Installation & Reference

Hermes SEG Docker installation, upgrade, and technical reference material. Auto-synced from docs/install/ + docs/general/.

Get Started (Docker)

Get Started (Docker)

This page is the **minimum config needed to get mail flowing** on a fresh Hermes SEG Docker install. The install script (`scripts/install_hermes_docker.sh`) does most of the heavy lifting — this page covers the handful of admin-UI steps that still need a human.

Skip these and Postfix will silently bounce or reject mail. The admin dashboard also surfaces two universal nudges (placeholder hostname, self-signed cert) until those are addressed (see Dashboard nudges at the bottom).

Hermes supports three deployment topologies and you don't need all of them:

Topology	Required steps	Skip
Relay-only (Hermes forwards inbound mail to a downstream MX)	First Domain → Relay Networks → First Recipient	Mailboxes
Mail-server-only (Hermes hosts mailboxes)	Mailbox Domain (under Email Server) → Mailboxes	Relay Domains, Relay Networks, Relay Recipients
Hybrid	All of the above	(nothing)

> **Legacy reference**: this page replaces the pre-Docker 16-step page. The Docker install script absorbs ~6 of those steps, so the list below is shorter.

What the install script already did

You don't need to redo any of this — `install_hermes_docker.sh` handled it during the install run:

Component	Result
Containers	All Hermes containers running (`docker compose ps`)
Bootstrap admin	LDAP user in `cn=admins` + `cn=one_factor`, password in `INSTALL_SUMMARY`

Component	Result
TLS	Self-signed bootstrap cert in System Certificates, bound to Console / SMTP / Webmail roles
Databases	MariaDB schemas (hermes, djigzo, opendmarc, syslog, authelia, nextcloud) created + seeded
Console settings	`parameters2.console.host` set to the FQDN you entered at install time
Postfix identity	`myhostname` / `myorigin` set from the install-time mail-hostname prompt
Authelia	LDAP backend wired up; 2FA enrollment available on first login
Mail filtering	Amavis + SpamAssassin + ClamAV all initialized and listening

So **after the install you can log in, but mail won't actually flow** until you complete the steps below.

Required configuration

1. System Identity {#system-identity}

Page: System → Server Setup

The install script sets `myhostname` from what you typed at the FQDN prompt, but you should double-check it matches your DNS A / MX records. Also set:

- **Postmaster address** — where bounce messages and admin notifications go
- **Admin email** — where alerts (license, system events) get delivered
- **Time zone** — affects log timestamps and report scheduling

> **Dashboard nudge**: an orange callout `Placeholder hostname` fires (any topology) if `myhostname` still equals the seed default `hermes.domain.tld` or `console.host` equals `smtp.domain.tld`. Both should never appear on a Docker install (the install script overrides them), but if they do, this is the page to fix.

2. First Relay Domain {#first-domain}

Page: Email Relay → Domains

Add **at least one domain** so Hermes knows what mail to accept on the SMTP port. Without this, every inbound message gets rejected with `Relay access denied`.

For each domain you'll choose:

Field	What it controls
Domain	The recipient domain (e.g. `@example.com`)
Recipient delivery mode	Where validated mail goes next — relay forwards to a downstream MX; local delivers to a Hermes-hosted mailbox
Destination address / port	The downstream MX (relay mode) or which mailbox-hosting-domain (local mode)
Policy	Encryption policy applied to outbound mail for this domain (Pro only)

3. Relay Networks {#relay-networks}

Page: Email Relay → Relay Networks

Add the IP addresses or CIDR blocks of any **upstream MTA** (your customer's mail server, an application server that sends notification mail, etc.) that should be allowed to relay outbound mail through Hermes.

By default Hermes only trusts `127.0.0.1` and the Docker bridge subnet (`172.16.32.0/24`). Anything else needs to be added here.

4. First Recipient OR Mailbox {#first-recipient}

You need **at least one of these two** depending on topology:

Relay topology (Hermes forwards to a downstream MX)

Page: Email Relay → Relay Recipients

Add the individual recipients (or wildcards) that Hermes should accept mail for. The validated mail is then forwarded to the destination set on the domain row in step 2.

Hermes-hosted mailboxes topology

Page: Email Server → Mailboxes

Skip Relay Recipients entirely and create mailboxes on Hermes itself. Each mailbox row creates an LDAP user, a Dovecot maildir, and (optionally) a Nextcloud account.

Optional but recommended

5. Relay Host (Outbound Smarthost)

Page: Email Relay → Relay Host

If outbound mail should route through Gmail / M365 / SendGrid / etc. instead of being sent directly to recipient MXes, configure the smarthost here. Authentication credentials are encrypted at rest using the Hermes install's key material.

6. Pro License Activation

Page: System → Server Setup → License section

Enter your serial number to unlock Pro features (organizational signatures, encrypted mail, ACME / Let's Encrypt automation, ARC sealing, etc.). Validation hits `validate.hermesseg.io` over HTTPS; the result is cached locally so Pro stays available during brief network outages.

7. Real TLS Certificate {#optional-tls-cert}

Page: System → System Certificates

Replace the bootstrap self-signed certificate with a real one before going live. Three paths:

Path	Tier	Workflow
Request ACME	Pro only	Click → enter domain → Let's Encrypt issues automatically, auto-renews
Import Certificate	Both tiers	Paste cert + key + chain from any CA you already have
Generate CSR	Both tiers	Generate signing request → submit to CA → import the result via *Import Certificate*

For mailbox hosting domains, see the in-app guide "Choosing the Right Certificate Type" panel on the System Certificates page — mailbox certs need SAN coverage for `autoconfig.<domain>` and `autodiscover.<domain>`.

> **Dashboard nudge**: blue informational callout `Self-signed cert` fires when the only row in `system_certificates` is the install-generated bootstrap (no real cert has been imported yet). Lower priority than the other nudges — Hermes still works on bootstrap, just produces a TLS warning in clients.

8. DNS for Mail Flow

Beyond the gateway itself, DNS is what makes mail actually arrive. The install script does **not** touch DNS — you do this at your registrar.

Record	Where it points	Why
`MX` for each relay domain	The Hermes mail hostname (e.g. `mail.example.com`)	Inbound mail routing
`A` for the mail hostname	Hermes public IP	Resolves the MX target
Reverse DNS (PTR) for the IP	The mail hostname	Outbound deliverability — most receivers reject mismatched PTR
`SPF` for each sending domain	Includes Hermes IP	Authenticates outbound; reduces spam-folder rate
`DKIM` selector → public key	Generated under Content Checks → DKIM	Cryptographic signing of outbound
`DMARC` policy	TXT at `_dmarc.example.com`	Defines what receivers do with SPF/DKIM failures

---

Dashboard nudges

The admin dashboard surfaces two universal callout banners under the navbar — these apply regardless of topology:

Color	Priority	Trigger
Orange `Placeholder hostname`	2	`myhostname` or `console.host` still at the seed placeholder (`hermes.domain.tld` / `smtp.domain.tld`)

Color	Priority	Trigger
Blue `Self-signed cert`	3	Only the bootstrap cert exists in System Certificates — no real cert imported yet

Each banner links directly to the page where you'd fix the underlying condition and disappears automatically as soon as the check passes. The topology-specific gates (relay domains, networks, recipients, mailboxes) intentionally don't have dashboard nudges — they depend on which topology you're using and the guidance above covers them.

After you finish these steps

1. Send a **test message** from an external account to a recipient on one of your relay domains (or to a mailbox if mail-server topology). Check Reports → Mail Log to confirm it arrived at Hermes and was handed off / delivered correctly.
2. (Relay topology) Send a **test outbound message** from your customer MTA (the one whose IP you added to Relay Networks) to an external recipient. Confirm DKIM/SPF pass on the receiver side.
3. Visit **System → Dashboard** and confirm both setup nudges are gone (placeholder hostname + self-signed cert).
4. If you set up Pro features, verify `session.edition` reads "Pro" in the top-right corner of any admin page.

You're done. Welcome to Hermes SEG.

Release and Update Methodology

Release and Update Methodology

This document is the canonical reference for how Hermes SEG (Docker era) is released, distributed, and upgraded. It covers both sides of the loop: the **developer side** (how a release is cut) and the **admin side** (how a running install is updated). Read this if you are:

- An operator deciding how to upgrade a running Hermes install
- A developer adding a schema change, a one-shot migration, or a service config edit and need to know where it goes
- A maintainer cutting a new release tag
- A future Claude session that needs the mental model

Linked work: #218 (release engineering pivot), #221 (`system_update_docker.sh` orchestrator), #231 (beta scope).

Core concepts

Calendar versioning

Versions are `vYYMMDD` — a **label** named for a target/planning date. The digits are **not** the ship date and **not** the git tag date; a release is often named early and tagged months later as the work lands.

```
v260119  →  named for 2026-01-19, actually tagged 2026-05-30   (first Docker release, the
baseline)
v260609  →  named for 2026-06-09                                (backup/restore/DR + upgrade-
```

```
tooling release)
v270315  →  hypothetical: a release named for 2027-03-15
```

The label does **not** auto-increment — there is no `v260120` just because a day passed. For the **actual** release timing of any tag, read the git tag date (`git log -1 <tag>`) or the GitHub Release page — never infer it from the digits.

The version stamp is stored in two `system_settings` rows:

Parameter	Value	Meaning
`version_no`	`'Docker'`	Code train identifier (post-bare-metal era)
`build_no`	`'v260119'`	Specific release tag the install is currently at

Image registry

Container images live at `ghcr.io/deeztek/hermes-<service>:<tag>`. The registry hostname is configurable via the `IMAGE_REGISTRY` env var in `.env` so the legacy GitLab CR (`hub.deeztek.com/dedwards/hermes-seg-docker-gl`) can still be used during the bootstrap period before ghcr.io is fully populated.

Distribution repo

Code lives on GitLab (dev). Distribution is via GitHub (`deeztek/Hermes-Secure-Email-Gateway`) — issues, releases, container packages all there. A two-remote `scripts/git_release.sh` helper codifies the dev-push / release-push split.

Artifact taxonomy — where does what go?

Hermes ships **three categories** of changes between releases. Each has a dedicated home in the repo.

```
+-------------------------------------------------------------+
|  CATEGORY 1: BASELINE                                       |
|  Lives at: config/database/hermes_install.sql               |
```

```
|  Purpose:  Self-contained snapshot of the schema AT v260119.     |
|            Imported once on fresh install. Never modified by     |
|            upgrades.                                             |
+------------------------------------------------------------------+

+------------------------------------------------------------------+
|  CATEGORY 2: PER-RELEASE ARTIFACTS                               |
|  Lives at: updates/v<DATE>/                                      |
|    ├── README.md                  (what this release does)      |
|    ├── sql/schema_updates.sql     (schema deltas)               |
|    ├── cfml/*.cfm                 (one-shot CFML migrations)    |
|    └── scripts/*.sh               (one-shot bash one-shots)     |
|  Purpose:  Everything that changed between the PREVIOUS release |
|            and THIS release. Applied once, in order, by the     |
|            update orchestrator. Discoverable: one directory     |
|            tells you everything that release does.               |
+------------------------------------------------------------------+

+------------------------------------------------------------------+
|  CATEGORY 3: PERSISTENT POST-UPGRADE HOOK                        |
|  Lives at: config/hermes/var/www/html/schedule/post_upgrade.cfm  |
|  Purpose:  Cross-release CFML migrations gated by a `migrations` |
|            table. Each named block runs ONCE EVER per install.  |
|            Used for retroactive backfills and migrations that    |
|            don't cleanly bind to a single release.               |
+------------------------------------------------------------------+
```

Categories side-by-side

	Baseline	Per-release	Persistent hook
File location	`config/database/hermes_install.sql`	`updates/v<DATE>/`	`schedule/post_upgrade.cfm`
Scope	Whole v260119 schema	One release's deltas	Cross-release backfills
When applied	Once, at fresh install	Once, when upgrading past that version	At every upgrade; per-block gated
Lifecycle	Replaced wholesale at major rebaselines (rare)	Lives forever in repo as history	Blocks accumulate over time; stay as documentation

	Baseline	Per-release	Persistent hook
Idempotency	Required (`IF NOT EXISTS` / `INSERT IGNORE`)	Required per artifact	Required per block (`migrations` table check)
Discoverability	One file, mysqldump-style	One dir per release	One file with named blocks

Choosing the right category

Use this decision tree when adding a schema/data/config change:

```
Is this a one-off "we noticed something needs fixing across all installs"
that doesn't bind to any specific release?
  YES → Persistent post-upgrade hook (category 3)
  NO  → continue

Is this a schema change, seed data, or config that NEW INSTALLS should have?
  YES → Add to baseline (category 1) — ALSO add to the current in-flight
        release's updates/v<DATE>/ (category 2) so existing installs get it
        on upgrade.
  NO  → continue

Is this a one-shot migration that only matters when transitioning past a
specific release (e.g., re-encrypt rows after a key rotation)?
  YES → Per-release artifact (category 2):
          - Pure SQL → updates/v<DATE>/sql/schema_updates.sql
          - Needs CFML (encryption, file I/O, API calls) → updates/v<DATE>/cfml/<name>.cfm
          - Needs host shell (docker exec, file moves outside containers) →
updates/v<DATE>/scripts/<name>.sh
```

Idempotency rules

Every artifact, in every category, MUST be safely re-runnable. The orchestrator does not track "did I apply this already" at fine granularity — it relies on each artifact to no-op when there's nothing to do.

SQL artifacts

Statement	Idempotency pattern

`CREATE TABLE`	`CREATE TABLE IF NOT EXISTS`
`ALTER TABLE ADD COLUMN`	`ADD COLUMN IF NOT EXISTS`
`ALTER TABLE DROP COLUMN`	`DROP COLUMN IF EXISTS`
`ALTER TABLE MODIFY COLUMN`	Naturally idempotent (re-applying same type is no-op)
`CREATE INDEX`	`CREATE INDEX IF NOT EXISTS`
`INSERT`	`INSERT IGNORE` (relies on UNIQUE key) OR `INSERT ... SELECT ... WHERE NOT EXISTS`
`UPDATE`	Value-gated `WHERE` clause that only matches pre-fix state (e.g., `WHERE value = 'Fail'`)
`DELETE`	Targeted `WHERE` that becomes empty after first application
`CREATE USER` / `GRANT`	Wrap in `PREPARE` block that checks `mysql.user` / `mysql.db` first (see `config/database/hermes_install.sql` for canonical pattern)

When in doubt, look at how an existing block in `hermes_install.sql` or a prior `schema_updates.sql` handled the same pattern. The CLAUDE.md "Schema Updates" section also documents these patterns inline.

CFML artifacts

Each `.cfm` file in `updates/v<DATE>/cfml/` or each block in `post_upgrade.cfm` must:

1. **Check before doing**: query something (a row's encryption state, a file's existence, a setting's value) to decide whether to act.
2. **Act**: do the work.
3. **Record completion**: either update a row to a state the check in step 1 will reject, OR insert into the `migrations` table.

The `migrations` table (created by Phase 5 framework, see below):

```
CREATE TABLE IF NOT EXISTS `migrations` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    `completed_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`),
    UNIQUE KEY `uq_name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Bash artifacts

Each `.sh` file in `updates/v<DATE>/scripts/` must:

1. **Precondition check**: test whether the operation has already been performed (file exists, container running, value set, etc.).
2. **Act**: do the work.
3. **Exit 0 on no-op**: a script invoked when its preconditions say "already done" must exit 0, not error.

The update orchestrator (`scripts/system_update_docker.sh`)

This script (target: #221) is the **single command an admin runs** to upgrade a Hermes install. It runs five phases in order; each is independently idempotent.

Phase 1 — Pull new code

```
cd /opt/hermes-seg-docker-gl
git fetch --tags
git checkout <target-tag>
```

The target tag is either passed as an argument (`./system_update_docker.sh v260601`) or auto-resolved by polling GitHub Releases API for `releases/latest`. If the working tree is dirty, the script refuses to continue.

Phase 2 — Update containers

```
docker compose pull          # only when HERMES_DOCKER_IMG_VERSION in .env bumped
docker compose up -d         # always (idempotent: only restarts services whose config
changed)
```

Image registry comes from `IMAGE_REGISTRY` in `.env`. Containers whose images didn't change are NOT restarted by `compose up -d`.

Phase 3 — Apply per-release artifacts

The orchestrator globs `updates/v*/` directories, sorts chronologically (calendar versioning sorts correctly as a string), filters to versions newer than current `build_no`, and applies each release in

order:

```
For each release dir D in (sorted release dirs above build_no):
    For each file F in D/sql/*.sql       (alphabetical):
        docker exec -i hermes_db_server mysql -u root hermes < F
    For each file F in D/cfml/*.cfm      (alphabetical):
        curl -sf http://localhost:8888/updates/<dir>/cfml/<name>.cfm
    For each file F in D/scripts/*.sh    (alphabetical):
        bash F
    # Each release's schema_updates.sql advances build_no at its end via
    #   UPDATE system_settings SET value='<DATE>' WHERE parameter='build_no';
    # giving the orchestrator a natural cursor across releases.
```

If any artifact fails, the orchestrator aborts. Because `build_no` advances at the END of each release's `sql/` step, a failed-mid-release re-run picks up at the failed release and re-applies its full set (idempotency makes that safe).

Why this ordering: SQL changes the schema first so CFML and bash artifacts that depend on new columns/tables work. CFML before bash because CFML migrations often need the schema landing but Lucee's lifecycle is already running. Bash last for host-level fixups.

Phase 4 — Standard finalize steps

```
- Restart hermes_commandbox so it picks up admin/2/, schedule/, schema changes
- NC version-drift detection (#264): if .env's NCVERSION differs from
  the live `occ status` versionstring → run `occ upgrade` + rehydrate
  Hermes-required NC apps (user_oidc, mail, twofactor_totp,
  twofactor_backupcodes, external) + verify needsDbUpgrade=false
- Re-render *.HERMES config templates (reminder only in MVP; v2 will
  detect modified template files and trigger each regen endpoint)
```

NC upgrade is fully automated — no manual `occ upgrade` step needed after a release that bumps `NCVERSION`. The rehydrate loop covers NC's tendency to auto-disable apps it thinks are incompatible with a new core version. Service restart triggers (other than commandbox) will be detected in v2 by diffing the working tree against the previous tag.

Phase 5 — Persistent post-upgrade hook

Last step: orchestrator calls the persistent hook.

```
curl -sf http://localhost:8888/schedule/post_upgrade.cfm
```

`post_upgrade.cfm` contains N named migration blocks. Each block:

1. Queries `migrations` table: has this block run before?
2. If yes: log "skipped (already applied)" and move on.
3. If no: do the work, then `INSERT INTO migrations (name) VALUES ('<block-name>')`.

This hook is for migrations that:

- Don't bind cleanly to a release boundary
- Need to be retroactive across all existing installs regardless of when they upgrade
- Replace existing inline-in-CFML migrations (lazy backfills currently scattered across page handlers)

Examples (some hypothetical):

Block name	What it does
`encrypt-relay-credentials-v1`	Re-encrypts `parameters` rows whose values were stored as plaintext before the key-rotation policy landed
`move-pgp-keys-to-vault-dir`	One-time file move from `/opt/hermes/keys/` to `/opt/hermes/keys/pgp/` after a refactor
`backfill-mailbox-domain-id`	Populates a `domain_id` column added in a prior release where the existing data needs computed values

The admin update procedure

For the period BEFORE `system_update_docker.sh` ships (#221), updates are manual. After #221 lands, the same procedure collapses into one command.

Today (pre-#221)

```
cd /opt/hermes-seg-docker-gl
git fetch --tags
git checkout <new-tag>                                  # or: git reset --hard origin/main
docker compose pull
docker compose up -d
./scripts/install_hermes_docker.sh --apply-schema       # SQL deltas
# Manual steps below ONLY if release notes call them out:
docker exec hermes_commandbox occ upgrade               # if NCVERSION bumped
```

```
# Service-specific restarts per release notes
```

`install_hermes_docker.sh --apply-schema` globs `updates/v*/sql/schema_updates.sql` and applies any with version newer than current `build_no`. On a v260119-only install, it logs "No per-release schema_updates.sql files found" and exits cleanly.

After #221

```
cd /opt/hermes-seg-docker-gl
./scripts/system_update_docker.sh           # latest release
./scripts/system_update_docker.sh v260601   # specific tag
```

The orchestrator runs all 5 phases. Idempotent + re-runnable. Logs every step. Refuses to run if working tree is dirty.

Pre-upgrade: take a hypervisor snapshot

Before running `system_update_docker.sh`, take a hypervisor / VM snapshot of the entire Hermes host (Proxmox, VMware, KVM, AWS EBS, Azure Disk, GCE, Hyper-V). If the upgrade fails partway through, reverting the snapshot is the only currently-supported rollback path — Docker-aware backup/restore tooling is in development at #219 and #220 and is not yet shipped. See `docs/admin/01-system/backup-restore.md` for the full interim-backup guidance.

Post-upgrade: hard-refresh the browser (any release that bumps NCVERSION or admin/2 assets)

After the orchestrator finishes, **operators must hard-refresh their browser** (Ctrl-Shift-R on Linux/Windows, Cmd-Shift-R on macOS) on any open Hermes admin tab or Nextcloud tab. The browser cache often serves the pre-upgrade CSS/JS bundle even though the server is now serving the new one, which presents as broken layouts — most visibly NC's top navbar collapsing to a vertical stack of icons when the NC 30 → 31 bundle swaps under it.

Release notes that bump `NCVERSION` or land changes under `config/hermes/var/www/html/admin/2/` should call this out explicitly. One sentence in the release notes prevents the "Nextcloud top bar broke" support ticket.

The release-cut procedure (developer side)

For maintainers preparing a release:

1. **Code complete on `main`**: all PRs for the release have landed on GitLab `main`.
2. **Create the release directory**: `mkdir -p updates/v<DATE>/{sql,cfml,scripts}` (only the subdirs you actually need). Add `updates/v<DATE>/README.md` documenting what changed.
3. **Add SQL deltas**: write `updates/v<DATE>/sql/schema_updates.sql`. ALSO add equivalent rows/columns to `config/database/hermes_install.sql` so fresh installs after this release ship with them baked in. **Last block of `schema_updates.sql`** must be:

   ```
   UPDATE system_settings SET value='v<DATE>' WHERE parameter='build_no';
   UPDATE system_settings SET value='Docker'  WHERE parameter='version_no';
   ```

4. **Add CFML / bash migrations** as needed under `updates/v<DATE>/cfml/` and `updates/v<DATE>/scripts/`.
5. **Update `.env.template`**: bump `HERMES_DOCKER_IMG_VERSION=v<DATE>` and (if NC bumped) `NCVERSION=...`. `NCVERSION` is release-managed per #261 — operators never edit it; bumps land here only after the integration check in step 6 passes.
6. **(If `NCVERSION` was bumped in step 5) Run the NC integration check on a Test box**:

   ```
   # On Test, after a fresh clone or pull that includes the new .env.template
   docker compose pull hermes_nextcloud
   docker compose up -d hermes_nextcloud
   sleep 30                                    # let NC finish first-start init
   ./scripts/test_nc_integration.sh
   ```

   The script (read-only `occ` queries + log scan) verifies the Hermes-NC integration surface: container responsive, `occ status` reports the expected version, no `needsDbUpgrade`, required apps enabled and not flagged incompatible (`user_oidc` / `mail` / `twofactor_totp` / `twofactor_backupcodes` / `external`), `trusted_domains` populated, theming URL set, `user_oidc` provider `Hermes_SEG` registered, and no ERROR/FATAL entries in the last 200 `nextcloud.log` lines. Exit code 0 if no FAIL.
   If anything fails, fix the integration (or revert the `NCVERSION` bump and pin to the prior NC) before continuing. Do not publish a release that ships a failing NC integration.
7. **Draft the GitHub Release body for `v<DATE>`**: list every change in the release. Per-release notes live on the GitHub Release page (created when the tag is pushed) — the cumulative `RELEASE-NOTES.md` was retired because release notes belong to a specific tag, not an ever-growing file.
8. **Build + push images**: `./Docker/build-all-ghcr.sh && ./Docker/push-all-ghcr.sh` (or wait for GitHub Actions per #218 Session C).

9. **Tag + push**: `./scripts/git_release.sh --release v<DATE>` (pushes branch + tag to both GitLab and GitHub).
10. **Verify**: GitHub Release page exists, ghcr.io packages updated, run `./scripts/system_update_docker.sh v<DATE>` on Test box and confirm clean upgrade.

Common scenarios

"I'm adding a new feature that needs a new table."

1. Add `CREATE TABLE IF NOT EXISTS ...` to `config/database/hermes_install.sql` (baseline gets it for fresh installs).
2. Add the same `CREATE TABLE IF NOT EXISTS ...` to the in-flight release's `updates/v<DATE>/sql/schema_updates.sql` (so existing installs get it on upgrade).
3. Add seed rows the table needs in BOTH files following the same pattern.

"I'm adding a new system_settings row with a default value."

Same as above: baseline + per-release. Use auto-assigned IDs (omit the `id` column); UNIQUE KEY on `parameter` handles dedup. See [[feedback-no-hardcoded-row-ids-on-new-seeds]] in memory.

"I'm changing a default value an existing install might have customized."

`UPDATE ... WHERE value = '<old-default>'` only in `updates/v<DATE>/sql/schema_updates.sql`. The `WHERE` clause preserves admin customizations. Also update `hermes_install.sql` baseline to the new default for fresh installs.

"I rotated an encryption key and need to re-encrypt rows."

Per-release `updates/v<DATE>/cfml/reencrypt-foo.cfm`. Pure SQL can't reach the encryption key. Inside the script, check whether each row is already at the new format before re-encrypting.

"I need to move a config file from `/old/path` to `/new/path` on the host."

Per-release `updates/v<DATE>/scripts/move-config.sh`. Precondition check: if `/new/path` exists, exit 0.

"I just realized all installs need a one-time backfill but I'm not cutting a release for it."

Add a named block to `schedule/post_upgrade.cfm`. It runs at the next upgrade (or at every upgrade, gated by `migrations` table).

"Fresh install at v260119, what runs?"

Just `config/database/hermes_install.sql`. The baseline is self-contained. `updates/v260119/` has no `sql/` subdirectory by design. `--apply-schema` finds nothing to apply and reports current `build_no=v260119`.

"Upgrading from v260119 to v260601 (hypothetical), what runs?"

In order:

1. `git checkout v260601`
2. `docker compose pull && docker compose up -d`
3. `updates/v260601/sql/schema_updates.sql` (advances `build_no` to `v260601` at its end)
4. `updates/v260601/cfml/*.cfm` (each)
5. `updates/v260601/scripts/*.sh` (each)
6. Standard finalize (restarts, occ upgrade, template regen)
7. `schedule/post_upgrade.cfm` (any unapplied blocks)

"Upgrading from v260119 to v260801, skipping v260601 (hypothetical)."

The orchestrator applies BOTH release directories in order:

1. Phase 1-2 once
2. Phase 3 walks `v260601/` first (sql → cfml → scripts), then `v260801/`
3. Phase 4-5 once at end

`build_no` advances after each release's `sql/` completes — `v260601` then `v260801`. Each release's artifacts are idempotent so the in-between cursor advancement is safe.

State of the methodology as of v260119

Piece	Status
Baseline (`hermes_install.sql`)	Self-contained at v260119 (audit completed 2026-05-26)
Per-release directory (`updates/v260119/`)	Empty by design — v260119 IS the baseline; first real per-release directory is the NEXT release
`install_hermes_docker.sh --init-db`	Imports baseline only; no schema_updates.sql call
`install_hermes_docker.sh --apply-schema`	Globs `updates/v*/sql/schema_updates.sql`, applies in version order; no-op on v260119-only
`scripts/system_update_docker.sh` (#221)	**MVP shipped 2026-05-26** — phases 1-5 functional; some v2 polish deferred (see "Known MVP limitations" below)
`schedule/post_upgrade.cfm` framework	Stub shipped 2026-05-26 — framework + helpers + `migrations` table in place, zero blocks registered (first one lands when first migration is needed)
`migrations` table	Added to baseline 2026-05-26
GitHub Actions release workflow	NOT YET BUILT — image pushes are manual via `Docker/push-all-ghcr.sh`
Image registry (ghcr.io)	Empty as of 2026-05-25; bootstrap is a pre-Session B task

Known MVP limitations of `system_update_docker.sh`

The MVP that shipped 2026-05-26 is deliberately scoped. These limitations are tracked for v2:

Limitation	What happens today	v2 plan
Service-restart detection	Always restarts `hermes_commandbox` in Phase 4; logs a reminder that admin may need to re-save certain config pages	Diff config files between previous and new tag; only restart containers whose volume-mounted config changed
`*.HERMES` template re-render	Phase 4 logs a reminder; admin must re-save the corresponding admin page manually	Detect modified template files and invoke each one's regen endpoint via curl into commandbox
`occ upgrade`	**Phase 4 auto-detects NCVERSION drift, runs `occ upgrade`, rehydrates Hermes-required NC apps, verifies post-upgrade state** (#264)	— already MVP-complete —
`updates/v<DATE>/cfml/*.cfm` artifacts	Phase 3 logs a WARN and skips them	First release that ships a CFML migration must add a host→container mount for `updates/` plus a Lucee mapping; the orchestrator already curls the right URL pattern, just needs the mount to exist
Mid-upgrade resume	`set -e` fail-fast; operator re-runs from scratch and idempotency handles re-application	Track per-phase + per-release-artifact completion; resume from last successful step
GitHub Releases API auth	Anonymous polling; subject to GitHub's unauthenticated rate limit (60 req/hr per IP)	Honor a `GITHUB_TOKEN` env var if present

Storage Topology (5 tiers)

Storage Topology (5 tiers)

Hermes SEG splits storage into five independent tiers so each can live on the right kind of disk for its workload. Four are operator-chosen at install time; the fifth (Config) is implicit — chosen by where the operator `git clone`d the repo.

Tier	Default path	Contents	Storage profile
1. Config	install root (implicit)	Repo working tree, `config/` subtrees, install script, secrets in `config/hermes/opt/hermes/keys/`, `.env`, `.hermes_install_config`	Fast SSD, modest size — chosen by where the repo lives
2. Data	`/mnt/data` (`DATA_MOUNT`)	DBs (MariaDB, Authelia, OpenLDAP), Amavis runtime state, ClamAV signatures, Fangfrisch state, Lucee server home, sieve scripts, all service logs, OpenDMARC, Postfix queue	Fast SSD; sized for DB growth + log retention
3. Archive	`/mnt/archive` (`ARCHIVE_MOUNT`) — added in #260	Amavis quarantine archive	Cheap bulk; sized for retention policy × quarantine inflow
4. Vmail	`/mnt/vmail` (`VMAIL_MOUNT`)	Dovecot mailboxes	Cheap bulk; sized for users × quota
5. Nextcloud	`/mnt/files` (`FILES_MOUNT`)	Nextcloud app + user files + Nextcloud's Redis cache	Cheap bulk; sized for user file storage

Each tier is **one host path**; the install script lays out the canonical sub-directory structure underneath it.

Why split storage

Tier	Why it gets its own disk

Config	Frequent reads (every container start); small footprint; lives with the install script + version control
Data	High write rate (logs + DBs); benefits from fast SSD; backup hot spot
Archive	Grows unboundedly with retention policy; cold reads (admin browses quarantine occasionally); cheaper bulk storage; backup cadence independent of Data
Vmail	Grows linearly with user count × quota; cheaper bulk storage; separate backup cadence (often less frequent than Data)
Nextcloud	Same growth characteristics as Vmail but different access pattern; often shared across multiple Hermes installs in larger deployments

Smaller deployments can collapse tiers — point Archive, Vmail, and Nextcloud at the same path as Data for a single-disk install. Each tier is its own prompt so the operator picks per workload.

Canonical sub-directory layout

Tier 2 — Data (default `/mnt/data/`)

Sub-path	Named volume	Service
`dbase/`	`db_data`	MariaDB
`authelia/db/`	`authelia_db`	Authelia state DB
`authelia/logs/`	`authelia_logs`	Authelia logs
`authelia/redis/`	`authelia_redis`	Authelia Redis
`commandbox/serverhome/`	`commandbox_serverhome`	Lucee server home
`dmarc/logs/`	`dmarc_logs`	OpenDMARC logs
`dovecot/logs/`	`dovecot_logs`	Dovecot service logs
`dovecot/sieve/`	`dovecot_sieve`	Sieve scripts (shared by commandbox + dovecot)
`ldap/data/`	`ldap_data`	OpenLDAP data
`ldap/logs/`	`ldap_logs`	OpenLDAP logs
`mail_filter/data/amavis/`	`mail_filter_data_amavis`	Amavis runtime state (PID files, scan tmp dirs — small, latency-sensitive)
`mail_filter/data/clamav/`	`mail_filter_data_clamav`	ClamAV signatures

Sub-path	Named volume	Service
`mail_filter/data/fangfrisch/`	`mail_filter_data_fangfrisch`	Fangfrisch state
`mail_filter/logs/`	`mail_filter_logs`	Mail filter logs
`nginx/logs/`	`nginx_logs`	Nginx logs
`openarc/logs/`	`openarc_logs`	OpenARC logs
`postfix_dkim/logs/`	`postfix_dkim_logs`	Postfix logs
`postfix_dkim/queue/`	`postfix_dkim_queue`	Postfix mail queue

Tier 3 — Archive (default `/mnt/archive/`) — added in #260

Sub-path	Named volume	Service
`amavis/`	`amavis_data`	Amavis quarantine archive (admin-browsable, grows with retention)

Note: the Amavis runtime state (`mail_filter/data/amavis/` named volume `mail_filter_data_amavis`) stays on the Data tier — it's small, doesn't grow with retention, and benefits from fast SSD latency. Only the quarantine archive moved.

Tier 4 — Vmail (default `/mnt/vmail/`)

Sub-path	Named volume	Service
`dovecot/`	`dovecot_mail`	Dovecot mailboxes

Tier 5 — Nextcloud (default `/mnt/files/`)

Sub-path	Named volume	Service
`app/`	`nextcloud`	NC app + user files
`redis/`	`nextcloud_redis`	NC's Redis cache

How it works at install time

1. `prompt_mount_points()` asks the operator for four paths (Data / Archive / Vmail / Nextcloud) — Config is already chosen by where the repo lives. Defaults `/mnt/data`, `/mnt/archive`, `/mnt/vmail`, `/mnt/files`. Choices saved to `.hermes_install_config` at the install root.
2. `provision_mount_dirs()` creates the entire sub-directory layout under each chosen path with the correct UID/GID for the containers that will write to them. Critical: bind-mounted volumes (`type: none, o: bind` in `docker-compose.yml`) require the source directory to **pre-exist** — Docker refuses to start the container otherwise.
3. `generate_compose_override()` writes the four mount-point variables (`DATA_MOUNT` / `ARCHIVE_MOUNT` / `VMAIL_MOUNT` / `FILES_MOUNT`) to `.env` at the install root. `docker-compose.yml` references these variables directly in its `device:` lines (e.g. `device: ${ARCHIVE_MOUNT}/amavis`) — Docker Compose substitutes at runtime. The legacy `docker-compose.override.yml` approach was retired in #179; the function name was kept for backwards-compatibility with the `--generate-override` CLI flag.
4. All four mount points are **required**. Empty values would resolve to dangerous relative paths during `device:` substitution (e.g. empty `${ARCHIVE_MOUNT}/amavis` → `/amavis` at the host root). For single-disk installs, point all four prompts at the same path.

Self-locating scripts

`install_hermes_docker.sh`, `rotate_db_credentials.sh`, and any other Hermes script needing the install root use a **walk-up self-locator** pattern that finds `docker-compose.yml` by walking up from `BASH_SOURCE[0]`:

```
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
if [[ -z "${HERMES_ROOT:-}" ]]; then
    HERMES_ROOT="$SCRIPT_DIR"
    while [[ "$HERMES_ROOT" != "/" ]] && [[ ! -f "$HERMES_ROOT/docker-compose.yml" ]]; do
        HERMES_ROOT="$(dirname "$HERMES_ROOT")"
    done
    if [[ "$HERMES_ROOT" == "/" ]]; then
        echo "ERROR: Could not locate docker-compose.yml in any parent of $SCRIPT_DIR" >&2
        echo "Set HERMES_ROOT environment variable manually and retry." >&2
        exit 1
    fi
fi
```

This is depth-independent (works at 1 level or 5 levels deep in the tree) and survives the script being relocated. **Do not use a hardcoded `dirname/..` chain** — it depends on the script's exact depth and breaks silently if the script moves.

Reading topology at runtime

`.hermes_install_config` is the source of truth for which paths the operator chose. Scripts that need this (`system_backup.sh`, `system_restore.sh`) source the file via the `load_config()` helper. Format:

```
DATA_MOUNT=/mnt/data
ARCHIVE_MOUNT=/mnt/archive
VMAIL_MOUNT=/mnt/vmail
FILES_MOUNT=/mnt/files
ENABLE_NEXTCLOUD=true
```

The file lives in the Config tier (install root), so it's part of every Config-tier backup automatically.

Post-Restore Steps

Post-Restore Steps

Run after `scripts/system_restore.sh` completes. The restore replaces databases, LDAP, and the storage tiers, but a few things must be reconciled **by hand** depending on whether you restored onto the **same host** or a **different host** (cross-host disaster recovery). The restore script prints a pointer to this page at the end of its run.

Same-host restore

You restored a backup onto the **same** machine it came from (e.g. rolling back).

1. **Log into the admin console** — `https://<console-host>/admin/`.
2. **Verify mail flow** — send and receive a test message.
3. **Check the log** — `install-logs/system_restore_*.log` for any `WARN` lines.

Nothing else is usually required: `.env`, `creds/`, and host identity already match this machine.

Cross-host restore (restore to new hardware)

You restored a backup taken on host **A** onto a different host **B** (DR to new hardware, migration, etc.). The restored data carries host A's identity and credentials, so **B serves A's configuration until you reconcile it**. Do these in order:

1. Rewire host identity — `system_rehost.sh` (REQUIRED)

The restored `.env`, `parameters2` rows, nginx vhost, Authelia cookie domain, and Postfix hostname all reference host A. Until you rewire them, the console is **unreachable** at host B's address.

```
sudo /opt/hermes-seg/scripts/system_rehost.sh
```

It auto-detects this host's IP/hostname (pass `--to-hostname=` / `--to-ip=` to override) and rewrites identity across `.env`, the database, and the rendered configs. See `system_rehost.sh --help`. The restore script prints this reminder automatically when it detects a host-identity mismatch.

> **Note:** `system_rehost.sh` also reconciles cross-host DB credentials for Nextcloud (its `config.php` rides in the restored data tier). All other services use install-generated configs that hold this host's credentials.

2. Re-activate the Pro license

Hermes **Pro** licenses are bound to a hardware-derived UUID. On new hardware the UUID changes, so the license reads `INVALID` / `VIOLATION` on the next validation and the console drops to **Community Edition**.

- Re-activate the license, or contact Hermes support to re-bind your serial to this host.
- **Community Edition installs are unaffected.**

3. Re-save Content Checks (re-apply the milter chain)

`smtpd_milters` (the DKIM / DMARC / ARC / SPF milter chain) is **not** restored — it is dynamic config applied by the Content Checks settings pages, not part of the restored `main.cf`. After a cross-host restore the database still records which checks are enabled, but the live Postfix config does not reflect them, so **outbound mail is not DKIM-signed and inbound is not DMARC/ARC-checked** until you re-apply them.

For each **enabled** Content Checks page, open it in the admin console and click **Save** (no changes needed) to re-apply its Postfix config:

- Content Checks → **DKIM**
- Content Checks → **DMARC**
- Content Checks → **SPF**
- Content Checks → **ARC** (if used)

Verify afterward:

```
docker exec hermes_postfix_dkim postconf smtpd_milters
```

It should list the milter sockets (e.g. `inet:hermes_dmarc:54321`, OpenDKIM, OpenARC, body_milter), not be empty.

> This re-save step is an interim workaround. Automatic re-application of the milter chain on restore is tracked as a follow-up enhancement (#268).

4. Clear Nextcloud maintenance mode (if needed)

The restore clears it automatically, but if Nextcloud still shows maintenance mode:

```
docker exec -u www-data hermes_nextcloud php occ maintenance:mode --off
```

5. Verify

- Admin console login at host B's address.
- Mail flow (send + receive).
- Nextcloud login.
- `docker exec hermes_postfix_dkim postconf smtpd_milters` is populated (step 3).

Health check

After either path, run the smoke test to confirm container health, per-service DB authentication, and the mail chain:

```
bash /opt/hermes-seg/scripts/hermes_smoke_test.sh
```

A `WARN` on `smtpd_milters is empty` after a cross-host restore is the symptom that step 3 has not been completed yet.

Hermes SEG Email Flow

Hermes SEG Email Flow

Reference diagram for the inbound, outbound, and CipherMail-originated mail paths inside the Docker stack. Includes every listening port, every container, the milter chain at each smtpd service, and where body modifications occur relative to DKIM signing.

Container + port map (at a glance)

Container	Service / Daemon	Port(s)	Role
`hermes_postfix_dkim`	postfix smtpd (`:25`)	25	Inbound MX
`hermes_postfix_dkim`	postfix smtpd (`submission`)	587	Authenticated outbound (STARTTLS)
`hermes_postfix_dkim`	postfix smtpd (`smtps`)	465	Authenticated outbound (implicit TLS)
`hermes_postfix_dkim`	postfix smtpd (`:10026`)	10026	Re-injection (post-CipherMail)
`hermes_postfix_dkim`	postfix smtpd (`:10027`)	10027	CipherMail web-GUI originated mail
`hermes_postfix_dkim`	OpenDKIM **primary** (sv mode)	8891	Verify inbound / sign outbound at `:25` / `:587` / `:465`
`hermes_postfix_dkim`	OpenDKIM **sign-only** (s mode) — #232	8892	Sign post-CipherMail egress at `:10026` only
`hermes_mail_filter`	amavisd-new (filter)	10021	SpamAssassin + ClamAV + policy
`hermes_mail_filter`	amavisd-new (pickup / bypass)	10030	`BYPASSALLCHECKS` lane
`hermes_body_milter`	Python pymilter (#214/#226/#228/#230)	8893	Disclaimer + signature + banner + CID inline
`hermes_dmarc`	OpenDMARC	54321	DMARC verify / SPF alignment header

Container	Service / Daemon	Port(s)	Role
`hermes_openarc`	OpenARC (#229, flowerysong v1.3.0 built from source)	8893	ARC sealing at `:10026` only — RFC 8617 chain preservation across body mods
`hermes_ciphermail`	CipherMail SMTP	25	Encryption decisions + MIME rebuild
`hermes_dovecot`	LMTP	24	Local mailbox delivery

Milter listening side: the OpenDKIM/OpenDMARC/body_milter/OpenARC daemons listen on TCP and postfix smtpd connects to them per the `smtpd_milters` line in effect for each port.

> Note: `hermes_body_milter` and `hermes_openarc` both listen on internal port `8893`, but they are separate containers with their own network namespaces. Postfix reaches each by container name (`inet:hermes_body_milter:8893` vs `inet:hermes_openarc:8893`), so the shared port number causes no conflict.

Milter chains by smtpd port

The `smtpd_milters` chain order is set globally in `main.cf` (built from the `parameters` DB table by `generate_postfix_configuration.cfm`) and overridden per-service in `master.cf` for `:10026` and `:10027`.

smtpd port	Milter chain (in order)	Why
`:25` (inbound)	OpenDKIM **:8891** → OpenDMARC **:54321** → body_milter **:8893**	Verify DKIM, verify DMARC, then inject External Banner / disclaimer
`:587` `:465` (submission)	OpenDKIM **:8891** → OpenDMARC **:54321** → body_milter **:8893**	Sign DKIM first (before body mods), then disclaimer/signature inject — **wrong order; see #232 outbound fix history**
`:10026` (re-inject)	OpenDKIM **:8892** sign-only → OpenARC **:8893** (`hermes_openarc`)	Re-sign body that CipherMail mutated, then ARC-seal the **final** form so downstream verifiers trust the cumulative auth chain even after body modification (#229)
`:10027` (CipherMail GUI)	OpenDKIM **:8891**	Sign GUI-originated mail; no body mods on this path

Why OpenARC sits ONLY at `:10026` (and NOT in `main.cf`)

OpenARC's `ARC-Message-Signature` includes a hash of the message body. If ARC sealed at `:25`, the body would later be mutated by body_milter (`:8893`) and CipherMail (MIME rebuild), so the seal's body hash would be invalid by the time downstream verifiers received the message — `cv=fail`.

`:10026` is the only point where the body is in its **final** form (all body modifications + CipherMail MIME rebuild complete), so it's the only correct hop to apply the ARC seal. Adding ARC to `main.cf`'s default `smtpd_milters` would cause two problems:

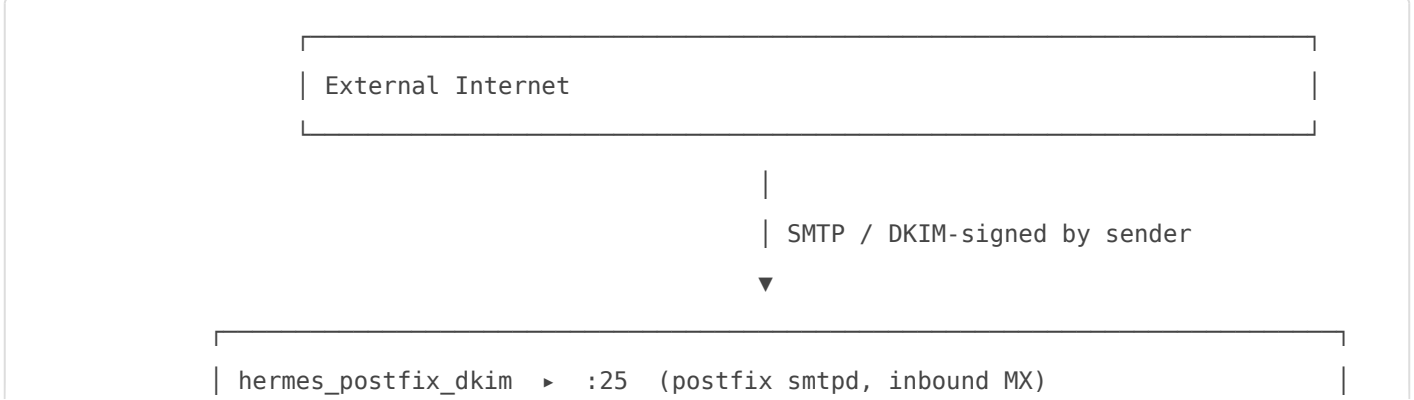
1. **Pre-modification sealing at `:25`** → broken seal at the recipient.
2. **Double-sealing**: mail going through `:25` → amavis → `:10026` would be sealed twice by the same gateway (`i=1` at `:25`, `i=2` at `:10026`), producing redundant chain bloat / verification ambiguity.

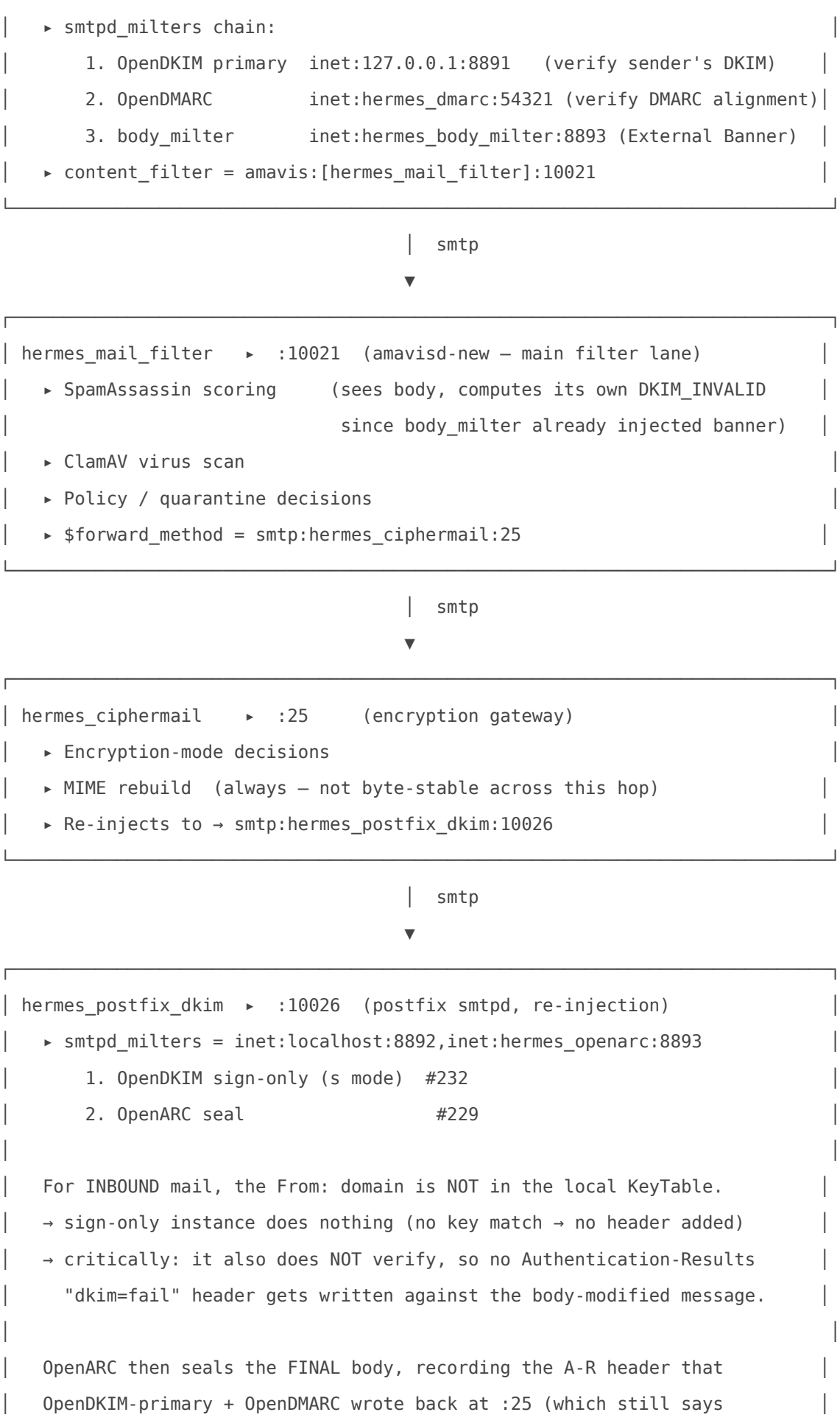
ARC stays out of `main.cf` deliberately. Master.cf `:10026` override is the single point of truth.

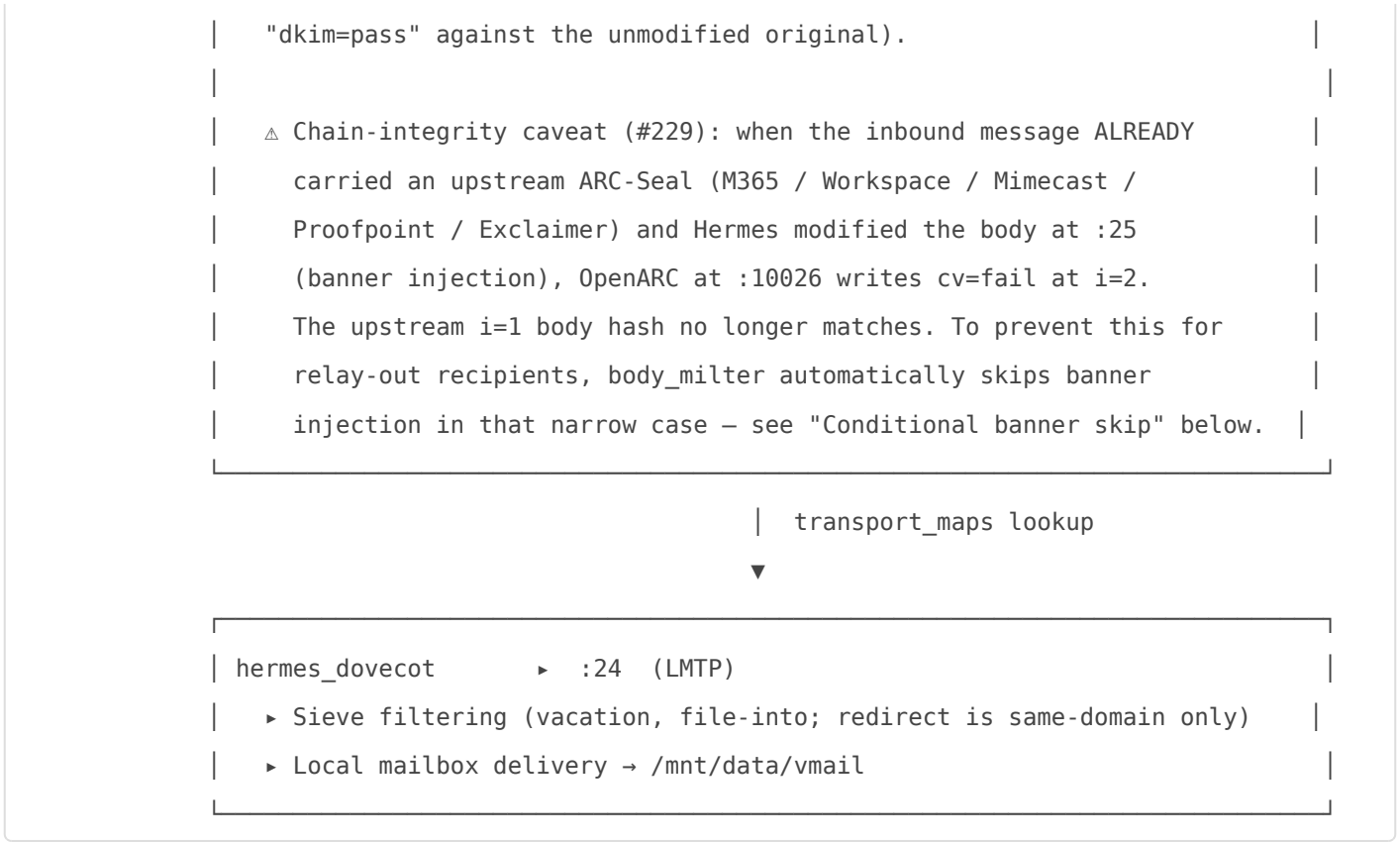
> The body_milter ordering is recorded in `parameters` as `order1=3.1` so it sits AFTER OpenDKIM signer (`0.5`) and OpenDMARC. The retro-fix `UPDATE` in `updates/hermes-260119/sql/schema_updates.sql` corrects existing installs that had `0.5` for body_milter (which placed body mods BEFORE signing, the root cause of #232 outbound DKIM failures).

Inbound flow

External MTA → local mailbox.




▸ smtpd_milters chain:
1. OpenDKIM primary inet:127.0.0.1:8891 (verify sender's DKIM)
2. OpenDMARC inet:hermes_dmarc:54321 (verify DMARC alignment)
3. body_milter inet:hermes_body_milter:8893 (External Banner)
▸ content_filter = amavis:[hermes_mail_filter]:10021
smtp
hermes_mail_filter ▸ :10021 (amavisd-new — main filter lane)
▸ SpamAssassin scoring (sees body, computes its own DKIM_INVALID
since body_milter already injected banner)
▸ ClamAV virus scan
▸ Policy / quarantine decisions
▸ $forward_method = smtp:hermes_ciphermail:25
smtp
hermes_ciphermail ▸ :25 (encryption gateway)
▸ Encryption-mode decisions
▸ MIME rebuild (always — not byte-stable across this hop)
▸ Re-injects to → smtp:hermes_postfix_dkim:10026
smtp
hermes_postfix_dkim ▸ :10026 (postfix smtpd, re-injection)
▸ smtpd_milters = inet:localhost:8892,inet:hermes_openarc:8893
1. OpenDKIM sign-only (s mode) #232
2. OpenARC seal #229
For INBOUND mail, the From: domain is NOT in the local KeyTable.
→ sign-only instance does nothing (no key match → no header added)
→ critically: it also does NOT verify, so no Authentication-Results
"dkim=fail" header gets written against the body-modified message.
OpenARC then seals the FINAL body, recording the A-R header that
OpenDKIM-primary + OpenDMARC wrote back at :25 (which still says



Inbound paths that diverge

- **Relay-only domains** (in `transport_maps`) skip Dovecot and `smtp:`-forward to the next-hop MX listed in `/etc/postfix/transport`.
- **Quarantined / virus / bad-header** mail is held by amavis at `:10021` (final destinies: `D_BOUNCE` virus/banned, `D_DISCARD` spam/bad_header per `50-user` config).
- **Whitelisted senders** bypass via `BYPASSALLCHECKS` at `:10030`.

Architectural principle: Hermes is the auth boundary (#229)

Hermes is the authoritative auth / security boundary for every domain it relays for. Inbound auth checks (DKIM, SPF, DMARC, ARC verify, spam, virus) happen at Hermes. Body modifications (External Sender Banner, disclaimer, signature insertion, encryption) also happen at Hermes. **Customer downstream mail servers (the relay-target MX) must be configured to trust Hermes implicitly:** allowlist Hermes by IP or hostname, accept forwarded mail without re-running DKIM / SPF / DMARC / ARC checks. This is the same deployment model Mimecast, Proofpoint, and Barracuda customers use — the SEG IS the trust boundary.

When the inbound message arrives carrying an upstream `ARC-Seal:` header (M365, Workspace, Mimecast, Proofpoint, Exclaimer, etc.) and Hermes modifies the body (banner, disclaimer), the

upstream chain's body hash is invalidated. OpenARC at `:10026` honestly records `cv=fail` on Hermes's own seal because it can no longer validate the upstream chain against the modified body. The original sender's `DKIM-Signature` body hash is also invalidated.

This is by design and is not a Hermes problem. A correctly-configured customer downstream MX is allowlisting Hermes and not re-checking auth on forwarded mail; the `cv=fail` and broken DKIM signal never gates delivery. If a customer's downstream MX is doing redundant auth checks on mail Hermes forwards, that's a misconfiguration on the customer's end — the fix is to allowlist Hermes there, not to silence Hermes here.

For external receivers Hermes does NOT have a trust relationship with (third-party MXes encountered via sieve redirect or alias forwarding, should those leak past the same-domain validation), the cv=fail and broken DKIM signals do reach a non-trusting receiver. That's why sieve redirects from the user portal are validated to require a same-domain target (see `inc/sieve_user_rule_actions.cfm`) — keeping forwarded mail inside Hermes's auth boundary. Aliases configured by admins are constrained to internal Hermes mailboxes by the existing CFML check in `inc/add_mailbox_alias_action.cfm`.

Why not lift the chain by stripping the upstream ARC?

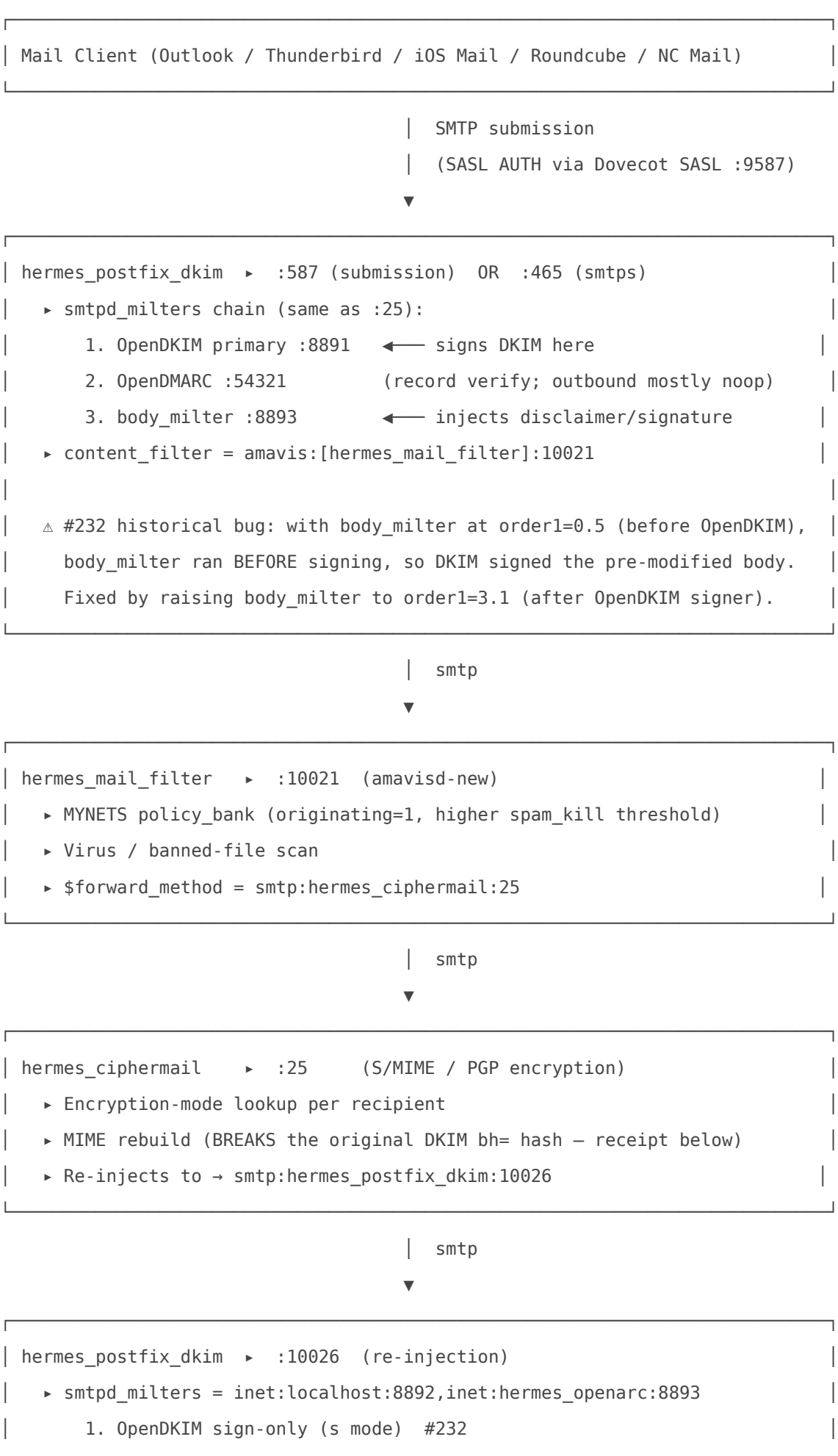
`cv=fail` is honest — Hermes correctly admits the chain we received no longer body-validates against the message we're about to send. The verifier walks the chain backward and recomputes the upstream `i=1` body hash against the **current** body, so stripping our `i=2` admission does not repair anything. The only mechanisms that could restore trust for body-modifying gateways are:

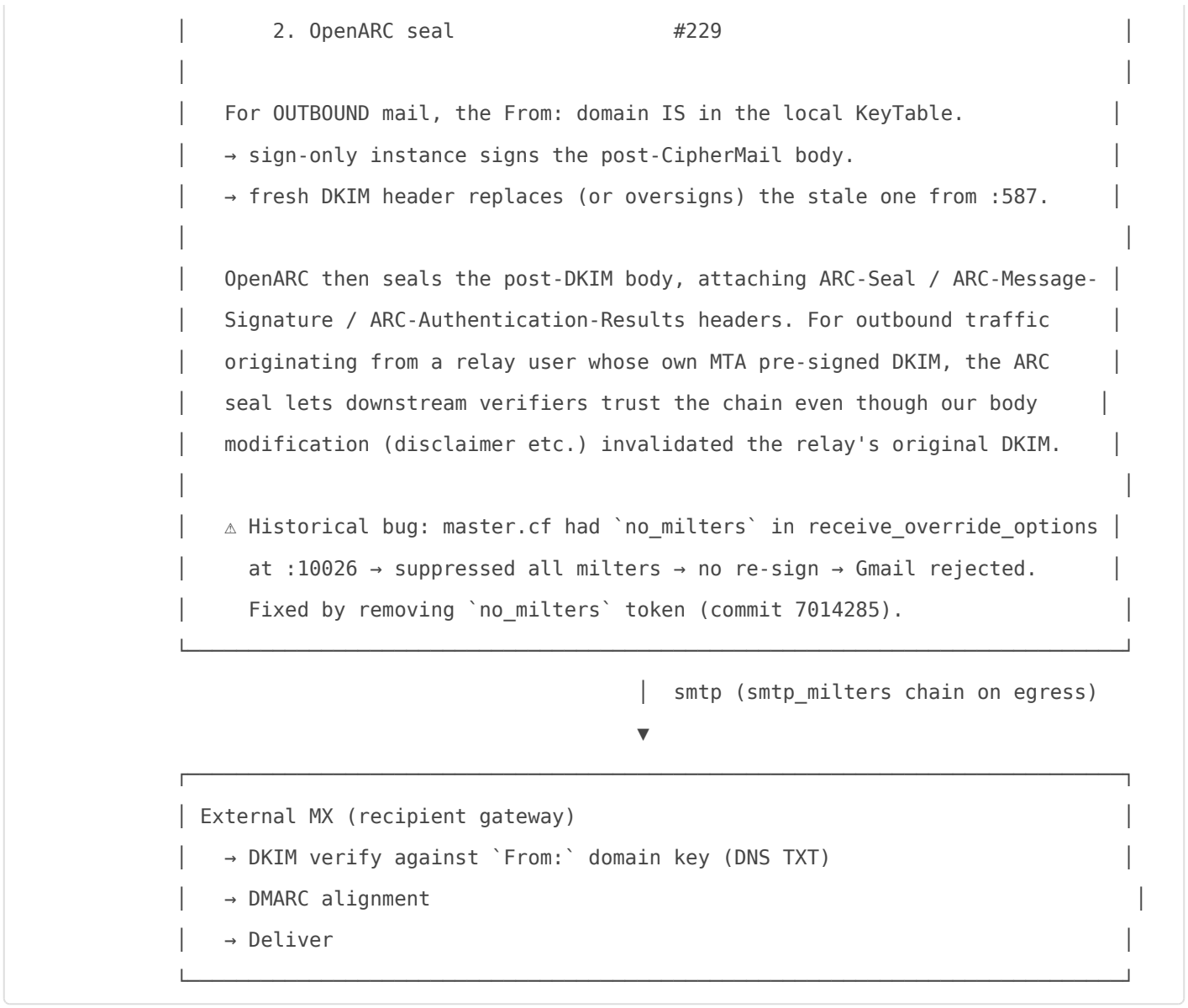
1. **Receiver-side trust configuration** — Microsoft 365's "Trusted ARC Sealers" feature, Gmail's internal trust list, etc. Useful when forwarding to receivers OTHER than the customer's own MX. See the Trusted ARC Sealers — M365 guide for cross-org scenarios.
2. **Don't modify the body** — defeats the purpose of having body modification features.

Multi-instance OpenARC (separate verify-only + sign-only daemons) does **not** help: OpenARC v1.3.0's sign-only mode re-validates the chain at sealing time and ignores AAR cached by the verify instance. This was empirically tested on DEV on 2026-05-14; see the closed #229 discussion.

Outbound flow

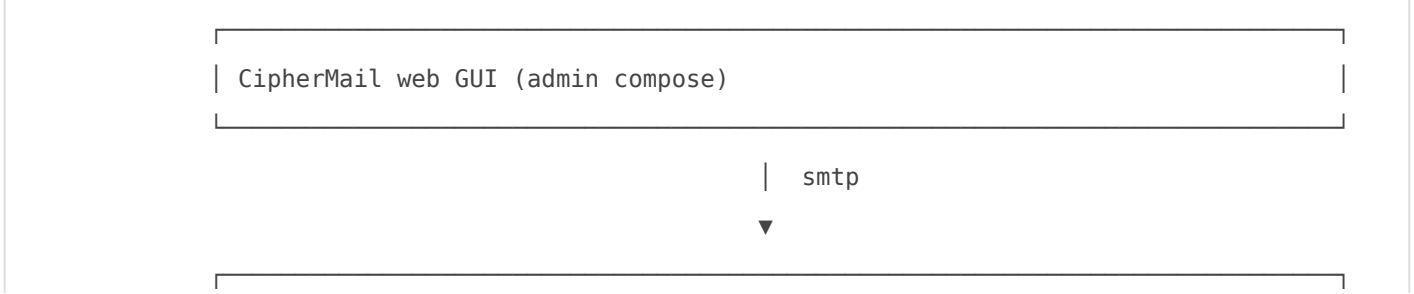
Authenticated local user → external recipient.

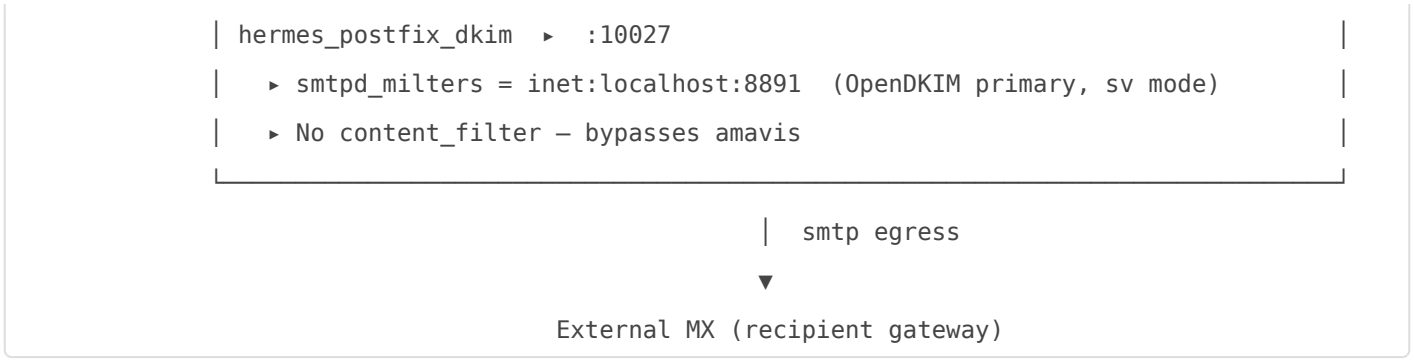

Mail Client (Outlook / Thunderbird / iOS Mail / Roundcube / NC Mail)
SMTP submission
(SASL AUTH via Dovecot SASL :9587)
hermes_postfix_dkim ▸ :587 (submission) OR :465 (smtps)
▸ smtpd_milters chain (same as :25):
1. OpenDKIM primary :8891 ◀── signs DKIM here
2. OpenDMARC :54321 (record verify; outbound mostly noop)
3. body_milter :8893 ◀── injects disclaimer/signature
▸ content_filter = amavis:[hermes_mail_filter]:10021
⚠ #232 historical bug: with body_milter at order1=0.5 (before OpenDKIM),
body_milter ran BEFORE signing, so DKIM signed the pre-modified body.
Fixed by raising body_milter to order1=3.1 (after OpenDKIM signer).
smtp
hermes_mail_filter ▸ :10021 (amavisd-new)
▸ MYNETS policy_bank (originating=1, higher spam_kill threshold)
▸ Virus / banned-file scan
▸ $forward_method = smtp:hermes_ciphermail:25
smtp
hermes_ciphermail ▸ :25 (S/MIME / PGP encryption)
▸ Encryption-mode lookup per recipient
▸ MIME rebuild (BREAKS the original DKIM bh= hash — receipt below)
▸ Re-injects to → smtp:hermes_postfix_dkim:10026
smtp
hermes_postfix_dkim ▸ :10026 (re-injection)
▸ smtpd_milters = inet:localhost:8892,inet:hermes_openarc:8893
1. OpenDKIM sign-only (s mode) #232



CipherMail web-GUI originated mail

When admins send mail directly from CipherMail's web GUI (rare), it enters postfix at `:10027`, bypasses amavis content filtering entirely, and is signed by the **primary** OpenDKIM (`:8891`) — not the sign-only instance — because this path doesn't traverse any body-modification hop and the standard sv-mode instance is appropriate.





Why two OpenDKIM instances? (#232 architecture decision)

A single OpenDKIM instance in sv mode (verify + sign) cannot satisfy both requirements at `:10026`:

Requirement	Default sv instance at :8891	Sign-only instance at :8892
Verify inbound at `:25`	✅ does this	❌ wouldn't (correctly)
Sign outbound at `:587` / `:465`	✅ does this	❌ wouldn't (correctly)
Sign outbound re-inject at `:10026` (post-CipherMail body rebuild)	❌ would sign	✅ signs
Skip verify on inbound re-inject at `:10026` (post-body-milter banner)	❌ verifies → `dkim=fail` Authentication-Results	✅ never verifies

OpenDKIM has no per-port mode override, and `InternalHosts` / `ExternalIgnoreList` control signing-vs-verification only by sender IP — not by smtpd inet service. Per OpenDKIM's own project guidance, running a second daemon with a different config file is the supported way to get differential behavior on a single host.

The sign-only instance ignores inbound mail naturally: its `KeyTable` and `SigningTable` only contain entries for local domains, so inbound mail (where the From: domain matches no local key) is a no-op pass-through — it neither signs nor adds Authentication-Results.

Receipts (forensic proof captured during #232 diagnosis)

Symptom	Evidence	Root cause
Gmail rejected outbound from `tina@getwithme.com`	bounced `.eml` had `bh=z0Xb...` in DKIM header but body hashed to `bh=vJCS...`	CipherMail MIME rebuild after `:587` DKIM sign; `:10026` had `no_milters` so no re-sign
Inbound `dkim=fail` only when External Banner enabled	Same gmail→tina test: banner-on `dkim=fail`, banner-off `dkim=pass`. CipherMail held constant.	body_milter modifying body before the next milter hop saw it; primary OpenDKIM at `:10026` re-verified the modified body
Spam score ~+1.3 on banner-injected inbound	`DKIM_INVALID=0.1` + `NML_ADSP_CUSTOM_MED=0.9` when banner on; `DKIM_VALID*=-0.3` when banner off	amavis runs its own SpamAssassin DKIM check on the post-body-milter body. **Not fixed by multi-instance OpenDKIM** — separate problem
Downstream forwarder DMARC fail	Recipient forwards to gmail; gmail re-verifies original sender DKIM against modified body → fail	Solved by ARC sealing (#229)

Open follow-ups that touch this flow

- **#228** External Sender Banner re-enable — blocked on this diagram's `:10026` sign-only wiring landing.
- **#229** ARC sealing at perimeter — required so downstream forwarders trust Hermes' original-sender verdict.
- **amavis SpamAssassin DKIM scoring** (separate issue, not yet filed) — options A/B/C documented in the #232 handoff doc.
- **Dead-weight config-file audit** — `Docker/postfix_dkim/config/`, `Docker/mail_filter/config/`, `Docker/opendmarc/config/` are shadowed by volume mounts at runtime; ~85 files to consolidate or relocate.
- **Install-template drift** — `config/hermes/opt/hermes/conf_files/master.cf` still has the pre-#232 `no_milters` token on `:10026` and `smtpd_milters=:8891` for `:10026`. Fresh installs would regress until this template is brought into line with the active runtime config.