Passwordless SSH Authentication Between Two Linux/EC2 Servers

OR

SSH Public Key Authentication (Method name)

1. Problem Statement

In a production environment, servers often need to communicate with each other for tasks such as:

- File transfers using SCP
- Data synchronization using Rsync
- Automated backups
- CI/CD deployments
- Configuration management using tools like Ansible

When using traditional password-based SSH authentication, an administrator must manually enter the password every time a connection is established. This creates challenges for automation, security, and operational efficiency.

Therefore, a secure and automated method is required to allow one server to connect to another without manually entering passwords.

---

2. Objective

To establish a secure passwordless SSH connection between two Linux/EC2 servers using **SSH Public Key Authentication**.

---

3. Understanding the Problem

Consider the following scenario:

Server A (Application Server)

|

| SSH
|
Server B (Backup Server)

Suppose Server A needs to copy backup files to Server B every night.

Using password authentication:

ssh ec2-user@172.31.22.11

Output:

Password:

Every connection requires a password.

Issues with Password-Based Authentication

Manual Intervention

Automated scripts cannot enter passwords.

Example:

0 2 * * * rsync data/ serverB:/backup

The cron job will fail because no one is available to enter the password.

Security Risks

Passwords:

- Can be guessed
- Can be shared
- Can be leaked
- Are vulnerable to brute-force attacks

Poor Scalability

Managing passwords across multiple servers becomes difficult.

Example:

1 Server -> Easy
10 Servers -> Manageable
100 Servers -> Difficult

A more scalable solution is required.

---

4. Proposed Solution

SSH provides a feature called **Public-Key Authentication**.

Instead of using passwords, authentication is performed using a pair of cryptographic keys:

Private Key

Stored on the source machine.

Example:

~/.ssh/id_rsa

Characteristics:

- Secret
- Never shared
- Used to prove identity

Public Key

Stored on the destination machine.

Example:

~/.ssh/id_rsa.pub

Characteristics:

- Can be shared safely
- Used by the destination server to verify identity

---

5. Theory: How Passwordless SSH Works

Step 1: Key Pair Generation

Server A generates:

Private Key
Public Key

Example:

id_rsa
id_rsa.pub

Step 2: Public Key Distribution

The public key is copied to Server B.

Server B stores it in:

~/.ssh/authorized_keys

Step 3: Authentication Process

When Server A initiates an SSH connection:

Server A ----------> Server B

Server B checks:

Is the client's public key present
inside authorized_keys?

If yes:

Challenge Generated

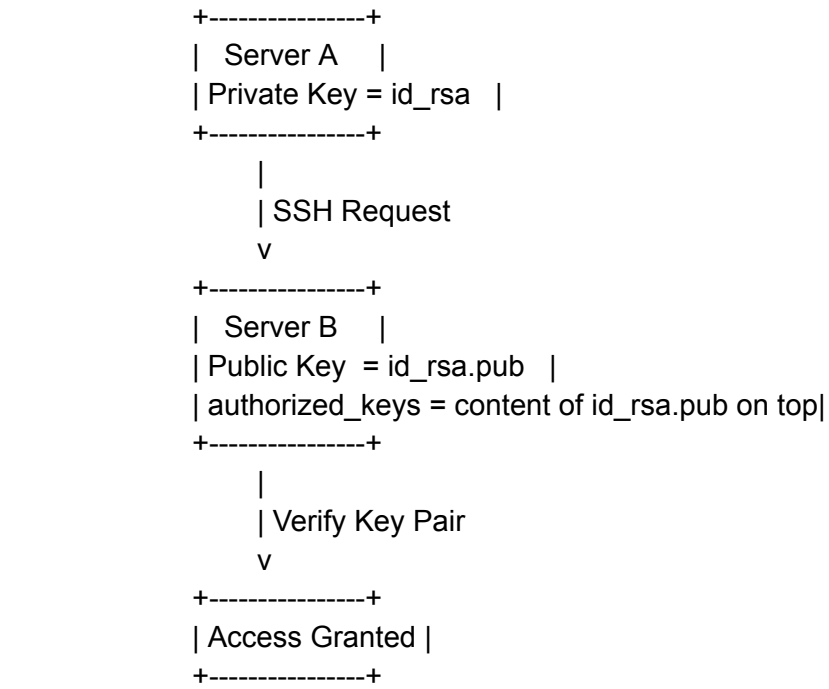
Server A proves ownership of the matching private key.

If verification succeeds:

Access Granted

No password is required.

Authentication Flow Diagram



---

6. Production Use Cases

Passwordless SSH is widely used in:

Automated Backups

rsync

Server-to-server data synchronization.

CI/CD Pipelines

Tools such as:

- Jenkins
- GitLab CI/CD
- GitHub Actions

use SSH keys to deploy applications to remote servers.

Configuration Management

Tools such as:

- Ansible
- Chef
- Puppet

rely heavily on SSH key authentication.

Automated File Transfers

scp
sftp
rsync

can run without human intervention.

---

7. Architecture Used in This Lab

Server A (Source)
172.31.31.174

SSH

Server B (Destination)
172.31.22.11

Server A will initiate the connection.

Server B will accept the connection.

---

8. Prerequisites

Before implementation, ensure:

Security Group Rules

Port 22 (SSH) is allowed.

Example:

Type: SSH
Protocol: TCP
Port: 22
Source: Appropriate CIDR or Security Group

---

9. Implementation Steps

Step 1: Login to Server A

ssh -i key.pem ec2-user@<server-a-public-ip>

Step 2: Generate SSH Key Pair

Run:

ssh-keygen -t rsa

> Means:
> 1. ssh-keygen: to generate key pair
> 2. -t: type
> 3. rsa: algorithm
> So it means generate key pair (private and public i.e id_rsa , id_rsa.pub) of type rsa algorithm.

Press Enter for all prompts.

Step 3: Verify Generated Keys

ls -a
Output main .ssh folder hoga.
Go inside that folder: cd .ssh
List all the files: ls

Expected output:

id_rsa
Id_rsa.pub
authorized_keys (ismain AWS ne “aditi.pem” private key ki corresponding public key ko rakha hua h)

Step 4: Display Public Key

We are talking about the key pairs that we generated, we have nothing to do with “aditi.pem”. It was used to access the ec2 only.(ssh -i “aditi.pem” ec2-user@<server-a-ki-public-ip>)

cat ~/.ssh/id_rsa.pub

Copy the complete output.

Example:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQ...

Step 5: Login to Server B

ssh -i key.pem ec2-user@<server-b-public-ip>

Step 6: Create SSH Directory (If Not Present)

mkdir -p ~/.ssh
Usually it will present, just do ls -a and then move into the **.ssh** folder.

Step 7: Add Public Key to authorized_keys

vim ~/.ssh/authorized_keys

Paste the public key copied from Server A.

Save and exit.

Step 8: Configure Permissions only if authentication fails and it shows too many permissions are open

chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys

Verify:

ls -ld ~/.ssh
ls -l ~/.ssh/authorized_keys

Meaning of these commands:

chmod 700 ~/.ssh

Means:

Owner : rwx = 7
Group : --- = 0
Others: --- = 0

Diagram:

drwx------

Meaning:

User	Permission
Owner	Read, Write, Execute
Group	No Access
Others	No Access

Why Execute Permission on Directory?

For directories:

r = view contents
w = create/delete files
x = enter/access directory

Without execute permission:

cd ~/.ssh

won't work.

So owner needs:

rwx

on `.ssh`.

---

chmod 600 ~/.ssh/authorized_keys

Means:

Owner : rw- = 6
Group : --- = 0
Others: --- = 0

Diagram:

-rw-------

Meaning:

User	Permission
Owner	Read, Write
Group	No Access
Others	No Access

Why No Execute?

Because:

authorized_keys

is a file, not a program.

Nobody needs to execute it.

Owner only needs:

Read
Write

Step 9: Test Passwordless Login

Return to Server A and execute:

ssh ec2-user@<server-B-ki-public-ip>

Expected result:

Last login...

No password prompt should appear.

Step 10: Validation

ssh -v ec2-user@<server-B-ki-public-ip>

Look for:

Offering public key
Authentication succeeded

This confirms key-based authentication is being used.

---

10. Benefits of Passwordless SSH

Feature	Password Authentication	SSH Key Authentication
Automation	No	Yes
Security	Lower	Higher
CI/CD Support	Difficult	Excellent
Backup Automation	Difficult	Easy
Scalability	Limited	High
User Experience	Manual	Seamless

---

14. Conclusion

Passwordless SSH authentication eliminates the need for manual password entry by using a public-private key pair. The public key is stored on the destination server, while the private key remains securely on the source server. This method improves security, enables automation, simplifies server management, and is widely used in production environments for backups, deployments, and configuration management.