD or manually add the user in Duo (for details, see Enroll user with Duo).

Configure RADIUS Server Attributes for MFA

This topic describes how to configure RADIUS server attributes for MFA on the Cisco IOS XR router (for more details, see configure-your-radius-client(s)).

Set the IP address of the RADIUS server to the IP address of your authentication proxy, the RADIUS server port to 1812, and the RADIUS secret to the appropriate secret that you configured in the *radius_server_auto* section in the authproxy.cfg file.

```
Router#configure
Router(config)#radius-server host 209.165.200.225auth-port 1812 acct-port 1813
Router(config-radius-host)#key test@1234
Router(config-radius-host)#commit
```

Verify MFA Set-up for SSH Connection

Once you complete the Duo MFA configurations, follow these steps to verify the set-up:

- Initiate an SSH connection from the SSH client router that is already added in the ISE, using the **ssh** command.
- Use the AD credentials for the admin user to log in.
- Upon successful authentication, confirm that the user received a **Duo Push/Passcode** notification on the Duo authentication device based on what is set in the Duo authentication proxy configuration file, authproxy.cfg.
- After approving the **Duo Push** or entering the correct Passcode, the admin user must be authenticated and authorized to access the router through the SSH connection.
- The live logs of RADIUS in the ISE server must show authentication requests against the Duo authentication proxies.

- Check the authproxy log file in your authentication proxy for any errors or issues.

DSCP Marking for SSH Packets

Table 19: Feature History Table

Feature Name	Release Information	Feature Description
DSCP Marking from TCP Connection Phase for SSH Packets	Release 24.1.1	We now prevent SSH client packet drops in the TCP connection (initial handshake) phase as they travel across transit routers in the network. This is because you can mark the DSCP values for SSH client packets in the TCP connection phase, which overrides the transit routers' policies to filter and drop packets with no DSCP value marked. Using a new command, you can also set the DSCP value from the TCP connection phase for SSH server packets. The feature introduces these changes: **CLI:** • **ssh server set-dscp-connection-phase** **YANG Data Model:** • New XPath, `set-dscp-connection-phase`, for `Cisco-IOS-XR-crypto-ssh-cfg.yang` (see GitHub, YANG Data Models Navigator)

Supported Platforms:

- N540-24Q8L2DD-SYS
- N540X-16Z4G8Q2C-D
- N540X-16Z4G8Q2C-A
- N540X-16Z8Q2C-D
- N540-28Z4C-SYS-D

- N540-28Z4C-SYS-A
- N540X-12Z16G-SYS-D
- N540X-12Z16G-SYS-A
- N540-12Z20G-SYS-D
- N540-12Z20G-SYS-A
- N540X-4Z14G2Q-D
- N540X-4Z14G2Q-A
- N540X-8Z16G-SYS-D
- N540X-8Z16G-SYS-A
- N540X-6Z18G-SYS-D
- N540X-6Z18G-SYS-A
- N540-6Z18G-SYS-D
- N540-6Z18G-SYS-A
- N540-6Z14S-SYS-D
- N540-FH-AGG-SYS
- N540-FH-CSR-SYS

CiscoSSH is based on OpenSSH version 8.0 in which the the DSCP marking of the SSH packets happens only after the authentication phase of SSH session establishment. Hence, the SSH packets originating from the CiscoSSH routers did not have the DSCP value set in the initial handshake or the TCP connection phase. This led to SSH packet drops during the TCP connection phase if routers in the transit network have specific rules or filters to drop packets with zero or incorrect DSCP value.

From OpenSSH version 8.5 and later, the DSCP marking of SSH client packets happens from the TCP connection phase itself. Cisco IOS XR Software Release 24.1.1 brings in this behavior change for DSCP marking of SSH client packets into CiscoSSH. Whereas there is no change in behavior of the DSCP marking for SSH server packets. The CiscoSSH routers that function as SSH servers continue to mark the DSCP value for the packets only after the authentication phase. You can use the **ssh server set-dscp-connection-phase** command to set the DSCP value for the SSH server packets from the TCP connection phase.

Set DSCP Marking for SSH Packets from TCP Connection Phase

To set the DSCP marking for SSH server packets from TCP connection phase, use the **ssh server set-dscp-connection-phase** command in XR Config mode.

Note Although the **ssh server set-dscp-connection-phase** command is available on routers with CiscoSSH and routers with Cisco IOS XR SSH, this configuration is relevant only on routers with CiscoSSH.

Configuration Example

```
Router#configure
Router(config)#ssh server set-dscp-connection-phase
Router(config-ssh)#commit
```

Running Configuration

```
Router#show run ssh
!
ssh server set-dscp-connection-phase
!
```

SSH server timeouts

Unused connection timeout for SSH sessions

An unused connection timeout is a secure shell (SSH) configuration setting that

- terminates SSH connections with no active channels after a specified period
- prevents accumulation of stale connections, and
- protects routers from reaching SSH session limits due to inactive sessions.

An SSH connection is a secure communication session that

- establishes an encrypted link between a client and a server
- enables authentication and data exchange, and
- serves as the foundation for creating multiple SSH channels.

Table 20: Feature History Table

Feature Name	Release Information	Feature Description
Unused connection timeout for SSH sessions	Release 25.3.1	You can prevent session limit exhaustion and maintain optimal system performance by automatically disconnecting SSH connections with no active channels. The feature introduces a configurable timeout for unused SSH connections, ensuring stale sessions do not occupy resources on your routers. The router monitors each SSH connection and terminates it when all channels remain closed and SSH clients do not create new channels within the configured timeout period. The feature introduces these changes: CLI: • **ssh server timeout** YANG Data Models: • `Cisco-IOS-XR-crypto-ssh-cfg.yang` data model was modified. • `Cisco-IOS-XR-um-ssh-cfg.yang` data model was modified. (see GitHub)

SSH connections form the basis for secure management and file transfer sessions in Cisco IOS XR routers. These connections persist after successful authentication and host one or more SSH channels for various operations. When all channels within a connection close and no new channels open, the unused connection timeout begins counting down.

The unused connection timeout functionality addresses issues where client automation scripts fail to properly close SSH connections, causing stale connections that remain active indefinitely and consume session resources.

Set unused connection timeout for SSH sessions

Follow this procedure to enable automatic disconnection of unused SSH connections on your routers to enhance device security and resource management.

Procedure

Step 1 Configure connection timeout for the unused SSH sessions on your router.

Example:

```
Router#configure
Router(config)#ssh server timeout connection 600
Router(config)#commit
```

The timeout begins counting down when all channels within a connection close and no new channels open.

Step 2 Verify the unused connection timeout configuration on the router.

Example:

```
Router#show run ssh
Thu Jul  3 09:45:41.201 UTC
ssh server v2
ssh server timeout
 connection 600
!
ssh server netconf vrf default
Router#


Router#show ssh server
Mon Aug 11 13:37:15.363 IST
---------------------
SSH Server Parameters
---------------------

Current supported versions := v2
                  SSH port := 22
                  SSH vrfs := vrfname:=default(v4-acl:=, v6-acl:=)
              Netconf Port := 830
              Netconf Vrfs := vrfname:=default(v4-acl:=, v6-acl:=)
        Netconf on SSH-port:= True (Netconf requests allowed on SSH port)

 Algorithms
---------------
        Hostkey Algorithms :=
x509v3-ssh-rsa,ssh-rsa-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,


ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa,ssh-dss

   Key-Exchange Algorithms :=
ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group14-sha1,curve25519-sha256,


diffie-hellman-group14-sha256,diffie-hellman-group16-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

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

 Others
------------
                     DSCP  := 48
```

```
              Ratelimit  := 6000
           Sessionlimit  := 110
              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  :=
    OpenSSH HIBA enabled  := No
       OpenSSH HIBA role  :=
            HIBA version  := 1
Punt Rx SSH pkts to NetIO := False
         Channel Timeout  := 300
      Connection Timeout  := 600
```

Step 3 Check the SSH session details on the router to confirm that the unused SSH session disconnects after the specified period of inactivity.

Example:

This sample command output shows that there is an active channel and the connection timeout has not yet triggered.

```
Router#show ssh
Mon Aug 11 08:10:39.133 UTC
SSH version : Cisco-2.0

id       chan pty     location        state           userid    host          ver
authentication connection type
-----------------------------------------------------------------------------------------------------------
Incoming sessions
2         0   vty0    0/RP0/CPU0      SESSION_OPEN    cisco     198.51.100.1  v2  password
       Command-Line-Interface

Outgoing sessions
```

The connection timeout triggers when there is no active channel. The **connection type** field then displays blank.

```
Router#show ssh
Mon Aug 11 08:10:47.675 UTC
SSH version : Cisco-2.0

id       chan pty     location        state           userid    host          ver
authentication connection type
-----------------------------------------------------------------------------------------------------------
Incoming sessions
2         0   XXXXX   0/RP0/CPU0      SESSION_OPEN    cisco     198.51.100.1  v2  password


Outgoing sessions
Router#
```

This is the sample output when the connection timeout expires and entire connection closes.

```
Router#show ssh
Mon Aug 11 08:10:53.097 UTC
```

```
SSH version : Cisco-2.0

id       chan pty     location        state           userid    host                  ver
authentication connection type
-----------------------------------------------------------------------------------------
Incoming sessions

Outgoing sessions
Router
```

Step 4 Check the system logs on the router console that indicates the termination of the inactive SSH sessions.

Example:

```
RP/0/RP0/CPU0:Aug 11 08:10:51.557 UTC: ssh_syslog_proxy[1215]:
%SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[42522]: terminating inactive connection
from user user1 198.51.100.1 port 25372
```

The router automatically disconnects unused SSH connections after the configured connection timeout period.

Channel timeout for SSH sessions

A channel timeout is a secure shell (SSH) configuration setting that

- closes an SSH channel if it remains idle for a specific duration
- applies to all types of channels, including Shell, SFTP, and Netconf, and
- ensures unused channels are closed promptly.

An SSH channel is a virtual communication path within an SSH connection that

- allows concurrent operations such as shell access or file transfer
- operates independently within its parent SSH connection, and
- can be opened and closed without affecting the overall connection.

SSH channels allow multiple management or file operations to run concurrently over a single SSH connection on the router.

Table 21: Feature History Table

Feature Name	Release Information	Feature Description
Channel timeout for SSH sessions	Release 25.3.1	You can improve resource efficiency and minimize potential security risks by automatically closing idle SSH channels on the routers after a specific period of inactivity. The feature introduces a configurable timeout for SSH channels which ensures that unused channels do not persist while the parent SSH connection remains active. The router monitors each SSH channel and closes any channel where no data is sent or received within the configured timeout period. The feature introduces these changes: CLI: • **ssh server timeout** YANG Data Models: • `Cisco-IOS-XR-crypto-ssh-cfg.yang` data model was modified. • `Cisco-IOS-XR-um-ssh-cfg.yang` data model was modified. (see GitHub)

The SSH client opens one or more SSH channels such as Shell, SFTP, and so on, within the SSH connection as needed. The channel timeout begins counting down after a specific period of inactivity.

Channel timeouts help enforce resource efficiency by closing idle channels, while the parent SSH connection remains active.

Set channel timeout for SSH sessions

Follow this procedure to enable automatic disconnection of idle SSH channels belonging to active SSH sessions on your routers to enhance device security and resource management.

Procedure

Step 1 Configure channel timeout for the SSH channels on your router.

Example:

```
Router#configure
Router(config)#ssh server timeout channel 300
Router(config)#commit
```

The timer expires when the SSH channel remains idle for the specified timeout period.

Step 2 Verify the SSH channel timeout configuration on the router by running these commands.

Example:

```
Router#show run ssh
Thu Jul  3 09:45:41.201 UTC
ssh server v2
ssh server timeout
 channel 300
 connection 600
!
ssh server netconf vrf default
Router#


Router#show ssh server
```

See the sample command output here.

Step 3 Check the SSH session details on the router to confirm if the idle SSH channel closes after the specified period of inactivity.

Example:

Before timer expiry:

```
Router#show ssh
Thu Jul  3 09:46:29.807 UTC
SSH version : Cisco-2.0

id       chan pty     location        state           userid    host             ver
authentication connection type
-----------------------------------------------------------------------------------------------
Incoming sessions
8         0   vty0    0/RP0/CPU0      SESSION_OPEN    user1     198.51.100.1     v2  password
       Command-Line-Interface

Outgoing sessions
Router#
```

After timer expiry:

```
Router#show ssh
Thu Jul  3 09:51:33.818 UTC
SSH version : Cisco-2.0

id       chan pty     location        state           userid    host             ver
authentication connection type
-----------------------------------------------------------------------------------------------
Incoming sessions
8         0   XXXXX   0/RP0/CPU0      SESSION_OPEN    user1     198.51.100.1     v2  password


Outgoing sessions
Router#
```

Step 4 Check the system logs on the router console that indicates the closure of the idle SSH channels.

Example:

```
RP/0/RP0/CPU0:ios#RP/0/RP0/CPU0:Jul  3 09:46:32.002 UTC: ssh_syslog_proxy[1222]:
%SECURITY-SSHD_SYSLOG_PRX-6-INFO_GENERAL : sshd[39648]: Closing channel 0 of user user1
198.51.100.1 port 34454 after 300 seconds of inactivity
```

The router automatically closes inactive SSH channels after the configured channel timeout period.