SSH Access Through Keys

Author: Zayan Ahmed | **Estimated Reading time:** 6 min

Introduction

SSH (Secure Shell) is a widely used protocol to secure remote access to systems over a network. It allows administrators to securely manage and control remote servers. One of the most secure and preferred methods of SSH authentication is using SSH keys instead of passwords.

What are SSH Keys?

SSH keys are a pair of cryptographic keys used for authenticating to an SSH server. They consist of:

1. **Public Key**: This key is shared with the remote server and placed in the server's `~/.ssh/authorized_keys` file.
2. **Private Key**: This key is kept on the client machine and should remain secret. It is used to authenticate the client when connecting to the server.

Why Use SSH Keys Instead of Passwords?

- **Enhanced Security**: Passwords can be brute-forced, but private keys are practically impossible to guess.
- **Convenience**: SSH keys eliminate the need to manually input a password each time you connect to a server.
- **Automation**: SSH keys are essential for automating server access in DevOps environments, such as with scripts or CI/CD pipelines.

Setting Up SSH Key Authentication

1. Generate SSH Keys

The first step is to generate a key pair (public and private) on your local machine.

Step 1: Open a terminal

On your local machine (client), open a terminal window and run the following command to generate an SSH key pair:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

- `-t rsa`: Specifies the RSA algorithm for the key.
- `-b 4096`: Sets the length of the key to 4096 bits (more secure than the default 2048 bits).
- `-C "your_email@example.com"`: Adds a comment to the key (usually your email or identifier).

Step 2: Choose the file location

You'll be prompted to specify a location to save the key. Press Enter to accept the default location (`~/.ssh/id_rsa`).

If you'd like to store the key in a different location or give it a specific name, you can specify the full path (e.g., `~/.ssh/mykey`).

Step 3: Enter a passphrase (optional)

You will be prompted to enter a passphrase for an added layer of security. While optional, it's recommended to use one. If you prefer no passphrase, simply press Enter.

2. Copy the Public Key to the Server

Once the key pair is generated, the next step is to copy the public key to the remote server.

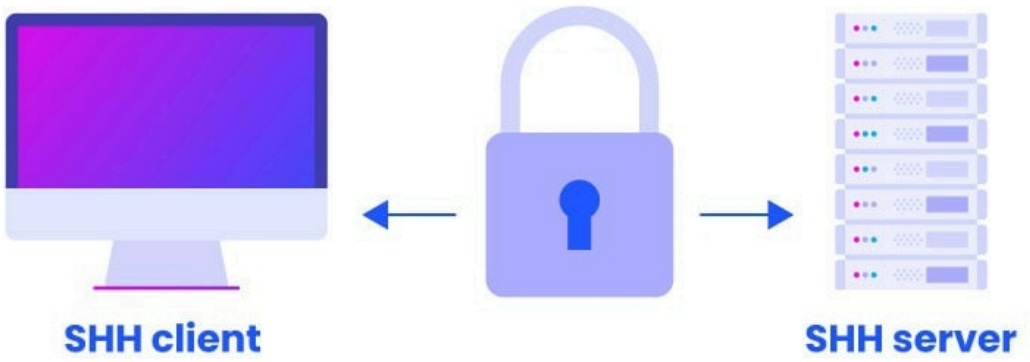
Step 1: Copy the Public Key Using `ssh-copy-id`

To copy the public key to the remote server's authorized keys file, you can use the `ssh-copy-id` tool. Run the following command:

```
ssh-copy-id username@remote_server
```

Replace `username` with the actual user on the remote server and `remote_server` with the server's IP address or hostname.

This command will copy the public key (`~/.ssh/id_rsa.pub`) to the remote server and add it to the `~/.ssh/authorized_keys` file for the user.



Step 2: Manually Copy the Public Key (if `ssh-copy-id` is unavailable)

If `ssh-copy-id` is not available or you're managing keys manually, you can do the following:

1. On the local machine, display the public key content with:

```
cat ~/.ssh/id_rsa.pub
```

2. Copy the output (the public key string).

3. On the remote server, open the `~/.ssh/authorized_keys` file with a text editor:

```
nano ~/.ssh/authorized_keys
```

4. Paste the public key into the file and save it.

3. Verify SSH Access

Now that the public key is installed on the server, test the connection:

```
ssh username@remote_server
```

You should be able to access the server without being prompted for a password, though you might be asked for your private key's passphrase if you set one.

Advanced SSH Key Configuration

1. Configuring the `~/.ssh/config` File

The `~/.ssh/config` file allows you to simplify SSH connections and manage multiple remote servers. Here's an example configuration:

```
Host server1
   HostName remote_server_1
   User username
   IdentityFile ~/.ssh/id_rsa
   Port 22

Host server2
   HostName remote_server_2
   User username
   IdentityFile ~/.ssh/another_key
   Port 2222
```

In this configuration:

- `Host`: A nickname for the server.
- `HostName`: The remote server's hostname or IP address.
- `User`: The user to log in as on the remote server.
- `IdentityFile`: The private key to use for authentication.
- `Port`: The port to use for SSH (default is 22, but you can customize it).

Now, you can use `ssh server1` instead of `ssh username@remote_server_1`.

2. Restricting SSH Key Usage

You can restrict what an SSH key can do by adding commands or limitations in the `authorized_keys` file. For example, to limit the key to only allowing a specific command, add a line like:

```
command="/path/to/command" ssh-rsa AAAAB3... user@host
```

You can also restrict key usage to specific IP addresses or enforce time-based access.

3. Using Multiple SSH Keys

If you have multiple SSH keys for different purposes, specify the key to use for a specific server in the `~/.ssh/config` file:

```
Host server1
```

```
    HostName remote_server_1
    User username
    IdentityFile ~/.ssh/id_rsa

Host server2
    HostName remote_server_2
    User username
    IdentityFile ~/.ssh/another_key
```

This ensures that the appropriate private key is used for each server.

Troubleshooting SSH Key Authentication

1. Permissions Issues

Ensure that your SSH key files have the correct permissions. The private key (`~/.ssh/id_rsa`) should only be readable by the user, and the `~/.ssh/authorized_keys` file on the server should be readable and writable only by the user.

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

2. SSH Agent Issues

Sometimes, SSH may not pick up the correct private key. Ensure the SSH agent is running and has the key loaded:

```
ssh-add ~/.ssh/id_rsa
```

If the key is not added to the agent, you may encounter authentication issues.

3. Debugging SSH Connections

Use the `-v` flag with SSH to get verbose output and troubleshoot connection problems:

```
ssh -v username@remote_server
```

This will provide detailed information about the connection process and help identify issues.

Best Practices for SSH Keys

- **Use Strong Passphrases**: If you use a passphrase for your private key, ensure it is strong and secure.
- **Use Different Keys for Different Servers**: Use separate key pairs for different servers or roles to minimize risk.
- **Backup Your Private Keys**: Keep a secure backup of your private keys, as losing them can prevent access to your servers.
- **Rotate Keys Periodically**: For enhanced security, rotate SSH keys regularly and update the `authorized_keys` file accordingly.
- **Limit Key Usage**: Restrict keys to specific commands, IPs, or time periods as needed.

Conclusion

SSH key-based authentication is a powerful and secure method for managing access to remote servers. It provides enhanced security over password-based authentication and is essential for automating access in DevOps environments. By following best practices for key management and configuration, you can ensure a secure and efficient setup for your SSH connections.

Follow me on **LinkedIn** for more 😊