

LDAP+PAM+SSH CA

- [LDAP + SSSD Client Enrollment Guide](#)
- [SSH CA setup](#)
- [Auto-Generate Signed SSH Keys](#)
- [SSH Client and Host Onboarding Guide](#)

LDAP + SSSD Client Enrollment Guide

How to enroll Linux machines into the `drheck.dev` Authentik LDAP directory service using SSSD, with offline credential caching, dual-stack LAN failover, and GDM graphical login integration.

1. Architecture & Authentication Pipeline

\$\$ \text{Client PAM / NSS} \quad \longrightarrow \quad \text{SSSD Backend (\texttt{sssd})} \quad \longrightarrow \quad \text{HAProxy LDAPS (\texttt{:636})} \quad \longrightarrow \quad \text{Rootless Outpost (\texttt{authentik-ldap :6636})} \quad \$\$

Parameter	Value
Primary LDAPS URI (IPv6 ULA)	<code>ldaps://ldap.drheck.dev:636</code>
Backup LDAPS URI (Dual-Stack IPv4/IPv6)	<code>ldaps://ul.drheck.dev:636</code>
Base DN	<code>dc=ldap,dc=goauthentik,dc=io</code>
Linux User Search Base & Filter	<code>ou=users,dc=ldap,dc=goauthentik,dc=io?sub?(memberOf=cn=locallogin,ou=groups,dc=ldap,dc=goauthentik,dc=io)</code>
Bind Service Account DN	<code>cn=ldapservice,ou=users,dc=ldap,dc=goauthentik,dc=io</code>

Parameter	Value
Local SSSD Cache DB	<code>/var/lib/sss/db/cache_goauthentik.io.ldb</code>

Offline & LAN Resilience: SSSD is configured with `enumerate = true`, `ldap_enumeration_refresh_offset = 0`, and `cache_credentials = true`. Immediately upon enrollment on the local network, SSSD enumerates and caches all members of `locallogin` into `/var/lib/sss/db/cache_goauthentik.io.ldb` and pre-populates `/var/lib/AccountsService/users/` so graphical GDM logins and `getent passwd` work even when the internet is offline.

2. Enrollment Steps

Step 0: Automated One-Liner (`ldap_deploy.py`)

Run the enrollment script from a local administrator account with `sudo`:

```
git clone -b main https://git.drheck.dev/luke/ldap_deploy.git /tmp/ldap_deploy && sudo python3 /tmp/ldap_deploy/ldap_deploy.py && rm -rf /tmp/ldap_deploy
```

The automated installer performs all steps below, verifies live LDAPS TLS connectivity (`ldap.drheck.dev:636` / `ul.drheck.dev:636`), confirms `sssctl domain-status goauthentik.io` is `Online`, verifies that LDAP users resolve via `getent passwd -s sss`, and pre-provisions GDM `AccountsService` profiles.

Step 1: Install SSSD & OpenLDAP Client Tools

Install both `sss` and the OpenLDAP client utilities (`openldap` on Arch Linux, `ldap-utils` on Debian/Ubuntu) so `ldapsearch` and `sssctl` are available for offline diagnostics:

- **Arch Linux:**

```
sudo pacman -S --needed sssd openldap
```

- **Debian / Ubuntu:**

```
sudo apt install sssd sssd-tools sssd-ldap libnss-sss libpam-sss ldap-utils
```

Installing When Internet Is Down (Arch Linux): On Arch Linux, the `sssd` package depends on `libldap` (shared libraries) rather than `openldap` (CLI tools like `ldapsearch`). If the internet is down and a package is already in the local pacman cache, install it directly with `sudo pacman -U /var/cache/pacman/pkg/sssd-*.pkg.tar.zst`. If `ldapsearch` is not installed, use the built-in `openssl s_client` and `ssctl` diagnostic commands in Section 4 below.

Step 2: Configure SSSD (`/etc/sssd/sssd.conf`)

Create backups of existing authentication configs:

```
sudo cp /etc/nsswitch.conf /etc/nsswitch.conf.bak
sudo cp /etc/pam.d/system-auth /etc/pam.d/system-auth.bak
```

Create `/etc/sssd/sssd.conf`:

```
sudo vim /etc/sssd/sssd.conf
```

Paste the following configuration:

```
[sssd]
services = nss, pam
config_file_version = 2
domains = goauthentik.io
debug_level = 1

[nss]
debug_level = 1

[pam]
debug_level = 1
```

```

[domain/goauthentik.io]
debug_level = 1
id_provider = ldap
auth_provider = ldap
sudo_provider = none
ldap_uri = ldaps://ldap.drheck.dev:636
ldap_backup_uri = ldaps://ul.drheck.dev:636
ldap_id_use_start_tls = false
ldap_search_base = dc=ldap,dc=goauthentik,dc=io
ldap_user_search_base =
ou=users,dc=ldap,dc=goauthentik,dc=io?sub?(memberOf=cn=locallogin,ou=groups,dc=ldap,dc=goauthentik,dc=io)
ldap_default_bind_dn = cn=ldapservice,ou=users,dc=ldap,dc=goauthentik,dc=io
ldap_default_authtok = hgNbhl3t1hzyV4jdfsiQuAeYS

# Secure Production TLS Configuration (Let's Encrypt)
ldap_tls_reqcert = demand
ldap_tls_cacertdir = /etc/ssl/certs

ldap_schema = rfc2307bis
ldap_user_object_class = posixAccount
ldap_user_name = sAMAccountName
ldap_user_uid_number = uidNumber
ldap_user_gid_number = gidNumber
ldap_user_home_directory = homeDirectory
ldap_user_shell = loginShell
ldap_group_object_class = posixGroup
ldap_group_name = cn
ldap_group_gid_number = gidNumber
ldap_group_member = member
ldap_ignore_unreadable_references = true
default_shell = /bin/bash
fallback_homedir = /home/%u
enumerate = true
ldap_enumeration_refresh_offset = 0
ldap_entry_usn = entryUSN
cache_credentials = true

```

Set strict ownership and `0600` permissions (SSSD refuses to start if `/etc/sss/sss.conf` is group- or world-readable):

```
sudo chmod 600 /etc/sss/sss.conf
sudo chown root:root /etc/sss/sss.conf
```

Step 3: Configure NSS (/etc/nsswitch.conf)

Open /etc/nsswitch.conf:

```
sudo vim /etc/nsswitch.conf
```

Ensure `sss` is appended to `passwd`, `group`, and `shadow`:

```
passwd: compat mymachines systemd sss
group:  compat mymachines systemd sss
shadow: compat systemd sss
```

Step 4: Configure PAM (/etc/pam.d/system-auth, su, su-l)

Open /etc/pam.d/system-auth:

```
sudo vim /etc/pam.d/system-auth
```

Replace the contents with the SSSD + `pam_mkhomedir.so` PAM stack:

Automatic Home Directory Creation: The `pam_mkhomedir.so` module in the `session` block automatically creates `/home/<username>` on first login, copying skeleton files from `/etc/skel/` with `umask=0022` (`0755` permissions).

```
##PAM-1.0
```

auth	required	pam_faillock.so	preauth
auth	sufficient	pam_sss.so	forward_pass
auth	[success=1 default=bad]	pam_unix.so	try_first_pass nullok
auth	[default=die]	pam_faillock.so	authfail

auth	optional	pam_permit.so	
auth	required	pam_env.so	
auth	required	pam_faillock.so	authsucc
account	sufficient	pam_sss.so	
account	required	pam_unix.so	
account	optional	pam_permit.so	
account	required	pam_time.so	
password	sufficient	pam_sss.so	
password	required	pam_unix.so	try_first_pass nullok sha512
shadow			
password	optional	pam_permit.so	
session	required	pam_limits.so	
session	required	pam_unix.so	
session	required	pam_mkhomedir.so	skel=/etc/skel/ umask=0022
session	optional	pam_exec.so	seteuid /usr/local/bin/gdm-ldap-
user-session.sh			
session	optional	pam_sss.so	
session	optional	pam_permit.so	

Ensure `/etc/pam.d/su` and `/etc/pam.d/su-l` include `system-auth` so `su - <ldap-user>` works from a terminal:

```
#%PAM-1.0
auth            sufficient    pam_rootok.so
auth            include       system-auth
account         include       system-auth
session         include       system-auth
password        include       system-auth
```

Step 5: Unlock GDM Login Screen & Session Selector (`AccountsService`)

Arch Linux compiles `accountsservice` without native SSSD enumeration support (see [NSS + GDM Issue](#)). Without a profile in `/var/lib/AccountsService/users/<username>`, GDM may hide remote LDAP

users from the login list or hide the desktop session selector (gear icon).

1. Create the PAM Auto-Provisioning Script (`/usr/local/bin/gdm-ldap-user-session.sh`)

```
sudo tee /usr/local/bin/gdm-ldap-user-session.sh >/dev/null << 'EOS'
#!/bin/bash
# Automatically provision AccountsService files for remote SSSD/LDAP users
# on their first login, unlocking GDM session selection on Arch Linux.

USER="$PAM_USER"
if [ -z "$USER" ]; then
    exit 0
fi

# 1. Skip local users (already in /etc/passwd)
if ! grep -q "^$USER:" /etc/passwd && id "$USER" &>/dev/null; then
    UID_NUM=$(id -u "$USER")

    # 2. Only process standard users (UID >= 1000)
    if [ "$UID_NUM" -ge 1000 ]; then
        FILE="/var/lib/AccountsService/users/$USER"
        if [ ! -f "$FILE" ]; then
            cat <<EOF > "$FILE"

[User]
Session=gnome
Icon=/home/$USER/.face
SystemAccount=false
EOF

            chown root:root "$FILE"
            chmod 600 "$FILE"
        fi
    fi
fi

fi

EOS

sudo chmod 755 /usr/local/bin/gdm-ldap-user-session.sh
```

2. Pre-Provision Existing LDAP Users Immediately (Before First Login)

Once SSSD is started in Step 6, run the following command (handled automatically by `ldap_deploy.py`) to pre-populate `/var/lib/AccountsService/users/<user>` for all enumerated LDAP users so they appear on the GDM login screen immediately without needing a prior terminal login:

```
if [ -d /var/lib/AccountsService/users ]; then
    for u in $(getent passwd -s sss | awk -F: '{print $1}'); do
        if ! grep -q "^${u}:" /etc/passwd && [ ! -f "/var/lib/AccountsService/users/${u}" ];
    then
        sudo tee "/var/lib/AccountsService/users/${u}" >/dev/null <<EOF
[User]
Session=gnome
Icon=/home/${u}/.face
SystemAccount=false
EOF
        sudo chmod 600 "/var/lib/AccountsService/users/${u}"
    fi
done
fi
```

Step 6: Start SSSD & Verify Before Logging Out

Clear any stale SSSD cache from previous attempts, then enable and start `sssd`:

```
sudo systemctl stop sssd
sudo rm -f /var/lib/sss/db/cache_goaauthentic.io.ldb
sudo systemctl enable --now sssd
```

3. Verification Checklist (Run Before Logging Out)

1. Verify SSSD Backend Online Status:

```
sudo sssctl domain-status goauthentik.io
```

Expected output:

```
Online status: Online

Active servers:
LDAP: ldap.drheck.dev

Discovered LDAP servers:
- ldap.drheck.dev
- ul.drheck.dev
```

2. **Verify SSSD LDAP User Enumeration:** Query only the `sss` NSS module to list all LDAP POSIX users resolved from Authentik:

```
getent passwd -s sss
```

Expected output:

```
akadmin*:2004:2004:akadmin:/home/akadmin:/bin/bash
brett*:2024:2024:brett:/home/brett:/bin/bash
ikura*:2022:2022:ikura:/home/ikura:/bin/bash
haruka*:2021:2021:haruka:/home/haruka:/bin/bash
luke*:2013:2013:luke:/home/luke:/bin/bash
```

(With `enumerate = true`, bare `getent passwd` also includes all five lines above at the bottom of the output. If `enumerate = false` was used on an older config, bare `getent passwd` only shows local `/etc/passwd` users and you must pass a username explicitly, e.g. `getent passwd ikura`.)

3. **Verify UID/GID and Group Memberships:**

```
id ikura
id haruka
```

4. **Verify PAM Authentication & Automatic Home Directory Creation:**

```
sudo su - ikura -c "pwd && id"
```

Or test full password authentication via SSH/PAM:

```
ssh ikura@localhost
```

4. Offline & LAN Troubleshooting Runbook

No Internet Required: All troubleshooting steps below work completely offline on the local LAN using standard system binaries (`getent`, `sssdctl`, `openssl`, `python3`, `journalctl`) even if `ldapsearch` (`openldap`) is not installed.

Issue A: `getent passwd` Only Shows `root` and Local User (`luke`), Nothing for LDAP Users

1. Test Direct Username Lookup vs. Enumeration:

```
getent passwd -s sss ikura
getent passwd -s sss haruka
```

- If `getent passwd -s sss ikura` returns a line (`ikura*:2022:...`) while bare `getent passwd` does not: `/etc/sss/sss.conf` has `enumerate = false` (or the initial 10-second enumeration delay has not finished). Set `enumerate = true` and `ldap_enumeration_refresh_offset = 0` in `/etc/sss/sss.conf` and restart `sss`.
- If `getent passwd -s sss ikura` returns nothing (exit code 2): SSSD is failing to reach or bind to `ldap.drheck.dev:636`. Proceed to **Issue B** below.

Issue B: SSSD Domain Is `Offline` or Cannot Resolve Any LDAP Users

Check SSSD's domain status and backend log:

```
sudo sssctl domain-status goauthentik.io
sudo tail -n 35 /var/log/sss/sss_goauthentik.io.log
```

Step 1: Verify DNS Resolution (`ldap.drheck.dev` and `ul.drheck.dev`)

`ldap.drheck.dev` resolves to `ul`'s ULA IPv6 address (`fd77:7777:5010:0:1337:f1ac:d00d:aaaa`), while `ul.drheck.dev` resolves to both IPv4 (`10.0.0.3`) and IPv6 (`fd77:7777:5010:0:1337:f1ac:d00d:337`):

```
getent ahosts ldap.drheck.dev
getent ahosts ul.drheck.dev
```

- If the client machine does not have a local ULA IPv6 address (`ip -6 addr show`), ensure `ldap_backup_uri = ldaps://ul.drheck.dev:636` is present in `/etc/sss/sssd.conf` so SSSD connects over IPv4 (`10.0.0.3:636`).

Step 2: Test LDAPS TCP & TLS Certificate Handshake (Without `ldapsearch`)

Use `openssl` (installed on all base systems) to verify that port `636` on `ul` is reachable and presenting a valid Let's Encrypt certificate for `*.drheck.dev`:

```
openssl s_client -connect ldap.drheck.dev:636 -servername ldap.drheck.dev </dev/null 2>&1 |
grep -E "CONNECTED|subject=|issuer=|Verify return code"
```

Or test over IPv4 via `ul.drheck.dev:636`:

```
openssl s_client -connect 10.0.0.3:636 -servername ul.drheck.dev </dev/null 2>&1 | grep -E
"CONNECTED|subject=|issuer=|Verify return code"
```

Expected output:

```
CONNECTED(00000003)
subject=CN=drheck.dev
issuer=C=US, O=Let's Encrypt, CN=YE1
Verify return code: 0 (ok)
```

- If **Verify return code is non-zero or clock is skewed**: Check `timedatectl status` on the client laptop (if the CMOS clock is wrong when offline, TLS certificate validation against Let's Encrypt will fail).

Step 3: Check for Invalid credentials (49) / Bind Password Mismatch

If `openssl s_client` succeeds with `Verify return code: 0 (ok)`, check `/var/log/sss/sssd_goaauthentik.io.log` (with debug level 6 if needed):

```
sudo sssctl debug-level 6
sudo systemctl restart sssd
sleep 3
sudo grep -i -E "bind|credential|tls|error|fail|offline" /var/log/sss/sss_goauthentik.io.log
| tail -n 30
sudo sssctl debug-level 1
```

If `openldap` (`ldapsearch`) is installed, test the `ldapservice` bind directly:

```
LDAPTLS_REQCERT=demand ldapsearch -LLL -x \
-H ldaps://ldap.drheck.dev:636 \
-D "cn=ldapservice,ou=users,dc=ldap,dc=goauthentik,dc=io" \
-w "hgNbhl3t1hzyV4jdfsiQuAeYS" \
-b "ou=users,dc=ldap,dc=goauthentik,dc=io" "(cn=ikura)" cn uidNumber gidNumber
```

- If `ldapsearch` returns `ldap_bind: Invalid credentials (49)` (or `sudo -u docker-data XDG_RUNTIME_DIR=/run/user/1100 journalctl --user -u authentik-ldap.service` on `ul` shows `"error": "flow error password: Invalid password"` for `cn=ldapservice,ou=users,dc=ldap,dc=goauthentik,dc=io`): The `ldapservice` password in Authentik on `ul` does not match `ldap_default_authtok` in `/etc/sss/sss.conf` (or another service overwrote `ldapservice`'s password instead of using its own dedicated service account). To verify and restore `ldapservice` on `ul`:

```
ssh ul '(cd /tmp && sudo -u docker-data HOME=/home/docker-data
XDG_RUNTIME_DIR=/run/user/1100 podman exec -i authentik-server ak shell -c "
from authentik.core.models import User
from authentik.providers.ldap.models import LDAPProvider
p = LDAPProvider.objects.first()
u = User.objects.get(username="ldapservice")
print("Password match:", u.check_password("hgNbhl3t1hzyV4jdfsiQuAeYS"))
print("LDAPProvider perms:", u.get_all_permissions(p))
" 2>/dev/null)'
```

If `Password match:` is `False`, restore it on `ul` and restart `authentik-ldap.service` so its cached bind session is flushed:

```
ssh ul '(cd /tmp && sudo -u docker-data HOME=/home/docker-data
XDG_RUNTIME_DIR=/run/user/1100 podman exec -i authentik-server ak shell -c "
from authentik.core.models import User
u = User.objects.get(username="ldapservice")
u.set_password("hgNbhl3t1hzyV4jdfsiQuAeYS")
u.save()
```

```
" 2>/dev/null) && sudo systemctl --user -M docker-data@ restart authentik-  
ldap.service'
```

Issue C: Flushing Stale SSSD Cache & Forcing SSSD Online

If SSSD marked the domain offline during a temporary network glitch, force it back online and clear the cache:

```
sudo systemctl stop sssd  
sudo rm -f /var/lib/sss/db/cache_goauthentik.io.ldb  
sudo systemctl start sssd  
sudo sssctl domain-status goauthentik.io  
getent passwd -s sss
```

Issue D: User Resolves in `getent passwd` But Cannot Log In on GDM / Laptop

1. Check if the user appears on the GDM login screen: Verify

`/var/lib/AccountsService/users/<username>` exists and has `SystemAccount=false`:

```
sudo cat /var/lib/AccountsService/users/ikura
```

If missing, run the pre-provisioning snippet in **Step 5.2** above and restart `accounts-daemon`:

```
sudo systemctl restart accounts-daemon
```

2. Test PAM authentication from a terminal (`su` / `login`):

```
su - ikura
```

If `su - ikura` fails while `sudo su - ikura` works, inspect `/var/log/sss/sss_pam.log` and `journalctl -n 40 --no-pager`:

- Ensure `/etc/pam.d/system-auth` contains `pam_sss.so` in `auth`, `account`, `password`, and `session`.
- Ensure the user is active in Authentik and their password was set in Authentik (`pwdChangedTime` present on their LDAP entry).

SSH CA setup

OVERVIEW & ARCHITECTURE

- **Authentication Flow:** Incoming SSH clients authenticate using an OpenSSH user certificate signed by the server's User Certificate Authority (CA). Similarly, clients verify host identity using an OpenSSH host certificate signed by the Host CA.
- **LDAP & Authentik Integration:** Host **ul** derives user accounts from Authentik via LDAP (managed via SSSD). When a client presents an SSH certificate, `sshd` validates the certificate signature against the trusted CA public key and checks that one of the certificate's valid principals matches the requested LDAP login username. No per-user `~/.ssh/authorized_keys` file is required.
- **Key Storage & Backup:**
 - The User CA private key (`drheck_ca_user_key`) and Host CA private key (`drheck_ca_host_key`) must **never** reside on unencrypted client machines or internet-facing servers.
 - Store the offline CA private keys and passphrases securely in Vaultwarden under account `akadmin` (`admin@drheck.dev`).
 - Deploy only the CA public keys (`drheck_ca_user_key.pub` and `drheck_ca_host_key.pub`) to host **ul** and client machines.
- **Operational Separation:**
 - **Architecture & Foundational CA Setup:** Documented on this page.
 - **Day-to-Day Client & Host Onboarding:** See the dedicated operational runbook: [SSH Client and Host Onboarding Guide](#).

PHASE 1: CA GENERATION & VAULTWARDEN BACKUP

Generate the standard Ed25519 CA Keypairs on a secure offline or administrative workstation:

When prompted by `ssh-keygen`, enter a strong, unique passphrase to encrypt each CA private key.

```
# Generate the User CA keypair
ssh-keygen -t ed25519 -f drheck_ca_user_key -C "drheck-user-ca"

# Generate the Host CA keypair
```

```
ssh-keygen -t ed25519 -f drheck_ca_host_key -C "drheck-host-ca"
```

Storage in Vaultwarden:

1. Create a secure item in Vaultwarden under `akadmin` (`admin@drheck.dev`):
 - `drheck_ca_user_key` (multi-line private key in Notes field)
 - `drheck_ca_user_key_pw` (passphrase)
 - `drheck_ca_host_key` (multi-line private key in Notes field)
 - `drheck_ca_host_key_pw` (passphrase)
2. Remove the private CA keys from the unencrypted local disk of your workstation after verifying the Vaultwarden backup.

PHASE 2: MAIN SERVER CONFIGURATION (ul)

1. Deploy Trusted User CA Public Key

- Copy the public CA file `drheck_ca_user_key.pub` to host **ul** at `/etc/ssh/drheck_ca_user_key.pub`.
- Set secure permissions:

```
sudo chmod 644 /etc/ssh/drheck_ca_user_key.pub
sudo chown root:root /etc/ssh/drheck_ca_user_key.pub
```

2. Configure Host Certificate (Signed by Host CA)

1. Copy **ul**'s public host key `/etc/ssh/ssh_host_ed25519_key.pub` to your signing environment.
2. Sign the host key using the vaulted `drheck_ca_host_key` (defining all valid FQDNs and IP addresses as principals):

```
ssh-keygen -s drheck_ca_host_key -I "drheck-ul-host-cert" -h -n
ul,ul.drheck.dev,10.0.0.3,fd77:7777:5010:0:1337:f1ac:d00d:337 -V +52w
/etc/ssh/ssh_host_ed25519_key.pub
```

3. Place the resulting certificate at `/etc/ssh/ssh_host_ed25519_key-cert.pub`.
4. Set secure permissions:

```
sudo chmod 644 /etc/ssh/ssh_host_ed25519_key-cert.pub
sudo chown root:root /etc/ssh/ssh_host_ed25519_key-cert.pub
```

3. Configure OpenSSH Daemon on ul

- Create or edit `/etc/ssh/sshd_config.d/10-drheck-certs.conf`:

```
TrustedUserCAKeys /etc/ssh/drheck_ca_user_key.pub
HostCertificate /etc/ssh/ssh_host_ed25519_key-cert.pub
```

4. Validate and Reload SSHD on ul

- Validate the configuration syntax before reloading:

```
sudo sshd -t
```

- Reload the daemon:

```
sudo systemctl reload sshd
```

PHASE 3: AUDITING & KEY REVOCATION (KRL)

OpenSSH natively supports Key Revocation Lists (KRL) to block compromised keys or expired certificates before their expiration timestamp.

1. Key Inventory Architecture

Every key signed by the signing utilities is automatically archived under:

- `/etc/ssh/ca_inventory/hosts/<label>.pub` and `<label>-cert.pub`
- `/etc/ssh/ca_inventory/users/<label>.pub` and `<label>-cert.pub`
- `/etc/ssh/ca_inventory/manifest.tsv`

2. Listing Active & Revoked Keys

Run the inventory helper to view all issued certificates and their revocation state:

```
revoke-key --list
```

Example Output:

LABEL	TYPE	FINGERPRINT	STATUS
danknasty	host	SHA256:oj796Z0eYmi6hAm+pG	[ACTIVE]
u1	host	SHA256:1a4QDcX1p/QT4qbFdc	[ACTIVE]
borrowbook	user	SHA256:HPu6XQU1ARHnLWgMWo	[ACTIVE]
metropolis	user	SHA256:220ChYHL9QxAe57Noe	[ACTIVE]

3. Revoking a Key

To immediately revoke any key or certificate:

```
sudo revoke-key <label>
```

Actions performed automatically:

1. Locates the archived public key in `/etc/ssh/ca_inventory/`.
2. Appends the key hash into the OpenSSH binary KRL file at `/etc/ssh/revoked_keys`.
3. Ensures `RevokedKeys /etc/ssh/revoked_keys` is enabled in `/etc/ssh/sshd_config.d/10-drheck-certs.conf`.
4. Tests configuration syntax and reloads `sshd`.

PHASE 4: CLIENT-SIDE GLOBAL TRUST & ENFORCEMENT

1. Trusting the Host CA Across Clients

To trust all hosts bearing a DrHeck Host CA certificate without TOFU prompts, deploy this line to `/etc/ssh/ssh_known_hosts` (or `~/.ssh/known_hosts`):

```
@cert-authority * ssh-ed25519 <drheck_ca_host_key.pub_content>
```

2. Enforcing Certificates (Disabling Raw Keys)

Once all clients have valid user certificates, you can disable uncertified public keys while preserving an `authorized_keys` backdoor for LAN-only bootstrapping:

```
# /etc/ssh/sshd_config.d/10-drheck-certs.conf
AuthorizedKeysFile /dev/null

# Allow authorized_keys ONLY on the internal trusted network
Match Address 10.0.0.0/24,fd77:7777:5010::/64
    AuthorizedKeysFile .ssh/authorized_keys
```

OPERATIONAL RUNBOOK

For the step-by-step instructions on adding new machines and signing certificates, see:

- [SSH Client and Host Onboarding Guide](#)

Auto-Generate Signed SSH Keys

1. Authentik Stores the User CA Private Key:

Upload `drheck_ca_user_key` (unencrypted or with a known passphrase) into Authentik under System -> Certificates.

2. Create an Authentik Python Policy/Execution Flow:

You create an API endpoint in Authentik (or a custom Policy/Flow) that accepts a POST request containing a public key and a username.

- * Authentik validates the API request.

- * Using its internal Python Expression Policies, Authentik signs the public key using the loaded `drheck_ca_user_key` and returns the signed certificate text.

3. Deploy `pam_exec.so` on the Desktop:

On your desktop machine, you add a line to `/etc/pam.d/system-auth` to execute a script upon successful login:

```
1 session optional pam_exec.so /usr/local/bin/mint-ssh-keys.sh
```

4. The `mint-ssh-keys.sh` Script:

Whenever any user logs in, this script runs under their local user context:

- * Checks if they already have keys: If `~/.ssh/id_ed25519-cert.pub` already exists, it exits (doing nothing).

- * Generates local keys: If they are logging in for the first time, it runs `ssh-keygen -t ed25519 -N "" -f ~/.ssh/id_ed25519` to generate a passwordless keypair.

- * Requests a Signature from Authentik: It uses `curl` to send their new `id_ed25519.pub` to your Authentik API endpoint.

- * Saves the Certificate: It saves the returned signed certificate to `~/.ssh/id_ed25519-cert.pub`.

SSH Client and Host Onboarding Guide

Quickest

ul

```
sudo -iu akadmin vw unlock
```

n00b

```
cat /etc/ssh/ssh_host_ed25519_key
```

ul

```
vim ~/.ssh/authorized_keys  
# type-y type-y
```

n00b

```
ssh -i /etc/ssh/ssh_host_ed25519_key ul  
# test ^  
cat /etc/ssh/ssh_host_ed25519_key.pub | ssh ul "sudo sign-host-key n00b -l n00b0x" | sudo tee  
/etc/ssh/ssh_host_ed25519_key-cert.pub  
sudo chmod 644 /etc/ssh/ssh_host_ed25519_key-cert.pub  
printf 'HostCertificate /etc/ssh/ssh_host_ed25519_key-cert.pub\n' | sudo tee  
/etc/ssh/sshd_config.d/10-cert.conf  
sudo sshd -T -f /etc/ssh/sshd_config | grep -q HostCertificate || echo fail  
sudo systemctl reload sshd
```

UNLOCK VAULTWARDEN

The CA signing keys reside in Vaultwarden and are kept in volatile memory for 1 hour once unlocked. Before onboarding, log into **ul** and unlock the vault:

```
sudo -iu akadmin vw unlock
```

WORKFLOW 1: ONBOARDING A NEW HOST (Server Host Certificate)

Host certificates allow clients to verify the authenticity of a server automatically without manual fingerprint checks ("The authenticity of host ... can't be established").

1. Generate Host Keypair on New Machine

On the machine being onboarded "n00b", generate its standard Ed25519 host key (if it does not already exist):

```
sudo ssh-keygen -t ed25519 -f /etc/ssh/ssh_host_ed25519_key -N ""
```

2. Out-of-Band Bootstrap & Verification

Verify it using `authorized_keys` first:

1. Copy the n00b's public key text (`/etc/ssh/ssh_host_ed25519_key.pub`) out-of-band
2. Add it to `~/.ssh/authorized_keys` on ul.
3. Test SSH connectivity from the new host to **ul**:

```
ssh -i /etc/ssh/ssh_host_ed25519_key ul
```

Verified!

3. Mint Signed Host Certificate

From the new host, stream the public key over that verified SSH connection directly into `sign-host-key`:

```
cat /etc/ssh/ssh_host_ed25519_key.pub | ssh ul "sudo sign-host-key bmo95 -l studio-pc" | sudo tee /etc/ssh/ssh_host_ed25519_key-cert.pub
```

- **Custom Principals:** To specify custom DNS names or IP addresses:

```
cat /etc/ssh/ssh_host_ed25519_key.pub | ssh -t ul "sudo sign-host-key danknasty danknasty,danknasty.drheck.dev,10.0.0.1,fd77:7777:5010::1" | sudo tee /etc/ssh/ssh_host_ed25519_key-cert.pub
```

- Set file permissions on the new host:

```
sudo chmod 644 /etc/ssh/ssh_host_ed25519_key-cert.pub
```

4. Configure sshd on the New Host

Add the host certificate to the SSH daemon configuration on the new host (`/etc/ssh/sshd_config` or `/etc/ssh/sshd_config.d/10-cert.conf`):

```
HostCertificate /etc/ssh/ssh_host_ed25519_key-cert.pub
```

Reload the SSH daemon:

```
sudo systemctl reload sshd
```

5. De-provision Bootstrap Key

Remove the temporary bootstrap entry from `~/.ssh/authorized_keys` on ul.

WORKFLOW 2: ONBOARDING A NEW USER (Client User Certificate)

User certificates allow clients to log into servers without putting per-user public keys in `~/.ssh/authorized_keys`.

1. Generate User Keypair on Client Workstation

On your laptop or workstation:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -C "workstation-label"
```

2. Out-of-Band Bootstrap & Verification

1. Copy `~/.ssh/id_ed25519.pub` out-of-band and temporarily append it to `~/.ssh/authorized_keys` on **ul**.
2. Confirm login works:

```
ssh ul
```

3. Mint Signed User Certificate

From your client workstation, pipe your public key into `sign-user-key`:

```
cat ~/.ssh/id_ed25519.pub | ssh -t ul "sudo sign-user-key <ldap_username>" >
~/.ssh/id_ed25519-cert.pub
chmod 644 ~/.ssh/id_ed25519-cert.pub
```

Note: The principal `<ldap_username>` must exactly match the Authentik/LDAP username on **ul**.

4. De-provision Bootstrap Key

Remove the raw public key from `~/.ssh/authorized_keys` on **ul**.

5. Verify Certificate Authentication

Test login with verbose output:

```
ssh -v ul
```

Confirm the server accepts the certificate:

```
Server accepts key: pka1g ssh-ed25519-cert-v01@openssh.com
```

CLIENT CONFIGURATION: TRUSTING THE HOST CA

To ensure client workstations trust all servers signed by the DrHeck Host CA without fingerprint prompts, add this line to `/etc/ssh/ssh_known_hosts` (system-wide) or `~/.ssh/known_hosts`:

```
@cert-authority * ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIC0m1wzW2e50529d+E3iA7x8iM3Q8qS5q7iVjGvF8e7d
```

(Replace with the actual public key from `/etc/ssh/drheck_ca_host_key.pub`).

QUICK REVOCATION REFERENCE

Every signed key is automatically archived in `/etc/ssh/ca_inventory/`.

- **List all signed keys:**

```
revoke-key --list
```

- **Revoke a key immediately:**

```
sudo revoke-key <label>
```

WORKFLOW 3: ANDROID / TERMUX ONE-TOUCH QR ONBOARDING

For Android devices using Termux, an entire SSH CA environment (device Ed25519 keypair, CA-signed user certificate, DrHeck Host CA trust, and SSH configuration) can be minted and imported using a single QR code.

1. Unlock Vaultwarden on `ul`

```
sudo -iu akadmin vw unlock
```

2. Mint & Render QR Code on `ul`

Run `/home/luke/bin/ssh-addclient`:

```
# Syntax: ssh-addclient <device_name> [ldap_username]
ssh-addclient pixel9 luke
```

This utility:

- Generates a private Ed25519 keypair in RAM (`/dev/shm`).
- Signs the user certificate via `sign-user-key` with the DrHeck User CA.
- Embeds the Host CA `@cert-authority` trust and pre-configured hosts (`ul`, `danknasty`).
- Compresses and encodes the payload into a self-extracting Termux bootstrap one-liner.
- Renders an ASCII QR code directly in the terminal.

3. Ingest in Termux (Android)

In Termux on the phone: In Termux with WireGuard connected:

```
pkg install openssh
curl -s http://10.0.0.3:7700 | sh
```

(The one-shot listener runs strictly on the internal WireGuard interface `10.0.0.3:7700` and shuts down immediately upon retrieval).

4. Connect

```
ssh ul  
# or  
sftp ul
```

Authentication succeeds immediately using the CA user certificate with zero host fingerprint prompts.