SSH public-key authentication

Keypairs, agent use, server validation and practical threat boundaries

The authentication exchange

In public-key SSH authentication, the server has a copy of the client's public key, while the client retains the private key. During authentication, the client proves possession of the private key by signing protocol data bound to the current SSH session and authentication request. The private key is not transmitted to the server.

A passphrase encrypts the private key file at rest. Unlocking it locally does not send the passphrase to the server. An ssh-agent can hold unlocked key material and perform signing on behalf of the client. Agent access must be protected, because a process able to use the agent may request signatures even without extracting the private key.

```
ssh-keygen -t ed25519 -a 100 -f ~/.ssh/id_ed25519
ssh-keygen -lf ~/.ssh/id_ed25519.pub
ssh-add -l
```

Enrolling and restricting keys

The server typically stores allowed public keys in ~/.ssh/authorized_keys for each account. Ownership and permissions matter when StrictModes is enabled. Each line contains a key type, base64 key and optional comment, possibly preceded by restrictions. Comments are labels, not security controls.

Per-key restrictions can limit source addresses, commands, port forwarding and PTY allocation. They are especially valuable for automated backup or deployment keys. Keep a separate interactive administrative key from unattended automation credentials.

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
# Example authorised_keys prefix:
from="192.0.2.0/24",no-agent-forwarding,no-port-forwarding ssh-ed25519 AAAA... comment
```

Client selection and host authenticity

The client uses Host entries in ~/.ssh/config to select identities and options. IdentitiesOnly yes prevents the agent from offering unrelated keys. Host-key checking authenticates the server, a separate issue from authenticating the user. Treat an unexpected host-key change as a security event until verified, not as an inconvenience to suppress.

OpenSSH certificates and hardware-backed FIDO keys are options for larger deployments or stronger protection. A hardware-backed key can reduce private-key exposure, although device and policy details matter.

```
Host production
    HostName server.example.net
    User admin
    IdentityFile ~/.ssh/id_ed25519
    IdentitiesOnly yes
    ForwardAgent no
ssh -v production
```

Troubleshooting key login

Use ssh -vvv on the client and journalctl -u ssh (or sshd) on the server. Common failures include wrong file ownership, an unexpected authorized_keys location, incorrect user, a key not offered by the agent, a denied algorithm, or an overriding Match block.

Do not assume that disabling password authentication automatically enforces a second factor. AuthenticationMethods controls which methods must succeed and in what order. Inspect the effective configuration with sshd -T and the connection-specific Match context.

```
sudo sshd -t
sudo sshd -T -C user=admin,host=server.example.net,addr=192.0.2.10
ssh -vvv admin@server.example.net
```

Key lifecycle and compromise

Keep private keys off shared storage, use distinct credentials where practical, and remove public keys promptly when access is revoked. Back up recovery access separately. A stolen encrypted private key may still be attacked offline; passphrase quality and key derivation settings matter.

Rotate keys after suspected compromise and review all places where the old public key was authorised. SSH access control also depends on account privileges, sudo policy, network reachability and patching; keys alone are not a complete hardening strategy.

```
ssh-keygen -y -f ~/.ssh/id_ed25519
ssh-add -D
```

Reference scope: commands and defaults vary by distribution and software version. Consult the locally installed manual pages for exact syntax and behaviour.