HOW-TO GUIDE

# How to Secure SSH on Ubuntu with Key-Based Authentication

Use key-based authentication to harden SSH on Ubuntu with a practical workflow: generate keys, install them safely, verify access, and disable password logins only after validation.

**Operating Systems - July 05, 2026**

## Quick version

If your Ubuntu server still allows SSH password logins, the fastest meaningful hardening step is to move to key-based authentication and then turn off password authentication only after you have confirmed key access works. The operational goal is simple: reduce the risk of brute-force attacks, credential stuffing, and reused passwords without locking yourself out.

The safe sequence is:

- Create an SSH key pair on your admin workstation.
- Copy the public key to the Ubuntu server.
- Open a second SSH session and verify key-based login works.
- Update the SSH server configuration to disable password authentication.
- Reload SSH and test again before closing the original session.

If you also manage firewall policy, make sure port 22 remains reachable from your admin network before you change authentication settings. If you need a broader baseline for the host, Ubuntu Security Hardening Checklist for Production Servers is a useful companion while you verify access controls and recovery readiness.

## Why this matters operationally

SSH is often the first and most privileged remote access path on Ubuntu servers. Password-based logins increase exposure because they depend on human-generated secrets, are harder to rotate safely at scale, and are frequently reused elsewhere. Key-based authentication replaces that with cryptographic credentials that are significantly harder to guess or brute-force.

In practice, the value is not just stronger authentication. It is also more predictable administration: keys can be provisioned per operator, revoked individually, and protected with a passphrase and agent. That makes access reviews, incident response, and offboarding simpler than with shared or reused passwords.

The only operational risk is self-inflicted lockout. That is why this guide emphasizes validation before disabling passwords.

## Prerequisites

Before changing anything on the server, confirm the following:

- You have console or out-of-band access in case SSH becomes unavailable.
- You have sudo privileges on the Ubuntu server.
- You can connect to the server from a trusted admin workstation.
- Port 22 or your chosen SSH port is allowed through the network path and host firewall. If you need a quick, safe check of the firewall side, How to Configure UFW Firewall Rules on Ubuntu Server covers the validation points that matter before production use.
- You know which account you will use for initial testing.

On the client side, you need an SSH key pair. OpenSSH is installed by default on most Linux and macOS systems. On Windows, use a supported SSH client such as the built-in OpenSSH tooling in a terminal environment.

## Step 1: Generate an SSH key pair on the client

Create the key pair on the admin workstation, not on the server. That keeps the private key under the control of the person or automation system that will use it.

A common modern choice is Ed25519:

What the options do:

- -t ed25519 creates an Ed25519 key, which is a strong and compact default for most deployments.
- -a 100 increases passphrase derivation cost during key protection.
- -C adds a label to help identify the key later.

You will be prompted for a file location and an optional passphrase. For administrative access, use a passphrase unless you have a controlled automation context with compensating safeguards.

Expected output is typically a pair of files such as:

- ~/.ssh/id_ed25519 ? private key
- ~/.ssh/id_ed25519.pub ? public key

Check permissions if you want to verify the client-side state:

The private key should be readable only by your user.

## Step 2: Install the public key on Ubuntu

The safest and simplest method is ssh-copy-id:

This appends your public key to the target user?s ~/.ssh/authorized_keys file and sets basic permissions.

Expected output usually includes a summary that the number of keys added and a prompt for the account password one last time. After that, try a new SSH login:

If the key is accepted, you should connect without being prompted for the account password. If your key has a passphrase, you will still be prompted for that passphrase locally.

If ssh-copy-id is unavailable, you can copy the public key manually. On the server, ensure the user has a secure ~/.ssh directory and append the public key to ~/.ssh/authorized_keys:

Paste the public key, then press Ctrl+D to finish input.

## Step 3: Validate key-based login before changing the server

Do not disable password authentication until you have confirmed that a second SSH session succeeds with the key.

Use a new terminal window and connect again:

If you want to see how authentication was negotiated, add verbose output:

Useful indicators in verbose output include lines showing the identity file being offered and the server accepting a public key method. If the connection falls back to password prompts, stop and fix that before making server-side changes.

Common causes of failure at this stage include:

- Incorrect ownership or permissions on ~/.ssh or authorized_keys
- The wrong public key copied to the server
- Connecting as the wrong user
- A restrictive server-side SSH policy
- Network access or firewall issues that were not actually related to authentication

If you are unsure whether the host is reachable at the network layer, verify that SSH port access is allowed from the admin network before proceeding.

## Step 4: Harden the SSH daemon configuration

On the Ubuntu server, edit the SSH daemon configuration:

Set or confirm these directives:

Notes on these settings:

- PubkeyAuthentication yes keeps public key auth enabled.
- PasswordAuthentication no disables password logins.
- KbdInteractiveAuthentication no helps prevent keyboard-interactive fallback paths if they are not needed in your environment.

- PermitRootLogin no prevents direct root SSH login, which is generally safer operationally.

If your system uses additional included configuration files under /etc/ssh/sshd_config.d/, check for conflicting overrides. The effective setting is what matters, not only the main file.

You can inspect the active SSH configuration with:

This is a useful validation step because it shows the daemon?s effective values after includes and defaults are applied.

## Step 5: Reload SSH safely and keep a live session open

Before reloading the daemon, keep at least one existing SSH session open. That gives you a fallback if the new settings are too strict.

First validate the configuration syntax:

If there is no output, the syntax check passed. If there is an error, fix it before reloading.

Then reload the SSH service:

A reload is usually enough for configuration changes. Use a restart only if reload is not supported in your environment or if you are addressing a specific problem that requires it.

After reloading, open a new terminal and test login again with the key:

If that works, try a deliberate password-authentication test. The expected result is that password login should be refused and the client should not be able to authenticate that way.

## Step 6: Verify the hardening outcome

At this point, you should confirm both success and failure cases.

From the client:

- Key-based login succeeds.
- Password-only login fails.
- The server accepts connections from the intended source network.

From the server, review the authentication logs if needed. On Ubuntu, SSH authentication messages are typically available through the system journal or auth logs:

or

Look for accepted public key entries and ensure there are no unexpected authentication failures or lockout symptoms.

A practical production rule is: do not close your original administrative session until a fresh session has succeeded after the configuration change.

## Common issues and safe fixes

If the key is not accepted, troubleshoot in this order:

- Confirm the client is offering the expected private key.
- Confirm the public key is present in the correct user?s authorized_keys file.
- Check permissions and ownership on ~/.ssh and authorized_keys.
- Review the effective SSH daemon configuration with sudo sshd -T.
- Inspect the auth log for the specific rejection reason.

A useful permissions baseline on the server is:

These settings are not optional in many environments. OpenSSH will ignore or reject keys if the directory and file permissions are too permissive.

If you locked yourself out of password auth but still have console access, you can roll back by re-enabling password login in /etc/ssh/sshd_config:

Then validate and reload again:

Use rollback only as a recovery action. If the server is exposed to the internet, treat the rollback window as temporary and reapply key-only access after fixing the cause.

## Operational improvements after the basic rollout

Once the core setup is working, you can improve control and auditability without changing the main access model.

One effective improvement is to use separate keys per person and per system. That lets you revoke a single key when someone leaves or a machine is retired without affecting everyone else.

You can also constrain authorized keys in ~/.ssh/authorized_keys with options such as source restrictions or command restrictions where appropriate. Those controls are useful for automation accounts, but they require careful testing because an overly strict rule can break legitimate workflows.

For stronger hygiene, consider:

- Setting a passphrase on administrative keys.
- Storing private keys in a secure agent rather than copying them around.
- Avoiding shared accounts for day-to-day access.
- Reviewing authorized_keys during access audits.

If you need to track patch compliance after SSH hardening, pairing access control review with a script-based audit workflow can help you keep host posture visible over time. The same discipline used for key management works well for security patch verification too.

## What to verify before production use

Before you declare the host hardened, verify these conditions explicitly:

- At least one non-root account can log in with the intended key.
- Password authentication is disabled in the effective SSH configuration.
- A separate session was used for validation before the original session was closed.
- Console or emergency access exists if SSH becomes unavailable.
- The firewall allows SSH only from the intended networks.
- Root login is disabled unless you have a documented exception.
- Authentication logs show the expected public key login path.

That checklist keeps the change safe enough for production because it confirms both the security intent and the operational recovery path.

## Final takeaway

Key-based SSH authentication on Ubuntu is one of the highest-value access hardening changes you can make, but only if you deploy it in the right order: generate a key pair, install the public key, validate login, then disable passwords and verify again. If you keep a live fallback session open and confirm the effective SSH settings before reloading, you can harden remote access without risking an avoidable lockout.

Use this guidance together with CentOS SSH key-based authentication and secure C# APIs with JWT to connect the workflow with related operational context already available on the site.