Encryption

Encryption Settings

Encryption Settings

Admin path: **Encryption > Encryption Settings** (`view_encryption_settings.cfm`, `inc/edit_encryption_settings.sh`).

This is the **global Ciphermail policy page** — a thin CFML wrapper over a fixed set of CipherMail "global" properties that govern subject-based encryption triggering, the PDF reply-sender identity, and three internal shared secrets used by the Secure Email Portal back-channel. Per-recipient policy lives on External Recipients; CA / S/MIME issuance lives on Internal CA. This page is the small set of gateway-wide toggles that affect every encrypted send.

> **Important: not a full encryption-mode picker.** The page does NOT pick "always encrypt vs opportunistic vs off" at the system level — CipherMail does that per-recipient via the user's `user.encryptMode` property (set when the admin creates the recipient on External Recipients). The only system-wide opt-in/opt-out exposed here is the **Subject Trigger** mechanism: whether `[encrypt]` (or whatever keyword is configured) in a message subject promotes that one message to an encryption attempt.

What the page persists

Every setting on the page is stored twice: once in the Hermes `encryption_settings` table (so the UI can re-render the current state on next load) and once in CipherMail's own global property store via the `CLITool --set-property ... --global` invocation. The two are kept in sync by re-running the full apply script on every save.

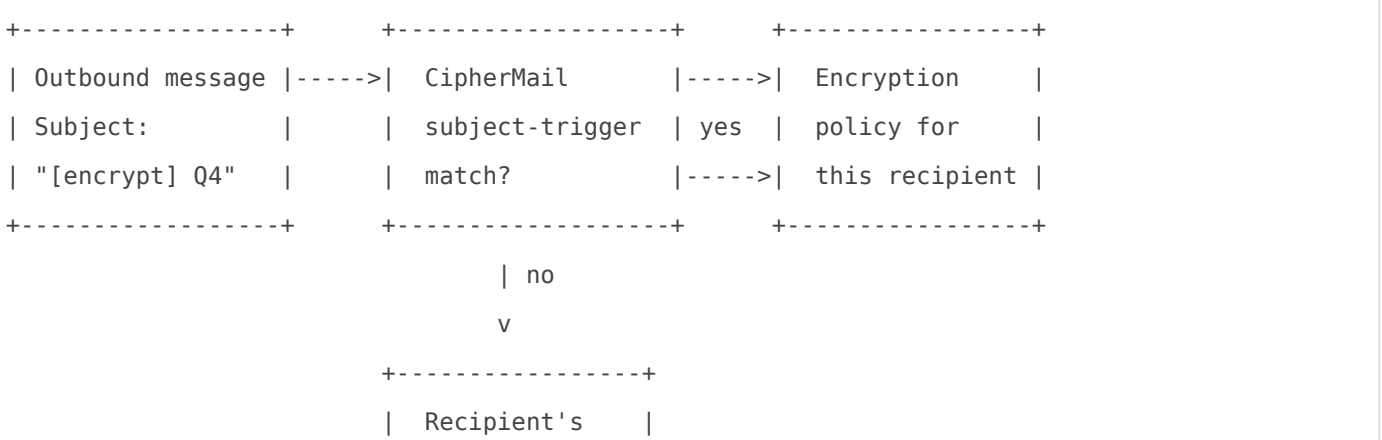
Field	`encryption_settings.property`	CipherMail property	Notes
Trigger Encryption by Subject (Enabled / Disabled)	`user.subjectTriggerEnabled`	`user.subjectTriggerEnabled`	`true` / `false` string
Subject Trigger Keyword	`user.subjectTrigger`	`user.subjectTrigger`	Free text, e.g. `[encrypt]`

Field	`encryption_settings.property`	CipherMail property	Notes
Remove Trigger After Encryption (Yes / No)	`user.subjectTriggerRemovePattern`	`user.subjectTriggerRemovePattern`	When `true`, the keyword is stripped before the recipient sees the message
PDF Reply Sender Email	`user.pdf.replySender`	`user.pdf.replySender`	Email validated as `IsValid("email", ...)` before save
Portal URL (read-only, derived)	`user.portal.baseURL`	`user.portal.baseURL`	Built at save time as `https://<console.host>/web/portal` — NOT directly editable on this page; change Console Host on System Settings
Server Secret Keyword	`user.serverSecret`	`user.serverSecret` (encrypted)	64-char auto-generated, masked in UI
Client Secret Keyword	`user.clientSecret`	`user.clientSecret` (encrypted)	64-char auto-generated, masked in UI
Mail Secret Keyword	`user.systemMailSecret`	`user.systemMailSecret` (encrypted)	64-char auto-generated, masked in UI

Additionally, the script always sets `user.otpEnabled = true --global` on every save — a fixed override that ensures CipherMail's one-time password feature is on globally regardless of any prior state.

Subject Trigger: how it actually works

When **Trigger Encryption by Subject** is enabled, CipherMail inspects each outbound message's `Subject:` header during processing:



```
                    |  user.encryptMode|
                    |  decides         |
                    +------------------+
```

Setting combination	Behavior
Trigger ENABLED + Keyword present + Recipient `user.encryptMode = allow`	Message encrypted using whichever protocol the recipient has enabled (S/MIME / PGP / PDF). If none, CipherMail falls back to its protocol-selection rules.
Trigger ENABLED + Keyword present + Recipient `user.encryptMode = mandatory`	Already always-encrypted; the keyword is redundant. If **Remove Trigger** is on, the keyword is still stripped from the visible subject.
Trigger ENABLED + Keyword NOT present + Recipient `user.encryptMode = allow`	Message sent plaintext (the recipient is configured "by subject" and the sender did not opt in).
Trigger ENABLED + Keyword NOT present + Recipient `user.encryptMode = mandatory`	Encrypted regardless (recipient policy overrides).
Trigger DISABLED	Subject line is never inspected; recipient `user.encryptMode` is the sole authority. Senders cannot opt-in per message.

Recipient `user.encryptMode` is set when the admin picks a mode (e.g. "PDF Mandatory" vs "PDF By Subject") on **Encryption > External Recipients > Create**. See External Recipients — Encryption modes.

PDF Reply Sender

When a recipient receives a PDF-encrypted message and clicks the reply link in the encrypted PDF, the response comes back to Hermes via the Secure Email Portal. The **PDF Reply Sender Email** is the `From:` address CipherMail uses when delivering that reply back to the original internal sender (and on system notifications about PDF reply activity). Operators typically set this to a monitored address like `postmaster@yourdomain.tld` or a dedicated `secure-reply@...` mailbox.

The field is validated: empty or non-email values trigger alerts `m=3` and `m=2` respectively and abort the save.

The three secret keywords

CipherMail uses three independent shared secrets to authenticate the back-channel between the encryption engine and the Secure Email Portal (`/web/portal/`). They are stored AES-encrypted in `encryption_settings.value` (using `/opt/hermes/keys/hermes.key` as the key) and pushed into CipherMail with the `--encrypt` flag so CipherMail encrypts them again with its own key.

Secret	Used by	Generated by
Server Secret (`user.serverSecret`)	CipherMail server-side validation of portal session tokens	Click the sync icon on the field; never user-entered
Client Secret (`user.clientSecret`)	Portal client-side validation handshake	Click the sync icon
Mail Secret (`user.systemMailSecret`)	Signing of system-generated email notifications (password delivery, portal invitations, etc.)	Click the sync icon

The UI masks the values to `**********<last 4 chars>` — full plaintext is never re-displayed after generation. To replace a secret, click the sync (`fa-sync-alt`) button on its row; a confirmation modal fires; on confirm Hermes:

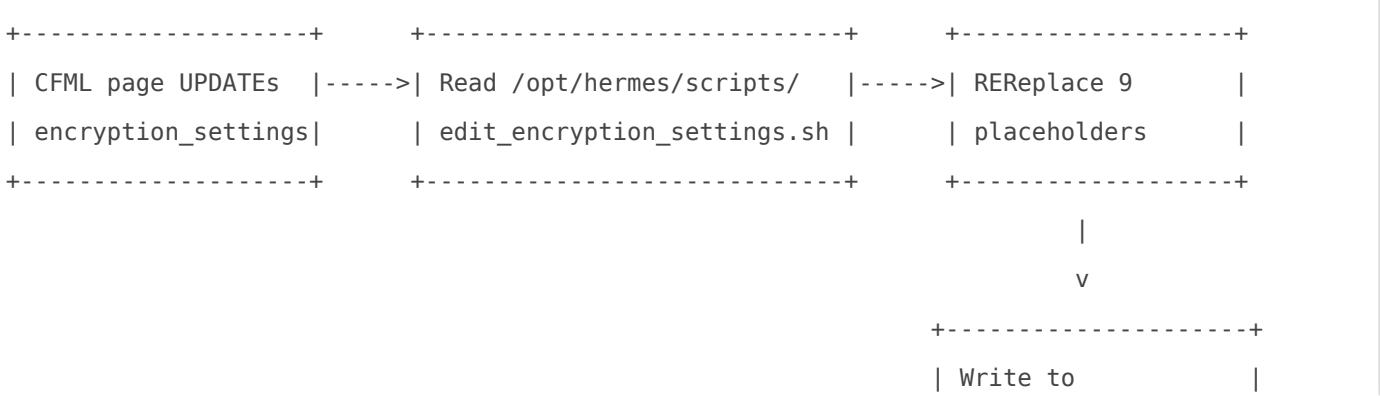
1. Generates 64 lowercase hex-ish characters by concatenating 8 rounds of the standard `customtrans3` token generator and truncating.
2. AES-encrypts that with `/opt/hermes/keys/hermes.key` and UPDATEs `encryption_settings.value` for the corresponding property.
3. Runs the full `edit_encryption_settings.sh` apply script (see below) to push **all three** secrets — plus the subject-trigger / PDF reply / portal URL settings — into CipherMail in one shot.

Rotating any one secret therefore re-applies the other two as a side-effect; in practice the values are stable across rotations because the script reads each from its already-decrypted form before writing.

Operational consequence: rotating a secret invalidates any in-flight portal sessions for that secret's role. Recipients with an active portal session may need to log in again; system notifications in transit may fail signature verification and be re-queued.

The apply pipeline

Both **Save Settings** and **Generate Secret** funnel through the same temp-script pattern documented across the Hermes admin:



```
                                    | /opt/hermes/tmp/    |
                                    | <token>_edit_...sh  |
                                    +---------------------+
                                              |
                                              v
                                    +---------------------+
                                    | chmod +x and execute|
                                    | (240s timeout) then |
                                    | delete the temp file|
                                    +---------------------+
                                              |
                                              v
                                    +---------------------+
                                    | 9 sequential        |
                                    | docker exec         |
                                    | hermes_ciphermail   |
                                    | CLITool --global    |
                                    +---------------------+
```

Placeholders substituted in the template:

Placeholder	Replaced with
`PDFREPLY-SENDER`	`user.pdf.replySender` value
`PORTAL-URL`	Derived `https://<console.host>/web/portal`
`SUBJECT-TRIGGER`	`user.subjectTrigger` value
`SUBJECT-ENABLE`	`true` / `false`
`TRIGGER-REMOVE`	`true` / `false`
`SERVER-SECRET`	Decrypted server secret (pushed with `--encrypt` so CipherMail re-encrypts)
`CLIENT-SECRET`	Decrypted client secret
`MAIL-SECRET`	Decrypted mail secret

On a CLITool execution failure the page sets `session.m_enc = 11` and surfaces "Settings saved to database but failed to apply to Ciphermail. Please check the logs." — the DB write succeeds first, so the UI state matches what the operator entered even when the CipherMail-side push fails. Re-save (with no edits) re-runs the apply script.

What's NOT on this page

Several things an operator might reasonably expect from a global "Encryption Settings" page that live elsewhere:

Expectation	Where it actually lives
Per-recipient "always encrypt vs by subject vs never"	External Recipients (`user.encryptMode` per CipherMail user)
Default cipher / algorithm selection (AES-128 vs AES-256, RSA key sizes)	CipherMail Advanced Settings (`/ciphermail/`, external link in sidebar)
Per-mailbox sign / encrypt action defaults	Email Server > Mailboxes (per-mailbox encryption action editor, `inc/edit_mailbox_encryption_action.cfm`)
TLS opportunistic vs DANE policy on outbound delivery	Email Relay > Relay Hosts and TLS Settings; this page is about message-content encryption only
Subject keyword for DLP-driven (content-based) encryption triggers	Not implemented in Hermes; CipherMail Advanced Settings can express custom DLP rules
Portal URL customization	Derived automatically from System > Console Settings (`parameters2.console.host`); editing console host updates this on next save
S/MIME signing of every outbound (gateway sign-and-forward)	CipherMail Advanced Settings; not surfaced here
Password complexity rules for the auto-generated portal / PDF passwords	Hardcoded in the modal JS on
External Recipients (16-char mixed alphanumeric)	

Body-modification interaction

The CipherMail encryption / signing pass runs **after** the `hermes_body_milter` disclaimer / signature / banner pipeline. That means PDF, S/MIME, and PGP envelopes always wrap the final body the recipient sees — including any appended disclaimer (see Disclaimers — Behavior with S/MIME, PGP, and DKIM-signed mail). The same milter-ordering rationale applies to ARC inbound sealing (see ARC Settings — Container and milter placement): the cryptographic envelope is the last thing applied so it always matches what the recipient downloads.

Container and database touch-points

Component	Container / path	Role
Page	`config/hermes/var/www/html/admin/2/view_encryption_settings.cfm` (`hermes_commandbox`)	CRUD UI + apply orchestration
Template script	`/opt/hermes/scripts/edit_encryption_settings.sh` (`hermes_commandbox` bind mount)	9-line shell with 9 placeholders
Temp scripts	`/opt/hermes/tmp/<token>_edit_encryption_settings.sh`	Substituted copy, executed once, deleted
Settings store (Hermes side)	`encryption_settings` in `hermes` DB (`hermes_db_server`)	One row per property; secrets stored AES-encrypted in `value`
Settings store (CipherMail side)	`cm_properties` in `djigzo` DB (`hermes_db_server`) — set indirectly via CLITool `--global`	CipherMail's authoritative global property store
Encryption engine	`hermes_ciphermail` (Java; CipherMail Community 5.x branded `djigzo`)	Performs S/MIME / PGP / PDF encryption at send time
Encryption key	`/opt/hermes/keys/hermes.key` (`hermes_commandbox` bind mount)	AES key used for CFML-side `encrypt()` / `decrypt()` of the three secrets
Console host source	`parameters2.console.host` in `hermes` DB	Drives the auto-derived `user.portal.baseURL`

Related

- External Recipients — per-recipient encryption modes; the page where `user.encryptMode = mandatory` vs `allow` is actually chosen
- Internal CA — where the S/MIME root CAs that mint per-recipient certs live; cross-referenced by recipient PDF / S/MIME / PGP rows on External Recipients
- PGP Key Servers — outbound key publishing list (note: publish-only, not lookup)
- Disclaimers — body-mod ordering against the CipherMail encryption pass
- ARC Settings — same milter-ordering pattern applied to inbound chain sealing
- DMARC Settings — cross-references the body-mod pipeline that also feeds DKIM signing
- **Advanced Settings** (sidebar link to `/ciphermail/`) — CipherMail's own admin UI; everything not surfaced on this page (per-protocol cipher selection, custom DLP, gateway-wide always-sign) lives there

External Recipients

External Recipients

Admin path: **Encryption > External Recipients** (`view_ext_rec_encryption.cfm`, `view_create_ext_recipient.cfm`, `view_ext_smime_certificates.cfm`, `view_ext_pgp_keyrings.cfm`, `view_ext_add_smime_cert.cfm`, `view_ext_add_pgp_keyring.cfm`, `inc/create_ext_recipient.cfm`, `inc/delete_ext_recipient.cfm`, `inc/reset_pdf_password.cfm`, `inc/reset_portal_password.cfm`).

This is the **per-counterparty encryption policy and key store** for external (non-managed) email addresses. Each row binds a single external email to one of three protocols (PDF / S/MIME / PGP) and to one of two trigger modes (Mandatory / By Subject). It is the page where the policy referenced by Encryption Settings actually takes effect — the global page chooses the **mechanism** (subject trigger keyword, shared secrets, PDF reply sender); this page chooses the **policy** for every external recipient the gateway encrypts to.

The DataTable is the master view across **both** Hermes-side metadata (`external_recipients` in the `hermes` DB) and CipherMail's own user table (`cm_users` in the `djigzo` DB), joined on email address. Rows are tagged **Admin-Configured** (explicitly created on this page, with a matching `external_recipients` row) or **Auto-Discovered** (materialized by CipherMail during message processing, no `external_recipients` row).

Schema: two tables, one view

```
            v
   Page renders Admin badge
            |
+-------------------------+
| If NO matching row,     |
| recipient is "Auto" with |
| inferred policy from    |
| cm_certificates_email /  |
| cm_keyring_email        |
+-------------------------+
```

The page never N+1's against CipherMail — three batch queries build struct lookups (`adminLookup`, `smimeLookup`, `pgpLookup`) and the row loop reads from those instead of per-row queries. That matters at any scale beyond a few hundred recipients.

`external_recipients` columns:

Column	Purpose
`id`	PK
`email`	External email address (joined to `cm_users.cm_email`)
`encryption_mode`	`pdf_mandatory` / `pdf_by_subject` / `smime_mandatory` / `smime_by_subject` / `pgp_mandatory` / `pgp_by_subject`
`pdf` / `smime` / `pgp`	Flag (1 / NULL) indicating which protocol is the active one for this recipient
`pdf_mode`	For PDF only: `static` / `random` / `backtosender`
`pdf_password`	AES-encrypted (with `/opt/hermes/keys/hermes.key`) copy of the static PDF password — for admin re-display only; CipherMail holds its own copy
`smime_mode` / `pgp_mode`	Reserved for parity; populated identically to `encryption_mode` for the matching protocol

Encryption modes

The 6 encryption modes map cleanly onto two axes (protocol × trigger):

Mode	CipherMail `user.encryptMode`	CipherMail `user.pdf.encryptionAllowed`	CipherMail `user.sMIMEEnabled`	CipherMail `user.pgp.enabled`
`pdf_mandatory`	`mandatory`	`true`	`false`	`false`
`pdf_by_subject`	`allow`	`true`	`false`	`false`

Mode	CipherMail `user.encryptMode`	CipherMail `user.pdf.encryptionAllowed`	CipherMail `user.sMIMEEnabled`	CipherMail `user.pgp.enabled`
`smime_mandatory`	`mandatory`	`false`	`true`	`false`
`smime_by_subject`	`allow`	`false`	`true`	`false`
`pgp_mandatory`	`mandatory`	`false`	`false`	`true`
`pgp_by_subject`	`allow`	`false`	`false`	`true`

"By Subject" requires **Encryption Settings > Trigger Encryption by Subject = Enabled** plus the configured keyword (default `[encrypt]`) in the message subject. See Encryption Settings — Subject Trigger for the decision tree.

PDF mode: three sub-policies

PDF encryption is the lowest-friction protocol (recipient needs only a PDF reader and a password — no certs, no keys, no portal account required up front), so it ships with three independent password-distribution sub-modes:

`pdf_mode`	How the password reaches the recipient	When to use
`random`	CipherMail auto-generates a one-time password per message and pushes it through the Secure Email Portal (`https://<console>/web/portal`); recipient self-registers on first use	Default. Best for ad-hoc / first-time external recipients
`static`	Admin sets a fixed password once (minimum 12 chars); recipient must already know it via out-of-band channel	Long-term partners who have agreed on a shared secret
`backtosender`	CipherMail generates a per-message password and emails it back to the **original internal sender** for them to relay to the recipient	Compliance scenarios where the sender must explicitly hand the password to the recipient (auditable trail)

For `backtosender`, two extra fields are configurable per recipient:

Field	Range	Purpose
Password Age (minutes)	15-240	How long the random password is valid
Password Length	16-bit / 20-bit	Bit-strength of the generated random password

Bulk vs single create

The **Create External Recipient** page (`view_create_ext_recipient.cfm`) exposes a Single / Bulk toggle:

Mode	Protocol options	Use case
Single	PDF, S/MIME, PGP (all three modes available)	One-off precise configuration including S/MIME / PGP recipients that need a cert/key uploaded afterward
Bulk	**PDF only** (Mandatory or By Subject)	Mass-onboard a list of external addresses, one per line; the UI auto-hides S/MIME and PGP because those protocols need per-recipient cert/key material that has no bulk equivalent

The bulk path validates and skips per-row (invalid format / internal domain / already-exists rows are reported but do not abort the batch); session variables `bulk_created`, `bulk_skipped`, `bulk_failed` feed a partial-success alert on return.

Both paths refuse internal domains. The check is a `COUNT(*) FROM domains WHERE domain = <recipient-domain>` — if Hermes is the authoritative MX for that domain, the recipient is a local mailbox or relay recipient, not an external recipient, and per-mailbox encryption policy belongs on Email Server > Mailboxes instead.

Auto-Discovered recipients

When CipherMail processes mail to an address it has never seen, it materializes a `cm_users` row with the global defaults. These recipients show up here with **Source = Auto** and no `external_recipients` row backing them. They:

- Use the global Subject Trigger policy (from Encryption Settings)
- Have no per-recipient password mode (PDF random is the CipherMail default)
- Display only the cert / keyring counts CipherMail actually holds
- Cannot be edited from this page (no Admin badge, no action buttons for cert / PGP / password reset) — managing them means **either** promoting them to Admin-Configured by creating an explicit row, **or** dropping into CipherMail's own admin UI at `/ciphermail/`

The Source dropdown defaults to **Admin-Configured** on page load — operators most often want to see what they explicitly configured, not the long tail of mail CipherMail has touched.

Per-row actions

The action column varies by what the recipient is configured for:

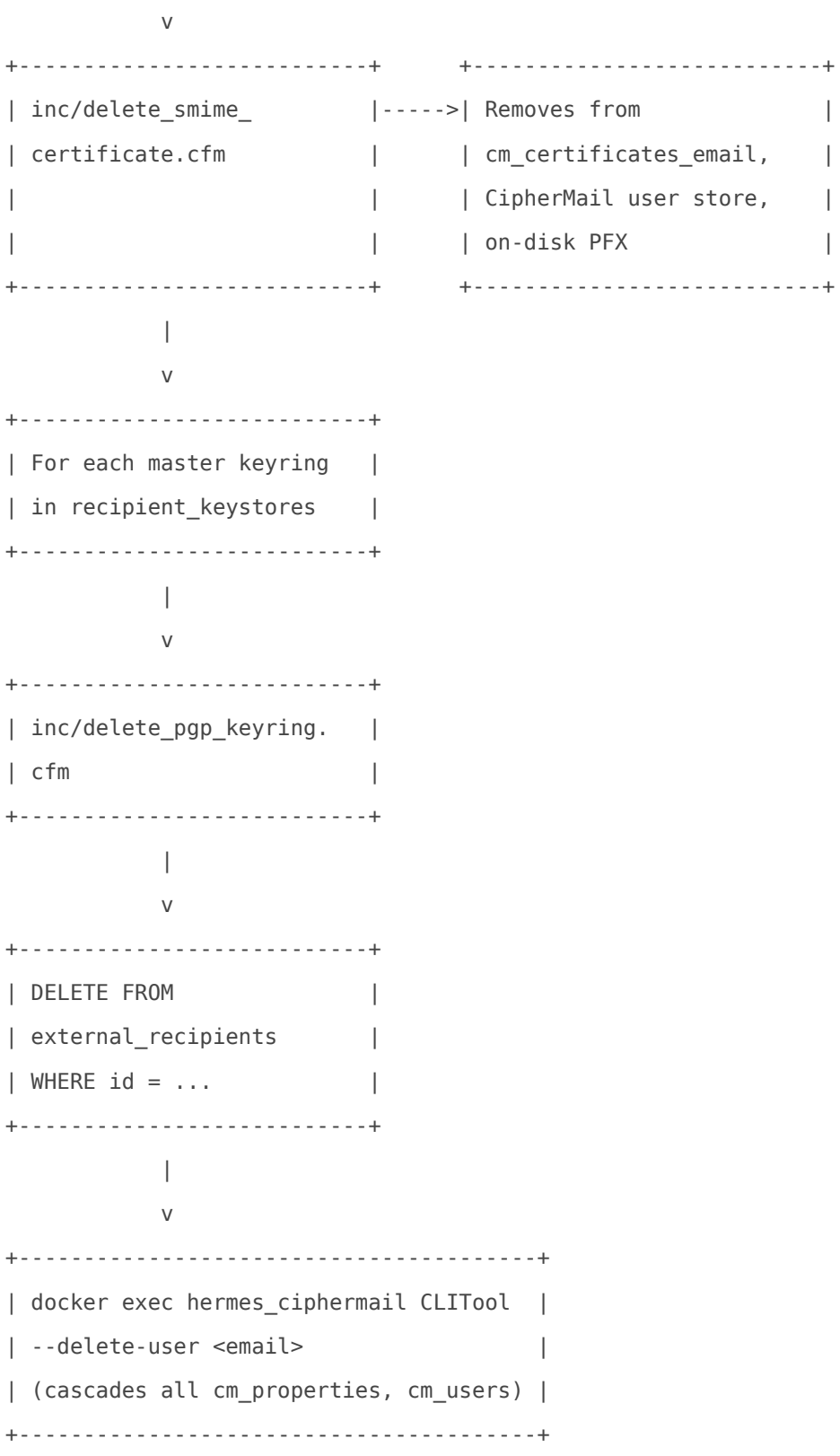
Action	Icon	Visible when	What it does
S/MIME Certificates	`fa-certificate` (green)	Admin row, `smime = 1`	Links to `view_ext_smime_certificates.cfm?email=...` for cert add / delete / send
PGP Keyrings	`fa-key` (blue)	Admin row, `pgp = 1`	Links to `view_ext_pgp_keyrings.cfm?email=...` for keyring add / delete / publish
Reset PDF Password	`fa-file-pdf` (yellow)	Admin row, `pdf = 1` AND `pdf_mode = static`	Opens modal; auto-generates a 16-char mixed-case-alphanumeric password client-side via `generatePassword(16)`; submits to `inc/reset_pdf_password.cfm`
Reset Portal Password	`fa-lock` (grey)	Admin row, `pdf = 1` AND `pdf_mode = random`	Opens modal; same 16-char generator; submits to `inc/reset_portal_password.cfm` (two-step: encode via `--encode-password`, then set `user.portal.password`)
Delete Recipient	`fa-trash-alt` (red)	Every row	Confirms, then submits to `delete_recipient` handler

The Cert Expiry column derives from a batch join of `cm_certificates_email + cm_certificates`, picking the **earliest** `cm_not_after` across all certs for that recipient. Color coding: red bold (already expired), yellow bold (within 30 days), grey muted (more than 30 days).

Delete cascade

Deleting an external recipient is a multi-table operation handled by `inc/delete_ext_recipient.cfm`:

```
+--------------------------+
| For each row in          |
| recipient_certificates   |
| where user_id = recipient |
+--------------------------+
            |
```



On success the page surfaces a callout reminding the operator that any **Sender Checks Bypass** mapping tied to this recipient must be re-created — that relationship is not auto-cascaded.

Password reset specifics

PDF static password reset (`inc/reset_pdf_password.cfm`):

1. Writes a one-liner `CLITool --set-property user.password --value <newpass> --encrypt --email <recipient>` to `/opt/hermes/tmp/<token>_reset_pdf_password.sh`.
2. `chmod +x`, executes (240s timeout), deletes.
3. AES-encrypts the new password with `/opt/hermes/keys/hermes.key` and UPDATEs `external_recipients.pdf_password` so the admin re-display path still works.

Portal password reset (`inc/reset_portal_password.cfm`) is **two-step** because CipherMail's portal password is stored as an encoded value, not the raw string:

1. **Step 1 — encode:** runs `CLITool --encode-password <newpass>`, captures stdout to `/opt/hermes/tmp/<token>_portal_password`, reads that file back into CFML, deletes the temp file.
2. **Step 2 — set:** runs `CLITool --set-property user.portal.password --encrypt --email <recipient> --value <encoded>` to push the encoded value into CipherMail.

Both modals auto-generate a 16-character mixed-case-alphanumeric password client-side and pre-populate the hidden confirm field; the operator can regenerate or type-in their own. Min length 12 is enforced server-side; the regenerator produces 16.

The modal text explicitly notes that **unencrypted voice calls and texts are NOT considered secure** for relaying the password to the recipient — operators are expected to use Signal, an in-person exchange, or a separately-encrypted channel.

CipherMail integration: every action is docker exec

Every CipherMail-side mutation on this page uses the same pattern documented across the Hermes admin:

The Hermes app container (`hermes_commandbox`) holds no JVM and no CipherMail libraries; everything reaches into `hermes_ciphermail` over the docker socket via `CLITool`. The temp-script pattern (write + chmod + execute + delete) survives the Lucee `cfexecute` quirks around stderr and quoting that would otherwise make a direct inline invocation unreliable.

What's NOT on this page

Expectation	Where it actually lives
Per-recipient cipher / algorithm selection (AES-128 vs AES-256, RSA / EC)	CipherMail Advanced Settings (`/ciphermail/`); per-recipient overrides live in `cm_properties` directly
Auto-lookup of recipient PGP keys from a keyserver at send time	Not implemented; see PGP Key Servers — that page is publish-only. Keys must be uploaded manually on the PGP Keyrings sub-page
Auto-lookup of recipient S/MIME certs via LDAP / public directory	Not implemented; certs must be uploaded manually on the S/MIME Certificates sub-page, OR minted from an Internal CA row and sent to the recipient
Per-recipient subject-trigger keyword override	Not implemented; the keyword is global (one row in `encryption_settings`)
Recipient-side enrollment / self-service for their own keys	The Secure Email Portal handles recipient password registration for PDF-random mode; there is no self-service cert / PGP upload UI
Bulk import from CSV with mixed protocols	Bulk path is PDF-only by design (S/MIME / PGP need per-recipient material that doesn't bulk-import cleanly)
Sender-side "force encrypt for this thread" UI	Senders use the subject trigger; there is no per-mailbox sender UI

Container and database touch-points

Component	Container / path	Role
Page	`config/hermes/var/www/html/admin/2/view_ext_rec_encryption.cfm` (`hermes_commandbox`)	List, filter, password resets, delete
Create page	`view_create_ext_recipient.cfm` + sub-pages for cert / keyring management	Single + bulk insertion
Action includes	`inc/create_ext_recipient.cfm`, `inc/delete_ext_recipient.cfm`, `inc/reset_pdf_password.cfm`, `inc/reset_portal_password.cfm`	One-liner CLITool dispatchers via temp script
Admin metadata	`external_recipients` in `hermes` DB (`hermes_db_server`)	Per-recipient policy choices + AES-encrypted static PDF password copy
CipherMail user store	`cm_users`, `cm_properties` in `djigzo` DB	Authoritative per-recipient state
CipherMail cert / key index	`cm_certificates_email`, `cm_certificates`, `cm_keyring_email` in `djigzo` DB	Joined batch into `smimeLookup` / `pgpLookup` for column rendering
Encryption engine	`hermes_ciphermail` (Java; CipherMail Community 5.x branded `djigzo`)	Actual S/MIME / PGP / PDF encryption + portal back-channel
AES key	`/opt/hermes/keys/hermes.key` (`hermes_commandbox` bind mount)	Encrypts `pdf_password` for re-display
Secure Email Portal	`https://<console.host>/web/portal/` (served by `hermes_ciphermail`)	Recipient-facing landing page for PDF random + portal account flows

Related

- Encryption Settings — global Subject Trigger, PDF reply sender, three shared secrets; the policy mechanism this page applies per recipient
- Internal CA — where the private CAs that can mint S/MIME certs for these recipients live (operator-issued S/MIME chain delivered out-of-band)
- PGP Key Servers — the publish list for the Publish action on the PGP Keyrings sub-page (note: publish-only, not lookup)
- System Certificates — distinct TLS cert store; not related to message-content S/MIME

- Disclaimers — body-mod ordering vs the CipherMail encryption pass (disclaimer is appended before encryption wraps the message)
- Organizational Signatures — same milter ordering applies to signature injection
- ARC Settings — same milter-ordering pattern applied to inbound chain sealing
- **Advanced Settings** (sidebar link to `/ciphermail/`) — CipherMail's own admin UI for everything not surfaced here (per-recipient cipher tuning, custom DLP, direct `cm_properties` editing)

Internal CA

Internal CA

Admin path: **Encryption > Internal CA** (`view_internal_ca.cfm`, `inc/download_ca_file.cfm`, `inc/create_certificate.cfm`, `inc/send_smime_certificate.cfm`, `inc/delete_smime_certificate.cfm`).

This is the **gateway's built-in Certificate Authority** for issuing S/MIME certificates to local users and relay recipients. Each CA row here corresponds to a private CA cert + key on disk under `/opt/hermes/CA/<directory>/root_ca/` and a matching `roots`-store entry in the CipherMail (`djigzo`) trust list. Per-recipient S/MIME certs minted from a CA on this page are stored in `recipient_certificates` and listed on Email Server > Relay Recipients (and Email Server > Mailboxes when S/MIME is enabled on a mailbox).

This page is **distinct from** System Certificates:

	System Certificates	Internal CA
What it stores	Operator-uploaded TLS leaf certs (nginx, Postfix, Dovecot)	Private CAs that *mint* S/MIME end-user certs
Trust direction	Hermes presents these to clients	Hermes issues certs that *recipients* present
Backing store	`system_certificates` table + `/opt/hermes/ssl/` or `/etc/letsencrypt/`	`ca_settings` table + `/opt/hermes/CA/<dir>/` + CipherMail `cm_certificates` (`roots` store) + `cm_ctl` trust list
Typical lifetime	90 d (ACME) or 1-3 yr (commercial)	5 yr root (recommended), extendable in place
Lifecycle owner	nginx / Postfix / Dovecot via TLS handshake	CipherMail S/MIME signer / encryptor for outbound; per-recipient cert issuance for inbound encrypt

The two ingest paths

The page exposes two collapsing cards (**Create Internal CA**, **Import External CA**) plus a DataTable of existing CAs. Both paths land a row in `ca_settings` and register the cert in CipherMail's `cm_certificates` table as a root (`cm_store_name = 'roots'`) plus an entry in `cm_ctl`

(Certificate Trust List) flagged `whitelisted`.

1. Create Internal CA

Operator fills the DN fields, picks a key size (2048 / 4096) and a validity (1-5 years; 5 years recommended). Hermes:

1. Validates inputs (regex-restricted character set per field, 2-char ISO country code, uniqueness against `ca_settings.ca_commonname`).
2. Materializes a per-CA on-disk skeleton at `/opt/hermes/CA/<sanitized-cn>/root_ca/` with the standard OpenSSL layout (`certs/`, `crl/`, `newcerts/`, `private/`, `requests/`, `PFX/`, `serial`, `index.txt`, `crlnumber`).
3. Materializes an `openssl.cnf` from `/opt/hermes/templates/rootca_openssl.cnf` with the directory placeholder substituted.
4. Snapshots `cm_certificates` into `cm_certificates_tmp`, runs the OpenSSL root-CA generation script as a one-shot temp script (`/opt/hermes/scripts/<token>_create_ca.sh`), then diffs to find the new cert.
5. Marks the new CipherMail row `cm_store_name = 'roots'`, inserts a `cm_ctl` row with status `whitelisted` and `allowExpired = false`, and back-fills `ca_settings.ca_djigzo_id` + `ca_djigzo_subject`.

2. Import External CA

For organizations that already have a private CA (commercial issuer, internal PKI, prior Hermes install). Operator uploads the **CA cert** (PEM) and the **CA private key** (PEM, unencrypted). Hermes:

1. Lands the files at `/opt/hermes/CA/<sanitized-cn>/root_ca/certs/cacert.pem` and `.../private/cakey.pem`.
2. Runs an OpenSSL validation script that checks:
   - Cert parses as X.509 (`openssl x509 -modulus`)
   - Key parses as RSA (`openssl rsa -modulus`)
   - Cert and key moduli match (private key matches public key)
   - Cert has `CA:TRUE` basic constraint
3. On any check failure the upload directory is removed and the operator gets a specific error alert (m=48 / 49 / 50 / 51).
4. Generates `openssl.cnf` from the template + `cachain.pem` = copy of the cert (needed for later PFX export of per-user certs).
5. Pipes the cert into CipherMail via `docker exec -i hermes_ciphermail /usr/bin/java -cp '/usr/share/djigzo/lib/*' mitm.application.djigzo.tools.CertStore --import-certificates` and back-fills the `ca_djigzo_id` exactly as the Create path does.

The Import path is the only way to migrate a CA that already has issued certs in the wild — re-creating a CA from scratch with the same DN does NOT reproduce the original key material, so

previously issued certs would not chain to it.

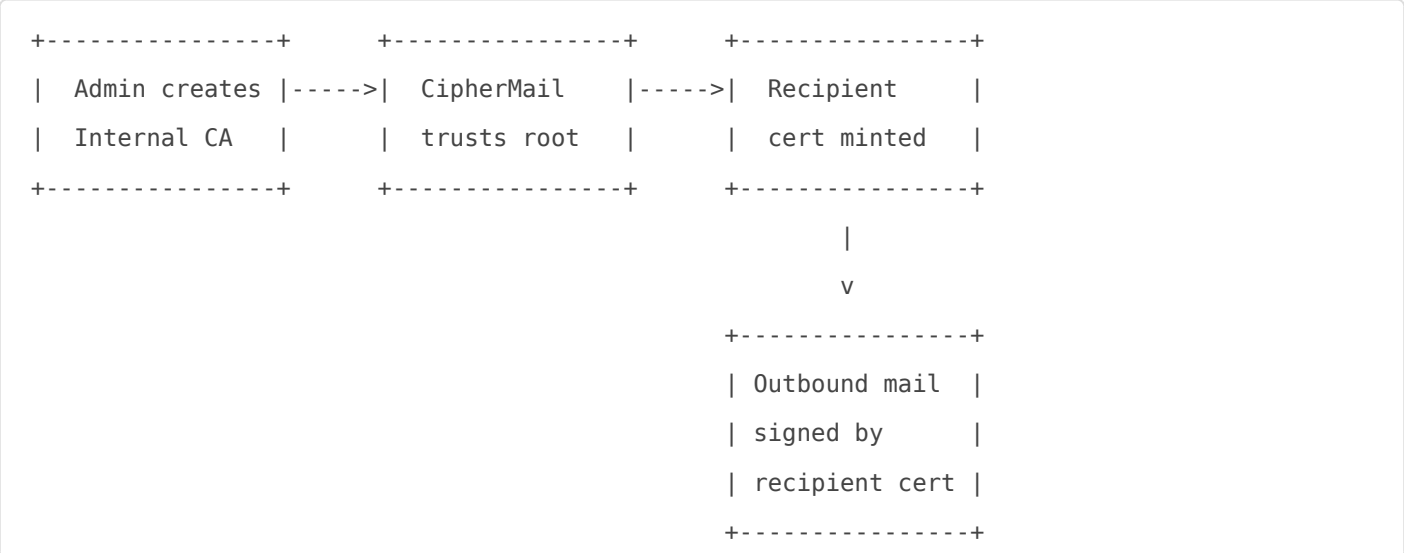
Default CA flag (`default2`)

Exactly one row in `ca_settings` has `default2 = '1'`; all others have `'2'`. The default CA is the one Hermes mints from when an admin clicks **Create Certificate** for a recipient on Email Server > Relay Recipients (or the mailbox equivalent) without explicitly choosing a CA. The page enforces single-default by:

- Setting all rows to `'2'` before flipping the new row to `'1'`
- Forcing the first-ever CA to default regardless of the checkbox
- Disabling the **Make Default** checkbox on the Create card when no default exists (forced default is implicit)

The DataTable Default column renders a green **YES** badge for the default row and a one-click **set default** button for the others.

CA lifecycle workflow



Stage	Where the data lives	Trigger
CA root created	`ca_settings` + `/opt/hermes/CA/<dir>/` + `cm_certificates` (`roots`) + `cm_ctl` (`whitelisted`)	Create / Import buttons on this page
Per-recipient cert minted	`recipient_certificates` (or `external_recipient_certificates`) + CipherMail user store	**Create Certificate** button on a recipient page; uses `default2 = '1'` CA unless overridden

Stage	Where the data lives	Trigger
Cert self-introduction	Bundled into the first signed outbound message the recipient sends	Automatic on first S/MIME-signed send
Cert revocation	`delete_smime_certificate.cfm` removes the row + CipherMail entry; CRL is maintained by CipherMail's own scheduled job	**Delete** button on the recipient cert row
CA renewal	Re-sign the existing cert + key with `openssl x509 -days <N>` and re-import into CipherMail; `ca_settings.expires` updated	**Renew** button (sync icon) on the CA row
CA deletion	Refused if any `recipient_certificates.ca_id` row references it; otherwise removes DB row + CipherMail `cm_certificates` / `cm_ctl` + on-disk tree	**Delete** button (only enabled when zero issued certs)

CA Renewal: 5-year extension in place

Clicking the **Renew** (sync) icon does NOT generate a new key pair — it re-signs the existing CA cert against its own key with an extended `notAfter`. The math:

```
new_expires = current_expires + 5 years
days_param  = max(1825, days_from_now_to_new_expires)
openssl x509 -in cacert.pem -days <days_param> -out cacert.pem.new -signkey cakey.pem
mv cacert.pem.new cacert.pem
cp cacert.pem cachain.pem
cat cacert.pem | docker exec -i hermes_ciphermail \
  /usr/bin/java -cp '/usr/share/djigzo/lib/*' \
  mitm.application.djigzo.tools.CertStore --import-certificates
```

Because the key stays the same, every previously issued recipient cert **still chains to a valid CA cert** — there is no need to re-mint recipient certs after a CA renewal. This is the operator-friendly path: recipients on the outside who already trust the CA root continue to trust it transparently.

The old CipherMail row is deleted and the renewed cert re-imported so the `cm_certificates` / `cm_ctl` rows reflect the new validity window (otherwise CipherMail would keep enforcing the old expiry).

Trust distribution to external recipients

A Hermes-issued S/MIME cert is signed by a private CA that **no operating system or mail client trusts by default**. External recipients see Hermes-signed mail as "signed by an unknown CA" until they explicitly install the Internal CA root in their trust store.

Two practical paths:

Path	Effort	Reach
Operator distributes the CA root out-of-band (download from this page, email or publish on a portal, recipient installs in Outlook / macOS Keychain / iOS Profile / Thunderbird)	Manual per recipient	Small fixed counterparty set (B2B, partner orgs)
Issue recipient certs from a publicly-rooted CA (commercial S/MIME issuer signs your CA, or you buy per-user S/MIME certs from a public issuer)	One-time cross-sign or per-user cost	Every MUA on the planet trusts the chain

For most Hermes deployments the Internal CA is the right answer (per-user public S/MIME costs $20-$80/yr/user); for high-volume B2C senders the publicly-rooted route is sometimes worth the cost.

Hermes does not generate a CRL distribution URL on this page; CipherMail maintains the revocation list internally and applies it when verifying inbound S/MIME from local recipients. External recipients have no automatic way to consume the CRL — revocation is effectively local-only unless the operator publishes the CRL manually.

CA file downloads (gated)

Each row's action column exposes a **Download Certificate** and **Download Private Key** button. These are **disabled by default** — downloading a CA private key off a web console is a high-risk operation. To enable, set

```
ALLOW_CA_DOWNLOAD=yes
```

in `/opt/hermes/config/security.conf` on the host filesystem. This is the same toggle pattern used by System Certificates (`ALLOW_CERT_DOWNLOAD`) — read on every page load, surfaced as a disabled-

button + tooltip when off. When enabled, downloads stream via a hidden iframe (`<iframe id="caDownloadFrame">`) so the page preloader doesn't get stuck.

Body-modification interaction with S/MIME

CipherMail-side S/MIME signing happens **after** the `hermes_body_milter` disclaimer / signature / banner insertion (see Disclaimers — Behavior with S/MIME, PGP, and DKIM-signed mail). That means outbound mail signed by an Internal-CA-minted recipient cert covers the **final body** the recipient sees — including any disclaimer or banner Hermes appended. The body milter passes already-S/MIME-signed mail through untouched, so end-to-end MUA-signed mail (Outlook + per-user S/MIME) is never re-signed or invalidated.

This is the same ordering rationale that drives ARC sealing placement (see ARC Settings — Container and milter placement): the cryptographic envelope is the last thing applied so it always matches the bytes the recipient sees.

Container and database touch-points

Component	Container / path	Role
Page	`config/hermes/var/www/html/admin/2/view_internal_ca.cfm` (`hermes_commandbox`)	CRUD + DataTable + action router
CA tree	`/opt/hermes/CA/<sanitized-cn>/root_ca/` (`hermes_commandbox` bind mount)	OpenSSL working tree per CA
Templates	`/opt/hermes/templates/rootca_openssl.cnf` + `/opt/hermes/scripts/create_ca.sh`	Placeholder-substituted at create time
Trust store	`cm_certificates` + `cm_ctl` + `cm_ctl_cm_name_values` in `djigzo` DB (`hermes_db_server`)	CipherMail's view of the root CA
Engine	`hermes_ciphermail` (Java; CipherMail Community 5.x branded `djigzo`)	Signing / encryption / decryption engine; reached via `docker exec -i hermes_ciphermail /usr/bin/java -cp '/usr/share/djigzo/lib/*' mitm.application.djigzo.tools.CertStore`

Component	Container / path	Role
Recipient certs	`recipient_certificates` + `external_recipient_certificates` in `hermes` DB	One-row-per-user, joined to a CA via `ca_id`
Security toggle	`/opt/hermes/config/security.conf` on host	`ALLOW_CA_DOWNLOAD=yes` to expose cert/key download buttons

Every CipherMail interaction is **temp-script + docker exec** rather than direct invocation — the `hermes_commandbox` container has no JVM of its own; the CipherMail Java tooling lives in `hermes_ciphermail` and is reached over the docker socket.

Related

- Encryption Settings — outbound encryption policy (force / opportunistic / off); decides whether Hermes signs at all and whether it falls back to plaintext when no recipient key is available
- External Recipients — per-counterparty key store; `external_recipient_certificates` rows pair with Internal-CA-issued or externally-issued S/MIME chains
- PGP Key Servers — sibling page for the PGP side of recipient key distribution
- System Certificates — distinct TLS cert store for nginx / Postfix / Dovecot
- Disclaimers — body-modification ordering vs the S/MIME signer
- ARC Settings — same milter-ordering pattern applied to inbound chain sealing
- **Advanced Settings** (sidebar link to `/ciphermail/`) — CipherMail's own admin UI, exposed for deep operations (CRL publishing, per-user policy tuning) not surfaced in the Hermes admin

PGP Key Servers

PGP Key Servers

Admin path: **Encryption > PGP Key Servers** (`view_pgp_key_servers.cfm`, `inc/publish_pgp_keyring.cfm`).

This page maintains the **HKP keyserver publish list** — the set of public OpenPGP keyservers Hermes will push (`gpg --send-keys`) recipient public keys to when an admin clicks **Publish** on a keyring row in **Encryption > External Recipients**. Each row is a hostname only (no scheme, no port, no path) stored in the `pgp_keyservers` table.

> **Important: publish, not lookup.** Despite the page name, the keyserver list is currently **outbound-only**. Hermes does NOT auto-query these servers to fetch a recipient's PGP key at send time — recipient keys must be imported manually (paste-in or file upload) on **Encryption > External Recipients > PGP Keyrings**. The keyservers configured here are used solely by the **Publish** action in `inc/publish_pgp_keyring.cfm`, which pushes a key the operator already holds (typically the local CipherMail server's public key or a recipient's key that was imported and now needs broader distribution).

What the page does

The page is a thin CRUD over a 3-column table:

`pgp_keyservers` column	Purpose
`id`	PK
`keyserver`	Hostname only, e.g. `keys.openpgp.org`
`note`	Free-text label, e.g. "Primary keyserver"

Three actions:

Action	Form value	Effect

Add	`action=add`	Validates hostname via `IsValid("email", "bob@" & ks)` (rejects URLs and `host:port`), checks for duplicate `keyserver`, INSERTs the row
Single delete	`action=delete` with `delete_id`	DELETE one row by id
Bulk delete	`action=bulk_delete` with `selected_ids` (CSV)	DELETE every selected id in a loop

The existing-servers card is a DataTable with select-all + per-row checkboxes + a **Delete Selected** button. There is no per-row enable flag, no protocol/port column, no priority ordering — every row in the table is offered as a publish target in the modal on the keyring page, indexed by `id`.

What "publish" actually runs

When the operator clicks **Publish** on a keyring row at **External Recipients > PGP Keyrings**, the `publish_pgp_keyring.cfm` include does the following for each selected keyserver:

```
/usr/bin/gpg --homedir /opt/hermes/.gnupg/ \
             --keyserver <hostname-from-pgp_keyservers> \
             --send-keys <recipient-PGP-key-id>
```

The temp script is written to `/opt/hermes/tmp/<token>_publish_pgp_key.sh`, chmod'd, executed, and deleted. The standard Hermes temp-script pattern. The keyserver hostname is substituted via `REReplace` of the `THE-KEY-SERVER` placeholder in `/opt/hermes/scripts/publish_pgp_key.sh`.

GPG itself picks the protocol — `gpg` defaults to `hkps://` (HKP over TLS on tcp/443) for a bare hostname when the local `dirmngr` is configured for it; otherwise it falls back to `hkp://` (tcp/11371). Hermes does not pass an explicit scheme.

Failure modes the include recognizes (sets `session.m` and redirects):

GPG stderr fragment	Meaning	`session.m`
`Server indicated a failure`	Keyserver rejected the upload (rate limit, policy, malformed key)	22
`No name`	Local GPG keyring has no user-id matching the requested key id	23
`Not found`	Local GPG keyring does not hold the requested key id	24
`Not a key ID`	The key id parameter was malformed	25

A successful publish returns no recognized fragment and falls through to the success branch.

Recommended seed list

The default install seeds one row:

Hostname	Note
`keyserver.ubuntu.com`	Ubuntu SKS OpenPGP Public Key Server

Practical 2026 replacements / additions the operator should consider:

Hostname	Network	Caveats
`keys.openpgp.org`	Identity-verified standalone (Hagrid)	Strips third-party signatures (no web-of-trust); requires email verification before a key becomes searchable by email address; **does not distribute revocation certificates the way SKS did**
`keyserver.ubuntu.com`	SKS-style federated	Was the last reliable SKS-network bridge; survives but is no longer broadly federated
`pgp.mit.edu`	Legacy SKS	Largely defunct in 2026 — uploads may not propagate; leave off unless legacy compatibility is required
`<your-org-keyserver>`	Internal HKP daemon (e.g. Hagrid)	Useful if the operator runs an authoritative keyserver for their own domain — same publish path

The page does NOT validate keyserver reachability at add time; an unreachable host simply produces a publish failure when the operator clicks Publish later.

What is NOT on this page

Several things an operator might reasonably expect from a "PGP Key Servers" page that are intentionally elsewhere or absent:

Expectation	Where it actually lives
Per-server enable/disable toggle	Not implemented — every row is a publish target
Search-order priority	Not applicable — publish iterates the explicit selection from the modal, not the full list

Expectation	Where it actually lives
Inbound recipient-key auto-lookup at send time (`gpg --search-keys` / `recv-keys`)	Not implemented anywhere in Hermes; recipient keys must be imported manually on **External Recipients > PGP Keyrings**
Automatic refresh of imported keys (re-fetch + merge updates)	Not implemented; operators must re-import a key if a recipient rotates
DANE `OPENPGPKEY` DNS lookup	Not currently surfaced in the Hermes admin or CipherMail engine config
WKD (Web Key Directory) discovery at `https://<domain>/.well-known/openpgpkey/...`	Not currently surfaced in the Hermes admin or CipherMail engine config
HKP port override	Not on this page; GPG picks the port
Encryption policy decisions ("fail closed vs send plaintext if no key")	Encryption Settings, not here

The page is deliberately scoped to one job: **a list of HKP endpoints the publish flow can push to.**

When the operator should populate this list

Two practical scenarios:

1. **The organization wants its own gateway PGP key to be publicly discoverable.** Add the operator's preferred public keyserver(s), then publish the local CipherMail key from **External Recipients > PGP Keyrings**. External counterparties running `gpg --recv-keys` against the same keyserver can then pull it for encrypting mail back to Hermes-served users.
2. **A specific recipient has asked for their key (which the operator already holds locally) to be pushed somewhere centralized.** Less common — usually recipients self-publish — but the workflow supports it.

If the deployment never publishes keys outward (typical Community deployments that use S/MIME exclusively, or PGP deployments that exchange keys out-of-band via attachment), this page can remain empty with no functional impact.

Container and database touch-points

Component	Location	Role
Page	`config/hermes/var/www/html/admin/2/view_pgp_key_servers.cfm` (`hermes_commandbox`)	CRUD UI
Publish include	`config/hermes/var/www/html/admin/2/inc/publish_pgp_keyring.cfm` (`hermes_commandbox`)	Builds + runs the temp `gpg --send-keys` script
Template script	`/opt/hermes/scripts/publish_pgp_key.sh`	Single line: `/usr/bin/gpg --homedir /opt/hermes/.gnupg/ --keyserver THE-KEY-SERVER --send-keys THE_KEY_ID 2>&1`
GPG home	`/opt/hermes/.gnupg/` (bind-mounted into `hermes_commandbox`)	Local GPG keyring holding the keys eligible for publish
Storage	`pgp_keyservers` in `hermes` DB (`hermes_db_server`)	The list itself
Engine	`hermes_ciphermail` (separate from publish — handles actual signing/encryption at send time)	NOT touched by this page; this page only manages the GPG outbound-publish list

The publish flow runs **gpg on `hermes_commandbox`** (which has the `/opt/hermes/.gnupg/` keyring bind-mounted) — not inside `hermes_ciphermail`. CipherMail keeps its own per-recipient PGP store in the `djigzo` DB for actual encryption/decryption operations.

Related

- External Recipients — per-counterparty key store; the **Publish** action that consumes this list lives on the keyring sub-page there
- Encryption Settings — outbound encryption policy that decides whether absence of a recipient PGP key blocks the message or falls through to plaintext
- Internal CA — sibling page for the S/MIME side of recipient key issuance and trust
- **Advanced Settings** (sidebar link to `/ciphermail/`) — CipherMail's own admin UI for the deep PGP keyring operations the Hermes admin does not surface