**openSSH Secure shell**

## **What is openssh ?**

OpenSSH is the premier connectivity tool for remote login with the SSH protocol. It encrypts all traffic to eliminate eavesdropping, connection hijacking, and other attacks. In addition, OpenSSH provides a large suite of secure tunneling capabilities, several authentication methods, and sophisticated configuration options.

Note, the OpenSSH packages require the OpenSSL package **openssl-libs**, which installs several important cryptographic libraries, enabling OpenSSH to provide encrypted communications

The OpenSSH suite consists of the following tools:

* Remote operations are done using [ssh](https://man.openbsd.org/ssh.1), [scp](https://man.openbsd.org/scp.1), and [sftp](https://man.openbsd.org/sftp.1).
* Key management with [ssh-add](https://man.openbsd.org/ssh-add.1), [ssh-keysign](https://man.openbsd.org/ssh-keysign.8), [ssh-keyscan](https://man.openbsd.org/ssh-keyscan.1), and [ssh-keygen](https://man.openbsd.org/ssh-keygen.1).
* The service side consists of [sshd](https://man.openbsd.org/sshd.8), [sftp-server](https://man.openbsd.org/sftp-server.8), and [ssh-agent](https://man.openbsd.org/ssh-agent.1).

## **What is SSH ?**

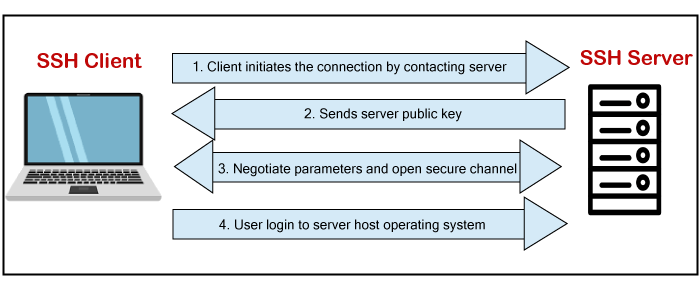
SSH (Secure Shell) is a protocol which facilitates secure communications between two systems using a client-server architecture and allows users to log in to server host systems remotely.

Unlike other remote communication protocols, such as FTP or Telnet, SSH encrypts the login session, rendering the connection difficult for intruders to collect unencrypted passwords.

A related program called scp replaces older programs designed to copy files between hosts, such as rcp.

## **How does SSH work ??**

* In order to establish an SSH connection, you need two components: a client and the corresponding server-side component. An SSH client is an application you install on the computer which you will use to connect to another computer or a server. The client uses the provided remote host information to initiate the connection and if the credentials are verified, establishes the encrypted connection.
* On the server’s side, there is a component called an SSH daemon that is constantly listening to a specific TCP/IP port for possible client connection requests. Once a client initiates a connection, the SSH daemon will respond with the software and the protocol versions it supports and the two will exchange their identification data. If the provided credentials are correct, SSH creates a new session for the appropriate environment.
* The default SSH protocol version for SSH server and SSH client communication is version 2.



## **Event Sequence of an SSH Connection**

The following series of events help protect the integrity of SSH communication between two hosts.

1. A cryptographic handshake is made so that the client can verify that it is communicating with the correct server.
2. The transport layer of the connection between the client and remote host is encrypted using a symmetric cipher.
3. The client authenticates itself to the server.
4. The client interacts with the remote host over the encrypted connection.

#### **Transport Layer**

The primary role of the transport layer is to facilitate safe and secure communication between the two hosts at the time of authentication and during subsequent communication. The transport layer accomplishes this by handling the encryption and decryption of data, and by providing integrity protection of data packets as they are sent and received. The transport layer also provides compression, speeding the transfer of information.

Once an SSH client contacts a server, key information is exchanged so that the two systems can correctly construct the transport layer. The following steps occur during this exchange:

* Keys are exchanged
* The public key encryption algorithm is determined
* The symmetric encryption algorithm is determined
* The message authentication algorithm is determined
* The hash algorithm is determined

During the key exchange, the server identifies itself to the client with a unique *host key*. If the client has never communicated with this particular server before, the server’s host key is unknown to the client and it does not connect.

OpenSSH gets around this problem by accepting the server’s host key. This is done after the user is notified and has both accepted and verified the new host key. In subsequent connections, the server’s host key is checked against the saved version on the client, providing confidence that the client is indeed communicating with the intended server

If, in the future, the host key no longer matches, the user must remove the client’s saved version before a connection can occur.

SSH is designed to work with almost any kind of public key algorithm or encoding format. After an initial key exchange creates a hash value used for exchanges and a shared secret value, the two systems immediately begin calculating new keys and algorithms to protect authentication and future data sent over the connection.

After a certain amount of data has been transmitted using a given key and algorithm (the exact amount depends on the SSH implementation), another key exchange occurs, generating another set of hash values and a new shared secret value. Even if an attacker is able to determine the hash and shared secret value, this information is only useful for a limited period of time.

#### **Authentication**

Once the transport layer has constructed a secure tunnel to pass information between the two systems, the server tells the client the different authentication methods supported, such as using a private key-encoded signature or typing a password. The client then tries to authenticate itself to the server using one of these supported methods.

SSH servers and clients can be configured to allow different types of authentication, which gives each side the optimal amount of control. The server can decide which encryption methods it supports based on its security model, and the client can choose the order of authentication methods to attempt from the available options

#### Channels

After a successful authentication over the SSH transport layer, multiple channels are opened via a technique called *multiplexing*[[1]](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-openssh#ftn.idm140646274602128). Each of these channels handles communication for different terminal sessions and for forwarded X11 sessions.

Both clients and servers can create a new channel. Each channel is then assigned a different number on each end of the connection. When the client attempts to open a new channel, the clients sends the channel number along with the request. This information is stored by the server and is used to direct communication to that channel. This is done so that different types of sessions do not affect one another and so that when a given session ends, its channel can be closed without disrupting the primary SSH connection.

Channels also support *flow-control*, which allows them to send and receive data in an orderly fashion. In this way, data is not sent over the channel until the client receives a message that the channel is open.

The client and server negotiate the characteristics of each channel automatically, depending on the type of service the client requests and the way the user is connected to the network. This allows great flexibility in handling different types of remote connections without having to change the basic infrastructure of the protocol.

## **OpenSSH installation**

*$ dnf install -y openssh*

***# installs both ssh server & client package***

***# Install the SSH server package***

*$ dnf install -y openssh*

*# Start and enable SSH daemon*

*$ systemctl start sshd*

*$ systemctl enable sshd*

*$ systemctl status sshd*

*$ dnf install -y openssh-clients*

***# installs ssh client package***

***#There are no services to start for OpenSSH clients***

## **SSH Commands**

**Syntax**

*$ ssh [option] [user]@[server\_address]*

*Options*

*-4/-6 Specify the ssh command that only IPv4(for -4) / IPv6 (for -6) addresses are permitted while creating an ssh connection.*

*-a This option is used to specify the ssh command that authentication agent connection forwarding should be disabled while creating an ssh connection. This is the default option for the ssh command.*

*-C Enable data compression while connecting over SSH. This can be helpful in case you are using a sluggish network*

*-p Specify a custom port number when the remote SSH server is not listening on the default port*

*-l login name*

*-q Quiet mode. Causes all warning and diagnostic messages to be suppressed.*

**Example**

*$ ssh remotehost/IP*

*#Create remote interative shell as the current user*

*$ ssh remote\_user@remotehost/IP*

*#Create remote interative shell as the current user*

*#****Executing remote commands on the server***

*$ ssh hostname command*

*$ ssh sample.ssh.com ls /tmp/doc*

## **SSH HOST KEYS**

* A host key is the server’s public key. The host key is used by the client to decrypt an authentication message sent from the server when connecting. The basic purpose of the host key is to ensure that when you connect to a remote host, it is actually the host that you intended to connect to.
* In Secure Shell, host keys can be used for host-based authentication.
* SSH clients store host keys for hosts they have ever connected to. These stored host keys are called *known host keys*, and the collection is often called *known hosts*. In OpenSSH, the collection of known host keys is stored in /etc/ssh/known\_hosts and in .ssh/known\_hosts in each user's home directory.
* SSH secures communication through public-key encryption. When an SSH client connects to an SSH server,
* It’s a protection against Man-In-The-Middle Attacks client & server.

**Host Key Generation**

* The host keys are usually automatically generated when an SSH server is installed but you can recreate them on the server with keygen.
* The host keys are usually automatically generated when an SSH server is installed. They can be regenerated at any time. However, if host keys are changed, clients may warn about changed keys. Changed keys are also reported when someone tries to perform a man-in-the-middle attack. Thus it is not advisable to train your users to blindly accept them. Changing the keys is thus either best done using an SSH key management tool that also changes them on clients, or using certificates.
* Host keys are just ordinary SSH key pairs. Each host can have one host key for each algorithm. The host keys are almost always stored in the following files:
* /etc/ssh/ssh\_host\_dsa\_key
* /etc/ssh/ssh\_host\_ecdsa\_key
* /etc/ssh/ssh\_host\_ed25519\_key
* /etc/ssh/ssh\_host\_rsa\_key

**$ sudo ssh-keygen -t rsa -f /etc/ssh/ssh\_host\_rsa\_key -C "RedHat 8"**

**Known HOST Keys [Server send to client]**

* SSH clients store host keys for hosts they have ever connected to. These stored host keys are called *known host keys.*
* *Stored in* ~/.ssh/known\_hosts
* *Fingerprints can be added to the known\_hosts file in multiple ways,*
* ***Automatically*** *when connecting to a server for the first time: When you connect to a server using SSH for the first time, if the fingerprint isn't already trusted, the SSH client will show a warning, and prompt you to accept or decline the fingerprint:*

*If you choose 'Yes', the full key fingerprint will be added to your ~/.ssh/known\_hosts file.*

* ***Manually****: You can also manually add fingerprints to your ~/.ssh/known\_hosts file.*
* In order to actually acquire the fingerprint to add to the file, you can use the ssh-keyscan command. This tool is included as part of OpenSSH on most Linux distributions, and is used to show the fingerprint(s) of a local or remote server
* **$ ssh-keyscan hostname # not recommended**

**Global Known Host :** /etc/ssh/ssh\_known\_hosts

**SSH KEY-BASED AUTHENTICATION**

**How Do SSH Keys Work?**

The SSH protocol employs a client-server model to authenticate two parties and encrypt the data between them.

The server component listens on a designated port for connections. It is responsible for negotiating the secure connection, authenticating the connecting party, and spawning the correct environment if the credentials are accepted.

The client is responsible for beginning the initial TCP handshake with the server, negotiating the secure connection, verifying that the server’s identity matches previously recorded information, and providing credentials to authenticate.

An SSH session is established in two separate stages. The first is to agree upon and establish encryption to protect future communication. The second stage is to authenticate the user and discover whether access to the server should be granted.

**Negotiating Encryption for the Session**

When a TCP connection is made by a client, the server responds with the protocol versions it supports. If the client can match one of the acceptable protocol versions, the connection continues. The server also provides its public host key, which the client can use to check whether this was the intended host.

At this point, both parties negotiate a session key using a version of something called the Diffie-Hellman algorithm. This algorithm (and its variants) make it possible for each party to combine their own private data with public data from the other system to arrive at an identical secret session key.

The session key will be used to encrypt the entire session. The public and private key pairs used for this part of the procedure are completely separate from the SSH keys used to authenticate a client to the server.

Session key >>>> Symetric

The generated secret is a symmetric key, meaning that the same key used to encrypt a message can be used to decrypt it on the other side. The purpose of this is to wrap all further communication in an encrypted tunnel that cannot be deciphered by outsiders.

After the session encryption is established, the user authentication stage begins.

**Authenticating the User’s Access to the Server**

The next stage involves authenticating the user and deciding access. There are a few different methods that can be used for authentication, based on what the server accepts.

The simplest is probably password authentication, in which the server simply prompts the client for the password of the account they are attempting to login with. The password is sent through the negotiated encryption, so it is secure from outside parties.

The most popular and recommended alternative is the use of SSH key pairs. SSH key pairs are asymmetric keys, meaning that the two associated keys serve different functions.

The public key is used to encrypt data that can only be decrypted with the private key. The public key can be freely shared, because, although it can encrypt for the private key, there is no method of deriving the private key from the public key.

Authentication using SSH key pairs begins after the symmetric encryption has been established as described in the last section. The procedure happens like this:

1. The client begins by sending an ID for the key pair it would like to authenticate with to the server.
2. The server check’s the authorized\_keys file of the account that the client is attempting to log into for the key ID.
3. If a public key with matching ID is found in the file, the server generates a random number and uses the public key to encrypt the number.
4. The server sends the client this encrypted message.
5. If the client actually has the associated private key, it will be able to decrypt the message using that key, revealing the original number.
6. The client combines the decrypted number with the shared session key that is being used to encrypt the communication, and calculates the MD5 hash of this value.
7. The client then sends this MD5 hash back to the server as an answer to the encrypted number message.
8. The server uses the same shared session key and the original number that it sent to the client to calculate the MD5 value on its own. It compares its own calculation to the one that the client sent back. If these two values match, it proves that the client was in possession of the private key and the client is authenticated.

As you can see, the asymmetry of the keys allows the server to encrypt messages to the client using the public key. The client can then prove that it holds the private key by decrypting the message correctly. The two types of encryption that are used (symmetric shared secret, and asymmetric public-private keys) are each able to leverage their specific strengths in this model.

**How to create ssh keys**

The first step to configure SSH key authentication to your server is to generate an SSH key pair on your local computer.

To do this, we can use a special utility called ssh-keygen, which is included with the standard OpenSSH suite of tools. By default, this will create a 2048 bit RSA key pair, which is fine for most uses.

**Each SSH key pair includes two keys:**

**A public key** that is copied to the SSH server(s). Anyone with a copy of the public key can encrypt data which can then only be read by the person who holds the corresponding private key. Once an SSH server receives a public key from a user and considers the key trustworthy, the server marks the key as authorized in its **authorized\_keys fil**e. Such keys are called authorized keys.

**A private key** that remains (only) with the user. The possession of this key is proof of the user's identity. Only a user in possession of a private key that corresponds to the public key at the server will be able to authenticate successfully. The private keys need to be stored and handled carefully, and no copies of the private key should be distributed. The private keys used for user authentication are called identity keys.

**SSH-KEYGEN**

*-t dsa | ecdsa | ed25519 | rsa | rsa1*

*Specifies the type of key to create. The possible values are ``rsa1'' for protocol version 1 and ``dsa'' ``ecdsa'' ``ed25519'' or ``rsa'' for protocol version 2.* ***rsa default***

*-b*

*Specifies the number of bits in the key to create. For RSA keys, the minimum size is 768 bits and the default is* ***2048*** *bits. 4096 is the next bit.*

*-C comment*

*-f U can specify the filename where you need to store the generated keys*

*-R*

*Removes all keys belonging to hostname from a known\_hosts file. This option is useful to delete hashed hosts*

*-v verbose*

*-p replace the existing passphrase*

*-N <new passphrase value> # generates new passphrase*

*Example*

*$ ssh-keygen -p*

*$ ssh-keygen -t rsa -b 4096 -f <path>*

*$ ssh-keygen -t rsa -b 4096 -N <newpassphrase> -f <file path>*

**SSH-COPY-ID**

**Once the ssh keys have been generated , they are stored by default in the .ssh/ directory of your home directory. Permission should be 600 on the private key and 644 on the publick key.**

***ssh-copy-id*** *utility copies public keys to a remote host's ~/.ssh/authorized\_keys file (creating the file and directory, if required).*

*-i file*

*Copy the public key contained in file. This option can be specified multiple times and can be combined with the -l option. If a private key is specified and a public key is found then the public key will be used.*

*-l*

*Copy the keys currently held by ssh-agent(1). This is the default if the -i option was not specified.*

*-p portnumber*

*EXIT STATUS*

*The ssh-copy-id utility exits 0 on success, and >0 if an error occurs.*

*EXAMPLES*

*To send a specific key to multiple hosts:*

*$ ssh-copy-id -i /path/to/keyfile.pub user@host1 user@host2*

*user@host3*

**Configuration Files**

Client program :  ssh, scp, and sftp

The server : sshd daemon

System-wide SSH configuration information is stored in the /etc/ssh/ directory

User-specific SSH configuration information is stored in ~/.ssh/ within the user’s home directory.

**Sytem wide configuration file**

|  |  |
| --- | --- |
| /etc/ssh/ssh\_config | The default SSH client configuration file. Note that it is overridden by ~/.ssh/config if it exists. |
| /etc/ssh/sshd\_config | The configuration file for the sshd daemon |
| /etc/sysconfig/sshd | Configuration file for the sshd service. |

**User-specific configuration files**

~/.ssh/authorized\_keys

Holds a list of authorized public keys for servers. When the client connects to a server, the server authenticates the client by checking its signed public key stored within this file.

~/.ssh/id\_rsa

The RSA private key used by ssh for version 2 of the SSH protocol.

~/.ssh/id\_rsa.pub

The RSA public key used by ssh for version 2 of the SSH protocol.

~/.ssh/authorized\_keys

Holds a list of authorized public keys for servers. When the client connects to a server, the server authenticates the client by checking its signed public key stored within this file.

~/.ssh/known\_hosts

Contains host keys of SSH servers accessed by the user. This file is very important for ensuring that the SSH client is connecting to the correct SSH server.