System Certificates

System Certificates

Admin path: **System > System Certificates** (`view_system_certificates.cfm`, `inc/cert_action.cfm`, `inc/import_certificate.cfm`, `inc/acme_request_certificate.cfm`, `inc/acme_request_san_certificate.cfm`, `inc/delete_system_certificate.cfm`, `inc/parse_certificate_details.cfm`, `inc/get_system_cert_ids.cfm`, `inc/get_active_cert_paths.cfm`).

This is the **canonical certificate store** for Hermes. Every X.509 certificate the gateway presents to the outside world is registered as a row in `system_certificates` and selected by ID from one of the binding pages: Console Settings (web console), SMTP TLS Settings (Postfix SMTP banner), Email Server > Settings (Dovecot IMAP/POP/Submission), and SAN Management (per-mailbox-domain autodiscover/autoconfig).

The page itself is purely a CRUD store plus a CSR generator and the Let's Encrypt (ACME) integration. It does **not** bind certs to services — that happens on the consuming pages, each of which writes to its own row in `parameters2`.

Where certificate files live

The store has two ingest paths plus a system-managed placeholder. Each lays down files in a different directory tree.

`type`	On-disk pattern	Source
`Imported`	`/opt/hermes/ssl/<file_name>_hermes.pem` (leaf), `.key`, `.chain.pem`, `.bundle.pem` (leaf + chain concatenated)	**Import Certificate** modal or **Generate CSR** → external CA → import
`Acme`	`/etc/letsencrypt/live/<file_name>/{fullchain,cert,privkey,chain}.pem`	**Request ACME Certificate** modal; renewals via Ofelia-scheduled certbot runs

`type`	On-disk pattern	Source
`Imported` (system)	`/opt/hermes/ssl/bootstrap_hermes.{bundle.pem,key,...}` (Docker fresh installs); `/etc/ssl/{certs,private}/ssl-cert-snakeoil.{pem,key}` (legacy non-Docker)	Installer (`install_hermes_docker.sh`) or Ubuntu `ssl-cert` package

The `bootstrap` cert is a self-signed snakeoil that ships with every fresh Docker install — Hermes needs **something** to bind to before the admin imports a real cert. It is reserved as a placeholder for newly-added mailbox domains; consumers that actually need a publicly-trusted cert (SMTP TLS, the console) refuse to bind to it (see SMTP TLS Settings § Selecting a certificate).

The `system` column and the SYSTEM badge

The `system_certificates.system` column (added by issue #252) is a boolean flag marking install-generated rows. The UI surfaces this two ways:

Surface	Behavior when `system = 1`
SYSTEM badge next to the friendly name	Rendered as a gray pill in the Name column
Delete button	Disabled with a tooltip ("System-managed certificate — cannot be deleted. Used as a placeholder when binding mailbox domains before a real cert is imported.")

The delete-protection gate lives in `cert_action.cfm` and re-checks `system = 1` server-side so a crafted POST cannot bypass the disabled button.

> **Legacy vs Docker file_name.** Fresh Docker installs have `file_name = 'bootstrap'`. Legacy non-Docker installs that survived a migration have `file_name = 'ssl-cert-snakeoil'` (from the Ubuntu `ssl-cert` package). Both are flagged `system = 1` on installs where the column exists. The `inc/get_system_cert_ids.cfm` helper resolves the row IDs at runtime — code that needs to know "is this a system cert" reads from the helper, never from a hardcoded `id = 1`. This is the only correct gating signal; `version_no = 'Docker'` does **not** tell you which file_name pattern applies because both DEV (Docker, legacy install vintage) and Test (Docker, fresh install) report the same version string.

Cert path resolver — `get_active_cert_paths.cfm`

Most consumers don't want the row ID — they want the actual on-disk paths to pass to `nginx ssl_certificate`, `openssl cms -sign`, Postfix's `smtpd_tls_cert_file`, etc. The path layout differs between Imported (`/opt/hermes/ssl/...`) and ACME (`/etc/letsencrypt/live/.../...`), and the same logical name maps to different files for different consumers (`fullchain.pem` for nginx vs `cert.pem` for openssl signer).

`inc/get_active_cert_paths.cfm` is the single place that knows this layout. It reads the active console certificate from `parameters2`, joins to `system_certificates`, and writes six caller-visible variables:

Variable	Purpose
`hermesCertType`	`"Imported"`, `"Acme"`, or `"Snakeoil"`
`hermesCertIsSnakeoil`	`true` when no real cert is bound (signing callers must skip)
`hermesCertNginxPath`	Cert for `nginx ssl_certificate` (bundle for Imported, fullchain for Acme)
`hermesCertKeyPath`	Private key
`hermesCertSignerPath`	Leaf cert only — for `openssl cms -sign`
`hermesCertChainPath`	Intermediates only — for `openssl cms -sign -certfile`

Any new code that touches certificate files should `cfinclude` this helper rather than reinventing the path arithmetic. The legacy hardcoded fallback (`/etc/ssl/certs/ssl-cert-snakeoil.pem`) was removed in #251 because the minimal Docker container doesn't have the `ssl-cert` package and nginx crashed with `BIO_new_file` errors on the missing file.

Three ingest paths

1. Request ACME Certificate

The **Request ACME Certificate** button issues a Let's Encrypt cert via an ephemeral certbot container. Available in all editions.

```
Admin clicks Request -> view_system_certificates.cfm action=requestacme
   -> inc/acme_request_certificate.cfm
```

```
docker run --rm --name hermes_certbot --network host \
  -v <repo>/config/hermes/var/www/html:/var/www/certbot \
  -v <repo>/config/certbot/conf:/etc/letsencrypt \
  -v <repo>/config/certbot/logs:/var/log \
  certbot/certbot:latest \
  certonly --webroot --webroot-path /var/www/certbot \
  --email <admin> --agree-tos --no-eff-email \
  [--dry-run]   # staging mode
  -d <domain>
```

- **Staging** mode adds `--dry-run` and never lands a real cert. Always test with Staging first to confirm DNS + ports 80/443 work; Let's Encrypt's production rate limits will lock the domain out for a week if you burn through them with broken HTTP-01 challenges.
- The webroot is mounted to `/var/www/certbot` so certbot can write the challenge file where the live nginx vhost expects it.
- Certs land in `config/certbot/conf/live/<domain>/` (bind-mounted to `/etc/letsencrypt/live/<domain>/` in the commandbox container).
- Renewals are driven by Ofelia (Scheduled Tasks). Each renewal runs the same ephemeral certbot container with `renew`; if the renewal succeeds, dependent services (nginx, Postfix, Dovecot) reload to pick up the new files.
- ACME certs cannot be renewed manually from this page — the row exists for binding and deletion only; renewals are scheduled and silent.

Per-mailbox-domain ACME SAN certs (autoconfig + autodiscover + custom prefixes) use a separate code path (`inc/acme_request_san_certificate.cfm`) wired to SAN Management. Both paths land rows in the same `system_certificates` table.

The ACME account

certbot registers an account the first time it talks to a given ACME endpoint, and registering means accepting the Terms of Service. Both request paths pass `--agree-tos --non-interactive --no-eff-email` so that happens without a prompt, because neither runs on a terminal that could answer one.

The automated SAN path did not always pass them. Without `--agree-tos` certbot stops to ask, finds no terminal, and exits on `EOFError` before it attempts a challenge, so the path only ever worked on a gateway where somebody had already requested a certificate by hand from this page. Adding a mailbox domain is normally the first ACME action a new install performs, so on a fresh install it failed every time.

The contact address comes from **Admin E-mail** in System Settings, and is passed only when it is a real address. It ships as a placeholder and nothing outside the console ever writes it, so if it has not been changed the account is registered with `--register-unsafely-without-email` instead. That is supported by Let's Encrypt, and it beats binding the account to an address that does not exist,

because an ACME account's contact address is awkward to correct afterwards. **Set a real Admin E-mail if you want Let's Encrypt's expiry warnings.**

Failures are recorded, not swallowed

certbot writes its failures to stderr. Both request paths capture stdout and stderr together, so a failure comes back as a value rather than an exception.

For the SAN path the reason is written to `mailbox_sans.dns_result_msg` and shown in the **Mailbox SAN Validation** sub-table below, so a certificate that will not issue tells you why on this page. The scheduled validator then retries on its next run.

Previously stderr had nowhere to go, so any certbot failure surfaced as an internal error: the scheduled job stopped where it stood, wrote nothing to the SAN rows, and mailed the administrator. A certbot success was the only outcome the code could observe.

2. Import Certificate

For certs issued by any CA other than Let's Encrypt (commercial CA, internal PKI, etc.). The admin pastes three PEM blobs in the **Import Certificate** modal:

Field	Contents
Certificate (PEM)	Leaf cert between `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`
Unencrypted Key (PEM)	Private key — must be **unencrypted** (no passphrase). Encrypted keys are rejected because nginx / Postfix cannot prompt for a passphrase at startup.
Root & Intermediate CA Certificates (PEM)	Chain — root + intermediates concatenated, leaf-omitted

On save, Hermes writes four files under `/opt/hermes/ssl/`:

```
<file_name>_hermes.pem            (leaf only)
<file_name>_hermes.key            (private key)
<file_name>_hermes.chain.pem      (CA chain, no leaf)
<file_name>_hermes.bundle.pem     (leaf + chain — for nginx ssl_certificate)
```

`<file_name>` is derived from the friendly name with special characters sanitized. The row is inserted with `type = 'Imported'` and the extracted Subject/Issuer/Serial/Fingerprint cached in the table for the expandable row preview.

3. Generate CSR

For admins who want to use their own CA but don't have a key+CSR yet. The modal collects DN fields (Country, State, Locality, Organization, Department) plus a **Certificate purpose** radio toggle that drives the rest of the form:

Purpose	CN source	SANs
Server certificate (single-name DV, ~$10/yr)	Admin enters Common Name field directly	Admin-entered FQDNs only
Mailbox certificate (SAN / UCC, $50–$200/yr)	Auto-derived as `<first-prefix>.<mailbox_domain>` matching the auto ACME path's first-`-d`-flag behavior	Mandatory: `autoconfig.<domain>`, `autodiscover.<domain>`, plus every prefix from `additional_sans`. Additional admin entries auto-expand bare prefixes against the mailbox domain.

Smart default: if `mailbox_domains` has any rows, the modal defaults to **Mailbox**; otherwise it defaults to **Server**. The page-level "Choosing the Right Certificate Type" card above the table walks the admin through the cost difference and the "a basic DV cert will not work for mailboxes" trap.

On submit, Hermes generates a 2048- or 4096-bit RSA key + matching CSR and bundles them into a `.rar` archive at `/opt/hermes/tmp/<token>_csr_key.rar`. The CSR-pending state is then surfaced as a **persistent callout** at the top of the page (added by #249) — the **Download CSR** button stays visible across page reloads until the admin clicks **Discard**. Submit the CSR to the chosen CA, receive the signed cert + chain, then come back and use **Import Certificate** (steps 1 above) to register it. The private key in the `.rar` is what you'll need for the import.

> **The CSR private key never leaves Hermes until the admin downloads the bundle.** If the admin clicks Discard without downloading, the key is gone — there is no recovery. The Discard button warns about this; the persistent callout pattern (#249) was introduced because the one-shot download button that used to live inside the success alert was easy to miss on a page reload.

Service binding cross-reference

The certificate-store rows are referenced from four service-binding locations. Each location keeps its **own** copy of the cert ID — there is no cascading delete, so the deletion guard (next section) walks all four before allowing a row to be removed.

Service	Where the binding lives	Set on page
Console (admin, user portal, NC, Ciphermail)	`parameters2.parameter = 'console.certificate', module = 'console'`	Console Settings

Service	Where the binding lives	Set on page
SMTP (Postfix `smtpd_tls_cert_file`)	`parameters2.parameter = 'smtp.certificate', module = 'certificates'`	SMTP TLS Settings
Webmail (Dovecot IMAP/POP)	`parameters2.parameter = 'mail.certificate', module = 'certificates'`	Email Server > Settings
Mailbox SAN (per-domain autodiscover/autoconfig)	`mailbox_domains.mailbox_certificate` (multiple rows possible)	Email Server > Domains, SAN Management

The page renders four YES/NO columns (Console / SMTP / Webmail / Mailbox SAN) so an admin can see at a glance which services a given cert is in use by.

Deletion guard

`inc/delete_system_certificate.cfm` walks every consumer before allowing a delete:

```
1. system column flag         -> system-managed, refuse
2. parameters2 console.certificate    -> assigned to Web Service, refuse
3. parameters2 smtp.certificate       -> assigned to SMTP Service, refuse
4. parameters2 mail.certificate       -> assigned to Mail Service, refuse
5. (mailbox_domains.mailbox_certificate check is in cert_action.cfm)
6. -> DELETE FROM system_certificates WHERE id = ?
7.    plus filesystem cleanup:
        Imported: rm /opt/hermes/ssl/<file_name>_hermes.{pem,key,chain.pem,bundle.pem}
        Acme:     docker run --rm certbot/certbot:latest delete --cert-name <file_name>
                  + DELETE FROM mailbox_sans WHERE certificate = ?
```

The guard is **stop-on-first-match** with a specific error message per case so the admin knows which binding is blocking the delete and where to go to unbind. There is no "force delete" — the only way past the guard is to unbind on the consuming page first.

Deleting used to fail after having deleted

Step 7's SAN cleanup named `mailbox_domains_sans`, a table that has never existed in this schema. The real table is `mailbox_sans` and the column is `certificate`.

Because the cleanup ran **after** the row had already been removed, the delete appeared to fail while having actually happened: the admin saw an error, the certificate was gone from the table, and the SAN rows that referenced it were left behind pointing at an id that no longer resolved. Those stranded rows are what the reconciliation described in

SAN Management now cleans up on upgrade.

Worth knowing why an FK-based cleanup could not find them all. `system_certificates` is InnoDB, so its auto-increment is recalculated as `MAX(id) + 1` after a restart, which means a deleted id gets reused. A stranded row then resolves to a perfectly valid but wrong certificate, and no orphan query can see it. Catching that class needed a filesystem check rather than a SQL one.

Certificate downloads (gated)

Each row has an expandable details panel with **Download Certificate**, **Download Private Key**, and **Download CA Chain** buttons. By default these are **disabled** for safety (downloading a private key over a web page is a sensitive operation). To enable, set

```
ALLOW_CERT_DOWNLOAD=yes
```

in `/opt/hermes/config/security.conf` on the host filesystem. The page reads this file on every load (cached in the local request). When the toggle is off, the buttons render disabled with a tooltip telling the admin where to set the flag.

Downloads are streamed via a hidden iframe + `class="no-preloader"` pattern (standard Hermes binary-download convention) so the page's spinner overlay doesn't get stuck.

SAN validation sub-table (Pro feature)

When a row is bound to one or more entries in `mailbox_sans` (autodiscover/autoconfig/custom subdomains for a mailbox domain), the expanded details panel includes a **Mailbox SAN Validation** sub-table showing IP-resolve and DNS-resolve status for each SAN. This is populated by the SAN Management validator and is read-only here — it answers "do all the SANs on this cert actually resolve to this server?" at a glance.

Failure semantics

What breaks	What happens

CSR field validation (Country != 2 chars, bad CN chars, etc.)	`session.m` set with the specific error, `cflocation` back to the page, no file/DB writes
Mailbox CSR with empty `additional_sans` table	Refused with "No SAN prefixes configured in SAN Management. Cannot generate a mailbox certificate without at least autoconfig + autodiscover."
ACME staging dry-run fails (DNS, port 80, rate limit)	Raw certbot stderr surfaced in the error alert; no DB row added
ACME production fails	Same as staging — error alert with raw stderr
Import with mismatched key + cert	The import script's openssl-modulus check fails; error alert with detail
Delete blocked by binding	"The Certificate you are attempting to delete is assigned to the X Service" — admin must unbind first on the consuming page
`certbot delete` fails on ACME row	DB row kept, error surfaced; manual cleanup of the `/etc/letsencrypt/live/<name>/` tree may be needed

Files and containers touched

Path	Owner	Role
`config/hermes/var/www/html/admin/2/view_system_certificates.cfm`	`hermes_commandbox`	Page
`config/hermes/var/www/html/admin/2/inc/cert_action.cfm`	`hermes_commandbox`	Action router (CSR, import, ACME, delete, discard)
`config/hermes/var/www/html/admin/2/inc/acme_request_certificate.cfm`	`hermes_commandbox`	Single-domain ACME via certbot container
`config/hermes/var/www/html/admin/2/inc/acme_request_san_certificate.cfm`	`hermes_commandbox`	Multi-SAN ACME (mailbox certs)
`config/hermes/var/www/html/admin/2/inc/import_certificate.cfm`	`hermes_commandbox`	PEM paste-in handler
`config/hermes/var/www/html/admin/2/inc/delete_system_certificate.cfm`	`hermes_commandbox`	Deletion guard + filesystem cleanup
`config/hermes/var/www/html/admin/2/inc/parse_certificate_details.cfm`	`hermes_commandbox`	Single `openssl x509` parse for subject/issuer/SAN/etc.
`config/hermes/var/www/html/admin/2/inc/get_system_cert_ids.cfm`	`hermes_commandbox`	Resolver — which rows are system-managed
`config/hermes/var/www/html/admin/2/inc/get_active_cert_paths.cfm`	`hermes_commandbox`	Resolver — on-disk paths for the active console cert
`/opt/hermes/ssl/`	`hermes_commandbox` (bind-mounted)	Imported cert files
`/etc/letsencrypt/live/<domain>/`	`hermes_commandbox` (bind-mounted from `config/certbot/conf/`)	ACME cert files

Path	Owner	Role
`/opt/hermes/tmp/<token>_csr_key.rar`	`hermes_commandbox`	Pending CSR bundle
`/opt/hermes/config/security.conf`	host filesystem	`ALLOW_CERT_DOWNLOAD` toggle
`system_certificates` table	`hermes_db_server` (`hermes` DB)	The canonical store
`certbot/certbot:latest` image	docker.io	Pulled on demand; ephemeral per request

Every certbot invocation is `docker run --rm` against the public `certbot/certbot:latest` image — Hermes never runs certbot directly on the host. The container shares the host network (`--network host`) so Let's Encrypt's HTTP-01 challenge can reach port 80 on the public IP.

Related

- SMTP TLS Settings — bind a System Certificate to Postfix SMTP TLS
- Console Settings — bind a System Certificate to the web console (nginx) and its hardening toggles
- Server Setup — Mail Server Hostname; should match the CN/SAN on the SMTP cert for STARTTLS verification
- Authentication Settings — Authelia; uses the console cert via its nginx-fronted vhost
- LDAP RemoteAuth — separate CA store at `/opt/hermes/certs/remoteauth/` for upstream LDAP; not a System Certificate
- SAN Management — per-mailbox-domain SAN prefixes that drive mailbox-cert CSR + ACME SAN issuance
- Intrusion Prevention — Fail2ban; not cert-related but documents the same nginx-restart cascade pattern this page avoids by not regenerating any nginx config
- Admin Console Firewall — IP allowlist for the console; layered above the TLS termination this page's certs drive

---

Revision #64
Created 2026-05-31 12:52:03 UTC by Dino Edwards
Updated 2026-08-22 11:48:30 UTC by Dino Edwards