anoBytes | TECHNOLOGY SOLUTIONS

Secure and harden SSH access on Linux

Key-only authentication, a CIS-aligned hardening drop-in, the brute-force defenses sshd now has built in, and where fail2ban still earns its place. For Fedora, RHEL, Ubuntu, and Arch-based systems.

OPENSSH · SSHD_CONFIG · FAIL2BAN · CIS- AND NIST-ALIGNED · HOW-TO GUIDE

General educational guidance, not affiliated with or endorsed by the Center for Internet Security, NIST, the OpenSSH project, or any vendor named here. Every step below can lock you out of a remote machine: keep a second session open throughout. Commands and defaults refer to OpenSSH 10.4, with release notes covered through 10.5. Published 2026-08-11.

3
brute-force defenses built into sshd
MaxAuthTries, LoginGraceTime, and OpenSSH 9.8's PerSourcePenalties

22 **CIS sshd checks**
section 5.1, RHEL 9 and 10 (Ubuntu adds two)

4 **distributions**
Fedora 44 · RHEL 9 and 10 · Ubuntu 26.04 · CachyOS

SSH exposed to the internet on port 22 attracts a steady stream of automated login attempts, whatever the machine is or who runs it. The fix that ends the problem is to remove the password path entirely, so there is nothing left to guess, and then to keep the port off the public internet where the guessing cannot even begin. This guide does both, adds the sshd settings the CIS Linux Benchmarks check for, and puts fail2ban in its proper place.

CONTEXT What each control actually buys you

These are in order of effect. Working down the list in this order means each step is useful on its own, and a step you skip costs you less than the ones above it.

1. Keeping SSH off the public internet is the strongest control for a server

A port the public internet cannot route to cannot be brute-forced, scanned, or hit by an internet-borne flaw in the daemon. Step 8 covers this. It maps to NIST SC-7 (Boundary Protection), CM-7 (Least Functionality), and AC-17(3) (Managed Access Control Points).

2. Key-only authentication is the core host control

With both `PasswordAuthentication no` and `KbdInteractiveAuthentication no` (Step 4), no password path remains. Both lines are needed: turning off only the first still leaves the keyboard-interactive path, which on most Linux systems asks PAM (Pluggable Authentication Modules, the system's login stack) for a password.

3. sshd already penalises repeat offenders, with no extra software

Since OpenSSH 9.8 (July 2024) the server has `PerSourcePenalties`, and it is **on by default**. Failed authentications, invalid usernames, and connections that never authenticate each earn the source address a short refusal that accumulates up to ten minutes. Step 6 shows how to see it working.

4. fail2ban is log-noise and load reduction

It bans repeat offenders for longer than sshd's own penalties and across all ports, which keeps auth logs readable. Once passwords are off, bots cannot get in either way, and once Step 8 is done there is little left for it to watch. Step 6 covers when it is worth running.

5. A non-standard port is obscurity

It cuts automated scan noise and stops no attacker who looks. Step 7 covers it last and treats it as optional, because an overlay network removes the public path instead of moving it.

```
client (your laptop)
     |  ssh, tcp/22
     v
host firewall: allow ssh
     |
     v
sshd, configured by /etc/ssh/sshd_config.d/10-hardening.conf
     PasswordAuthentication no          no password to guess
     KbdInteractiveAuthentication no
     AllowGroups ssh-users              only that group may log in
     PermitRootLogin no
     LogLevel VERBOSE                   logs the key fingerprint used
     |
     |  a failed attempt goes two places
     |
     +--> PerSourcePenalties        built in, on by default since 9.8
     |      authfail 5s, invaliduser 5s, accumulating to max 10m
     |
     +--> systemd journal --> fail2ban [sshd] --> firewall drop
            5 failures in 10m, ban 10m (package defaults)
```

sshd refuses passwords outright. The failed attempts it does see are penalised inside the daemon and, if you run fail2ban, banned at the firewall as well.

Lockout warning

READ THIS BEFORE YOU TOUCH A REMOTE HOST

Every step here can lock you out of a remote machine if done out of order. Keep your current SSH session open the entire time, test each change from a **second** terminal before you close the first, and on a cloud virtual machine find your provider's recovery console before you start. The order in this guide is chosen so that the setting which can lock you out (Step 4) comes after the group and the key it depends on.

MAP Distribution cheat-sheet

The procedure is the same everywhere; a handful of commands differ. Substitute from this table as you go. Fedora and RHEL share tooling (dnf, `sshd.service`, firewalld, SELinux), so they are grouped. Versions are those current in August 2026.

	FEDORA 44 / RHEL 10 (AND 9)	UBUNTU 26.04 LTS	CACHYOS (ARCH-BASED)
OpenSSH version	10.2p1 (Fedora); 9.9p1 (RHEL 10 and 9.8+)	10.2p1	10.5p1 (rolling)
Install the server	`sudo dnf install openssh-server`	`sudo apt install openssh-server`	`sudo pacman -S openssh`
systemd unit	`sshd.service`, persistent	`ssh.service`, socket-activated by `ssh.socket`; `sshd.service` is an alias	`sshd.service`, persistent
Firewall	firewalld, on by default	ufw, installed but **inactive**	ufw, **active** by default
Cipher policy	system-wide crypto policies	OpenSSH defaults	OpenSSH defaults
Install fail2ban	Fedora: `dnf install fail2ban`. RHEL: enable EPEL first (Step 6)	`sudo apt install fail2ban`	`sudo pacman -S fail2ban`
sshd jail after install	disabled, you enable it	**already enabled**	disabled, you enable it

All four read drop-in files from the same directory

Each ships an `Include /etc/ssh/sshd_config.d/*.conf` line near the top of `/etc/ssh/sshd_config`, so a file you add there is read before the settings in the main file, and a low-numbered name is read before the distribution's own drop-ins. That is what Step 4 relies on, and it is what the CIS remediation text tells you to use.

The unit name differs on Ubuntu

On Ubuntu and Debian the service is `ssh.service` and an `sshd.service` alias exists, so `systemctl reload ssh` is the safe form to type. On Fedora, RHEL, and CachyOS it is `sshd.service`.

Ubuntu starts sshd from a socket

systemd owns the listening port through `ssh.socket` and starts `sshd` when the first connection arrives; the daemon then stays resident. Reloads work as normal. Only a port change is different, and Step 7 covers it.

Client and server are one package on Arch

Arch and CachyOS ship both in `openssh`. Fedora, RHEL, and Ubuntu split the server into `openssh-server`.

On Fedora and RHEL, algorithms come from the system crypto policy

Ciphers, MACs, and key exchange are set with `update-crypto-policies` rather than in `sshd_config`. Step 5 explains why putting them in your drop-in silently overrides the policy.

PREP Prerequisites and conventions

- A non-root account on the server that can use `sudo`.

- Terminal access to the server and to the machine you connect from.
- The OpenSSH **client** (`ssh`, `ssh-keygen`, `ssh-copy-id`) on the machine you connect from.
- On a remote host: an SSH session you will keep open, a second terminal to test with, and the location of your provider's recovery console.

- **Two machines, and which one you are on**

  You work with the computer you sit at (the *client*, called "your laptop" below) and the *server* you are securing. Key generation in Step 2 runs on your laptop. Everything else runs on the server. Run `hostname` if you lose track.

- **Replace the placeholders**

  `you` is your login name on the server (`whoami` prints it) and `server` is its address or hostname, so `ssh you@server` becomes something like `ssh alex@203.0.113.10`. The names `ssh-users` and `10-hardening.conf` are chosen by this guide; keep them.

- **Keep a second terminal open**

  Open another terminal window on your laptop and start a fresh `ssh you@server` there, leaving your first session connected and untouched. After every change, prove you can still get in by opening a *new* connection in that second window. If a change breaks login, the original session is still in, so you can undo it. That first session is your way back in for the whole guide.

- **reload versus restart**

  `systemctl reload` makes the running daemon re-read its config without dropping current connections, so your session survives. `systemctl restart` stops and starts it, which can drop sessions, and it is only needed when the listening port changes.

- **Everything goes in one drop-in file**

  All the changes in Step 4 go in one new file under `/etc/ssh/sshd_config.d/`. Nothing in `/etc/ssh/sshd_config` is edited, so rolling back is one `rm` and a reload, and a package upgrade cannot quietly revert your work.

- **If you do lock yourself out**

  Cloud providers give you an out-of-band *recovery console* (labelled "Serial Console", "VNC console", or "Recovery mode", depending on the provider) that logs you in even when SSH is unreachable. Find yours before you start.

01 Install the server and allow it through the firewall

Do the firewall rule **before** anything else on a remote host. Install the package for your distribution (see the cheat-sheet), then enable the service:

```
# Fedora, RHEL, and CachyOS
sudo systemctl enable --now sshd

# Ubuntu (this enables ssh.socket as well)
sudo systemctl enable --now ssh
```

Then open the port:

```
# Fedora and RHEL (firewalld)
sudo firewall-cmd --add-service=ssh --permanent
sudo firewall-cmd --reload

# Ubuntu (ufw is inactive by default: add the rule, then enable it)
sudo ufw limit OpenSSH
sudo ufw enable

# CachyOS (ufw is already active)
sudo ufw limit ssh
```

`ufw limit` opens SSH and rate-limits repeated attempts from one address: ufw allows the connection normally, then denies a source that opens six or more connections in thirty seconds. It overlaps with the penalties sshd now applies by itself (Step 6) and costs nothing to keep. One catch for later: a rule added with `limit` is removed with `ufw delete limit`, and `ufw delete allow` will not match it.

Confirm the port is reachable before you go further: from your laptop, `ssh you@server` should still connect. Enabling a default-deny firewall on a remote host without an SSH rule in place ends the session, and there is no way back in except the recovery console.

02 Set up key-based authentication

This is the control everything else rests on. Do it carefully and in order, and do not move on until a key-only login works.

2.1 Generate a key pair on the machine you connect FROM

Run this on your laptop, not on the server. Ed25519 is the right choice in 2026, and it is what `ssh-keygen` produces when you give it no arguments at all.

```
ssh-keygen -t ed25519 -C "you@your-laptop-2026"
```

The `-C` text is a label to help you recognise this key later; it need not be an email address and it does not affect security. Accept the default file location and **set a passphrase** when prompted. The passphrase encrypts the private key on disk with a slow key-derivation function (16 rounds by default), so a stolen laptop does not hand a usable key to whoever took it. NIST IR 7966 treats SSH keys as authenticators under NIST IA-5: protect them with a passphrase, never copy or share a private key, and rotate on a defined schedule and on suspected compromise.

- **When to use something other than Ed25519**
  Use RSA only for systems too old to understand Ed25519, which arrived in OpenSSH 6.5 (2014): `ssh-keygen -t rsa -b 4096`. Avoid ECDSA for new keys. DSA is gone: OpenSSH 10.0 removed it entirely. For high-value administrative accounts, a hardware-backed FIDO2 key (`ssh-keygen -t ed25519-sk`, needs OpenSSH 8.2 or newer and a security token) keeps the private key on the token where it cannot be copied, and a physical touch is required for each use.

2.2 Copy the public key to the server

```
ssh-copy-id you@server
```

`ssh-copy-id` logs in the ordinary way to install the key, so it needs a login that already works: your account password, or an existing key. Cloud images usually ship with password authentication off and one key already injected, in which case this step is done for you, and a second key goes in through the provider's console or beside the first in `~/.ssh/authorized_keys`.

2.3 Test the key login in a NEW terminal, before changing any server config

```
ssh you@server
```

You should be logged in **without being asked for your account password**. If the server still asks, fix the key before going further. Permissions are the usual cause: on the server, `~/.ssh` must be mode `700` and `~/.ssh/authorized_keys` must be `600`, because sshd refuses to use `authorized_keys` when that file, `~/.ssh`, or your home directory is writable by anyone else. That check is `StrictModes`, and it is on by default.

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

Stop here until this works

A KEY-ONLY LOGIN IS THE PRECONDITION FOR STEP 4

Step 4 turns off password authentication. If your key login is not working when you apply it, your next connection attempt has no way to authenticate and you are locked out. Prove the key works from a fresh terminal first.

03 Create an SSH access group

The hardening drop-in restricts SSH to members of one group and disables root login. CIS asks for SSH access to be limited by `AllowUsers`, `AllowGroups`, `DenyUsers`, or `DenyGroups`; a group is the version that keeps working as people come and go. This maps to NIST AC-6 (Least Privilege). Under the principle of least privilege, an account gets the access it needs and no more. Set it up and confirm your own membership **before** applying Step 4.

```
sudo groupadd -f ssh-users
sudo usermod -aG ssh-users you
id you            # confirm 'ssh-users' is listed
```

Every account that needs SSH access must be in `ssh-users`. Because root login is about to be disabled, make sure this non-root user has both a working key login (Step 2) and `sudo`. A brand-new SSH login is checked against the updated group immediately, so `AllowGroups` will accept you without a logout. Log out and back in anyway. Your interactive session only picks up the new group for `sudo` and file access when it starts fresh, and nothing that can lock you out is applied until Step 4, so this is the safe moment.

04 Apply the hardening drop-in

Put every change in one file under `/etc/ssh/sshd_config.d/`. Files there are read in lexical order, and for any keyword **the first value obtained wins**, so a low-numbered file takes precedence over the distribution's own drop-ins (on Fedora and RHEL, `40-redhat-crypto-policies.conf` and `50-redhat.conf`; on Ubuntu cloud images, drop-ins such as `50-cloud-init.conf` and `60-cloudimg-settings.conf`). This is the file layout the CIS remediation text asks for.

```
sudo nano /etc/ssh/sshd_config.d/10-hardening.conf
```

Everything below goes in that one file. Authentication first:

```
# Authentication: keys only
PubkeyAuthentication yes
PasswordAuthentication no
KbdInteractiveAuthentication no
PermitEmptyPasswords no
HostbasedAuthentication no
GSSAPIAuthentication no
IgnoreRhosts yes
PermitRootLogin no
UsePAM yes

# Restrict who may log in
AllowGroups ssh-users
```

Then the session limits, the forwarding settings, and the logging, in the same file:

```
# Session and connection limits
MaxAuthTries 4
LoginGraceTime 60
MaxStartups 10:30:60
MaxSessions 10
ClientAliveInterval 300
ClientAliveCountMax 3

# Reduce attack surface. Comment out this whole block if you use
# SSH tunnels (ssh -L, -D) or agent forwarding to a jump host.
PermitUserEnvironment no
DisableForwarding yes
X11Forwarding no
AllowAgentForwarding no
AllowTcpForwarding no

# Logging and banner
LogLevel VERBOSE
Banner /etc/issue.net
```

What the less obvious lines do, and the caveats:

`PasswordAuthentication no` with `KbdInteractiveAuthentication no`

Together these are what actually ends password brute force. `KbdInteractiveAuthentication` is the current name for the setting older guides call `ChallengeResponseAuthentication`, now a deprecated alias. Ubuntu already ships the second line set to `no`; repeating it here costs nothing and keeps the file complete on every distribution.

`GSSAPIAuthentication no`

Kerberos-based authentication, off by default upstream and switched on by the RHEL family: Fedora 44, RHEL 9, and RHEL 10 all ship `GSSAPIAuthentication yes` (and `X11Forwarding yes`) in `/etc/ssh/sshd_config.d/50-redhat.conf`. Your `10-hardening.conf` sorts ahead of that file, so it wins. CIS requires the option disabled (Level 2 on a server, Level 1 on a workstation) to shrink the pre-authentication attack surface. OpenSSH 10.4 fixed a pre-authentication denial of service that only applied when this option was on, and which `MaxAuthTries` did not mitigate. Leave it off unless you actually use Kerberos single sign-on.

`PermitRootLogin no`

Disables direct root SSH entirely. Log in as your `ssh-users` member and use `sudo`. Where root key access is genuinely required (some single-account virtual private servers), `prohibit-password` is the fallback: it allows a root key but blocks root password and keyboard-interactive logins. That is also OpenSSH's own default, so leaving this line out is weaker than setting it.

`MaxAuthTries 4`, `LoginGraceTime 60`, `MaxStartups 10:30:60`, `MaxSessions 10`

These are the current CIS values, and each is tighter than or equal to the OpenSSH default (6, 120 seconds, 10:30:100, and 10). CIS asks for `MaxAuthTries` of 4 or less, `LoginGraceTime` between 1 and 60 seconds, `MaxStartups` of 10:30:60 or more restrictive, and `MaxSessions` of 10 or less. `MaxSessions 1` disables session multiplexing and `0` blocks shell and subsystem sessions altogether while still allowing forwarding, so 10 is both the CIS maximum and the value to leave alone.

`ClientAliveInterval 300` with `ClientAliveCountMax 3`

Drops an unresponsive connection after about 15 minutes (300 seconds times 3), which is NIST AC-12, session termination. CIS accepts any pair above zero "according to site policy" and prints 15 and 3 as its example, so substitute the exact pair your target benchmark or policy requires. These probes end genuinely dead links only. An idle but connected session stays up, because the client answers the probes automatically; SSH has never had a way to disconnect an idle *user*.

`DisableForwarding yes` plus the three explicit lines

The umbrella directive turns off X11, agent, TCP, and StreamLocal forwarding, and CIS rates it Level 2 on a server and Level 1 on a workstation. The three explicit lines are there because on OpenSSH before 10.0 the umbrella did not actually stop X11 and agent forwarding (CVE-2025-32728, Moderate, CVSS 4.3). That still matters on RHEL 9: Red Hat fixed it in RHEL 10 (openssh-9.9p1-11.el10, RHSA-2025:20126) and has marked it *Fix deferred* on RHEL 9, whose 9.9p1 build therefore still carries it. Red Hat's own mitigation is the first two of these lines, `X11Forwarding no` and `AllowAgentForwarding no`; the bug never affected TCP forwarding. Fedora, Ubuntu, and CachyOS are all on OpenSSH 10.x and unaffected, and the lines are harmless there.

`LogLevel VERBOSE`

Records the fingerprint of the key used to authenticate, alongside login and logout activity. NIST IR 7966 calls for logging key fingerprints, and it is what lets you answer "which key was that?" months later. This supports NIST AU-2, AU-3, and AU-12. Anything above VERBOSE (the DEBUG levels) logs user activity in a way that harms privacy and is not recommended.

No `Ciphers`, `MACs`, or `KexAlgorithms` lines here

Deliberate. On Fedora and RHEL a list in this file would silently override the system crypto policy, because this file is read first. Everywhere else the OpenSSH defaults are already strong. Step 5 covers both cases.

No `Protocol 2` line

Old guides still tell you to add one, and it does nothing. Server support for SSH protocol 1 was removed in OpenSSH 7.4 (2016), and the remaining protocol 1 code along with the `Protocol` option itself went in 7.6 (2017). Current sshd parses the keyword as a deprecated option, notes that fact at debug level, and ignores it, so `sshd -t` passes and nothing is configured.

4.1 Create the warning banner

CIS requires `Banner` to point at a file, and that the file match your own policy (NIST AC-8, System Use Notification). The text below is a reasonable default for personal use; replace it with your organization's approved notice if you have one. Keep the system's name, version, and patch level out of it.

```
sudo tee /etc/issue.net >/dev/null <<'EOF'
WARNING: Authorized access only. All activity may be monitored and recorded.
Disconnect now if you are not an authorized user.
EOF
```

4.2 Tighten the file permissions

```
sudo chmod 600 /etc/ssh/sshd_config /etc/ssh/sshd_config.d/10-hardening.conf
```

CIS 5.1.1 checks the mode and ownership of `sshd_config`, and the same reasoning applies to anything you add beside it. The host key files have their own checks (5.1.2 for the private keys, 5.1.3 for the public ones) and their packaged permissions are already correct; leave them alone.

4.3 Validate before you apply

```
sudo sshd -t
```

No output means the configuration is valid. A syntax error here does not affect the running service or your current session, which is the point of checking first. If you get `sshd: command not found`, use the full path: `sudo /usr/sbin/sshd -t`.

4.4 Apply it

```
# Fedora, RHEL, and CachyOS
sudo systemctl reload sshd

# Ubuntu
sudo systemctl reload ssh
```

Socket activation on Ubuntu changes nothing here. systemd starts `sshd` at the first connection and the daemon then stays resident, so it does not re-read its configuration until you reload it.

4.5 Test from the second terminal before closing your session

```
ssh you@server
sudo sshd -T | grep -Ei 'passwordauthentication|permitrootlogin|allowgroups|maxauthtries'
```

`sshd -T` prints the server's effective configuration after all includes and drop-ins are resolved, which is the only reliable way to know what is actually in force. Lines reading `PasswordAuthentication no` and `PermitRootLogin no` are success. Only close your original session once a fresh key-only login as an `ssh-users` member works.

- **Match case-insensitively: OpenSSH 10.4 changed this output**

  Before OpenSSH 10.4 (July 2026) the configuration dump printed keyword names in lower case, and audit snippets written against it use `grep maxauthtries`. From 10.4 the dump writes them in mixed case (`MaxAuthTries 4`), so those older snippets silently return nothing and read as a passing result when nothing was found at all. A 10.4 dump mixes the two forms: a handful of directives, `port` and `maxstartups` among them, stayed lower case until 10.5 converted those too. Published CIS audit text still uses the lower-case form throughout. Always add `-i`, as the command above does.

05 Cryptographic algorithms

In plain terms, a *cipher* encrypts the traffic, a *MAC* proves it was not tampered with, and *key exchange* is how the two ends agree on a shared secret. Every version here picks strong ones by default and drops weak ones each release, so the short answer is: on Ubuntu and CachyOS leave the defaults alone, on Fedora and RHEL set a system-wide policy, and touch FIPS only where a regulation requires it.

5.1 Fedora and RHEL: use the system-wide crypto policy

On these systems the policy reaches sshd through a packaged drop-in that includes `/etc/crypto-policies/back-ends/opensshserver.config`. Fedora 44 and RHEL 10 put that include in `40-redhat-crypto-policies.conf`; RHEL 9 keeps it at the top of `50-redhat.conf`. The file says so itself: "The changes to crypto properties (Ciphers, MACs, ...) will not have any effect in this or following included files." Because the first value obtained wins, algorithm lines in your `10-hardening.conf` are read first and would silently override the policy, while the same lines further down the main `sshd_config` would be ignored. CIS makes the same point and adds one of its own: RHEL 9's check 1.6.2 fails a system that opts sshd out of the policy with `CRYPTO_POLICY=` in `/etc/sysconfig/sshd`. Set a policy instead:

```
update-crypto-policies --show                # what is in force now
sudo update-crypto-policies --set FUTURE     # stricter; DEFAULT is the baseline
sudo systemctl restart sshd
```

`FUTURE` is a conservative policy meant to withstand near-term future attacks at the cost of interoperability: RSA keys and Diffie-Hellman parameters above 3071 bits, prime curves of at least 255 bits, and 256-bit authenticated symmetric ciphers only. It can reject older clients, so test connectivity from every machine that needs in before you close your session. To adjust one algorithm class rather than the whole level, write a sub-policy: a `.pmod` file in `/etc/crypto-policies/policies/modules/` applied on top of a base policy, which is also how CIS remediates its individual cipher and MAC checks.

5.2 Ubuntu and CachyOS: rely on the OpenSSH defaults

They are current and strong. If a scanner insists on a change, subtract the one algorithm rather than pinning a whole list, for example `MACs -hmac-sha1`. A hardcoded list copied from an older guide, including older CIS algorithm lists, tends to *remove* the newer and stronger algorithms that shipped after the list was written.

5.3 Post-quantum key exchange

On a current system, this needs nothing from you.

It is already the default on every version covered here

OpenSSH has defaulted to a hybrid post-quantum key exchange since 9.0 (2022), when `sntrup761x25519-sha512` became the default for exactly this reason: an adversary who records traffic today could decrypt it once a capable quantum computer exists. OpenSSH 10.0 moved the default to `mlkem768x25519-sha256`, pairing classical X25519 with NIST's module-lattice key encapsulation mechanism. The OpenSSH 9.9 in RHEL 10 and RHEL 9.8 prefers the earlier sntrup761 hybrid, which is post-quantum as well.

OpenSSH 10.1 and later will warn you when it is missing

The client prints a warning when a connection negotiates a key agreement that is not post-quantum, because traffic captured today can be decrypted later once a quantum computer exists. The `WarnWeakCrypto` option controls it and defaults to on. Seeing that warning means the other end is old, and it is a prompt to upgrade the other end rather than to silence the warning.

On RHEL, the crypto policy decides

RHEL 10.0 shipped post-quantum cryptography as a Technology Preview and RHEL 10.1 makes it fully supported; both carry `mlkem768x25519-sha256` and `sntrup761x25519-sha512` for SSH key exchange. Since the policy decides what sshd actually offers, check the server rather than assuming: `sudo sshd -T | grep -i kexalgorithms` and look for `mlkem768x25519-sha256` or `sntrup761x25519-sha512`. `ssh -Q KexAlgorithms` answers a different question: what the binary supports, rather than what is enabled.

CIS has begun checking for it

The CIS Ubuntu 24.04 Benchmark v2.0.0 added 5.1.23, a Level 1 check that a post-quantum key exchange algorithm is available to the SSH server. The RHEL 9 and RHEL 10 benchmarks have no equivalent yet.

5.4 FIPS mode, only if a regulation requires it

For federal or regulated environments that require FIPS 140-3 validated cryptography (NIST SC-13), FIPS mode constrains SSH and the rest of the system to approved algorithms. How you get there depends on the version, and on RHEL 10 the answer changed:

- **RHEL 9 and Fedora:** `sudo fips-mode-setup --enable`, then reboot. On RHEL this has been deprecated since 9.5 but still works as a post-install switch.
- **RHEL 10:** `fips-mode-setup` has been removed and switching to FIPS mode after installation is **not supported**. FIPS mode must be enabled during installation. On an already-installed RHEL 10 host this is a permanent finding until the host is rebuilt, so decide before you provision.

Setting the FIPS crypto policy on its own is not the same as enabling FIPS mode, and enabling the mode after installation does not by itself demonstrate validated compliance. Red Hat recommends installing in FIPS mode, and on RHEL 10 that is the only supported path.

06 Brute force: what sshd does already, and when to add fail2ban

Since OpenSSH 9.8 the server penalises repeat offenders itself, on by default, with no package to install and no log parsing involved. That covers much of what guides written before mid-2024 install fail2ban for. Start by seeing what you already have.

6.1 What sshd already does

`PerSourcePenalties` refuses connections from a source address, and from others in the same network block, for a period that grows as the offenses repeat. Confirm it is on:

```
sudo sshd -T | grep -i persourcepenalties
```

```
persourcepenalties crash:90 authfail:5 noauth:1 invaliduser:5
  grace-exceeded:10 refuseconnection:10 max:600 min:15
  max-sources4:65536 max-sources6:65536 overflow:permissive
  overflow6:permissive
```

Read as: a failed authentication earns 5 seconds, an attempt at a username that does not exist earns 5 seconds, failing to authenticate within `LoginGraceTime` earns 10 seconds, and crashing the daemon earns 90. Penalties accumulate to a maximum of 600 seconds (10 minutes) and are only enforced once 15 seconds have accrued, so one fumbled login costs you nothing. By default each address is treated individually (`PerSourceNetBlockSize` is 32:128). OpenSSH 10.3 and later print the same values with decimal places (`crash:90.000000`), and 10.5 prints the keyword itself as `PerSourcePenalties`. Both are display changes only, and the `-i` above already handles the second.

Penalties are on by default in OpenSSH 9.8 and later, which means Fedora 44, Ubuntu 26.04, CachyOS, RHEL 10, and RHEL 9.8 and later. RHEL 9.7 and earlier shipped OpenSSH 8.7 and do not have it.

- **Exempt your own address if you need to**
  If a shared address, a NAT gateway, or a monitoring probe generates enough failures to get itself refused, add `PerSourcePenaltyExemptList` to the drop-in with the addresses or ranges to leave alone. Turning penalties off entirely (`PerSourcePenalties no`) discards a free control; exempt the specific source instead.

6.2 When fail2ban is still worth running

fail2ban watches the logs and bans an address at the firewall. It implements part of NIST AC-7 (Unsuccessful Logon Attempts). Compared with what sshd now does on its own, it adds a longer ban, a ban that covers every port rather than only SSH, one place to see and manage bans across several services, and the option to ban on things sshd does not penalise. It is worth running where SSH is publicly reachable and you want the auth log quiet. It adds little on a host that is only reachable over an overlay network (Step 8), where there is no internet traffic left for it to observe.

6.3 Install it

```
# Fedora
sudo dnf install fail2ban

# Ubuntu
sudo apt install fail2ban

# CachyOS
sudo pacman -S fail2ban
```

On **RHEL**, fail2ban is in neither the base nor the AppStream repository; it comes from EPEL, the Fedora project's package repository for enterprise Linux. On RHEL 10 (substitute the matching number for RHEL 9):

```
sudo subscription-manager repos --enable codeready-builder-for-rhel-10-$(arch)-rpms
sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-10.noarch.rpm
sudo dnf install fail2ban
```

On Rocky Linux, AlmaLinux, and CentOS Stream, `sudo dnf install epel-release` replaces both of those lines and there is no subscription step. EPEL packages generally need the CodeReady Builder repository as well; AlmaLinux 10 turns it on for you, and where it is off the command differs by package manager: `sudo dnf config-manager --set-enabled crb` on the dnf 4 systems (the 9 series) and `sudo dnf config-manager enable crb` on dnf 5 (the 10 series). EPEL 10 currently carries fail2ban 1.1.0, the same version Fedora and Ubuntu ship; upstream's latest release is 1.1.1 (August 2026).

6.4 Configure it, and check what your package already did

Two of the settings most guides tell you to write are already set for you, and the third differs by distribution. Check before you write anything:

```
sudo fail2ban-client status                # which jails are running
sudo fail2ban-client get sshd actions      # how bans are applied
```

Ubuntu and Debian: the sshd jail is already on

The package ships `/etc/fail2ban/jail.d/defaults-debian.conf`, which enables the `[sshd]` jail, sets `backend = systemd`, and sets the ban action to `nftables`. Installing the package is enough to get SSH protection. The nftables action builds its own `f2b-table` at hook priority -1, ahead of the ufw rules, so the two coexist and you do not have to choose.

Fedora, RHEL, and CachyOS: every jail starts disabled

Upstream ships all jails off ("it should stay this way", as the shipped config puts it), so you enable `[sshd]` yourself. The journal backend is already correct: the Fedora and Arch path files both set `sshd_backend = systemd`. On Fedora and RHEL the `fail2ban-firewalld` subpackage drops in `/etc/fail2ban/jail.d/00-firewalld.conf`, which sets `banaction = firewallcmd-rich-rules`; if `get sshd actions` shows an iptables action instead, install that subpackage.

Never edit `jail.conf`

It is replaced on upgrade. Put your settings in `/etc/fail2ban/jail.local` or a file under `/etc/fail2ban/jail.d/`, which are read afterwards and left alone.

A minimal `jail.local`, holding only what you are actually changing:

```
sudo nano /etc/fail2ban/jail.local
```

```
[DEFAULT]
# Never ban these. Add the address you connect from so a run of fumbled
# logins cannot lock you out ('curl ifconfig.me' on your laptop shows it).
# Single addresses or ranges such as 192.168.1.0/24 both work. Skip your
# own address if it changes often, or you may end up trusting a stranger.
ignoreip = 127.0.0.1/8 ::1

# 4 failures in 10 minutes earns a 1 hour ban (matches MaxAuthTries 4).
maxretry = 4
findtime = 10m
bantime  = 1h

[sshd]
enabled = true
# Fedora, RHEL, and CachyOS need this line; Ubuntu already sets it.
backend = systemd
# If you changed the SSH port in Step 7, set it here, e.g. port = 2222
port    = ssh
```

```
sudo systemctl enable --now fail2ban
sudo fail2ban-client status sshd
```

The status output shows the filter's failure count and the current ban list. The package defaults are 5 failures in 10 minutes and a 10 minute ban; the values above are tighter and line up with the `MaxAuthTries 4` you set in Step 4. A jail that starts cleanly but never records a failure, on a host whose journal clearly shows failed logins, almost always means the jail is reading a log file instead of the journal. A host that does not run rsyslog has no `/var/log/auth.log` or `/var/log/secure` for it to read, which is exactly what `backend = systemd` fixes.

07 Optional: change the SSH port

A non-standard port reduces automated scan noise and is not a security control: anyone who scans the host finds the service. An overlay network (Step 8) removes the public path instead of moving it. If you want the port change anyway, it touches the firewall, SELinux on Fedora and RHEL, and the SSH configuration, in that order. The example uses 2222.

7.1 Open the new port, leaving 22 open for now

```
# Fedora and RHEL (firewalld)
sudo firewall-cmd --add-port=2222/tcp --permanent
sudo firewall-cmd --reload

# Ubuntu and CachyOS (ufw)
sudo ufw limit 2222/tcp
```

`limit` rather than `allow`, so the new port keeps the same rate limiting the old one had from Step 1. Remove it later with `ufw delete limit 2222/tcp`.

7.2 On Fedora and RHEL, label the port for SELinux

SELinux runs in enforcing mode by default on both and only lets sshd bind ports labelled as SSH ports, so without this the daemon cannot start on the new port:

```
sudo semanage port -a -t ssh_port_t -p tcp 2222
```

If `semanage` is missing, install `policycoreutils-python-utils`. Ubuntu and CachyOS need no equivalent.

7.3 Set the port

```
echo 'Port 2222' | sudo tee -a /etc/ssh/sshd_config.d/10-hardening.conf
sudo sshd -t
```

7.4 Apply it

```
# Fedora, RHEL, and CachyOS: the daemon owns the port, so restart it
sudo systemctl restart sshd

# Ubuntu: the socket owns the port, so re-run the generators and restart it
sudo systemctl daemon-reload
sudo systemctl restart ssh.socket
```

- **Ubuntu no longer needs a hand-written socket drop-in**
  Under socket activation, systemd holds the listening port, so for a long time a `Port` line in `sshd_config` did nothing and you had to write an `ssh.socket.d` override by hand. From Ubuntu 24.04 a systemd generator reads `/etc/ssh/sshd_config`, follows its `Include` lines into `sshd_config.d/`, and writes the socket's listen addresses for you, which is why `daemon-reload` is in the command above: it is what re-runs the generator. Guides that still tell you to write `ListenStream=` by hand are describing 22.10 through 23.10.

7.5 Test the new port from the second terminal, then close 22

```
ssh -p 2222 you@server
# only once that succeeds:
sudo firewall-cmd --remove-service=ssh --permanent && sudo firewall-cmd --reload   # Fedora/RHEL
sudo ufw delete limit OpenSSH    # Ubuntu
sudo ufw delete limit ssh        # CachyOS
```

Use `-p 2222` from now on, and set `port = 2222` in the fail2ban jail so it still watches the right service.

08 Take SSH off the public internet

This is the strongest control for a server, and it maps to NIST SC-7 (Boundary Protection), SC-7(5) (Deny by Default), CM-7 (Least Functionality), and AC-17(3) (Managed Access Control Points). A publicly reachable SSH port is scanned and attacked continuously. Make it unreachable from the public internet and none of that traffic can touch it.

The modern way to do this without a VPN appliance is an **identity-based overlay network**: every device gets a private address on an encrypted mesh, only enrolled and authenticated devices can reach each other, and you then firewall SSH so it is reachable over that mesh alone. Two options worth knowing:

Tailscale

A mesh VPN built on WireGuard. Each device joins your private network (a "tailnet"), gets a stable `100.x` address only tailnet members can reach, and works from behind home and office routers. Access between devices is governed by policy rules. It also offers **Tailscale SSH**, where Tailscale handles SSH authentication itself using device identity rather than SSH keys, including a "check mode" that forces re-authentication (and can trigger multi-factor) before high-risk connections such as connecting as root.

Nebula, and Defined Networking

Nebula is an open-source (MIT) overlay created at Slack. A certificate authority signs a certificate for each node, nodes mutually authenticate by validating certificates, and always-reachable "lighthouse" nodes let peers find each other and punch through NAT without opening inbound ports. It has a built-in group-based host firewall. Defined Networking sells a managed control plane for it, so you do not run the CA or distribute certificates by hand.

The procedure is the same for both. Get onto the overlay, prove a login over it works, and only then remove the public rule.

8.1 Install the overlay client and join

Install Tailscale from the vendor's repository for your distribution (the instructions are at **`tailscale.com/download`**), then bring the node onto your tailnet:

```
sudo tailscale up      # prints a URL; open it to sign in and authorize this device
tailscale ip -4        # shows this node's private 100.x address
```

8.2 From the SECOND terminal, confirm SSH works over the overlay

```
ssh you@100.x.y.z
```

Do not proceed until this succeeds

THE PUBLIC RULE IS YOUR ONLY WAY BACK IN UNTIL THE OVERLAY LOGIN WORKS

Removing the public SSH rule before an overlay login is proven leaves you with no route to the host at all. The public rule is still in place at this point, so your current session is safe; keep it that way until 8.2 passes.

8.3 Allow the overlay interface

- **These rules admit overlay traffic to every port on the host**
  Allowing the interface, or putting it in firewalld's `trusted` zone, opens the whole host to anything on your mesh. That is the usual intent on a small private network of your own devices, and it is worth deciding deliberately. To admit SSH alone, use `sudo ufw allow in on tailscale0 to any port 22 proto tcp`, or a firewalld zone that lists only the `ssh` service in place of `trusted`.

```
# Ubuntu and CachyOS (ufw)
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow in on tailscale0
sudo ufw reload

# Fedora and RHEL (firewalld): put the overlay interface in the trusted zone
sudo firewall-cmd --permanent --zone=trusted --change-interface=tailscale0
sudo firewall-cmd --reload
```

`tailscale0` is the virtual interface Tailscale creates; for Nebula substitute its interface, typically `nebula1`. On Fedora and RHEL the `--permanent` change only takes effect after the reload; confirm with `sudo firewall-cmd --zone=trusted --list-interfaces`. The interface is created by the overlay's own daemon and is normally not managed by NetworkManager, so the binding survives a reboot. Where NetworkManager does manage it, pin the zone as well so a reconnect cannot reset it to `public`: `sudo nmcli connection modify <connection> connection.zone trusted`.

8.4 Only now, remove the public SSH rule

```
# Ubuntu (rule named OpenSSH) or CachyOS (rule named ssh)
sudo ufw delete limit OpenSSH     # CachyOS: sudo ufw delete limit ssh
sudo ufw reload

# Fedora and RHEL (firewalld): drop ssh from the public zone
sudo firewall-cmd --permanent --zone=public --remove-service=ssh
sudo firewall-cmd --reload
```

After this, SSH to the server's public address times out, while SSH to its overlay address still works for authenticated members. Unauthenticated internet hosts have nothing to connect to, so the public attack surface for SSH is gone.

8.5 The limits of this

- You take on a control plane you have to trust. Tailscale's coordination server (or a self-hosted Headscale), or Defined Networking's service, which holds and rotates the Nebula CA key for you, becomes part of your trust boundary.
- Host hardening still matters, because an authorized or compromised overlay peer can still reach `sshd`. Keep the key-only authentication and the drop-in from Steps 2 to 5.
- Tailscale SSH is a separate mechanism from OpenSSH. It authenticates by node identity rather than SSH keys, claims port 22 on the Tailscale address only, cannot be moved to another port, and ties authorization to Tailscale's policy engine. Your normal `sshd` and its non-Tailscale connections keep working alongside it.

MFA Optional: a second factor

For higher-assurance servers, add a second factor (NIST IA-2(1), multi-factor authentication to privileged accounts). Configure either one with a second session open, the same as any other authentication change.

Hardware FIDO2 keys, the low-friction option

`ssh-keygen -t ed25519-sk` creates a key bound to a security token. The private key cannot leave the hardware and a physical touch is required for each use, which defeats both key theft and remote misuse of a forwarded agent. It needs OpenSSH 8.2 or newer (every version here qualifies) and a token that supports Ed25519; for older tokens use `ecdsa-sk`. Nothing else in your configuration changes, because it is still public key authentication.

A time-based one-time code through PAM

Install a PAM one-time-password module, enable it in `/etc/pam.d/sshd`, and require both factors with `AuthenticationMethods publickey,keyboard-interactive:pam` in your drop-in. That forces an SSH key **and** a code. Two catches. Step 4 set `KbdInteractiveAuthentication no`, and the code prompt cannot appear until you change it to `yes`. The other is that keyboard-interactive asks PAM: if `/etc/pam.d/sshd` still consults the account password alongside the one-time-password module, a password can satisfy the second factor in place of the code. The key is still required either way, but the factor is weaker than intended, so order the PAM stack to require the code. Run the enrollment once per user, validate with `sudo sshd -t`, and test in a second terminal.

CHECK Verification summary

```
# A key-only login as an ssh-users member works, with no password prompt
ssh you@server

# The effective configuration matches what you intended
sudo sshd -T | grep -Ei 'passwordauthentication|kbdinteractive|permitrootlogin'
sudo sshd -T | grep -Ei 'allowgroups|maxauthtries|logingracetime|loglevel|banner'

# sshd's own penalties are active
sudo sshd -T | grep -i persourcepenalties

# fail2ban is guarding sshd, if you installed it
sudo fail2ban-client status sshd

# The crypto policy in force (Fedora and RHEL)
update-crypto-policies --show

# A post-quantum key exchange is on offer
sudo sshd -T | grep -i kexalgorithms | tr ',' '\n' | grep -E 'mlkem|sntrup'
```

Keep the `-i` on every one of these greps, for the reason given in Step 4.5.

ALIGN CIS and NIST alignment

There is no standalone "CIS OpenSSH" benchmark. The SSH server controls live in section 5.1 of each per-distribution CIS Linux Benchmark, and the numbering differs between them: RHEL 10 and Ubuntu 24.04 list the settings alphabetically, while RHEL 9 puts the algorithm checks earlier. Verify against the exact benchmark you must meet. The numbers below are from CIS RHEL 9 v2.0.0, CIS RHEL 10 v1.0.1, and CIS Ubuntu 24.04 LTS v2.0.0. CIS has published no benchmark for Ubuntu 26.04 LTS as of August 2026, so 24.04 v2.0.0 is the nearest applicable edition; Fedora and Arch-based systems have no current benchmark at all, and their coverage here is by analogy.

Who may log in, and how they authenticate

SETTING IN THE DROP-IN	CIS RHEL 9 v2.0.0	CIS RHEL 10 v1.0.1 AND UBUNTU 24.04 v2.0.0	NIST 800-53 R5
`AllowGroups ssh-users`	5.1.7	5.1.4	AC-6
`GSSAPIAuthentication no` (Level 2 on a server)	5.1.11	5.1.9	CM-7
`HostbasedAuthentication no`	5.1.12	5.1.10	IA-2
`PermitEmptyPasswords no`	5.1.19	5.1.19	IA-5
`PermitRootLogin no`	5.1.20	5.1.20	AC-6

The remaining settings, plus files and algorithms

SETTING IN THE DROP-IN	CIS RHEL 9 v2.0.0	CIS RHEL 10 v1.0.1 AND UBUNTU 24.04 v2.0.0	NIST 800-53 R5
`Banner /etc/issue.net`	5.1.8	5.1.5	AC-8
`ClientAliveInterval` and `CountMax`	5.1.9	5.1.7	AC-12
`DisableForwarding yes` (Level 2 on a server)	5.1.10	5.1.8	CM-7
`IgnoreRhosts yes`	5.1.13	5.1.11	AC-6
`LoginGraceTime 60`	5.1.14	5.1.13	AC-12, SC-5
`LogLevel VERBOSE`	5.1.15	5.1.14	AU-2, AU-3, AU-12
`MaxAuthTries 4`	5.1.16	5.1.16	AC-7
`MaxStartups 10:30:60`	5.1.17	5.1.17	SC-5
`MaxSessions 10`	5.1.18	5.1.18	AC-10
`PermitUserEnvironment no`	5.1.21	5.1.21	CM-6
`UsePAM yes`	5.1.22	5.1.22	IA-2
File permissions on the config and host keys	5.1.1 to 5.1.3	5.1.1 to 5.1.3	CM-6, AC-6
Algorithms set by the crypto policy (Step 5)	5.1.4 to 5.1.6, 1.6.2	5.1.6, 5.1.12, 5.1.15; Ubuntu 5.1.23	SC-8, SC-13

The 5.2.x numbers quoted elsewhere come from the CIS Distribution Independent Linux Benchmark, which CIS has since archived. The per-distribution benchmarks above are the maintained ones.

Four things in this guide sit outside the benchmarks, because CIS leaves them to site policy:

- Key-only authentication: no CIS check requires `PasswordAuthentication no`, even though it is the highest-value setting here. NIST IA-2 and IA-5 cover it, and NIST IR 7966 covers the key management around it, meaning passphrases, no shared private keys, and rotation.
- Brute-force response, whether from sshd's own penalties or fail2ban: NIST AC-7.
- Taking SSH off the public internet: NIST SC-7, SC-7(5), CM-7, and AC-17(3).
- A second factor: NIST IA-2(1).

NIST IR 7966, *Security of Interactive and Automated Access Management Using Secure Shell (SSH)* (October 2015), is NIST's dedicated SSH publication and is still current. It covers SSH access and key management (provisioning, rotation, restrictions on automation keys, and logging key fingerprints) and places line-by-line `sshd` hardening out of scope, which is why it pairs with the CIS benchmarks rather than replacing them.

HELP Troubleshooting

- **Locked out of a remote server**
  Use the provider's serial or recovery console, then either delete `/etc/ssh/sshd_config.d/10-hardening.conf` outright or re-enable `PasswordAuthentication` in it temporarily, and reload sshd. This is why you keep a session open and test from a second terminal.

- **Locked out by `AllowGroups` or `PermitRootLogin no`**
  Your account is not in `ssh-users`, or root was the only account. Recover through the console, add the user to the group, confirm with `id`, and re-apply.

- **A `grep` of `sshd -T` returns nothing at all**
  On OpenSSH 10.4 and later the keyword names come back in mixed case. Add `-i`. The same problem catches older audit scripts and the published CIS audit snippets.

- **Your own address got refused for a while**
  That is `PerSourcePenalties` doing its job. Wait it out (the maximum is 10 minutes) or add the address to `PerSourcePenaltyExemptList`. If fail2ban banned you instead, clear it with `sudo fail2ban-client set sshd unbanip <your-ip>` and add the address to `ignoreip`.

- **fail2ban starts but never records a failure**
  The jail is reading a log file that does not exist on a journal-only system. Confirm `backend = systemd` is set for the jail, and on Ubuntu that `python3-systemd` is installed (it comes with the package).

- **Cipher settings appear to be ignored on Fedora or RHEL**
  Expected. The crypto-policy include is read first and the first value obtained wins. Use `update-crypto-policies` or a `.pmod` sub-policy (Step 5). Moving the lines into a low-numbered drop-in makes it worse: it overrides the policy instead of losing to it.

- **A port change on Ubuntu did not take effect**
  The socket owns the port, and the generator that reads your `Port` line only runs on `systemctl daemon-reload`. Run that, then `sudo systemctl restart ssh.socket`. Check the result with `systemctl show ssh.socket -p Listen`.

- **sshd will not start on a new port (Fedora or RHEL)**
  SELinux is blocking the bind. Label the port: `sudo semanage port -a -t ssh_port_t -p tcp <port>`.

- **Clients warn about a non-post-quantum key exchange**
  Your client is OpenSSH 10.1 or later and the far end offered no post-quantum key exchange, so it predates OpenSSH 9.0 or has been restricted to classical algorithms by its configuration or crypto policy. Fix the far end. Silencing the warning with `WarnWeakCrypto no` hides the finding without changing it.

ROLLBACK How to roll back

```
# Undo the SSH hardening: one file, one reload
sudo rm /etc/ssh/sshd_config.d/10-hardening.conf
sudo sshd -t && sudo systemctl reload sshd     # Fedora, RHEL, CachyOS
sudo sshd -t && sudo systemctl reload ssh      # Ubuntu

# Stop and disable fail2ban
sudo systemctl disable --now fail2ban

# Revert the crypto policy (Fedora and RHEL)
sudo update-crypto-policies --set DEFAULT && sudo systemctl restart sshd
```

- **Reverting a port change needs care**
  A running listener keeps the old port until it is fully restarted rather than reloaded, so do it in this order. First remove the `Port` line from the drop-in and **restart** (on Ubuntu: `systemctl daemon-reload` then `systemctl restart ssh.socket`), so the listener rebinds to 22. Second, re-open port 22 in the firewall and confirm a login on 22 works. Only then remove the custom-port firewall rule and, on Fedora and RHEL, the SELinux label: `sudo semanage port -d -t ssh_port_t -p tcp <port>`.

DONE What you achieved

- SSH accepts **keys only**, from members of one defined group, with **no root login**, so password brute force cannot succeed and access follows least privilege.
- The login window, attempt count, forwarding, and pre-authentication surface are set to current CIS values in a **single drop-in file** you can remove in one command.
- Authentication is logged with the key fingerprint, so you can tell later which key was used.
- Cryptography is left to strong, self-updating defaults, or to the system crypto policy on Fedora and RHEL, and every version covered here negotiates a hybrid post-quantum key exchange by default.
- Repeat offenders are penalised by **sshd itself**, and banned at the firewall by fail2ban where the port is still publicly reachable.
- If you took Step 8, SSH is reachable only **over an identity-based overlay**, so unauthenticated internet hosts have nothing to connect to, within the limits set out in Step 8.5.

REF References

OpenSSH release notes, and the `sshd_config(5)`, `sshd(8)`, and `ssh-keygen(1)` manual pages · `openssh.com/releasenotes.html` · `man.openbsd.org`

CIS Benchmarks: RHEL 9 v2.0.0, RHEL 10 v1.0.1, Ubuntu 24.04 LTS v2.0.0 · `cisecurity.org/cis-benchmarks`

NIST SP 800-53 Rev 5, and NIST IR 7966: Security of Interactive and Automated Access Management Using Secure Shell (SSH) · `csrc.nist.gov`

Red Hat: using the system-wide cryptographic policies, switching RHEL to FIPS mode, and the CVE-2025-32728 advisory · `docs.redhat.com` · `access.redhat.com/security/cve/CVE-2025-32728`

Ubuntu: socket-based sshd activation · `discourse.ubuntu.com/t/30189`

fail2ban documentation and shipped configuration · `github.com/fail2ban/fail2ban`

Tailscale SSH, and Nebula · `tailscale.com/kb/1193` · `github.com/slackhq/nebula` · `defined.net`