s

**SSH Passwordless login by Esirigho Otite**

SSH (Secure Shell) is one of the best ways to handle tasks such as automating backups, performing file synchronization, and also having the ability to perform remote server access and management.

Linux using SSH Passwordless login

**Passwordless ssh**

\*\*Please note that before we started this project we had to install Linux Rhel 9 on a custom partition; the pictures are under the appendix section.\*\*

**Functionality and Purpose of using Passwordless ssh:**

Passwordless SSH is the most secure, efficient way, to access​ Linux servers. You generate a single pair ‍of SSH keys‍ on your local workstation, and‌ then securely store them on a remote server. To authenticate, the remote server securely sends‍ an ‌encrypted message that can only be, unlocked with‍ your SSH keys. This‌ helps, safeguard against unauthorized access and keeps your data safe and secure.

From a practical viewpoint in my company, we use SSH to transfer bank files between my company and the bank. The SSH keys help to give us that secure path for move vital data across the internet without the worry of files being intercepted. These files contain current account numbers and customer details.

SSH is one of the best ways to handle tasks such as automated backups, file synchronization, and remote server access and management. SSH passwordless login is an SSH authentication method that employs a pair of public and private keys for asymmetric encryption. The public key resides on the server, and only a client that presents the private key can connect.

**Questions:**

1. Is Passwordless SSH secured?
   1. Yes because it requires the authentication of the private/public key before connection is established.
2. What permissions should the authorized key file have?
   1. ~/.ssh/authorized\_keys permissions should be 600, same ownership as the directory. ~/.ssh/authorized\_keys are also a file with all the authorized ssh keys you want to use, not a directory.
3. Can it be done between two servers with two different usernames?
   1. Yes
4. What are the benefits of using a passwordless ssh? => List at least 5.
   1. Easy of logging between client-host.
   2. Easy way of transferring files between client-host.
   3. Secure path that only opens due to the key authentication.
   4. No hassle entering usernames and passwords, each time.
   5. Quickly access to remote systems.
   6. Secure, access to ​privileged systems and data.
   7. As employees come and go, new credentials (whether passwords or keys) must be created and old ones destroyed.
   8. Credentials need to be rotated, a time-consuming and often overlooked process.
   9. Auditing access can be a challenge. Wrapping communication in an SSH tunnel makes it more secure but also more difficult to track and control.
5. Why is the public key and NOT the private key being shared between the servers?
   1. The relationship between the Public key and the Private keys are unique, when they match the connection or port can now be opened. Even though they are related in this case means that whatever is encrypted by the public key can only be decrypted by the related private key. A person cannot guess the private key based on knowing the public key. Because of this, a public key can be freely shared. The private key however belongs to only one person.
6. Do I need to put a password to secure my passwordless ssh connection?
   1. Not exactly, unless the client needs or request some extra security.

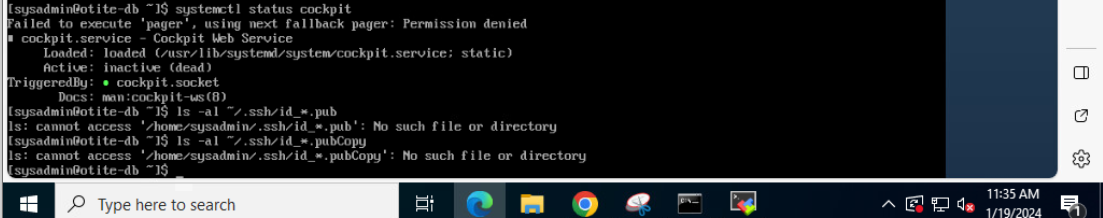


This tutorial will walk you step by step through how to manually set up SSH passwordless login to a Linux server. These commands should work on the majority of Linux distributions and Linux varieties.

**Step 1: Generate a key pair**

Use *ssh-keygen* to generate a key pair consisting of a public key and a private key on the client computer. This command can be run on any modern Linux client distribution, the Terminal in macOS, or in the Command Prompt in Windows 10/11.

Check to make sure no ssh already exists:



* ssh-keygen

I executed command ssh-keygen as we did in class for this step (In Linux command):

A screenshot of a computer

Description automatically generated

My ssh key private key was saved under root (/root/ .ssh/id\_rsa), while my public key was saved (/root/ .ssh/id\_rsa.pub)

To be sure that the SSH keys are generated you can list your new private and public keys with:

The default (*id\_rsa*in the *.ssh* directory under the user’s home directory) works perfectly in most cases (and if this is to be your primary key, or only key, it is often the best option). Hit enter to accept the default (if you already have a key by this name, use whatever name you choose here throughout this tutorial in place of *id\_rsa*). Then enter the passphrase when prompted.

Enter Passphrase (empty for no passphrase): I skipped this step

Adding a passphrase is an important step for securing the local key, which otherwise will be usable by anyone who acquires the key itself

With the initial step to set up SSH passwordless login using *ssh keygen* completed, you now have two files:

* *id\_rsa* contains the private key.
* *id\_rsa.pub* contains the public key.

**Step 2: Create SSH directory on server**

Next, add the public key on the server you want to connect to. With your existing username and password, connect to the server using SSH, using whatever command line or client program you normally use for such connections. Check to see if the *.ssh* directory already exists by attempting to list the files within it:

ls .ssh

A screenshot of a computer

Description automatically generated

If it does not, you will not be able to move into that directory and should instead create it:

mkdir -p .ssh

A screenshot of a computer screen

Description automatically generated

(Note the required dot at the beginning of the directory name, which makes this a hidden directory.)

**Step 3: Upload public key to remote server**

**Uploading your public key with a Linux**

On Linux client, use *ssh-copy-id* to propagate the public key to the server, like this:

ssh-copy-id [root@10.1.10.15](mailto:root@10.1.10.15) (I had to end up using our old box RHEL 8)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A) **If you had to create the .ssh directory yourself, or if the remote server doesn’t already have an authorized\_keys file**, on the client computer command line, enter the following to copy the public key to the *.ssh* directory on the server (if you changed the name of your key from *id\_rsa.pub*, change it here):

scp .ssh/id\_rsa.pub [user@somedomain:~/.ssh/authorized\_keys](mailto:user@somedomain:~/.ssh/authorized_keys)

scp .ssh/id\_rsa.pub [root@10.1.10.206](mailto:root@10.1.10.206) :~/.ssh/authorized\_keys

B) **If the remote server has an existing authorized\_keys file**, the new key must be appended rather than overwriting the existing file. This is very important so that existing users do not lose access unintentionally. First, you will copy the file to the remote server. Then on the remote server, use the cat command to append it to the existing file:

On the client:*scp .ssh/id\_rsa.pub user@somedomain:~/.ssh*

On the remote server: *cat .ssh/id\_rsa.pub >> .ssh/authorized\_keys*

On the remote server: *rm .ssh/id\_rsa.pub* (clean up after yourself and remove the now-unnecessary key file)

C) **If the ssh key was generated on the root directory:** I had to first create a backup copy then using “cp” I copied the files over to the destination /usr/bin/

**Step 4: Test connection and configure an SSH agent**

In your SSH session with the remote machine, update the permissions of the *.ssh* directory and *authorized\_keys* file in case they need it:

chmod 700 ~/.ssh  
  
chmod 600 ~/.ssh/authorized\_keys

Now, close your connection and your Terminal or Command Prompt. When you reopen it and try to connect to the remote server again from the client where you have your private key saved, you should receive a request to enter the passphrase instead of your username and password. Test it out:

ssh root@10.1.10.15

A screenshot of a computer

Description automatically generated

I wanted to check the directory on the server side:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

👍 Success!

Appendix:

Create a Red Hat 9 linux box:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

Now, to avoid entering the SSH key passphrase every time:

1. You will need to use an SSH agent of some kind.  
     
   **For Windows:**You will use the OpenSSH Authentication Agent. The agent can be started by searching in the Windows Start menu for "Services," then double click on "OpenSSH Authentication Agent." Set the startup type to "Automatic" and click "Start."; Click Ok and Exit.  
     
   **For macOS and Linux:** The ssh-agent program already runs on session start for most Linux/Unix distributions. It provides an agent that you can add keys to and save passphrases. Once set up, the program will not require further interaction.
2. At your command line prompt, in either case, type *ssh-add*. If you used the default *id\_rsa* naming for your key, that’s all you have to do. If you used a passphrase for the key, it will prompt you to enter it. Now the agent will remember your key and passphrase, and you won’t need to enter it on each use. You can also get more specific when adding keys (for example, if you used a different name for your key) with [ssh-add parameters](https://www.tutorialspoint.com/unix_commands/ssh-add.htm).

**Step 5: Back up SSH Keys**

[A public key can be re-derived from a private key](https://askubuntu.com/questions/53553/how-do-i-retrieve-the-public-key-from-a-ssh-private-key#53555), but not vice-versa, making it especially important to back up private keys. To do so, simply back up the directory where they reside, which in our above examples, was the*.ssh* directory in your user’s home directory. Both keys in the pair will be backed up because you generated them there, unless you removed one of them from the directory.

**Optional Step: Disable password authentication**

Most servers allow both username/password authentication and SSH key authentication, but if you want to allow only SSH key authentication, then you can disable the use of usernames and passwords. Be certain that you have thought through the ramifications before doing so, because once you take this action, successful certificate authentication will be the **only way to access your server.**

This is accomplished through the [sshd\_config](http://tldp.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/chap15sec122.html" \t "_blank) file. The exact location of this file varies by Linux distribution. Often it’s in the */etc/ssh* directory. Edit this file to include the following parameters:

PasswordAuthentication no                                         
ChallengeResponseAuthentication no                                
UsePAM no

**Drawbacks of SSH Passwordless Logins**

SSH passwordless login facilitates remote system login for off-site developers as well as on-site staff and scripted automation, but it comes with some potential complications:

* Private SSH keys sit on the client disk, where they can potentially be stolen (if passphrase protected, this is less of an issue.)
* SSH keys take a bit of work to set up and may require technical knowledge on the user’s end.
* Distributing a user’s public key to all servers the user wishes to connect to becomes a cumbersome requirement in large environments.
* Incorrect file permissions on the remote server can prevent SSH key authentication from working.
* Compatibility problems can arise between versions of SSH. For example, a system running an older version of OpenSSH might require a different key type, such as DSA instead of RSA
* Manual SSH key management consumes a lot of time and is open to errors, although this drawback is shared with the management of usernames and passwords for individual users and servers, as well.

**Eliminate passwords with a control plane**

Excellent access control and monitoring strategies are a crucial part of any infrastructure plan in today’s highly distributed environment. Yet manually distributing, revoking, rotating, and auditing SSH keys is a lot of work. Incorporating a good control plane is the answer:

* It eliminates the need to provide SSH keys to individual users and administer them across multiple machines.
* Centralizes credential management and access by role via an easy-to-use interface, making [onboarding](https://www.strongdm.com/blog/technical-staff-onboarding-checklist) and [offboarding](https://www.strongdm.com/blog/technical-staff-offboarding-checklist) simple.
* Enables quick, secure access to any server, any database, local or remote.
* Is [compatible with every database, server OS, SSO provider, and cloud](https://www.strongdm.com/connect).
* [Logs every session, capture every query, command, and permission change](https://www.strongdm.com/solution/cloud-observability-and-visibility) with a protocol-aware proxy. These logs can’t be beaten for auditing abnormal activity or as a [tool for meeting regulatory compliance](https://www.strongdm.com/solution/security-compliance).

StrongDM improves workflow and simplifies administration. If you’re ready to step away from the hassles of traditional [SSH key management](https://www.strongdm.com/blog/ssh-key-management) to more modern ways of authenticating, securing, and tracking access, [give StrongDM a try](https://www.strongdm.com/signup). You’ll only need five minutes to connect to your first database or server.

[How to set up certificate-based SSH](https://www.strongdm.com/blog/%7Btype=EXTERNAL,%20content_id=null,%20href=https:/www.youtube.com/watch?v=8Z2CNRawERg%7d)

**About the Author**

[**Jeff Smith**](https://www.strongdm.com/blog/author/jeff-smith)**, Lead Technical Writer,** has led projects and teams working on documentation in access and security for more than six years. Learning these technologies and helping other people do the same is his passion. Jeff contributes occasionally to various technical blogs and publications and sometimes writes on non-software topics such as productivity, project management, and tech news. To contact Jeff, [visit him on LinkedIn](https://www.linkedin.com/in/jeffreylees/).

**General information Online for reading:**

SSH is one of the best ways to handle tasks such as automated backups, file synchronization, and remote server access and management. **SSH passwordless login** is an SSH authentication method that employs a pair of public and private keys for asymmetric encryption. The public key resides on the server, and only a client that presents the private key can connect.

SSH does have its pain points, with or without SSH keys, including:

* As employees come and go, new credentials (whether passwords or keys) must be created and old ones destroyed.
* Credentials need to be rotated, a time-consuming and often overlooked process.
* Auditing access can be a challenge. Wrapping communication in an [SSH tunnel](https://www.strongdm.com/blog/ssh-tunneling) makes it more secure but also more difficult to track and control.

Fortunately, there’s an alternative to managing all of these things by hand, which we’ll go into later.

Bottom of Form

**You May Also Like**



[**AWS Authentication Best Practices (That Go Beyond MFA)**](https://www.strongdm.com/blog/aws-authentication-best-practices)

AWS authentication confirms the identity of users trying to access your resources, safeguarding against potential intrusions and data breaches. But weak authentication practices—like easy-to-guess passwords and single-factor authentication (SFA)—are far too common and they leave the door wide open for threat actors. Weak authentication often leads to data theft, resource misuse, financial and reputational nightmares…the list goes on. On the contrary, strong authentication measures like Multi-Factor Authentication (MFA) significantly reduce the risk of these incidents occurring. StrongDM takes AWS authentication to the next level, going beyond MFA to include granular access controls based on roles (RBAC), attributes (ABAC), and just-in-time approvals.



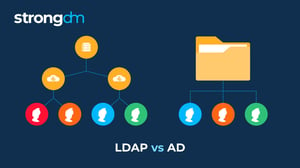
[**Connect to Even More Resources with StrongDM’s AWS Management Console**](https://www.strongdm.com/blog/aws-management-console)

We’ve just launched our AWS Management Console, adding yet another supported authentication method to improve control and auditability–so you can protect your business and improve employee productivity.



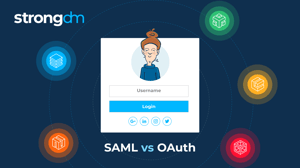
[**Token-based Authentication: Everything You Need to Know**](https://www.strongdm.com/blog/token-based-authentication)

Secured authentication to databases and applications is crucial to enterprise cybersecurity management. Unfortunately, 82% of all breaches involve human error, including misused or compromised credentials that give threat actors unauthorized access to network resources. Luckily, there’s a solution that ensures security without the risks that come with traditional, credential-based authentication. This article discusses token-based authentication and explains why it's a reliable and flexible alternative to verifying users, especially for cloud applications.



[**LDAP vs. Active Directory: Everything You Need to Know**](https://www.strongdm.com/blog/ldap-vs-active-directory)

Struggling to understand the difference between Active Directory and LDAP? Don't worry, we’ll make it simple. These are just two among many methods that can provide secure user authentication and authorization. The information in this article will help you decide if LDAP or Active Directory is right for your organization. Robust security and a seamless user experience are attainable, and you can have both!



[**SAML vs. OAuth: Everything You Need to Know**](https://www.strongdm.com/blog/saml-vs-oauth)

In this article, we will provide a high-level overview of the Security Assertion Markup Language (SAML) and Open Authorization (OAuth) information access frameworks. You’ll learn about the key similarities and differences between SAML and OAuth, the unique benefits of each framework, and specific use cases for each. By the end of this article, you’ll have a clear understanding of SAML and OAuth to help you determine which is right for your organization.