Authentication

- Credential Model

Credential Model

Credential Model

This page describes how Hermes authenticates users across all of its surfaces — the web admin console, the user portal, Nextcloud (mail / calendar / contacts), and direct mail-protocol clients (IMAP, SMTP, CalDAV, CardDAV). The model is **uniform**: it works the same way whether a user is authenticated locally (against Hermes's built-in LDAP) or remotely (against an external Active Directory or LDAP server).

Understanding this model is a prerequisite for everything else in the Authentication chapter — app password management, MFA, OIDC SSO, and the iOS device setup wizard all build on it.

The credentials a user has

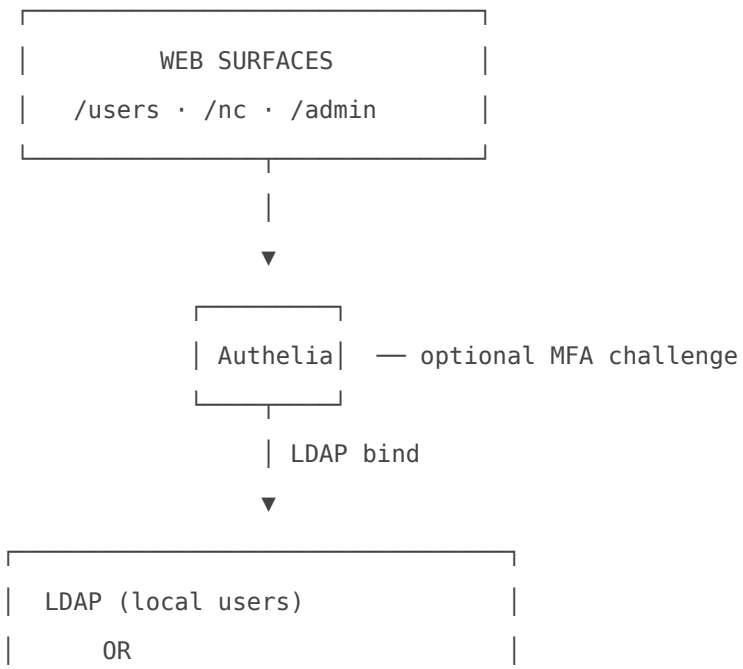
Every Hermes user — local or remote — has up to **four distinct credentials**, each with a single, well-defined purpose. None of them is a "master password."

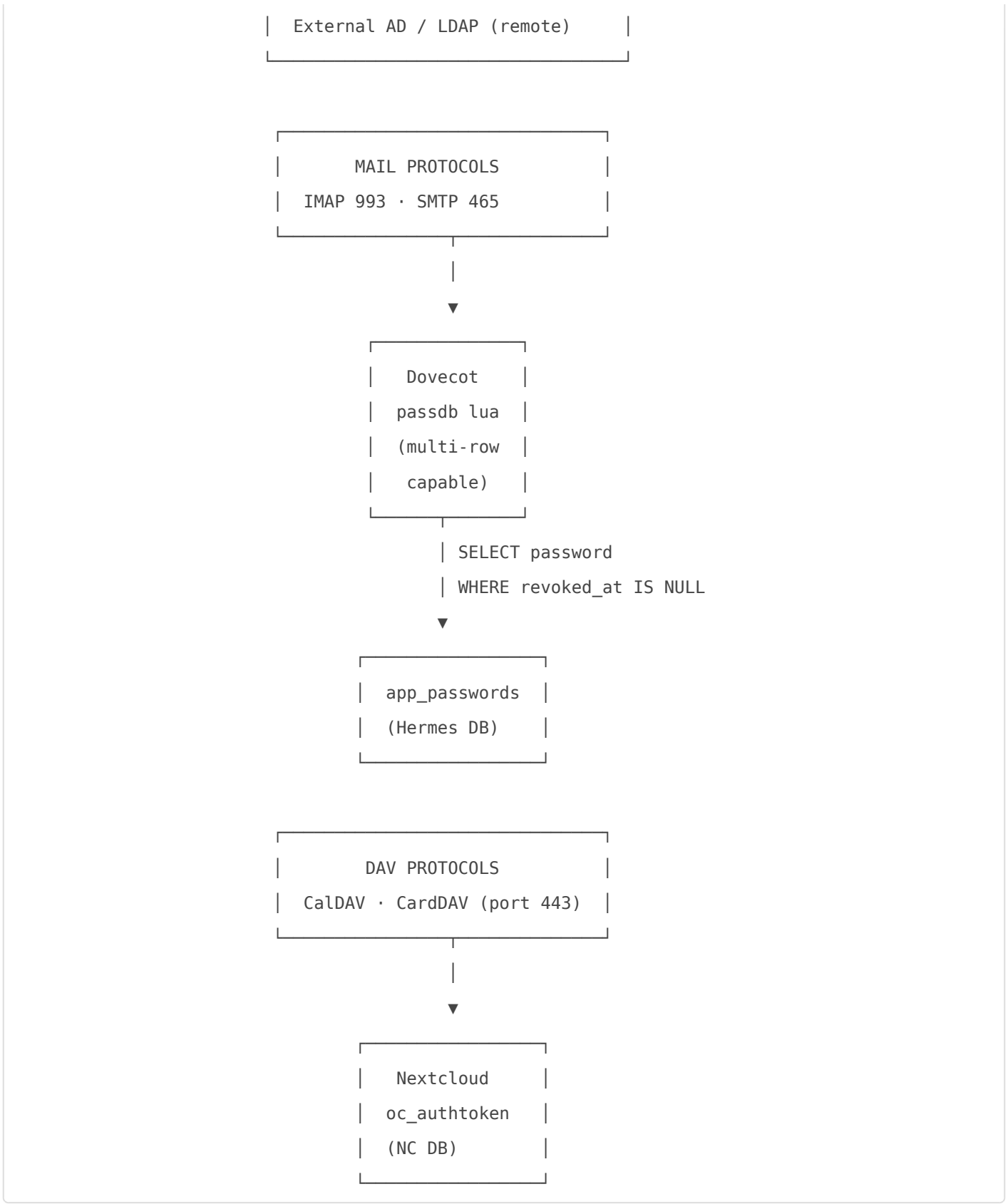
Credential	Where it's stored	What it logs you into	How a user obtains it
Web login password	LDAP (local users) or external AD/LDAP (remote users)	`/users` portal · `/nc` (Nextcloud web UI) · `/admin` (admin console, admins only)	**Local auth:** set by admin at mailbox creation · changed by user in the portal · forgotten-password reset flow available. **Remote auth:** set and managed entirely in your external AD/LDAP — Hermes never sees or stores it.
NC internal password	Nextcloud's `oc_users` table	Nothing the user ever needs. It exists purely as a defense-in-depth backstop.	The user **never sees it**. It is set to a random value at mailbox creation and is never disclosed to anyone.

Credential	Where it's stored	What it logs you into	How a user obtains it
"Hermes System" app password	`app_passwords` table with `is_system = 1`	Used internally by the Nextcloud Mail app to authenticate IMAP/SMTP against Dovecot. Nothing the user ever needs to type.	Generated automatically at mailbox creation. Hidden from the user portal so users can't accidentally revoke it and break webmail. Admin can revoke + regenerate it from the per-mailbox admin page if needed.
User app passwords	`app_passwords` table with `is_system = 0` (read by Dovecot) AND `oc_authtoken` table (read by Nextcloud DAV) — same plaintext, both stores	IMAP, SMTP, CalDAV, CardDAV — i.e. mail/calendar/contact apps on devices	User generates them in the portal under **My App Passwords**. Each one is shown **once** and labelled per-device ("iPhone", "Thunderbird"). On create, the credential is registered with both Hermes and Nextcloud atomically; on revoke, both sides are removed.

The two non-obvious parts here are the **NC internal password** (a back-channel-closing trick — see § Why a random NC internal password) and the **"Hermes System" app password** (admin-managed plumbing that lets webmail work without the user ever seeing a credential — see § The "Hermes System" app password).

High-level flow diagram





Three surfaces, three back-ends. **No credential is shared across surfaces.** The web login password never reaches Dovecot. App passwords never reach Authelia. The NC internal password is never accepted by anything a user holds.

Local-auth users vs. remote-auth users

The model applies identically to both. The only thing that changes is **what backs the LDAP bind** for the web login password.

Local-auth user

Web login password lives in:	Hermes's built-in OpenLDAP (cn=<user>,ou=users,dc=hermes,dc=local)
Set / changed by:	Admin (at create time) → user (in portal)
Reset path:	Forgot password → email link → reset
App passwords:	Generated and revoked by the user in /users (no external dependency)
NC internal password:	Random at create. Admin can rotate via the "Rotate NC Internal Password" action in the mailbox detail view.

Remote-auth user

Web login password lives in:	The customer's external AD or LDAP server. Hermes's Authelia binds against it. Hermes never stores or hashes it.
Set / changed by:	The customer's IT team in their own directory. Hermes has no control surface for it.
Reset path:	Customer's existing AD/LDAP reset workflow. Hermes does not handle it.
App passwords:	Generated and revoked by the user in /users —

```
                          same UI, same table, same lifecycle as local.
                          These ARE stored in Hermes; they have to be,
                          because IMAP/SMTP/DAV cannot speak the
                          protocols a corporate AD typically uses.

NC internal password:     Random at create. Same admin rotation path.
```

> **Key takeaway:** the operational surface a user touches — web login, app password mgmt, mail/calendar/contacts setup — looks **identical** between local and remote. The only difference is which directory their **login password** lives in.

Why three credentials? Why not just one?

The single-credential approach (using the login password everywhere) has three problems:

1. **You can't revoke a single device.** A user loses their phone — to lock that phone out, you have to change the password on every device they own and re-enter it everywhere. With per-device app passwords, you revoke just the one row.
2. **You can't enforce MFA on devices.** IMAP, SMTP, CalDAV, and CardDAV cannot prompt for a TOTP code or a Duo Push. They authenticate with one round trip and one secret. So if MFA matters at all, it can only live at the web gate (Authelia). A separate device credential lets you keep MFA on the web while devices use a non-MFA bearer token.
3. **You can't safely embed a login password on a device.** The login password is the user's keys to the kingdom — email, AD, often other corporate apps. Every device that holds it is a leak risk. App passwords are scoped (mail/DAV only), revocable, and have no other privilege.

The three-credential model makes each problem disappear:

- **Revoke a device:** revoke its app password row. Done.
- **MFA at the gate:** Authelia challenges on web login. Devices use bearer-token-style app passwords and are never prompted.
- **No login password on devices:** devices receive a 30-character random string with mail/DAV scope only. If it leaks, it leaks one mail account, not the whole identity.

The "Hermes System" app password

When a mailbox is created, Hermes mints **one** app password automatically and labels it `Hermes System`. It is stored in `app_passwords` with `is_system = 1`. It is used in exactly one place: as the IMAP credential that the **Nextcloud Mail webmail app** uses to read mail from Dovecot on the user's behalf.

Why it exists

Without it, the user couldn't read mail through the `/nc` Mail app on day 1, because:

1. The user has no app password yet (they generate their own from the user portal).
2. Their login password no longer works for IMAP (it never reaches Dovecot under the new model).
3. NC Mail needs *some* credential to authenticate IMAP on the server side — there's no SSO from NC Mail down into Dovecot in this stack.

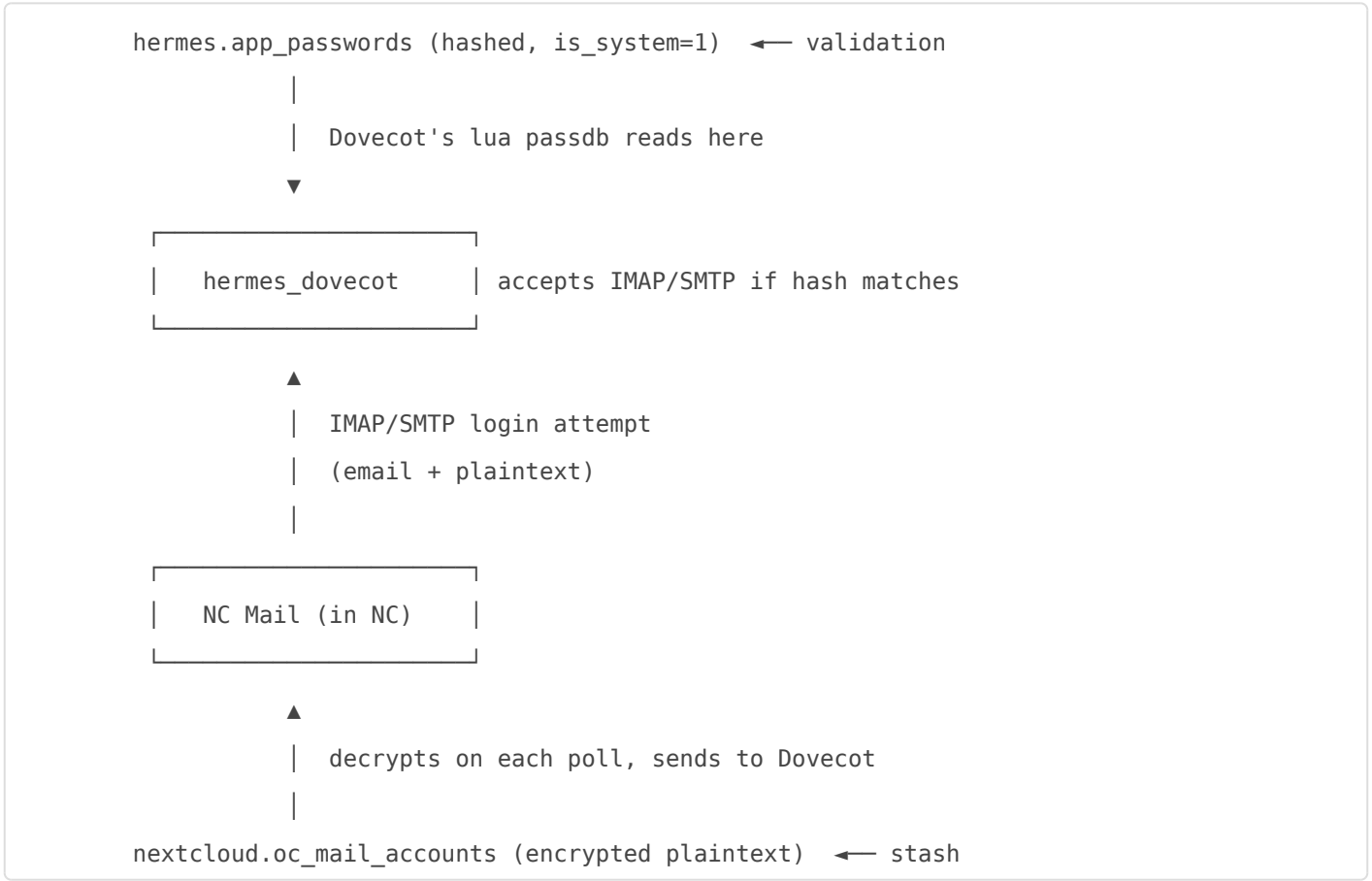
The Hermes System app password is that credential. The user never sees it, never types it, and never knows it exists.

Where it lives — two stores, two roles

The Hermes System credential is held in two databases at the same time, each playing a different role:

Location	Form	Used by	Role
`hermes.app_passwords` (`is_system = 1`)	ARGON2ID **hash**	Dovecot's Lua passdb	**Validation store.** Dovecot password-verifies incoming IMAP/SMTP attempts against this hash. This is "the credential" — its source of truth.

Location	Form	Used by	Role
`nextcloud.oc_mail_accounts`	NC-encrypted **plaintext**	NC Mail (the webmail app inside Nextcloud)	**Operational copy.** NC Mail decrypts this on each poll and presents `email + plaintext` to Dovecot. It does *not* authenticate against `oc_authtoken` — `oc_mail_accounts` is a separate NC Mail table for stored mail-server credentials.



Two important consequences:

- **Deleting the `app_passwords` row instantly disables NC Mail.** NC Mail's stored plaintext becomes garbage that never validates. Dovecot rejects every poll attempt. Webmail starts erroring.
- **Hermes System is *not* an `oc_authtoken` row.** That's a different NC table used for NC's own session/DAV authentication. NC Mail authenticates *to Dovecot* (an external IMAP server from NC's perspective), not *to NC*, so it has no business in `oc_authtoken`. Only user-generated app passwords (Phase 1b dual-write) live in `oc_authtoken`.

Why it's hidden from the user portal

If the user could see this row in **My App Passwords**, they might revoke it — at which point NC Mail breaks, webmail starts erroring, and the user has no obvious way to know what happened. The `is_system = 1` flag filters it out of the user's list. The admin sees it on the per-mailbox app-password page, marked with a "System" badge.

Lifecycle

- **Created:** at mailbox create, by `add_mailbox_action.cfm` (step 4h).
- **Used:** every time NC Mail polls IMAP for that user.
- **Rotated:** if the admin uses the "Regenerate Hermes System" action (planned), or if the admin manually revokes + creates a new one and updates the NC Mail account config. Phase 1c polish.
- **Deleted:** when the mailbox is deleted (`delete_mailbox_action.cfm` removes all `app_passwords` rows for that user).

How the Nextcloud `oc_authtoken` mirror works (Phase 1b)

User-generated app passwords are stored in **both** Hermes and Nextcloud at create time, with the same plaintext:

```
User clicks "Create App Password" labeled "iPhone"
        │
        ├─ occ user:auth-tokens:add → NC mints a fresh plaintext token
        │  (NC chooses the value; we don't pick it)
        │
        ├─ Token row gets renamed in oc_authtoken to "iPhone"
        │
        ├─ The new oc_authtoken.id is captured and stored in
        │  app_passwords.nc_token_id
        │
        ├─ The plaintext from NC is hashed via doveadm pw -s ARGON2ID
        │  and stored in app_passwords.password
        │
        └─ The plaintext is shown to the user once (one-shot callout)
                │
                ▼
        Same plaintext now authenticates:
          • IMAP/SMTP via Dovecot (lua passdb, app_passwords)
```

```
          • CalDAV/CardDAV via Nextcloud (oc_authtoken)
```

On revoke, the flow runs in the opposite direction:

```
User clicks Revoke
        │
        ├─ Look up the row's nc_token_id
        │
        ├─ occ user:auth-tokens:delete → removes the NC oc_authtoken row
        │  (DAV stops authenticating immediately)
        │
        └─ UPDATE app_passwords SET revoked_at = NOW()
           (Dovecot stops authenticating on next IMAP/SMTP attempt)
```

The "Hermes System" admin-managed app password (`is_system = 1`) is **not** mirrored to `oc_authtoken`. It exists purely as NC Mail's IMAP credential to Dovecot. Keeping it out of NC's auth store means it cannot be used for DAV access — defensive separation between admin plumbing and user-facing credentials.

Why a random NC internal password

This is the part that is non-obvious. Walk through it carefully.

When a Nextcloud user logs in via OIDC SSO, NC internally provisions a row in `oc_users` for them. That row has a `password` column. NC needs *something* there because some NC subsystems (notably the DAV endpoints, before app-password enforcement) will accept a password against `oc_users.password` as a valid auth.

The natural temptation is to set this `oc_users.password` to either (a) the user's login password or (b) some predictable derivative of it. **Both are wrong**, for the same reason: it creates a silent back-channel.

Picture the failure mode:

```
1. User authenticates to /nc via OIDC. Web is fine.
2. User configures their phone's CalDAV with the login password.
3. NC's DAV endpoint, finding a matching oc_users.password, accepts it.
4. From that moment on, the login password is now embedded on a device.
```

```
5. User loses the phone. Org password leaks. Worse, neither admin nor user
   realises DAV ever silently "worked" — they assumed only OIDC was in play.
```

Setting `oc_users.password` to a **random value that no one knows** removes the back-channel:

```
1. User authenticates to /nc via OIDC. Web is still fine.
2. User configures their phone's CalDAV with the login password.
3. NC's DAV endpoint compares the supplied password against
   oc_users.password. No match (the stored value is random).
   Auth fails.
4. User is forced to either (a) generate a NC app password through the
   normal app-password flow, or (b) realise they need to use a different
   credential. Either way, the login password is NOT on the device.
```

The random value is generated at mailbox creation and is **never disclosed**. There is no UI to view it. The admin can **rotate** it (regenerate to a fresh random value) via the **Rotate NC Internal Password** action on the mailbox detail page. Rotation is a defense-in-depth move — it costs the operator one click and protects against the unlikely event of a NC password-hash store leak.

Where each credential is checked

User action	Surface	Credential checked	Backend
Open `/users` in a browser	Web	Web login password (+ MFA)	Authelia → LDAP
Open `/nc` in a browser	Web	Web login password (+ MFA)	Authelia → LDAP
Open `/admin` in a browser (admins)	Web	Web login password (+ MFA)	Authelia → LDAP
Mail.app fetches from IMAP 993	Mail	User app password	Dovecot `passdb lua` → `app_passwords` (any non-revoked row)
Mail.app sends via SMTP 465	Mail	User app password	Dovecot SASL → `passdb lua` → `app_passwords`
Nextcloud Mail webmail fetches IMAP	Mail (server-side)	"Hermes System" app password	Dovecot `passdb lua` → `app_passwords` (`is_system = 1` row, set up at provisioning)
Calendar.app sync via CalDAV 443	DAV	User app password	Nextcloud → `oc_authtoken` (mirror of `app_passwords` row)

User action	Surface	Credential checked	Backend
Contacts.app sync via CardDAV 443	DAV	User app password	Nextcloud → `oc_authtoken` (mirror of `app_passwords` row)
Anything tries `oc_users.password` directly	(varies)	NC internal password	Nextcloud → `oc_users` (random — won't match anything a user holds)

Multi-active app passwords

A user can have **many** active app passwords at once. Each device gets its own row, with its own hash and its own label.

How the iteration works

Stock Dovecot's `passdb sql` driver looks at the **first returned row only** — it does not try multiple hashes against the supplied password. To support per-device app passwords, Hermes uses a Lua-backed passdb script (`/etc/dovecot/auth_app_passwords.lua`) instead. The script:

1. Connects to MariaDB on each authentication attempt.
2. Selects all `app_passwords` rows for the user where `revoked_at IS NULL`.
3. Iterates the rows, calling Dovecot's `password_verify()` against each hash.
4. Returns success on the first match; failure if none match.

A successful match also updates `last_used_at` on the matching row (rate-limited to once per hour per row, to avoid hammering the DB on chatty IMAP IDLE clients).

Why this matters

Multi-row support is what makes device swaps zero-downtime:

```
1. User creates "iPhone (new)" app password, enters into new phone.
2. New phone works (matches its own row).
3. Old phone keeps working in parallel (matches "iPhone (old)" row).
4. User revokes "iPhone (old)" — old phone immediately stops working.
5. No window during which either device is locked out.
```

A revocation is **instant**: setting `revoked_at` excludes the row from the next Lua lookup. There is no cache to wait on.

What this costs

- One extra MySQL connection per authentication attempt (per-call open/close — safe under Dovecot's `use_worker = yes` mode).
- One extra `UPDATE` per matched authentication, throttled to once per hour per row.
- A new Docker image dependency (`dovecot-lua` + `lua-sql-mysql`) on the `hermes_dovecot` container.

These are small. The win — true per-device revocation — is large.

What this model deliberately does NOT do

- **It does not support typing the login password on a phone for IMAP/SMTP.** That was never the goal. It is by design.
- **It does not let users set their own NC internal password.** That column is internal plumbing, not a user credential. There is no UI for it on either side.
- **It does not store anything that lets us recover a lost app password.** App passwords are shown once at creation. If a user loses one, they revoke it and create a new one. There is no "show me my app password again" path.

What happens when an admin creates a new mailbox

The flow runs through `admin/2/inc/add_mailbox_action.cfm`. Most steps are identical for local-auth and remote-auth mailboxes; the one place they branch is step 2.

1. A row is inserted into `mailboxes` (Dovecot userdb — quota, active flag, etc.).
2. An LDAP entry is created in Hermes's directory:
   - **Local auth**: with the login password the admin entered in the form. This is the user's only login password going forward — Hermes is the source of truth for it.
   - **Remote auth**: as a stub entry with `seeAlso` pointing at the user's account in your external AD/LDAP. The user's login password lives in that external directory; Hermes never stores or hashes it.
3. The Nextcloud user account is provisioned with a **random** local password (`add_mailbox_action.cfm` step 4c). This password is never disclosed and never used by

anyone — it exists only to close the back-channel risk where the user's login password could otherwise be silently accepted by NC's DAV endpoint. Same behavior for both auth types.

4. The "Hermes System" app password is minted (`is_system = 1`, label `Hermes System`) — step 4h. Same for both auth types.
5. The NC Mail account is provisioned with the Hermes System app password — step 4f. This is what makes webmail work on day 1 without the user having ever set up an app password themselves. Same for both auth types.
6. The welcome email is sent. **No credential is included in the email.** The email tells the user to sign in to the user portal with their login password (received out-of-band from the admin) and generate per-device app passwords from My App Passwords. The local-auth and remote-auth variants of the welcome email differ only in tone — local-auth says "your login password" while remote-auth specifies "your organization (AD/LDAP) password."

Related documentation

- My App Passwords (user) — what end users see and do
- Email Server > Mailboxes — the per-mailbox detail page is where an admin revokes a user's app passwords or rotates the per-mailbox Nextcloud internal password