



# Architecture & Deployment

2025-2026 v0.1.0 on branch main Rev: 03b1cdace14bb0b0720e24be862097b3792214ea

## Secure Shell (SSH)

Learn about the SSH cryptographic network protocol and how to use the SSH command line tool to connect to other computers.

### Table of contents

- [Presentation](#)
- [The ssh command](#)
- [SSH known hosts](#)
  - [Are you really the SSH server I'm looking for?](#)
    - [How can I solve this problem?](#)
  - [Known hosts file](#)
    - [Adding public keys to the known hosts file](#)
    - [Preventing future man-in-the-middle attacks](#)
- [Password authentication](#)
- [Logging in with SSH](#)
  - [Typing commands while connected through SSH](#)
  - [Disconnecting](#)
  - [Where am I?](#)
- [Logging in or running a command](#)
- [Public key authentication](#)
  - [How does it work?](#)
  - [How does public key authentication work?](#)
  - [Using multiple keys](#)
  - [Key management](#)
  - [Key protection](#)
- [SSH for other network services](#)
- [The birth \(or death\) of an SSH connection](#)
- [References](#)

## You will need

- A Unix CLI

## Recommended reading

- [Command Line Introduction](#)

# The `ssh` command

---

The `ssh` command is available on most Unix-like systems (e.g. Linux & macOS) and in Unix shells on Windows (e.g. Git Bash or WSL). Its basic syntax is:

```
ssh [user@]hostname [command]
```

Here's a few examples:

- `ssh example.com` - Connect to the SSH server at `example.com` and log in (with the same username as in your current shell).
- `ssh jde@example.com` - Connect to the SSH server at `example.com` and log in as user `jde`.
- `ssh jde@192.168.50.4 hostname` - Run the `hostname` command as user `jde` on the SSH server at `192.168.50.4`.

Run `man ssh` to see available options (or just `ssh` in Git Bash).

# SSH known hosts

---

When you connect to an SSH server for the first time, you will most likely get a message similar to this:

```
$> ssh example.com
```

```
The authenticity of host 'example.com (192.168.50.4)' can't be established.  
ECDSA key fingerprint is SHA256:colYVucS/YU0JSK7woiLAf5ChPgJYAR1BWJlET2EwDI=  
Are you sure you want to continue connecting (yes/no)?
```

What does this mean? *I thought SSH was **secure**?*

## Are you really the SSH server I'm looking for?

---

As we've seen, when SSH establishes a secure channel, a Diffie-Hellman asymmetric key exchange will occur to agree on a secret symmetric encryption key. To secure this exchange, the server will perform an asymmetric digital signature so that no attacker can impersonate the server.

To verify the signature, **your SSH client will ask the server for its public key. This is where a man-in-the-middle attack is possible.** SSH warns you that someone is sending you a public key, but it has no way of verifying whether it's actually the server's public key, or whether it's the public key of an attacker performing a man-in-the-middle attack.

Basically, **SSH makes the following guarantee:**

- Once you have established a secure channel to a given server, no third party can decrypt your communications. Forward secrecy is also guaranteed in the event your credentials are compromised in the future.

**SSH does not guarantee** that:

- You are connecting to the correct server. You may be connecting to an attacker's server.

## How can I solve this problem?

---

If you are using SSH to transmit sensitive data, **you should check that the server's public key is the correct one before connecting.**

One way to do this is to use the **key fingerprint** that is shown to you when first connecting. The key fingerprint is a cryptographic hash of the public key:

```
ECDSA key fingerprint is SHA256:colYVucS/YU0JSK7woiLAf5ChPgJYAR1BWJlET2EwDI=
```

Some services that allow you to connect over SSH, like GitHub, publish their SSH key fingerprints on their website so that users may check them. In other cases, the key may be physically transmitted to you, or dictated over the phone.

You should **check that both fingerprints match** before proceeding with the connection. If it does not, either you typed the wrong server address, or an attacker may be trying to hack your connection.

## Known hosts file

---

If you accept the connection, SSH will save the server's address and public key in its **known hosts file**. You can see the contents of this file with the following command:

```
$> cat ~/.ssh/known_hosts
example.com,192.168.50.4 ecdsa-sha2-nistp256 eTJtK2wrRzhW5RQzUHprbFJa...
```

The format of each line in this file is `[domain], [ipaddr] algorithm pubkey`.

The line above means that when SSH connects to `example.com` at IP address `192.168.50.4`, it expects the server to send this specific public key `(eTJtK2wrRzhW5RQzUHprbFJa...)` using the ECDSA algorithm.

### More information

ECDSA is another asymmetric algorithm like RSA, although ECDSA is based on elliptic curve cryptography, while RSA is based on prime numbers.

## Adding public keys to the known hosts file

---

Another solution to SSH man-in-the-middle attacks when first connecting is to put the server's public key in the known hosts file yourself.

If you have previously obtained the server's public key (the full key, not just the fingerprint), you can **add it to the known hosts file** before attempting to connect.

If you do that, SSH will consider that the server is already a *known host*, and will not prompt you to accept the public key.

## Preventing future man-in-the-middle attacks

---

The known hosts file has another purpose. Once SSH knows to expect a specific public key for a given domain or IP address, it will warn you if that public key changes:

```
$> ssh 192.168.50.4
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:FUwFoK/hcqRAvJgDFmljw0ur8t/mhfbm4tfIxdaVTQ==.
Please contact your system administrator.
Add correct host key in /path/to/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /path/to/.ssh/known_hosts:33
ECDSA host key for 192.168.50.4 has changed and you have requested strict checking
Host key verification failed.
```

As the message mentions, either the server changed its SSH key pair, or **an attacker may be intercepting your communications**.

If you're sure it's not an attack, for example if you know the server actually changed its key pair, you can eliminate this warning by putting the correct public key in the known

hosts file (or by removing the offending line).

## Password authentication

---

Establishing a secure channel is one thing, but that only ensures an attacker cannot intercept communications. Once the channel is established, you must still **authenticate**, i.e. **prove that you are in fact the user you are attempting to log in as**.

How you authenticate depends on how the SSH server is configured. **Password authentication** is one method. When enabled, the SSH server will prompt you for the correct password; in this example, the password of the user named `jde` in the server's user database:

```
$> ssh jde@192.168.50.4
```

```
The authenticity of host '192.168.50.4 (192.168.50.4)' can't be established.  
ECDSA key fingerprint is SHA256:E4GYJCEoz+G5wv+EdkPyRLytgP7aTj9BS9lr1d38Xg==.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '192.168.50.4' (ECDSA) to the list of known hosts.
```

```
jde@192.168.50.4's password:
```

### Tip

Most SSH clients will not display anything while you type your password. Simply press `Enter` when you're done to submit it.

## Logging in with SSH

---

If you run the `ssh` command with no extra arguments and authenticate with your password, **SSH will run the default shell** configured for that user, typically Bash on Linux

servers:

```
$> ssh jde@192.168.50.4
jde@192.168.50.4's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-33-generic x86_64)

System information as of Wed Oct 21 04:29:00 UTC 2015
...

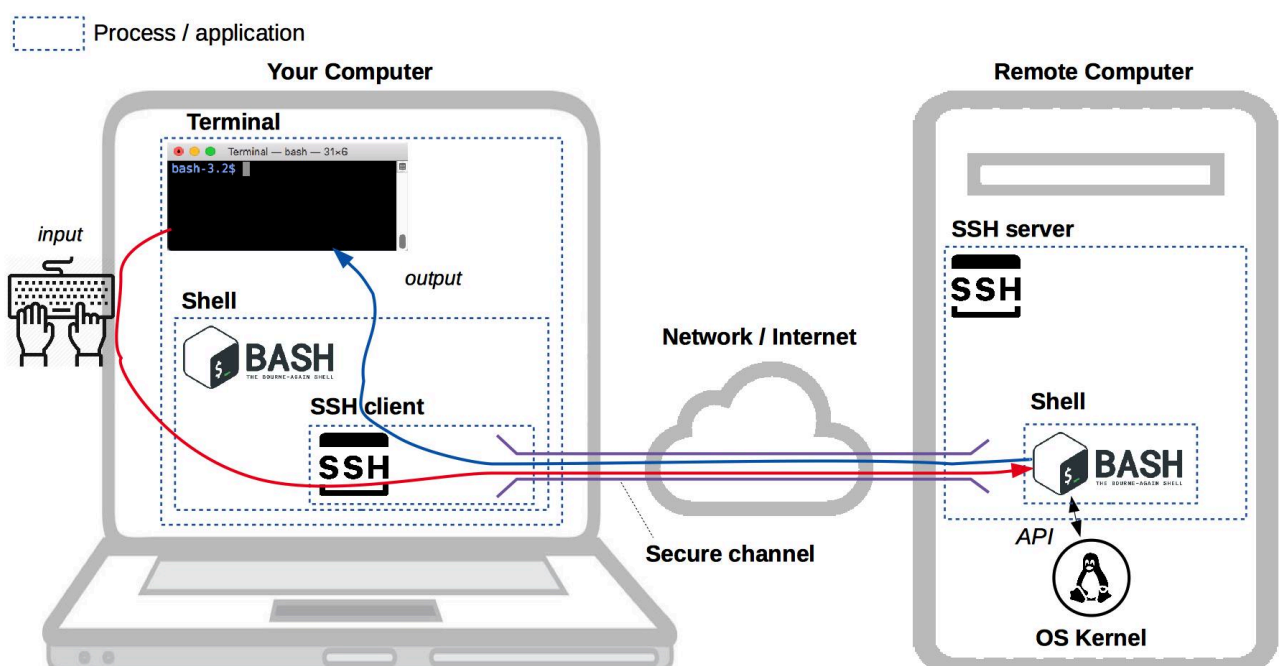
$
```

### Tip

Note that you may have a different command line prompt once you are connected, in this example `$` instead of `$>`.

## Typing commands while connected through SSH

You are now **connected to a Bash shell running on the server**. Anything you type is encrypted through SSH's secure channel and interpreted by that shell. Any data that Bash outputs is also encrypted, sent back through the channel and displayed in your terminal.



## Disconnecting

---

Disconnect with the command `exit` (or with `Ctrl-D` on Linux or macOS). You should be back to the shell running on your local computer, with your usual prompt:

```
$ exit
Connection to 192.168.50.4 closed.

$>
```

## Where am I?

---

Sometimes, you might forget what shell your terminal is connected to. Is it a shell on your local machine or one running on the server?

If you're not sure, the `hostname` command may help you. It prints the network name of the current machine:

```
$> hostname
MyComputer.local

$> ssh jde@192.168.50.4
jde@192.168.50.4's password:

$ hostname
example.com
```

In this example, the local computer is named `MyComputer.local`, while the server is named `example.com`.

As you can see, the `hostname` command returns different results before and after connecting to the server with SSH, because it's running on your local machine the first time, but is running on the server the second time.



# Logging in or running a command

---

When you execute `ssh` with the `[command]` option, it will execute the command and close the connection as soon as that command is done.

Run this from your local shell:

```
$> hostname  
MyComputer.local
```

```
$> ssh jde@192.168.50.4 echo Hello World  
Hello World
```

```
$> hostname  
MyComputer.local
```

As you can see, you are still in your local shell. The connection was closed as soon as the `echo` command completed.

## Public key authentication

---

Password authentication works, but it has some drawbacks:

- Attackers may try to brute force your password.
- If an attacker succeeds in performing a man-in-the-middle attack (for example if you forget to check the public key the first time you connect), he may steal your password.
- If the server is compromised, an attacker may modify the SSH server to steal your password.

As explained earlier, SSH uses asymmetric cryptography (among other techniques) to establish its secure channel. It's **also possible to use asymmetric cryptography to authenticate**.

## How does it work?

---

If you have a **private-public key pair**, you can **give your public key to the server**. Using **your private key**, your SSH client **can prove to the SSH server that you are the owner of that public key**.

This has advantages over password authentication:

- It's virtually impossible to brute-force (it is larger and probably has much more entropy than your password).
- Your private key will not be compromised by a man-in-the-middle attack or if the server is compromised, as it is never transmitted to the server, only used to solve mathematical problems based on the public key.

### **Warning**

Note that **public key authentication is only as secure as the file containing your private key**. If you publish that file anywhere or allow your local machine to be compromised, the attacker will be able to impersonate you on any server or service where you put your public key.

## How does public key authentication work?

---

To authenticate you, the server will need your **public key**. That way, you will be able to prove, using your **private key**, that you are the owner of that public key.



Remember, **your private key MUST remain private** (i.e. the `id_ed25519` file). You should **never** give it to any person, server or web service. Only give your public key (i.e. the `id_ed25519.pub` file).

## Using multiple keys

You may have multiple key pairs.

Some key pairs may have been generated by other programs or web services. For example, some Git user interfaces generate a key pair to access GitHub, or Amazon Web Services's Elastic Compute Cloud (EC2) generates key pairs to give you access to their virtual machines.

Having multiple key pairs may be part of a security strategy to limit the access an attacker might gain if one of them is compromised.

To generate a key with a custom name, use the `-f` (file) option when generating the key with the `ssh-keygen` command. To use a specific key pair, use the `ssh` command's `-i` (identity) option, which allows you to choose the private key file you want to use:

```
$> ssh-keygen -f custom_key
$> ssh -i ~/.ssh/custom_key jde@192.168.50.4
```

### Note

It is the private key file you want to use with the `-i` option, not the public key, as the private key is the one your SSH client will use to prove that it owns the public key.

## Key management

A few tips on managing your key pairs:

- You may disseminate your **public key** freely to authenticate to other computers or services.
- **NEVER give your private key to anyone.**
- Conversely, you may copy your private key to another computer of yours if you want it to have the same access to other computers or services.
- **Back up your private and public key files** (`id_rsa` and `id_rsa.pub`) to avoid having to regenerate a pair if you lose your computer or switch to another computer. (If you create a new key pair, you will have to replace the old public key with the new one everywhere you used it.)
- Use the `ssh-copy-id` command to copy your public key to other computers to use public key authentication instead of password authentication.

*You will see how to do that in the SSH exercises.*

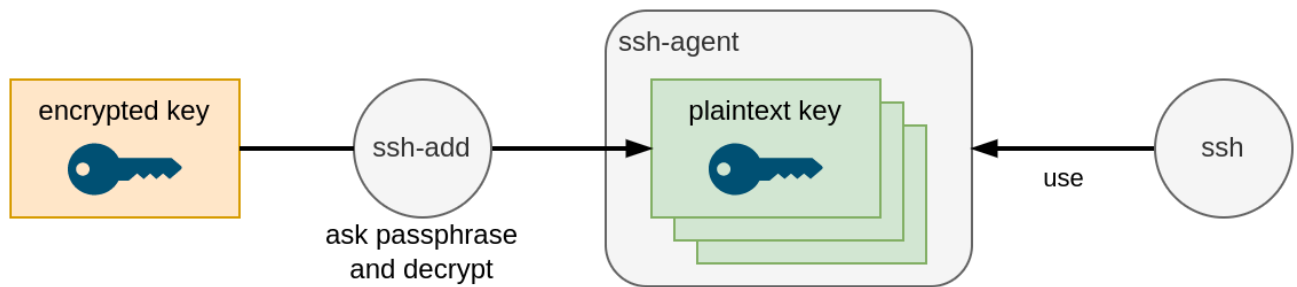
- For web services using public key authentication (e.g. GitHub), you usually have to manually copy the public key file's contents (`id_rsa.pub`) and provide it to them in your account's settings.

## Key protection

---

It's good practice to protect your private key with a passphrase. You can enter a passphrase when generating your key pair with the `ssh-keygen` command. You can also add a passphrase to an existing key later.

- **Without a passphrase, anyone who gains access to your computer has the potential to copy your private key.** For example, family members, coworkers, system administrators and hostile actors could gain access.
- The **downside** to using a passphrase is that **you need to enter it every time you use SSH**. You can temporarily cache your passphrase using `ssh-agent` so you don't have to enter it every time you connect.



- **If a private key is compromised** (e.g. your computer is hacked or stolen), you should **remove the corresponding public key** from computers and web services you have copied it to.

## SSH for other network services

---

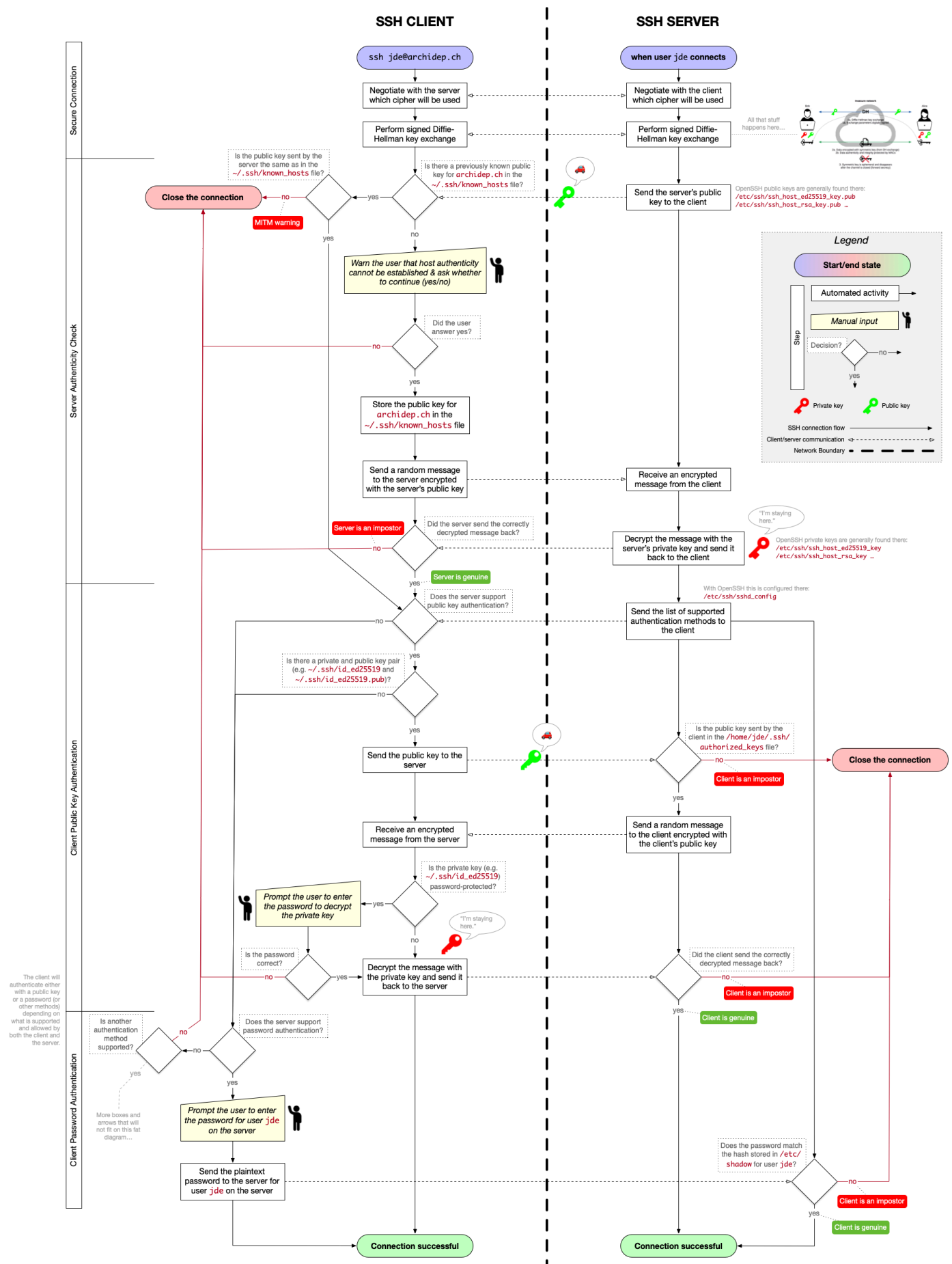
As mentioned initially, SSH is a network protocol. It can be used not only for command line login, but to secure other network services.

A few examples are:

- **Secure Copy** (scp) - A means of securely transferring computer files between a local and remote host.
- **rsync** - Utility for efficiently transferring and synchronizing files across computer systems.
- **SSH File Transfer Protocol** (SFTP) - Network protocol that provides file access, file transfer and file management.
- **Git** - Version control system that can use SSH (among other protocols) to transfer versioned data.

# The birth (or death) of an SSH connection

## THE BIRTH (OR DEATH) OF AN SSH CONNECTION



- [PDF version](#)
- [PNG version](#)

## References

---

- [How does SSH Work](#)
- [Demystifying Symmetric and Asymmetric Methods of Encryption](#)
- [Understanding the SSH Encryption and Connection Process](#)
- [Diffie-Hellman Key Exchange](#)
- [Simplest Explanation of the Math Behind Public Key Cryptography](#)
- [SSH, The Secure Shell: The Definitive Guide](#)
- [SSH Authentication Sequence and Key Files](#)

[↑](#) Back to top