SSH Cheatsheet

SSH Cheatsheet

A practical quick-reference guide for Secure Shell operations. Commands are organized by category with common flags and realistic examples covering key management, configuration, tunneling, file transfer, and server hardening.

Key Generation

SSH keys provide cryptographic authentication. Ed25519 is the modern default; RSA is used when legacy compatibility is required.

Generate Ed25519 Key (Recommended)

```
# Default ed25519 key
ssh-keygen -t ed25519 -C "user@host"

# Specify output file
ssh-keygen -t ed25519 -f ~/.ssh/mykey -C "user@host"

# Ed25519 with custom key derivation rounds
ssh-keygen -t ed25519 -a 100 -f ~/.ssh/mykey
```

Generate RSA Key

```
# 4096-bit RSA (minimum recommended size)
ssh-keygen -t rsa -b 4096 -C "user@host"

# 4096-bit RSA with specific file path
ssh-keygen -t rsa -b 4096 -f ~/.ssh/mykey_rsa -C "user@host"
```

ssh-keygen Options

```
# Change passphrase on existing key
ssh-keygen -p -f ~/.ssh/id_ed25519

# Change key type comment
ssh-keygen -c -f ~/.ssh/id_ed25519 -C "new-comment@example.com"
```

```
# Generate public key from private key
ssh-keygen -y -f ~/.ssh/id_ed25519 > ~/.ssh/id_ed25519.pub

# Generate fingerprint of a key
ssh-keygen -l -f ~/.ssh/id_ed25519

# Generate visual (randomart) fingerprint
ssh-keygen -lv -f ~/.ssh/id_ed25519

# View public key in RFC 4716 format (for some appliances)
ssh-keygen -e -f ~/.ssh/id_ed25519.pub

# Convert OpenSSH to PEM format (for older systems)
ssh-keygen -p -f ~/.ssh/id_rsa -m pem

# Convert PEM to OpenSSH format
ssh-keygen -p -f ~/.ssh/id_rsa -m RFC4716

# Generate host keys for a server
ssh-keygen -A

# Generate CDP/PKCS#11-compatible key
ssh-keygen -t ecdsa -b 521 -f ~/.ssh/id_ecdsa
```

Key Fingerprint Verification

```
# Fingerprint of remote server (verify on first connect)
ssh-keyscan -t ed25519 github.com | ssh-keygen -lf -

# Check all known_hosts fingerprints
ssh-keygen -l -f ~/.ssh/known_hosts

# Search known_hosts for a specific host
ssh-keygen -F github.com

# Remove a specific host from known_hosts
ssh-keygen -R github.com
```

SSH Agent

The SSH agent holds decrypted private keys in memory so you authenticate once per session.

Starting the Agent

```
# Start ssh-agent (macOS usually auto-starts it)
eval "$(ssh-agent -s)"

# Start with a specific socket
eval "$(ssh-agent -a /tmp/ssh-agent.sock)"

# Kill the agent
ssh-agent -k
```

ssh-add: Managing Keys in the Agent

```
# Add default key (~/.ssh/id_ed25519, ~/.ssh/id_rsa)
ssh-add

# Add specific key
ssh-add ~/.ssh/mykey

# Add key with a lifetime (seconds)
ssh-add -t 3600 ~/.ssh/mykey

# Add key and prompt for passphrase via stdin
cat passphrase.txt | ssh-add ~/.ssh/mykey

# List keys currently in agent
ssh-add -l

# List keys with full fingerprints
ssh-add -L

# Remove a specific key from agent
ssh-add -d ~/.ssh/mykey.pub

# Remove all keys from agent
ssh-add -D

# Lock the agent (requires password to unlock)
ssh-add -x

# Unlock the agent
ssh-add -X
```

Auto-Loading Keys with Config

```
# Add to ~/.ssh/config to auto-load keys
Host *
  AddKeysToAgent yes
  IdentityFile ~/.ssh/id_ed25519
  IdentityFile ~/.ssh/id_rsa_work
```

authorized_keys

The `~/.ssh/authorized_keys` file on the server lists public keys permitted to authenticate.

Format

Each line is one key with optional options:

```
[options] key-type base64-encoded-key [comment]
```

Managing authorized_keys

```
# Append your public key to a remote server (one-liner)
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server

# Append with custom port
ssh-copy-id -i ~/.ssh/id_ed25519.pub -p 2222 user@server

# Manually append (if ssh-copy-id unavailable)
cat ~/.ssh/id_ed25519.pub | ssh user@server 'mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys
```

authorized_keys Options

```
# Restrict key to a specific command
command="echo 'No shell access allowed'",no-port-forwarding,no-X11-forwarding,no-agent-forwar

# Restrict key to specific source IPs
from="10.0.0.0/8,192.168.1.*" ssh-ed25519 AAAA... user@host

# Restrict key to specific ports
permitopen="10.0.0.1:3306",permitopen="10.0.0.1:6379" ssh-ed25519 AAAA... user@host

# Restrict to no PTY allocation
no-pty ssh-ed25519 AAAA... user@host

# Combine multiple restrictions
command="/usr/bin/backup.sh",from="10.0.0.100",no-port-forwarding,no-X11-forwarding,no-agent-

# Expiry date (OpenSSH 6.5+)
expiry="2026-12-31" ssh-ed25519 AAAA... temporary-key
```

SSH Config File (~/.ssh/config)

The SSH client config file simplifies connections with aliases and default options.

Basic Host Aliases

```
# ~/.ssh/config

# Personal server
Host myserver
    HostName 192.168.1.50
    User admin
    Port 2222
    IdentityFile ~/.ssh/id_ed25519

# GitHub
Host github.com
    HostName github.com
    User git
    IdentityFile ~/.ssh/id_ed25519_github

# Work bastion
Host bastion
    HostName bastion.company.com
    User jumpuser
```

```
    ForwardAgent yes

# Internal server via bastion
Host web-internal
    HostName 10.0.1.50
    User deploy
    ProxyJump bastion
    IdentityFile ~/.ssh/id_ed25519_work
```

Wildcard and Pattern Matching

```
# Apply settings to all hosts in a domain
Host *.company.com
    User admin
    IdentityFile ~/.ssh/id_ed25519_work

# Match all hosts (defaults)
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
    AddKeysToAgent yes
    Compression yes
    ForwardX11 no

# Match by condition (supported from OpenSSH 7.6)
Host bastion
    HostName bastion.company.com
    User jumpuser
    ForwardAgent yes

# Apply settings only from a specific host (origin check)
Match host 10.0.0.0/8
    User admin
    IdentityFile ~/.ssh/id_ed25519_internal

# Match on local user
Match host gitlab.internal User git
    IdentityFile ~/.ssh/id_ed25519_gitlab
```

Common Config Options

```
# ~/.ssh/config examples

Host myhost
    HostName example.com
    User deploy
    Port 22
    IdentityFile ~/.ssh/id_ed25519

    # Connection settings
    ConnectTimeout 10
    ConnectionAttempts 3
    ServerAliveInterval 60
    ServerAliveCountMax 3
    TCPKeepAlive yes
```

```
    # Security settings
    StrictHostKeyChecking accept-new
    UserKnownHostsFile ~/.ssh/known_hosts
    IdentitiesOnly yes

    # Forwarding settings
    ForwardAgent yes
    ForwardX11 no
    ForwardX11Trusted no

    # Logging
    LogLevel INFO

    # Ciphers and algorithms (use only if needed for legacy compat)
    # Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com
    # MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com
```

Include Directive

```
# ~/.ssh/config

# Include shared config
Include ~/.ssh/config.d/*

# Include specific files
Include ~/.ssh/config.d/work
Include ~/.ssh/config.d/personal
```

Connecting

Basic Connection

```
# Connect with default key
ssh user@host

# Connect with specific port
ssh -p 2222 user@host

# Connect with specific identity file
ssh -i ~/.ssh/mykey user@host

# Connect with verbose output (for troubleshooting)
ssh -vvv user@host

# Connect with config alias
ssh myserver
```

Connection Options

```
# Disable strict host key checking (insecure, use sparingly)
ssh -o StrictHostKeyChecking=no user@host

# Run a single command on remote host
```

```
ssh user@host "uptime && df -h"

# Run a script on remote host
ssh user@host "bash -s" < local_script.sh

# Force pseudo-terminal allocation
ssh -t user@host

# Background SSH connection (for port forwarding)
ssh -fN -L 8080:localhost:80 user@host

# Connect with a specific cipher
ssh -c aes256-gcm@openssh.com user@host

# Escape sequence: press Enter then ~ then ? for help
# ~.  - terminate connection
# ~C  - open command line
# ~#  - list forwarded connections
# ~&  - background ssh
# ~?  - list escape sequences
```

ProxyJump and ProxyCommand

Access servers behind bastion hosts or jump servers.

ProxyJump (OpenSSH 7.3+)

```
# One hop via bastion
ssh -J bastion user@internal-server

# Multiple hops
ssh -J bastion1,bastion2 user@internal-server

# Via config file
Host internal-server
    HostName 10.0.1.50
    User admin
    ProxyJump bastion

# With specific port on bastion
ssh -J user@bastion:2222 user@internal-server
```

ProxyCommand

```
# Via netcat
ssh -o ProxyCommand="nc -x bastion:1080 %h %p" user@internal-server

# Via another SSH connection
ssh -o ProxyCommand="ssh -W %h:%p bastion" user@internal-server

# Via HTTP CONNECT proxy
ssh -o ProxyCommand="socat - PROXY:proxy.example.com:%h:%p,proxyport=8080" user@internal-serv

# Via config file
```

```
Host internal-server
    HostName 10.0.1.50
    User admin
    ProxyCommand ssh -W %h:%p bastion

# Via SOCKS5 proxy using nc (OpenBSD netcat)
Host internal-server
    ProxyCommand nc -X 5 -x proxy.example.com:1080 %h %p
```

ProxyJump with SSH Config (Multi-Hop Example)

```
# ~/.ssh/config
Host bastion
    HostName bastion.company.com
    User jumpuser
    IdentityFile ~/.ssh/id_ed25519_bastion

Host staging-db
    HostName 10.10.0.50
    User postgres
    ProxyJump bastion
    IdentityFile ~/.ssh/id_ed25519_db

Host prod-app
    HostName 10.20.0.10
    User deploy
    ProxyJump bastion
    ForwardAgent yes
    IdentityFile ~/.ssh/id_ed25519_prod

# Two-hop jump: local -> bastion -> jumphost -> target
Host final-target
    HostName 10.30.0.5
    User admin
    ProxyJump staging-db
    IdentityFile ~/.ssh/id_ed25519_final
```

Port Forwarding (Tunneling)

SSH can tunnel traffic through encrypted channels to access remote services.

Local Port Forwarding

Forward a port on your local machine to a port on the remote server (or a host reachable from it).

```
# Forward local port 8080 to remote host's port 80
ssh -L 8080:localhost:80 user@remote

# Forward local port 3306 to a remote database server
# (access MySQL on 10.0.0.5:3306 via localhost:3306)
ssh -L 3306:10.0.0.5:3306 user@bastion

# Forward local port 5432 to PostgreSQL on a private server
ssh -L 5432:db.internal:5432 user@bastion
```

```
# Bind to all interfaces (0.0.0.0) instead of localhost
ssh -L 0.0.0.0:8080:localhost:80 user@remote

# Background tunnel
ssh -fN -L 8080:localhost:80 user@remote
```

Remote Port Forwarding

Forward a port on the remote server back to your local machine.

```
# Allow remote server to access your local port 3000
ssh -R 3000:localhost:3000 user@remote

# Make the remote listen on all interfaces (GatewayPorts)
ssh -R 0.0.0.0:8080:localhost:80 user@remote

# Persistent remote tunnel (keep alive)
ssh -o ServerAliveInterval=60 -fN -R 2222:localhost:22 user@remote
```

Dynamic Port Forwarding (SOCKS Proxy)

Create a SOCKS proxy that routes all traffic through the SSH connection.

```
# SOCKS5 proxy on local port 1080
ssh -D 1080 user@remote

# SOCKS5 proxy bound to all interfaces
ssh -D 0.0.0.0:1080 user@remote

# Background SOCKS proxy
ssh -fN -D 1080 user@remote

# Use with curl
curl --socks5 localhost:1080 http://internal-service:8080

# Use with environment variable
export ALL_PROXY=socks5://localhost:1080
curl http://internal-service:8080

# Use with Firefox: Settings > Network > Manual proxy > SOCKS Host: localhost, Port: 1080
```

Port Forwarding via Config File

```
# ~/.ssh/config
Host tunnel-db
    HostName bastion.company.com
    User jumpuser
    LocalForward 3306 db.internal:3306
    LocalForward 5432 db.internal:5432
    LocalForward 6379 redis.internal:6379
    ServerAliveInterval 60
    ServerAliveCountMax 3
```

```
# Connect and establish all tunnels
ssh -fN tunnel-db
```

X11 Forwarding

Run graphical applications on a remote server and display them locally.

```
# Enable X11 forwarding
ssh -X user@remote

# Trusted X11 forwarding (less secure but needed by some apps)
ssh -Y user@remote

# Via config
Host gui-server
    HostName server.example.com
    ForwardX11 yes
    ForwardX11Trusted no
```

File Transfer

scp (Secure Copy)

```
# Copy local file to remote
scp file.txt user@remote:/path/to/destination/

# Copy remote file to local
scp user@remote:/path/to/file.txt ./

# Copy directory recursively
scp -r ./local_dir/ user@remote:/path/to/destination/

# Copy with specific port
scp -P 2222 file.txt user@remote:/path/

# Copy with specific identity
scp -i ~/.ssh/mykey file.txt user@remote:/path/

# Preserve permissions and timestamps
scp -p file.txt user@remote:/path/

# Compress during transfer
scp -C largefile.tar.gz user@remote:/path/

# Copy between two remote servers (from local machine)
scp user@remote1:/path/file.txt user@remote2:/path/

# Limit bandwidth (Kbit/s)
scp -l 1024 largefile.iso user@remote:/path/
```

sftp (SSH File Transfer Protocol)

```
# Interactive SFTP session
sftp user@remote

# SFTP with specific port
sftp -P 2222 user@remote

# SFTP with config alias
sftp myserver

# Batch mode (from a file of commands)
sftp -b batch_commands.txt user@remote
```

Common SFTP Commands

```
# Inside the sftp prompt:

# Navigate remote filesystem
ls                      # list remote files
cd /path                # change remote directory
pwd                     # print remote working directory

# Navigate local filesystem
lls                     # list local files
lcd /path               # change local directory
lpwd                    # print local working directory

# Transfer files
put localfile.txt       # upload file
put -r localdir/        # upload directory recursively
get remotefile.txt      # download file
get -r remotedir/       # download directory recursively

# File operations
mkdir remote_dir        # create remote directory
rm remotefile.txt       # remove remote file
rmdir remote_dir        # remove remote directory
chmod 644 remotefile    # change remote permissions
chown user remotefile   # change remote ownership

# Other
rename oldname newname # rename remote file
symlink src target      # create symbolic link
exit                    # close connection
```

rsync over SSH

```
# Sync local directory to remote
rsync -avz -e ssh ./local_dir/ user@remote:/path/to/destination/

# Sync remote to local
rsync -avz -e ssh user@remote:/path/to/source/ ./local_dir/

# Sync with progress and partial transfer support
rsync -avzP -e ssh ./local_dir/ user@remote:/path/
```

```
# Dry run (show what would be transferred)
rsync -avzn -e ssh ./local_dir/ user@remote:/path/

# Delete files on destination that don't exist locally
rsync -avz --delete -e ssh ./local_dir/ user@remote:/path/

# Use with custom port and key
rsync -avz -e "ssh -p 2222 -i ~/.ssh/mykey" ./dir/ user@remote:/path/

# Exclude patterns
rsync -avz --exclude='*.log' --exclude='node_modules/' -e ssh ./dir/ user@remote:/path/
```

sshd_config Hardening

Secure the SSH server by editing `/etc/ssh/sshd_config`.

Essential Hardening Settings

```
# /etc/ssh/sshd_config

# === Authentication ===

# Disable root login
PermitRootLogin no

# Disable password authentication (keys only)
PasswordAuthentication no

# Disable empty passwords
PermitEmptyPasswords no

# Disable keyboard-interactive authentication
KbdInteractiveAuthentication no

# Use PAM (set to no if not needed)
UsePAM no

# === Key and Algorithm Settings ===

# Accept only strong key types
PubkeyAcceptedAlgorithms ssh-ed25519,sk-ssh-ed25519@openssh.com,ecdsa-sha2-nistp256,rsa-sha2-

# Strong host key algorithms
HostKeyAlgorithms ssh-ed25519,ecdsa-sha2-nistp256,rsa-sha2-512

# Strong ciphers
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com

# Strong MACs
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com

# Strong KEX algorithms
KexAlgorithms curve25519-sha256,curve25519-sha256@libssh.org,diffie-hellman-group16-sha512,di

# === Access Control ===
```

```
# Allow only specific users
AllowUsers deploy admin

# Allow only specific groups
AllowGroups ssh-users

# Deny specific users
DenyUsers guest

# === Network Settings ===

# Change default port
Port 2222

# Listen on specific address only
ListenAddress 0.0.0.0
ListenAddress ::

# === Session Settings ===

# Set idle timeout (seconds)
ClientAliveInterval 300
ClientAliveCountMax 2

# Limit max authentication attempts
MaxAuthTries 3

# Limit max sessions per connection
MaxSessions 5

# Limit max unauthenticated connections
MaxStartups 3:50:10

# Disable .rhosts file
IgnoreRhosts yes

# Disable host-based authentication
HostbasedAuthentication no

# Disable X11 forwarding if not needed
X11Forwarding no

# Disable tunneling if not needed
PermitTunnel no

# Disable agent forwarding if not needed
AllowAgentForwarding no

# Disable port forwarding if not needed
AllowTcpForwarding no
PermitOpen none

# Disable reverse shell access
PermitUserEnvironment no

# === Logging ===
SyslogFacility AUTH
LogLevel VERBOSE
```

```
# === Banner ===
Banner /etc/ssh/banner
```

Applying sshd_config Changes

```
# Check configuration syntax before reloading
sudo sshd -t

# Restart sshd (systemd)
sudo systemctl restart sshd

# Reload sshd (preserves connections)
sudo systemctl reload sshd

# Check sshd status
sudo systemctl status sshd

# Check sshd is listening on expected port
ss -tlnp | grep sshd
```

Troubleshooting

Debug Connection Issues

```
# Verbose connection (show handshake, auth, and key exchange)
ssh -vvv user@host

# Show only auth debugging
ssh -vv user@host

# Test connection without executing a command
ssh -o BatchMode=yes -o ConnectTimeout=5 user@host echo "OK"

# Check which key is being offered
ssh -v user@host 2>&1 | grep "Offering"

# Check supported algorithms on remote server
ssh -Q key
ssh -Q cipher
ssh -Q mac
ssh -Q kex
```

Common Issues

```
# "Permission denied (publickey)"
# -> Check key permissions
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
chmod 600 ~/.ssh/authorized_keys

# Check if key is loaded in agent
ssh-add -l
```

```
# "Connection refused"
# -> Verify SSH is running and on expected port
systemctl status sshd
ss -tlnp | grep sshd

# "Connection timed out"
# -> Check firewall, security groups, or network routing
telnet host 22
nc -zv host 22
nmap -p 22 host

# "Host key verification failed"
# -> Server reinstalled or IP changed; remove old key
ssh-keygen -R hostname
ssh-keygen -R 192.168.1.50

# "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED"
# -> Possible MITM attack or server reinstall
# -> If you trust the change:
ssh-keygen -R hostname

# "Too many authentication failures"
# -> Server rejecting after MaxAuthTries
# -> Specify the correct key explicitly
ssh -i ~/.ssh/correct_key user@host

# Agent not forwarding through jump host
# -> Verify ForwardAgent yes is set in config
# -> Verify agent is running locally
echo $SSH_AUTH_SOCK
ssh-add -l
```

Test SSH Config

```
# Show effective config for a specific host
ssh -G myserver

# Show effective config for a specific host (verbose)
ssh -Gv myserver

# Test connection with specific config file
ssh -F /path/to/custom_config user@host

# Check which config file is being used
ssh -V
```

Certificate-Based Authentication

SSH certificates provide centralized key management, avoiding the need to distribute individual public keys.

Certificate Types

- **User certificates** — authenticate users to servers
- **Host certificates** — authenticate servers to clients

Generate a Certificate Authority (CA)

```
# Generate CA key (do this once, keep it secure)
ssh-keygen -t ed25519 -f ca_key -C "SSH Certificate Authority"

# The public key (ca_key.pub) is what you distribute
```

Sign User Keys

```
# Sign a user's public key, valid for 52 weeks
ssh-keygen -s ca_key \
  -I user_john_2026 \
  -n john,deploy \
  -V +52w \
  ~/.ssh/id_ed25519.pub

# Sign with specific validity period
ssh-keygen -s ca_key \
  -I user_john_2026 \
  -n john \
  -V 20260101:20261231 \
  -O no-port-forwarding \
  ~/.ssh/id_ed25519.pub

# View certificate details
ssh-keygen -L -f ~/.ssh/id_ed25519-cert.pub
```

Sign Host Keys

```
# Sign a host's public key
ssh-keygen -s ca_key \
  -I host_webserver_2026 \
  -h \
  -n webserver.example.com,10.0.0.50 \
  -V +52w \
  /etc/ssh/ssh_host_ed25519_key.pub

# View host certificate details
ssh-keygen -L -f /etc/ssh/ssh_host_ed25519_key-cert.pub
```

Configure Servers to Trust the CA

```
# /etc/ssh/sshd_config — add this line
TrustedUserCAKeys /etc/ssh/ca_user.pub

# Restart sshd
sudo systemctl restart sshd
```

Configure Clients to Trust the CA

```
# ~/.ssh/config — trust host certificates signed by CA
Host *.example.com
```

```
    @cert-authority *.example.com ssh-ed25519 AAAA...CA_PUBLIC_KEY_HERE

# Or in ~/.ssh/known_hosts:
@cert-authority *.example.com ssh-ed25519 AAAA...CA_PUBLIC_KEY_HERE
```

Best Practices

Key Management

```
# Use ed25519 for all new keys (smaller, faster, more secure)
ssh-keygen -t ed25519 -C "purpose-identifier"

# Use unique keys per service (rotate independently)
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_github -C "github"
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_work -C "work"

# Always set a passphrase on private keys
# Never share or commit private keys

# Rotate keys periodically (at least annually)
# Revoke old keys from authorized_keys after rotation

# Use hardware security keys (FIDO2/U2F) for critical access
ssh-keygen -t ed25519-sk -C "hardware-key@example.com"
```

Server Security

```
# Disable password authentication entirely
PasswordAuthentication no

# Disable root login
PermitRootLogin no

# Use fail2ban to block brute-force attempts
sudo apt install fail2ban
sudo systemctl enable fail2ban

# Restrict by user or group
AllowUsers deploy admin
AllowGroups ssh-users

# Change default port to reduce noise from scanners
Port 2222

# Keep OpenSSH updated
sudo apt update && sudo apt upgrade openssh-server

# Use firewall to restrict SSH access
sudo ufw allow 2222/tcp from 10.0.0.0/8
```

Operational Practices

```
# Use SSH config to standardize connections across team members
# Store team configs in version control (excluding private keys)

# Use ProxyJump instead of chaining SSH commands
# Good: ssh -J bastion target
# Avoid: ssh bastion "ssh target"

# Set connection keepalive to prevent idle disconnects
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3

# Use -o BatchMode=yes in scripts to prevent interactive prompts
ssh -o BatchMode=yes user@host "command"

# Audit authorized_keys regularly
# Remove stale or unused entries
# Use -V expiry dates on certificate-based auth

# Monitor auth logs
sudo journalctl -u sshd -f
sudo tail -f /var/log/auth.log
```

Key Revocation (Certificate-Based)

```
# Create a Key Revocation List (KRL)
ssh-keygen -k -f revoked_keys.krl revoked_key1.pub revoked_key2.pub

# Update an existing KRL
ssh-keygen -k -u -f revoked_keys.krl new_revoked_key.pub

# Configure server to check the KRL
# /etc/ssh/sshd_config
RevokedKeys /etc/ssh/revoked_keys.krl

# Check if a certificate is revoked
ssh-keygen -Q -f revoked_keys.krl ~/.ssh/id_ed25519-cert.pub
```